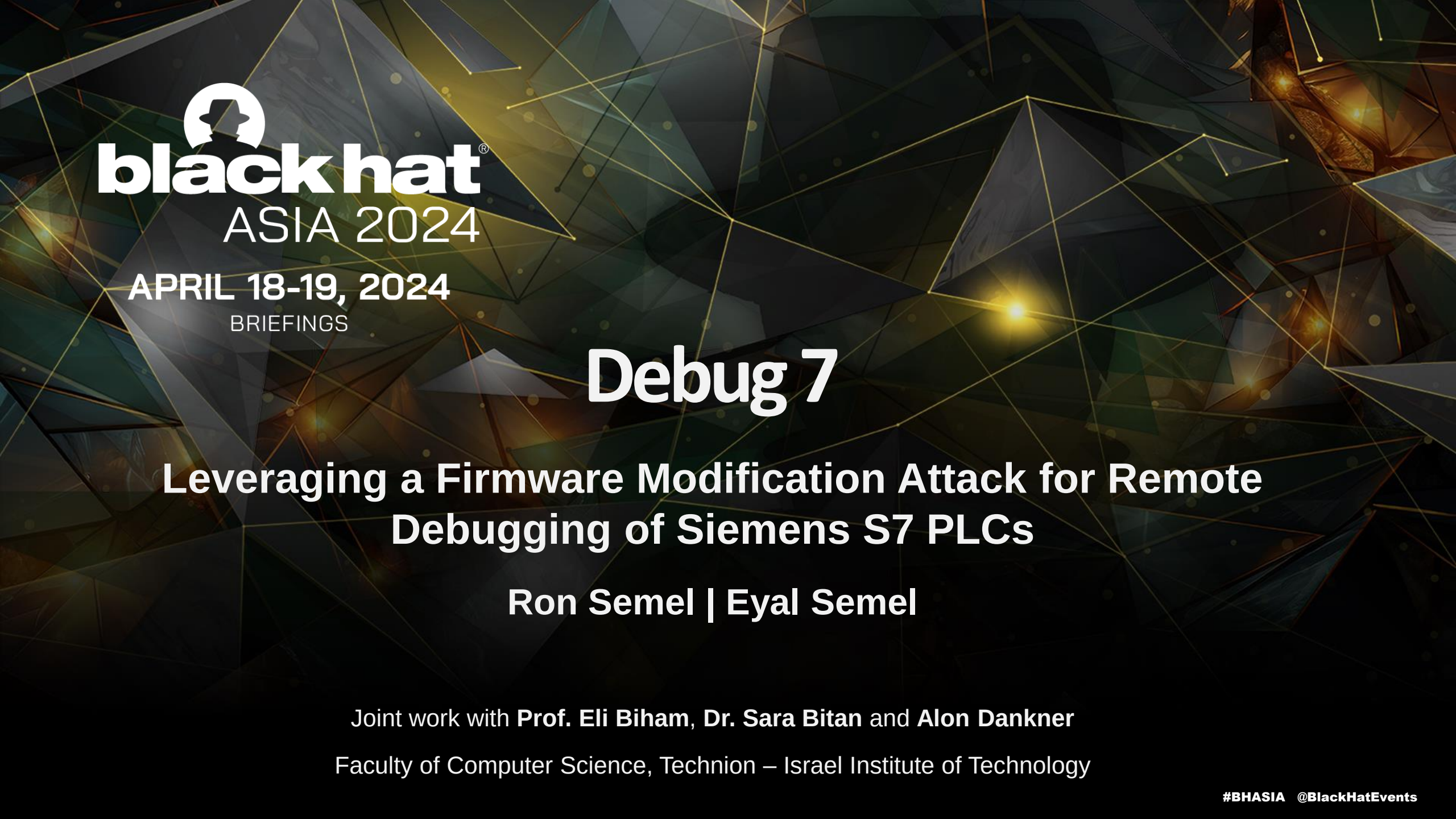


Debug 7

Leveraging a Firmware Modification Attack for Remote Debugging of Siemens S7 PLCs

Ron Semel | Eyal Semel

Joint work with **Prof. Eli Biham**, **Dr. Sara Bitan** and **Alon Dankner**

Faculty of Computer Science, Technion – Israel Institute of Technology

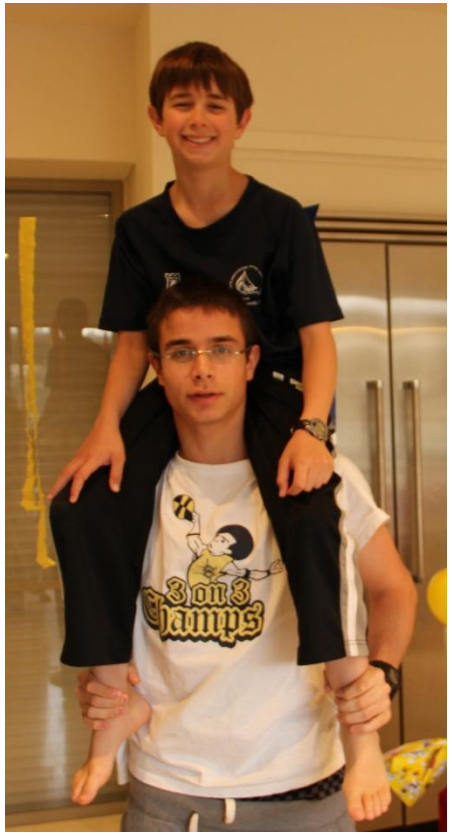
Who Are We?

Ron Semel

Software engineer at Microsoft
Microsoft Defender for Endpoint (MDE)

Security researcher
Computer science faculty, Technion

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



Eyal Semel

Security researcher
Computer science faculty, Technion

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

Talk Topics

- Introduction and Previous Research
- Runtime Manipulation of Siemens S7 PLCs Firmware
- Implementation of Debug 7 - a Remote Debugger for Siemens S7 PLCs
- Debugger Video Demo
- Conclusions

The 4th Industrial Revolution – Industry 4.0

- Can anyone imagine life without:

Drinking water

Transportation

Food

Amazon

- Our necessities are made accessible via automated industrial control systems.

Wastewater treatment plants purify water.

Complex signaling systems manage traffic.

Food is grown using automatic irrigation systems.

Automated warehouses manage our online purchases

The 4th Industrial Revolution – Industry 4.0

- These smart control systems include:
  - Mass integration of IOT devices.
  - Extensive cloud communication.
  - Smart automation

All these cool new features come with risks…

Attacks on Critical Infrastructure

- Cyber attacks on critical infrastructure can be catastrophic!

MIT Technology Review

https://www.technologyreview.com/2019/03/05/103328/cybersecurity-critical-infrastructure-triton-malware/

- We have a great responsibility securing these systems!

PLC – Programmable Logic Controller

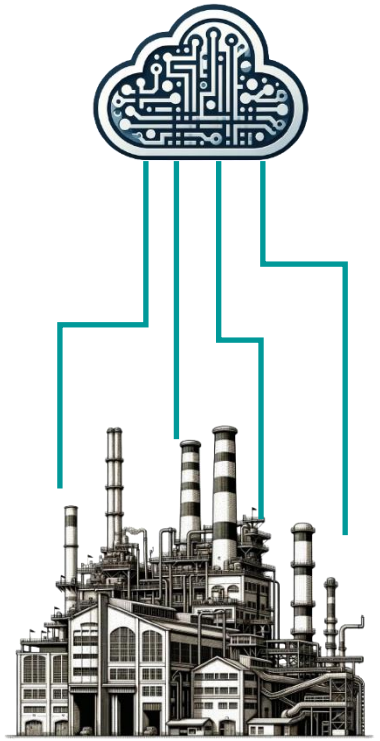
PLCs are rugged computers used for industrial automation.

- They are the core component of an ICS.

They read input data from field devices such as sensors.

- Outputs are triggered based on pre-programmed code.

A bridge between the virtual world and the physical world.



Everybody Loves Ice Cream

Industrial Control System Example

The PLC collects data from the industrial pot:

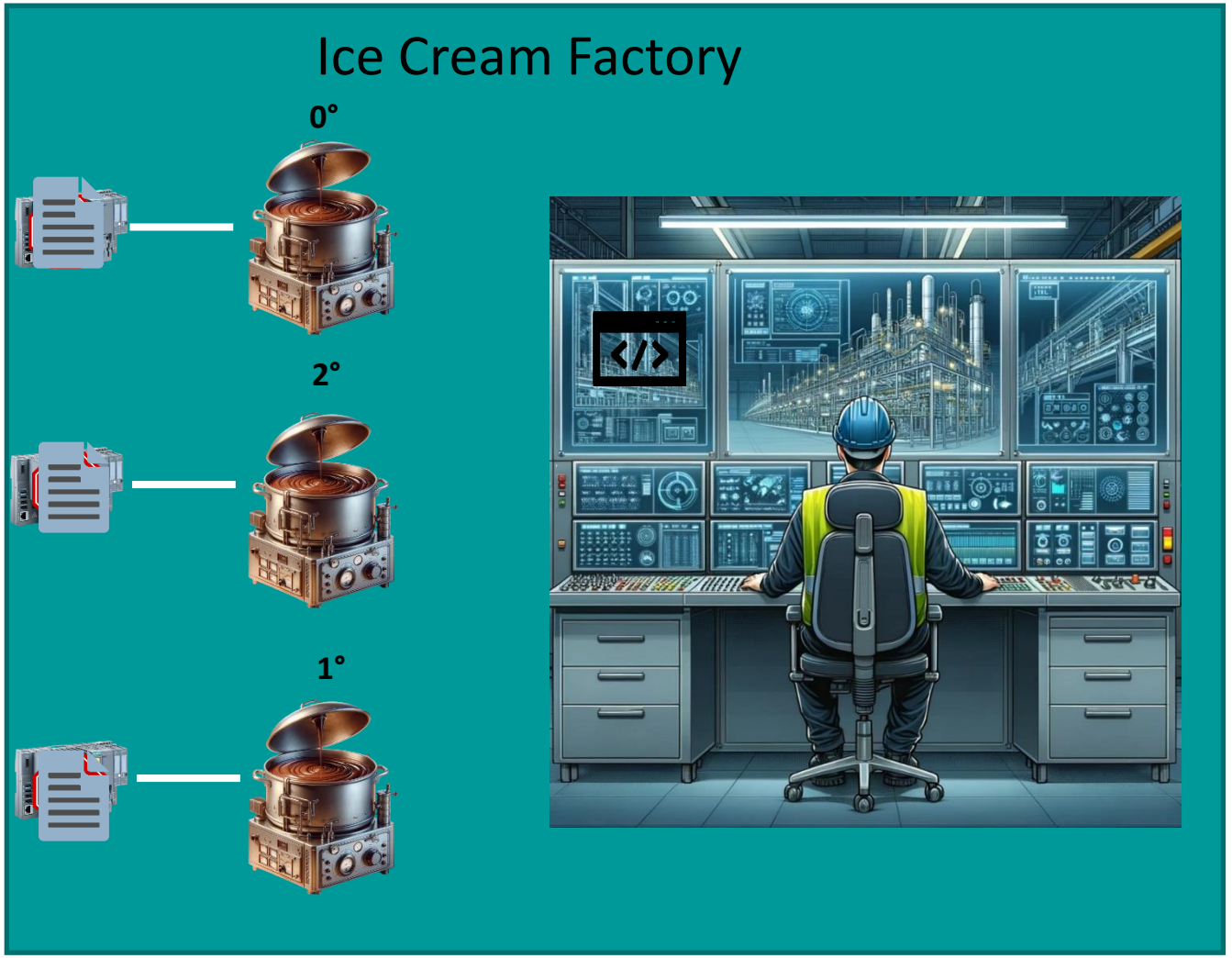
- **Temperature**
- **Etc.**

Based on the sensor input, the PLC commands the tubs:

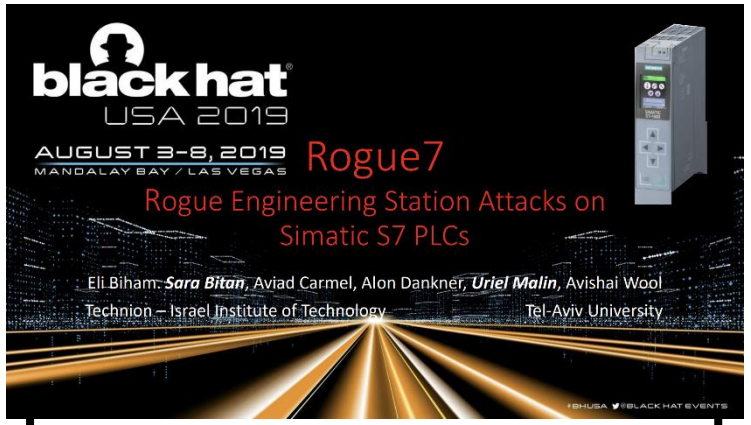
- **Start**
- **Stop**
- **Etc.**

The statuses of the tubs can be viewed in the engineering workstation.

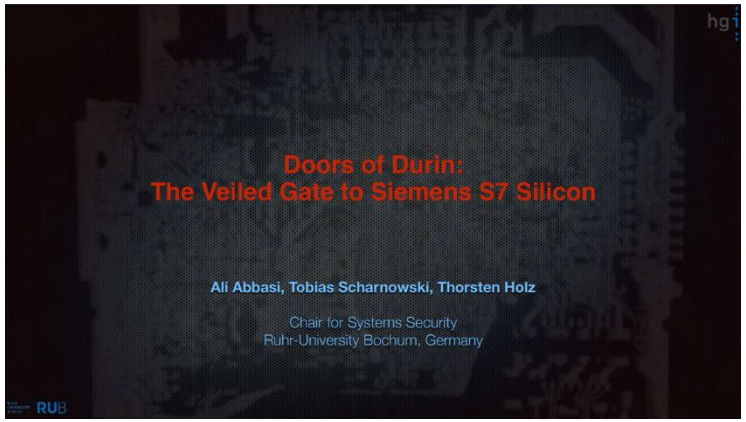
The custom control logic can be updated remotely, via the engineering workstation.



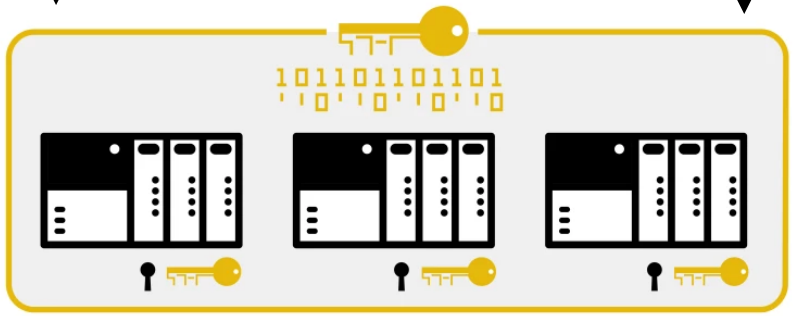
Previous Research



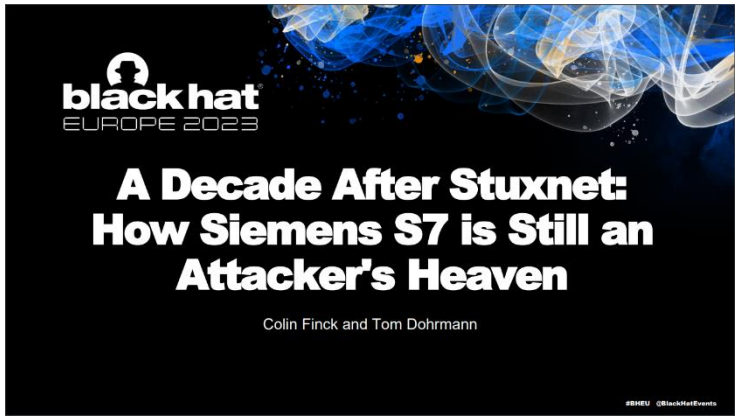
Biham et al.



Abbasi et al.



Team82 at Claroty



Colin Finck and Tom Dohrmann

The Siemens ET 200SP PLC Open Controller

- PLCs are the main target of attacks on critical infrastructure.
  - We focused on the largest PLC vendor – Siemens.
  - More specifically the **Siemens ET 200SP PLC Open Controller**.
  - One of the leading PLCs in the market.
  - It runs on standard hardware - an Intel Atom CPU with 4 cores.

- It includes a software PLC – the **S7-1500 Software Controller**:
  - It's a software application that simulates the functionality of a hardware PLC.
  - From now on we will call this software PLC – *SWCPU*.

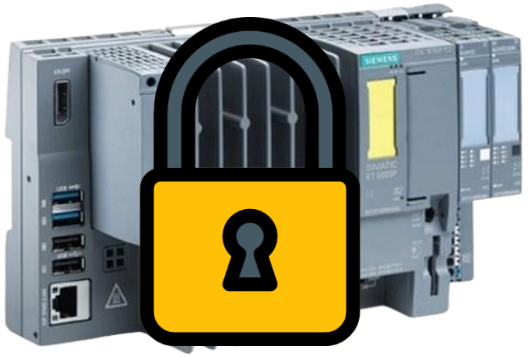
Top Secret

For years, Siemens kept the firmware of their S7 PLC's a secret!

They invested a lot of resources in IP protection:

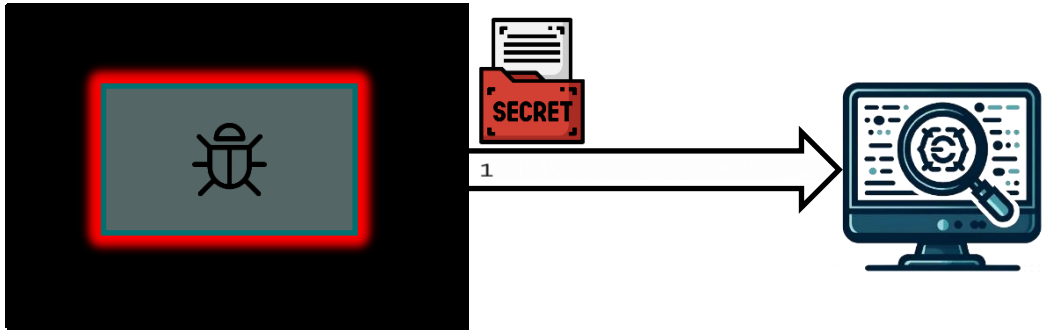
- The S7-1200 PLC self-destructs if a watchdog discovers a core was halted via the JTAG interface.
  - Thomas Weber, Hack In The Box, 2019
  - Ali Abbasi, CS3STHLM, 2020

- The SWCPU is encrypted on the open controller.
  - Soft7, Biham et al., 2022

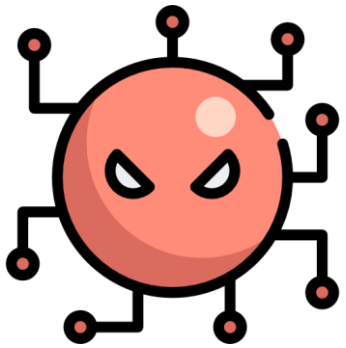


A Revolution in Siemens' S7 PLC Research

- For the first time S7 PLC history:
  - We remove from the Siemens' S7 PLCs from their many layers of obscurity.
  - We expose a powerful remote tool to dynamically analyze their firmware.
  - We can easily expose secrets kept hidden for years.



- As a byproduct, we installed persistent malware on it.
  - Which communicates with a malicious command and control server (C2), to enable convenient communication with the installed implant.



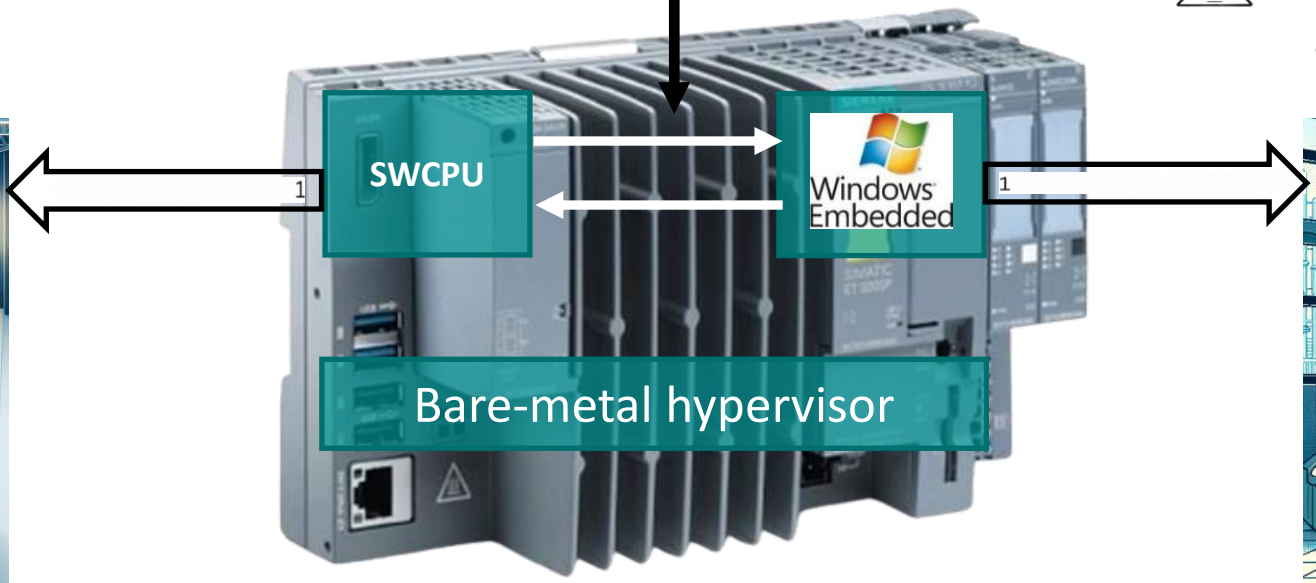
The PLC's Architecture

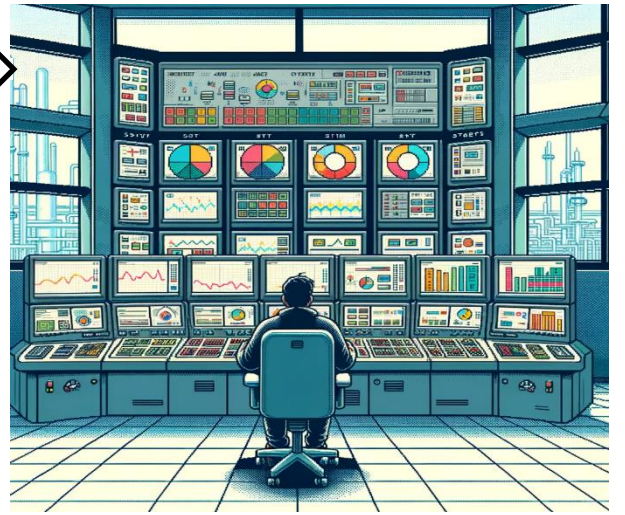
Communicates with the **field devices**

Inter OS communication

Communicates with the:

- **Engineering workstation**
- **Cloud platform**



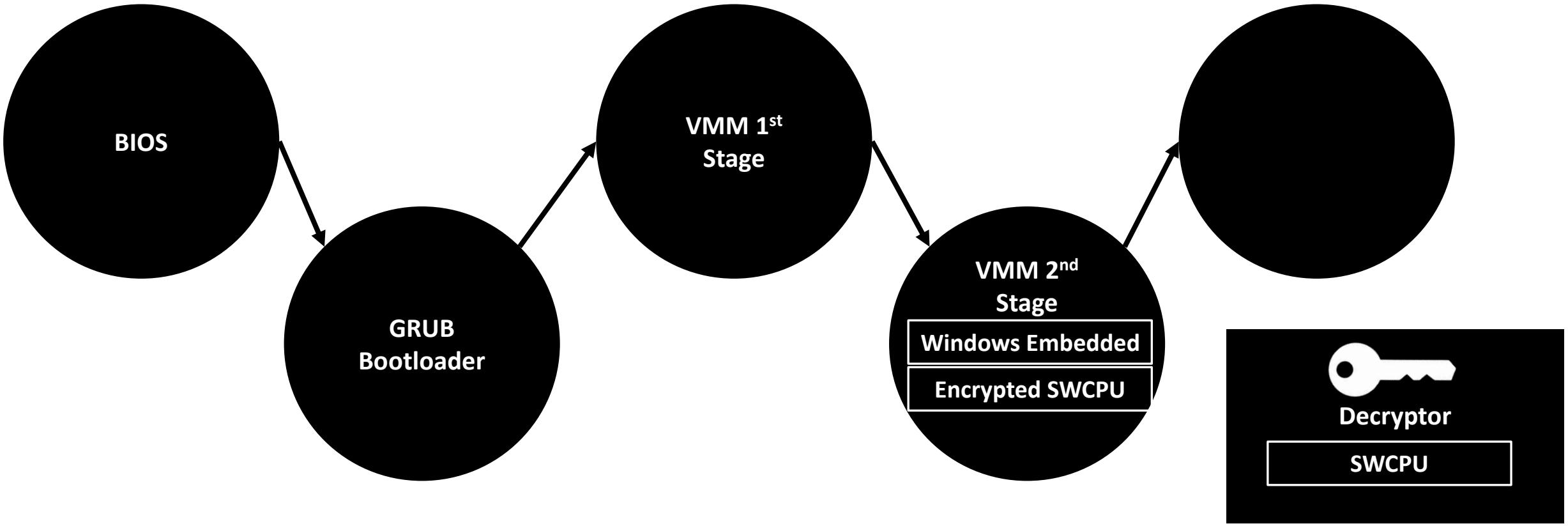


The isolation by virtualization, between the two guest OSs is an important **layer of security.**

Even if the Windows OS is compromised, the field devices should remain safe.

Soft7, Eli Biham et al., 2022

Boot Process

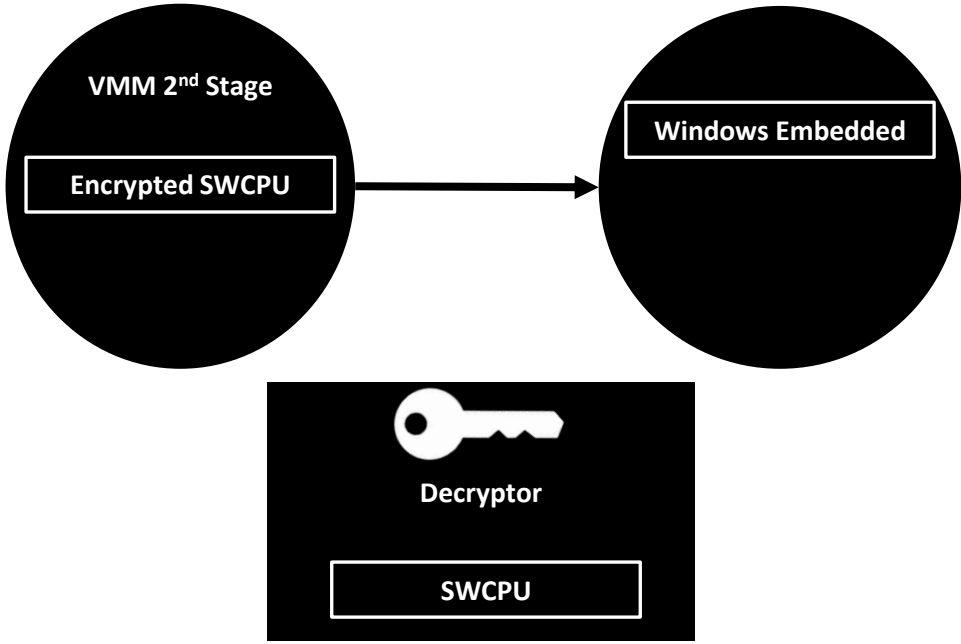


Soft7, Eli Biham et al., 2022

Previous Research

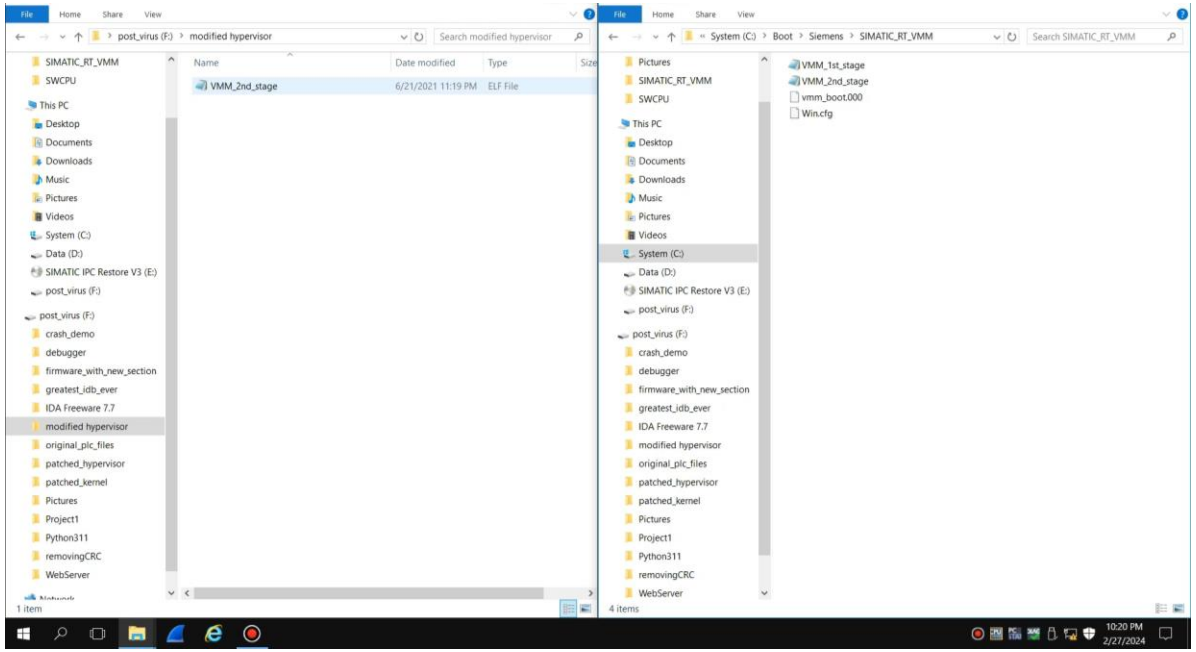
- Our research continues where Soft 7 ended.
  - Black Hat USA 22', Soft7, Biham et al.

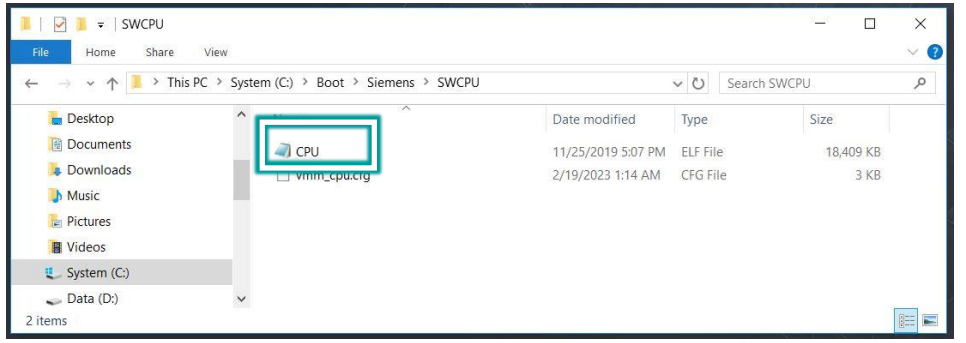
- Remember that the hypervisor decrypts the SWCPU and loads it into memory?
  - The Soft7 team, extracted the plaintext SWCPU firmware file.
  - We could finally start studying the SWCPU's assembly code!

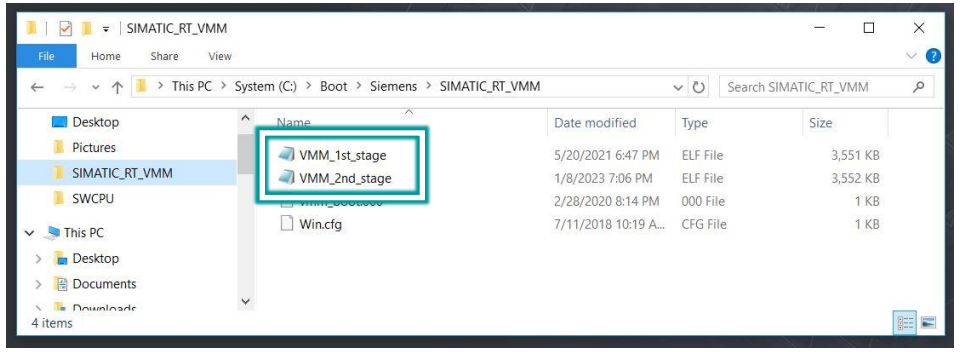


Previous Research

- Surprisingly, the HV and SWCPU firmware files may be accessed via the Windows OS.
  - *C:\Boot\Siemens\SWCPU*
- By simply dragging and dropping, we can replace the firmware files.







The OSs aren't that isolated…

Previous Research

- They leveraged this capability to crash debug the HV:
  - Override an instruction in the HV with an “int 3” command.
  - Replace the original HV with the modified version using the “drag and drop” method we showed earlier.
  - Once the “int 3” command executes, a crash dump will be produced (a hidden feature).

```
int     3
```

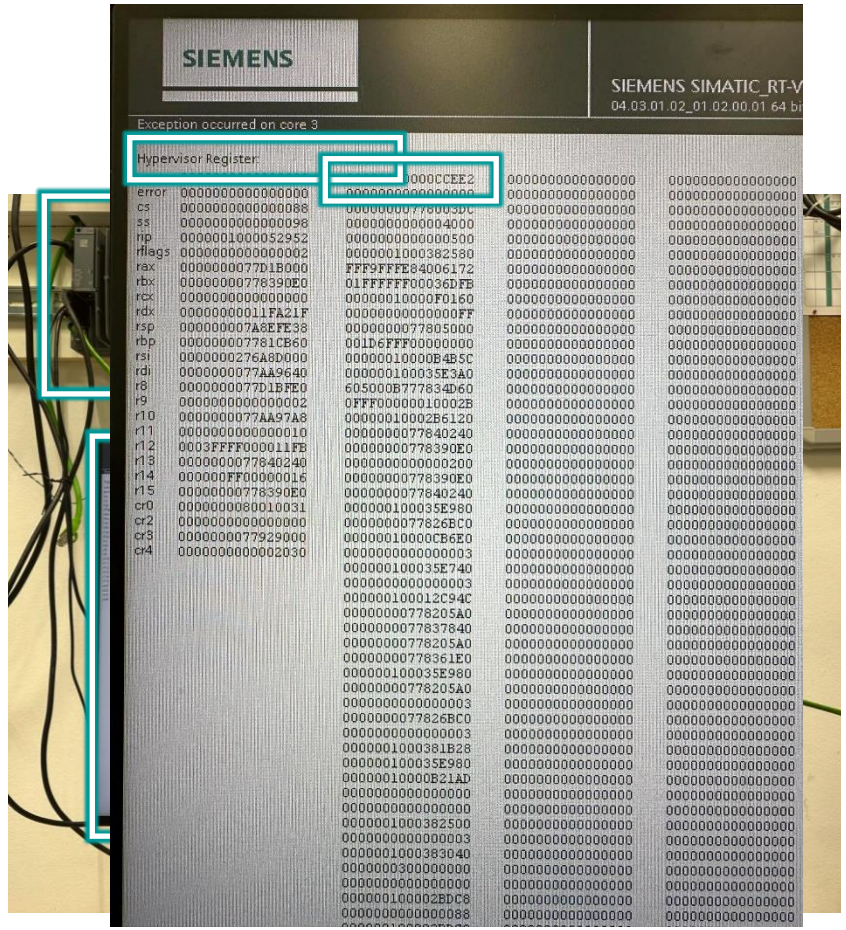
```
; __unwind {
push    r15
lea     r15, asc_10001508FF ; "?"
push    r14
push    r13
push    r12
mov     r12, rsi
push    rbp
push    rbx
```

The PLC

The crash dump

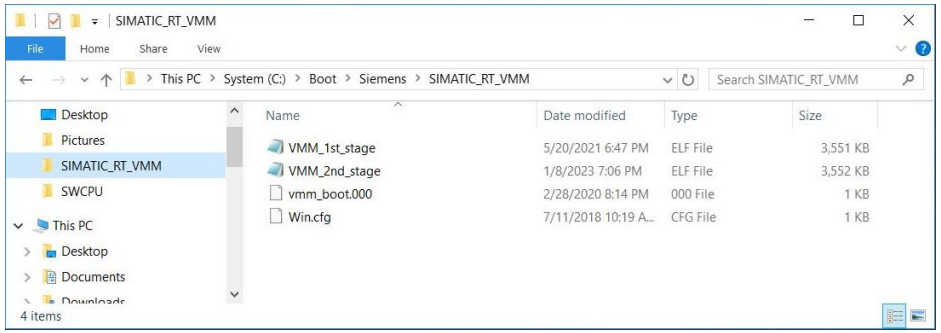
If we zoom in, we will see the:

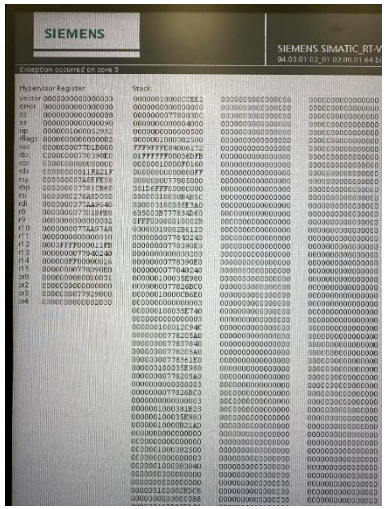
- **Registers and stack**
- **The return address on the stack**



A Quick Recap

- Soft7 enabled us to:
  - Study the SWCPU's assembly code.
  - Modify the hypervisor via the Windows filesystem
  - Crash debug the hypervisor.



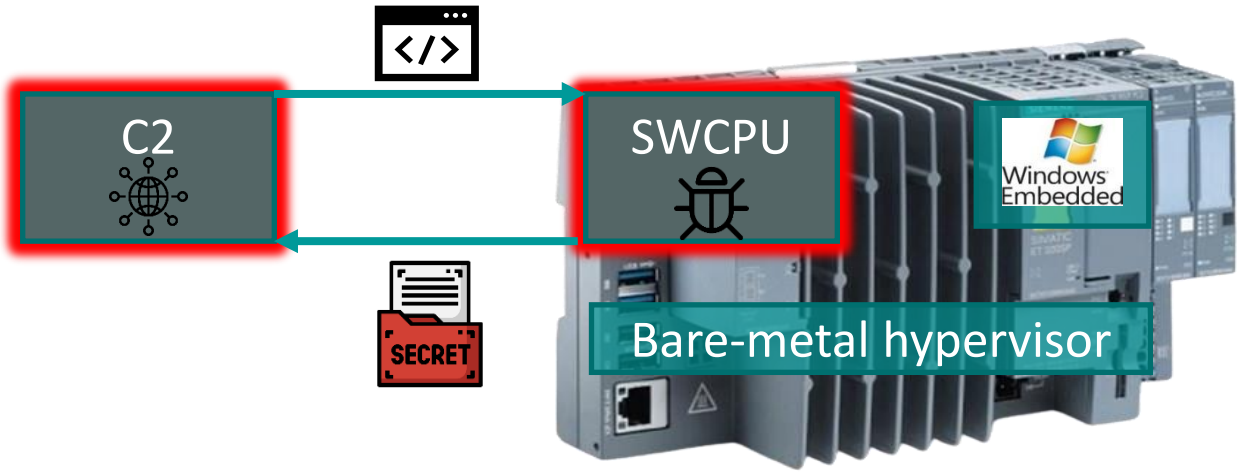


- Going forward, we'll present our contributions!

What Will We Show You?

How a bad actor could:

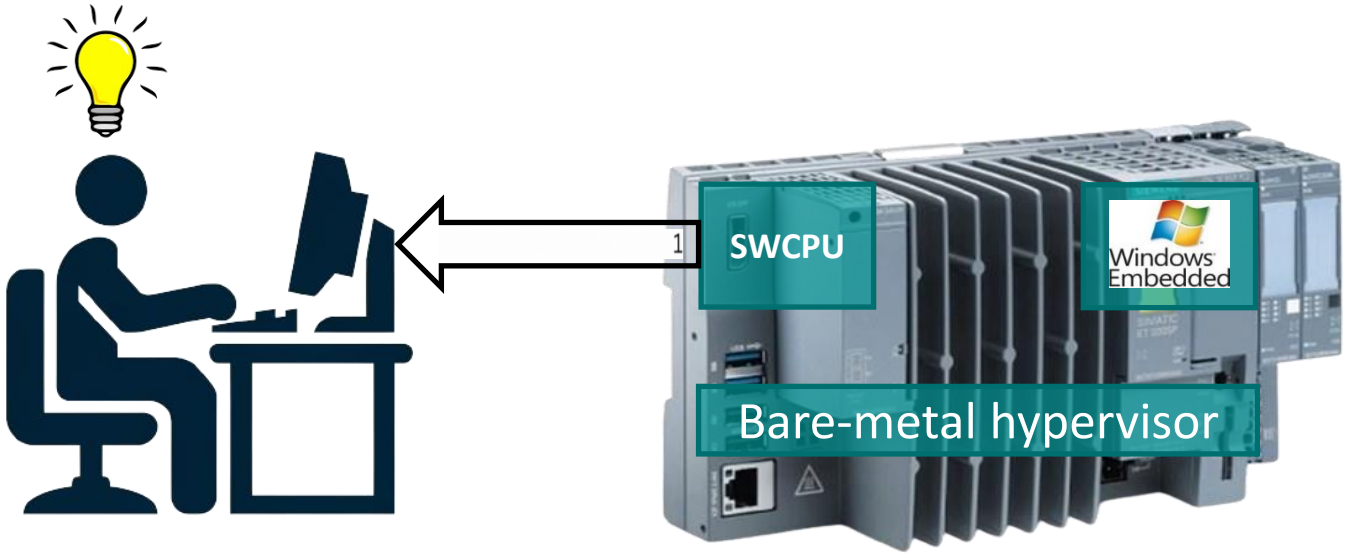
- Use remote exploitation to stealthily install persistent malware on the SWCPU.
- Establish communication between the malware and a remote C2 server.
- Dynamically inject commands into the SWCPU.
- Exfiltrate data from the SWCPU.



What Will We Show You?

If you're a researcher, you will learn how to:

- Read data from the SWCPU during runtime.
- Get a better understanding of the code flow.
- Expedite your research process exponentially.



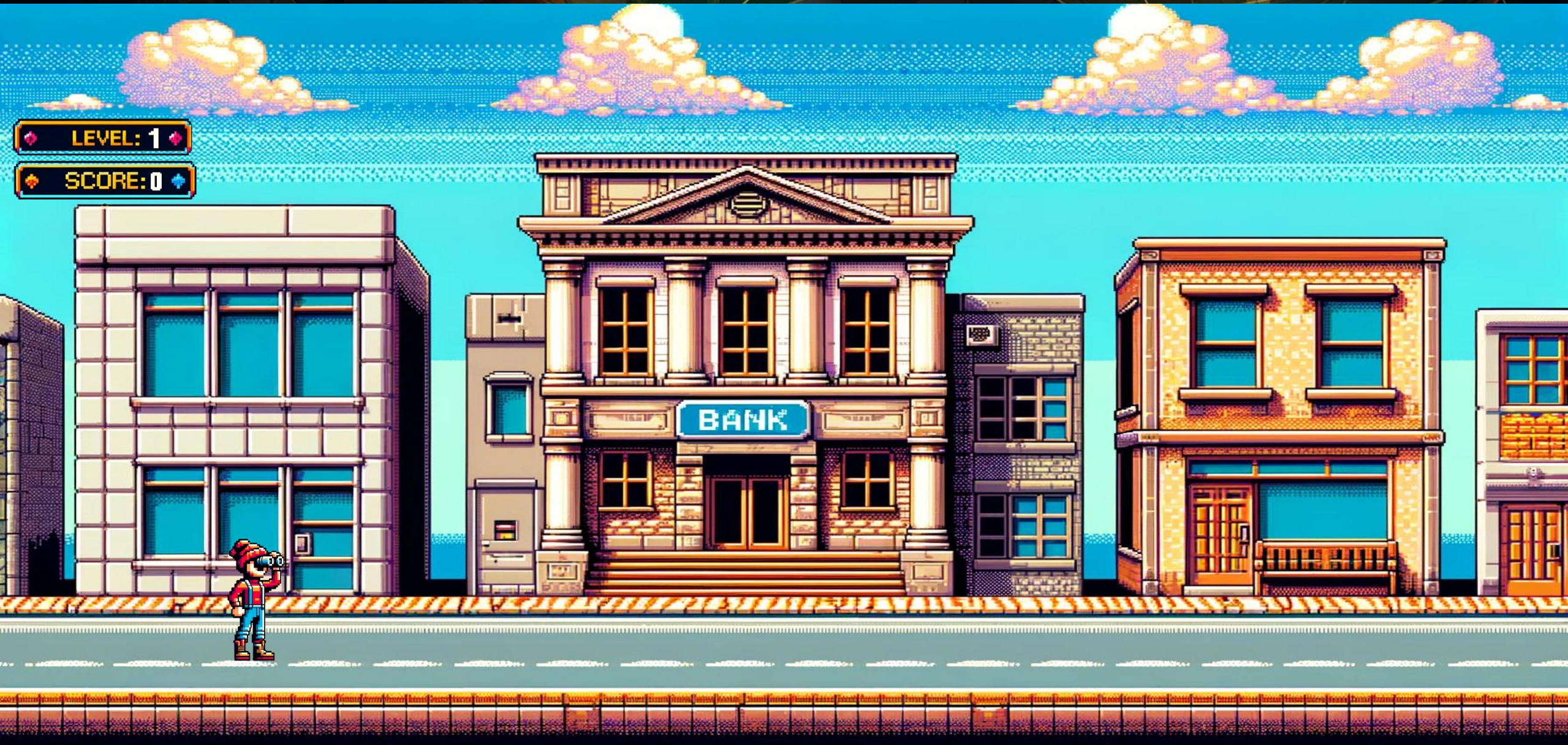
The Heist

- Our research process was like robbing a bank.
- Let me introduce you to the game “Bank Heist”
  - We’ll be playing this game throughout the talk.

Talk Topics

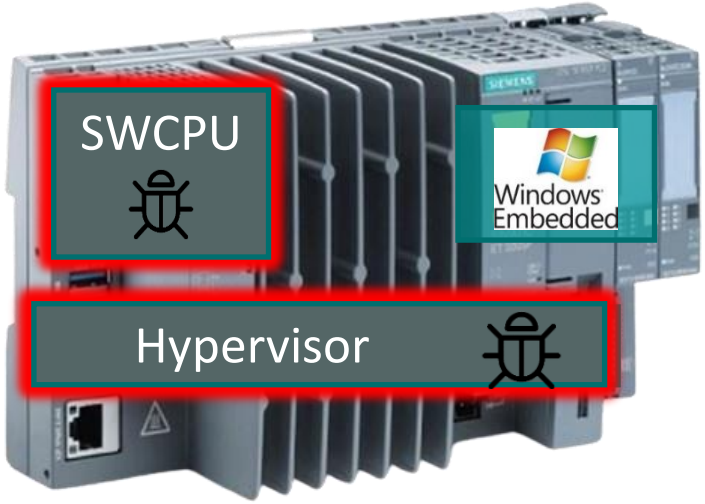
- Introduction and Previous Research
- Runtime Manipulation of Siemens S7 PLCs Firmware
- Implementation of Debug 7 - a Remote Debugger for Siemens S7 PLCs
- Debugger Video Demo
- Conclusions

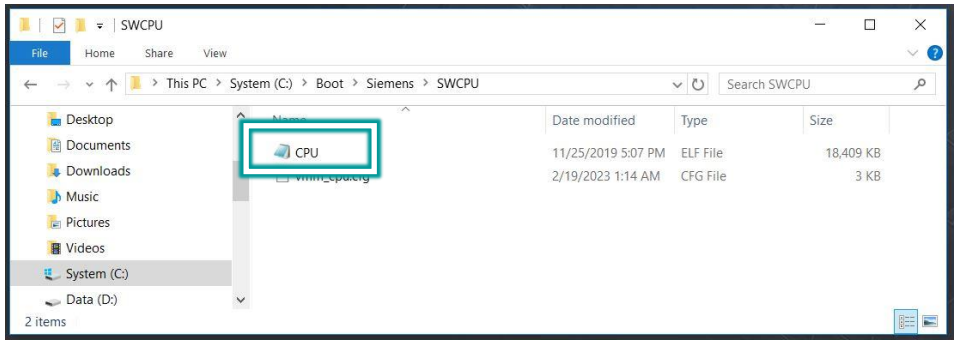
Level 1 – Gather Intel (Research the firmware)



A Bump in the Road

- Remember that the Soft7 team was able to modify the HV and run it?
- Similarly, we wanted to run our modified version of the SWCPU, but:
  - The SWCPU on the PLC is encrypted.
  - The Soft7 team didn't discover the decryption method or key.
  - After modifying the plaintext SWCPU, we couldn't encrypt it to match the hypervisor's decryptor.
  - Replacing the original SWCPU with a decrypted one should just make it crash…





The Hypervisor Vulnerability We Found

- We started by reversing the code that loads the SWCPU.
  - The hypervisor can load any ELF file as the SWCPU!!
  - Looks like a backdoor…

This "if" clause enables any ELF file to be loaded

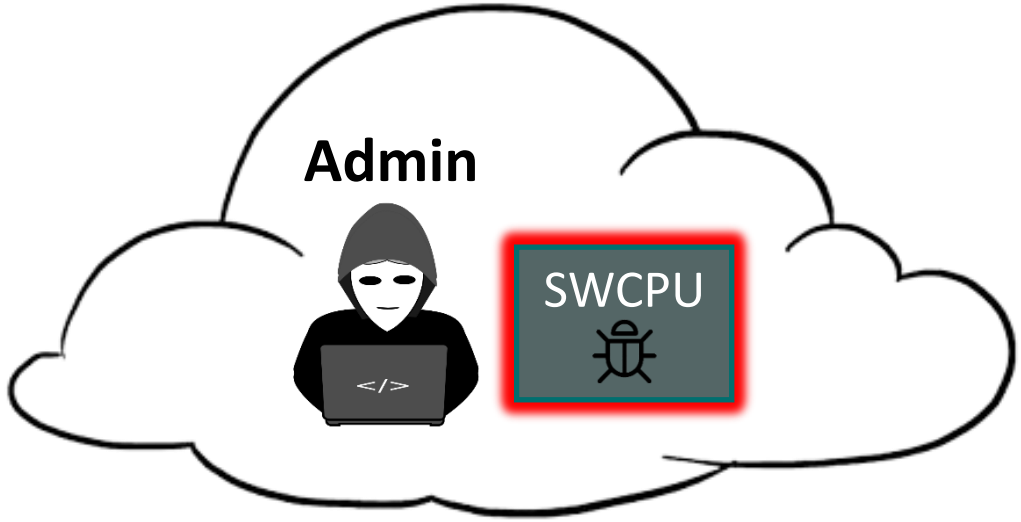
```
if ( *header == 127 )
{
  if ( header[1] == 'E' && header[2] == 'L' && header[3] == 'F' )// if elf file, load and run it
  {
    func_ptr = (__int64 (__fastcall *)(__int64, __int64, int, unsigned int *, unsigned int, unsigned int (__fastcall *)(__int64 *)))sub_1000051F70;
    func_ptr_copy = (__int64 (__fastcall *)(__int64, __int64, int, unsigned int *, unsigned int, unsigned int (__fastcall *)(__int64 *)))sub_1000051F70;
    goto LABEL_18;
  }
LABEL_6:
  print((char)"Error loading elf file (%s): invalid magic");
}
if ( *header != 'S' || header[1] != '3' || header[2] != '^' || header[3] != '\x9F' )// if encrypted elf, decrypt and run it
  goto LABEL_6;
```

This "if" clause loads the encrypted SWCPU file.
It begins with the "S3^\x9F" magic string.

The Vulnerability – A Visual Explanation

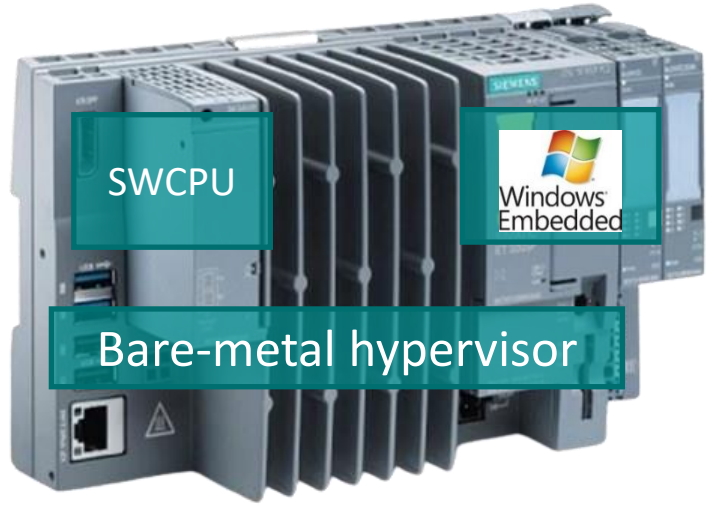
An admin on the Windows OS can replace the SWCPU with a malicious version, using the "drag and drop" method.

Upon PLC restart, the hypervisor loads the malicious SWCPU without checking for authenticity.

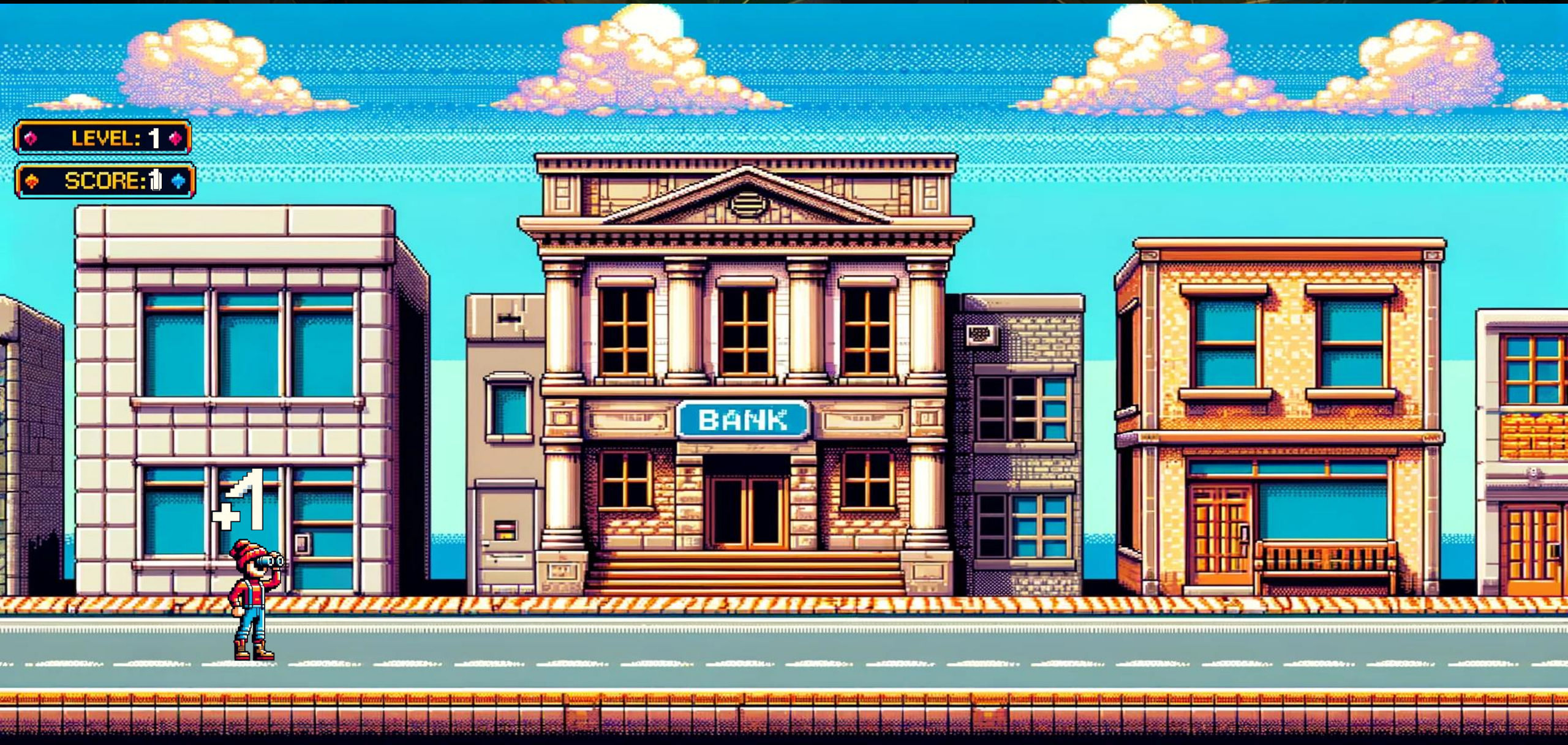


- This undermines a significant advantage of virtualization – isolation.
- Can be done remotely:
  - If an attacker gains remote admin access to the Windows OS.
- Changes withstand a reboot.

Sidenote: we owned our PLC so obviously we had an admin user

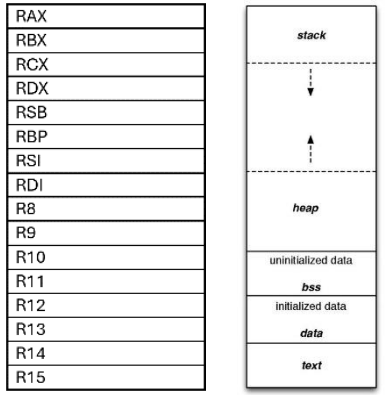


Level 1 – Gather Intel (Research the firmware)

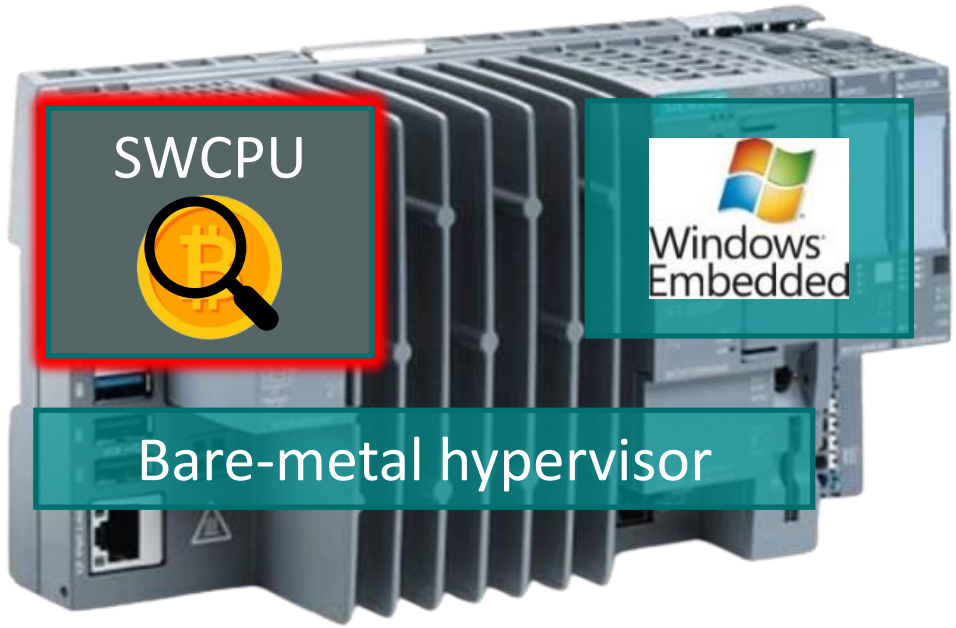


Exploiting the Vulnerability

- Now we can modify the SWCPU to our liking!!
  - Should we make it mine bitcoin?
  - Instead, let's modify it to help us understand its internal logic better.
- To understand the SWPCU better we needed to extract basic runtime info:
  - Registers
  - Stack



- But how can we do it?

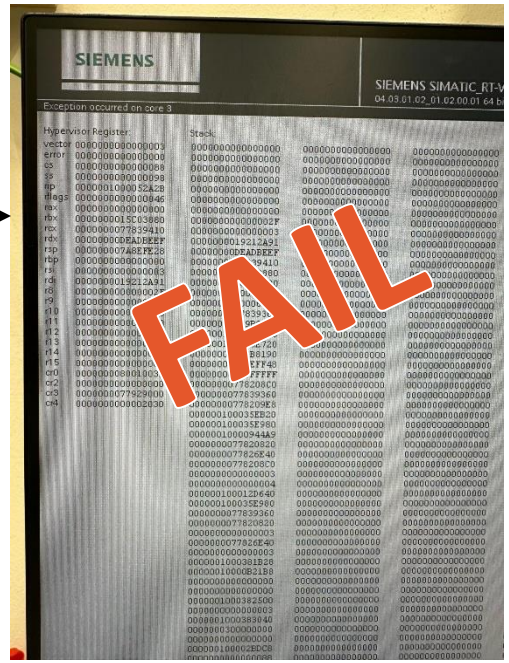


The Naïve Approach

- Remember that by injecting an ‘int3’ command into the hypervisor code we can generate a crash dump?
- If we do the same to the SWCPU, will it also generate a crash dump?
  - Thus, we will see the SWCPU’s registers and stack.

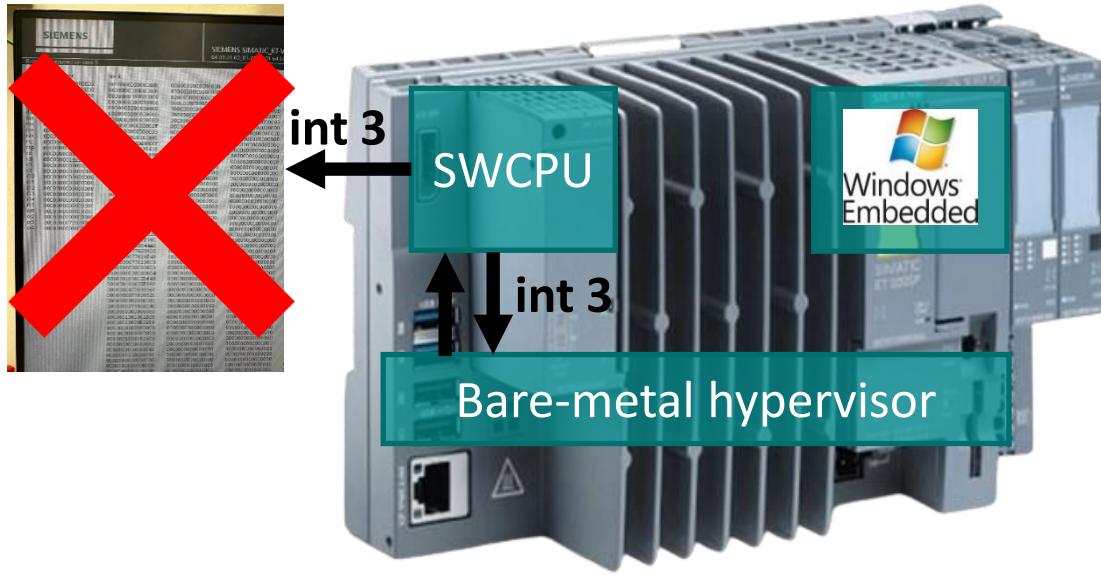
```
int     3
```

```
call    rax
test    eax, eax
mov     r14d, eax
jnz     short loc_167DD830
```



Improving the Naïve Approach

- The hypervisor launches the SWCPU.
  - During runtime control is shifted back and forth.
- What does an “int 3” command do when it executes in the SWCPU?
  - We thought it generates a crash dump.
  - We discovered that it shifts control back to the hypervisor!!
- Can we leak data from the SWCPU to the hypervisor’s crash dump?



A Visual Explanation

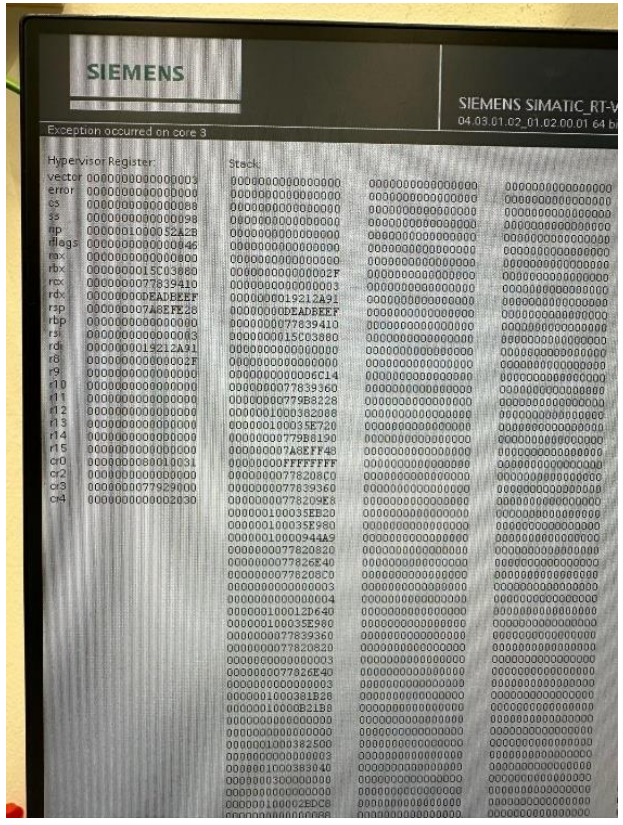
While running within the SWCPU, shift control back to the HV using an “int 3” command.

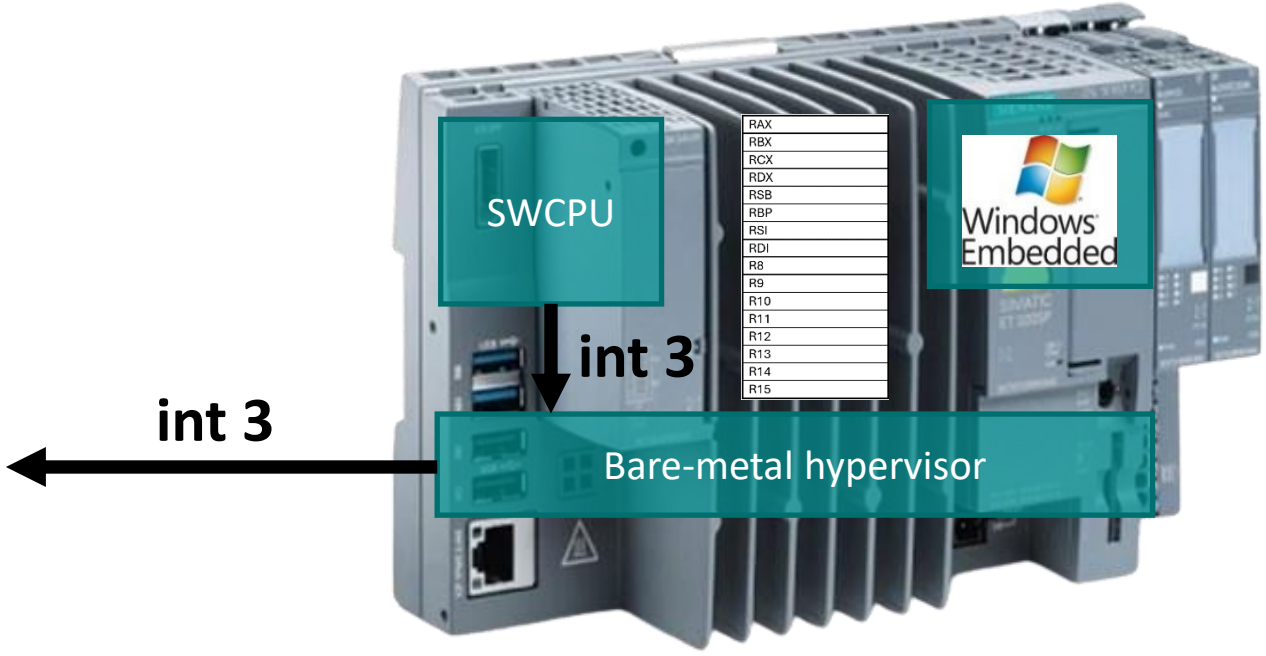
After the context switch, the registers will remain the same, since it’s a bare-metal HV.

Crash the hypervisor by using another “int 3” command.

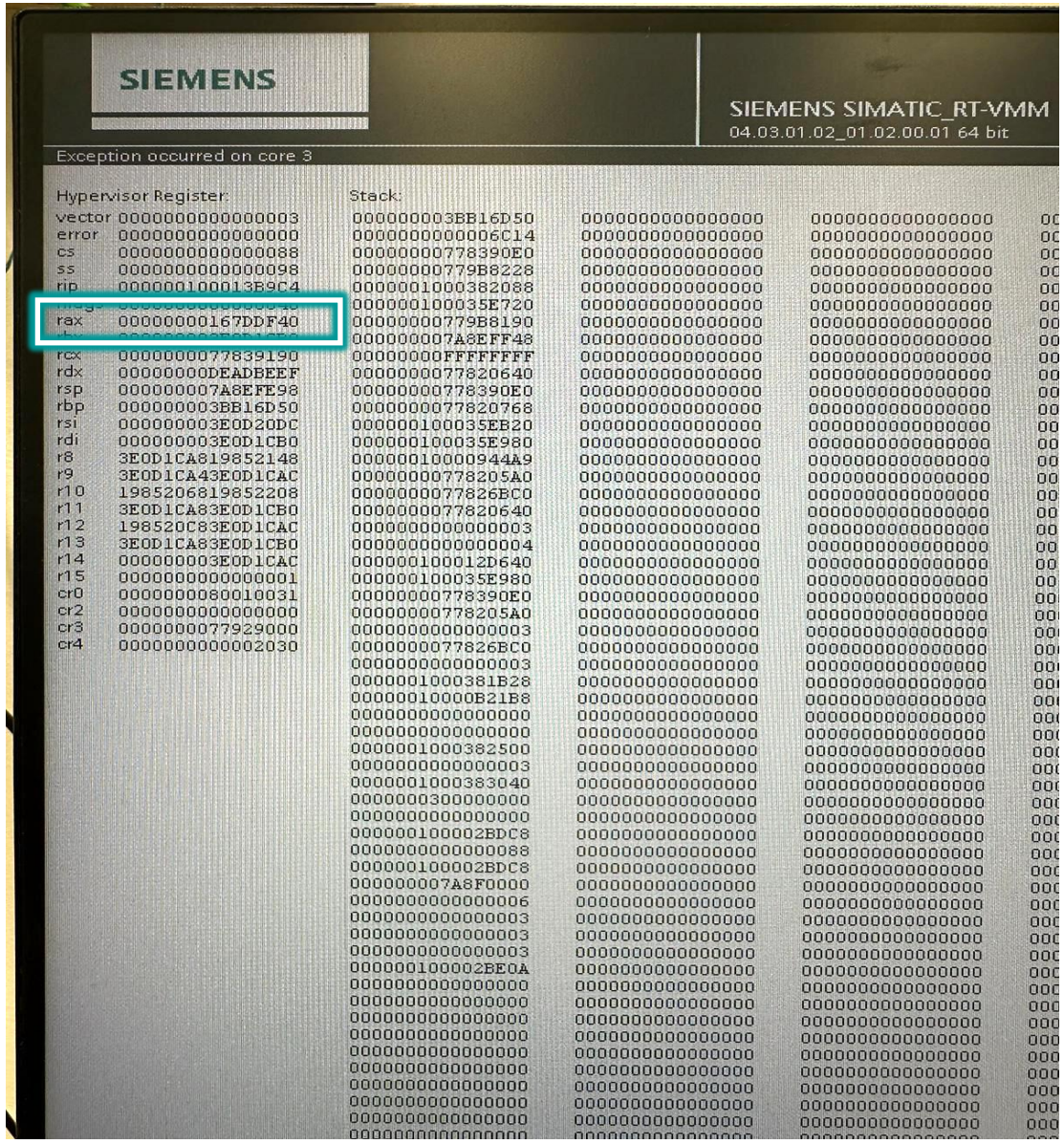
The registers in the hypervisor crash dump will be identical to the SWCPU’s registers.

We found a way to leak SWCPU information to the hypervisor’s crash dump, via the registers.





The Code We Injected

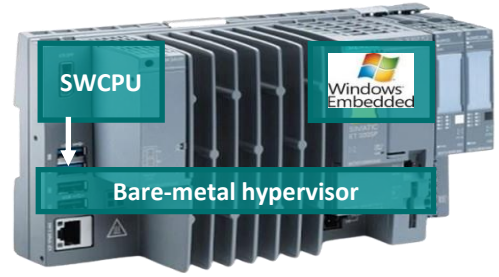



```
call    rax
test    eax, eax
mov     r14d, eax
jnz     short loc_167DD830
```

SWCPU code

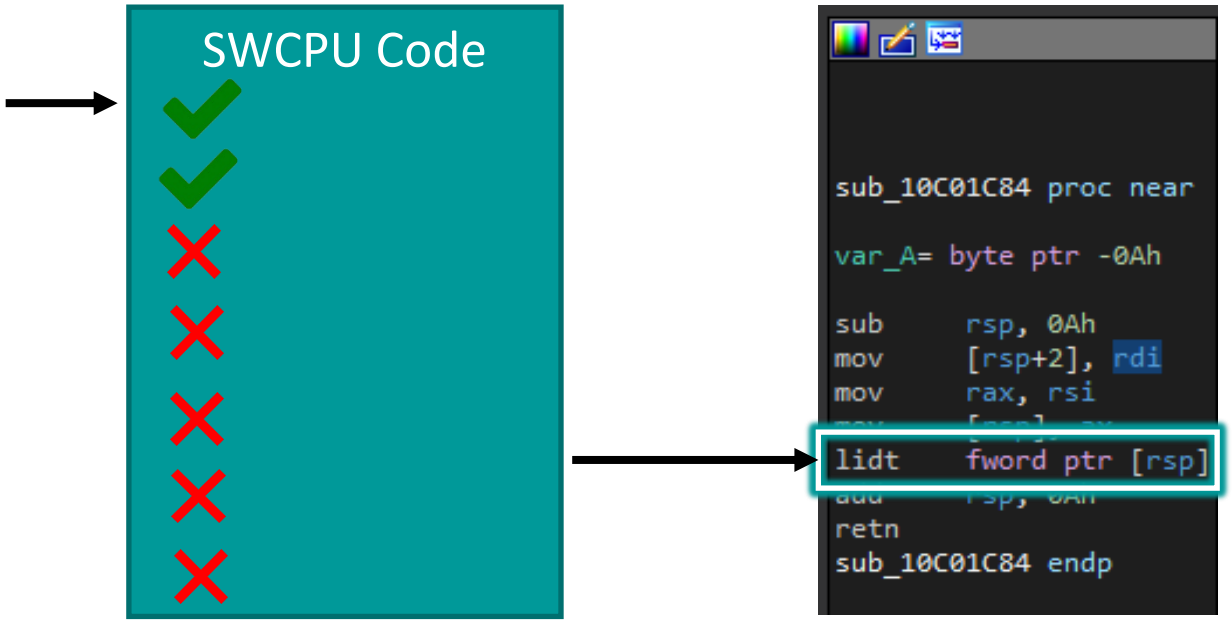
```
mov     edx, 0DEADBEEFh
int     3
```

Hypervisor code



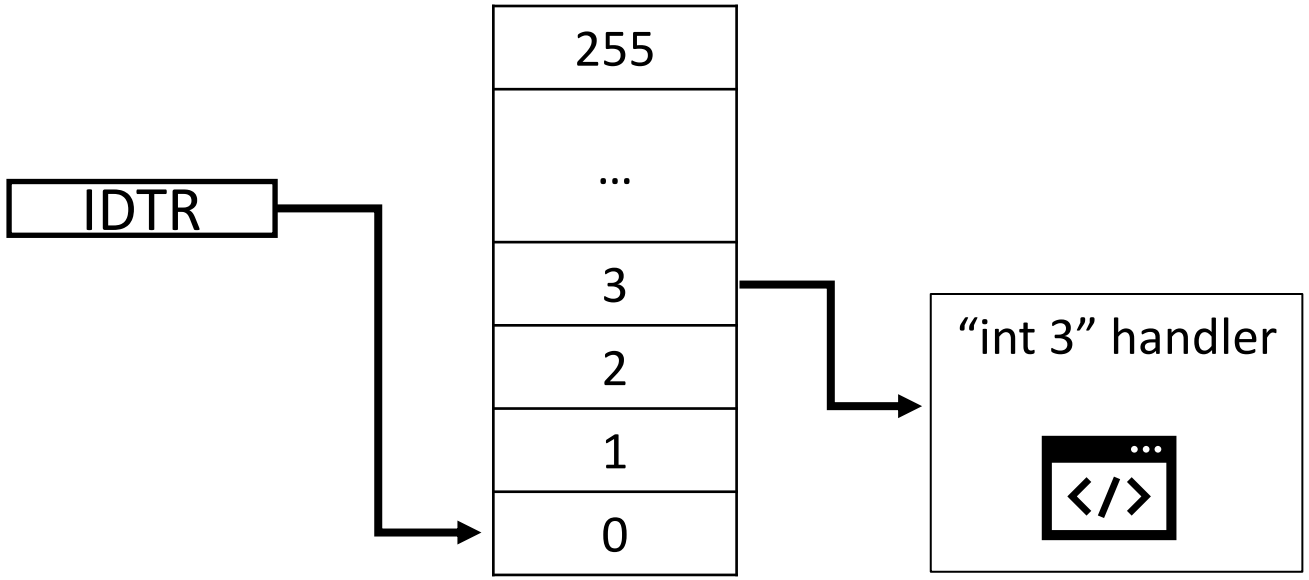
Results

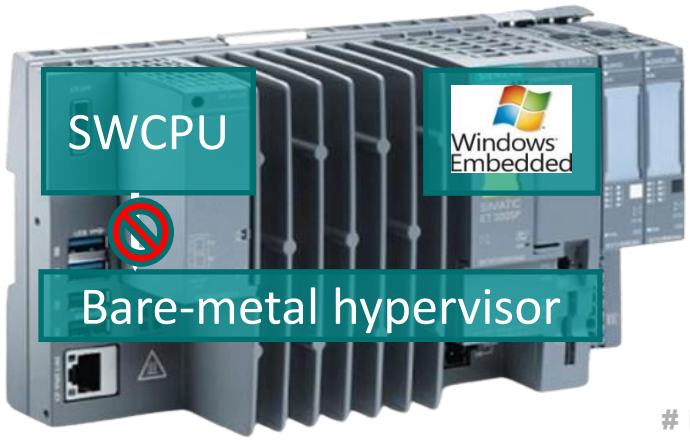
- This was a breakthrough! We managed to leak SWCPU runtime information
  - This worked only in some parts of the code.
- We decided to find the point of failure:
  - Using binary search, we attempted to crash the SWCPU until we found the point of failure.



Let's Improve Even More

- What is the IDT?
  - A kernel struct, pointed to by the IDTR register.
  - Each entry in the IDT points to a different interrupt handler
  - When "int 3" executes, the handler shifts control back to the hypervisor.
  - If modified, the handler won't shift control back to the hypervisor.





Let's Improve Even More

- We bypassed the issue by using the "*vmcall*" instruction instead of "*int 3*".
  - "*vmcall*" shifts control to the hypervisor without going through the IDT.

Our breakpoint looked like this

```
.text:00000000167DD7DB    mov     eax, [ebx]
.text:00000000167DD7DE    mov     eax, [eax+18h]
.text:00000000167DD7E2    mov     edx, 0DEADBEEFh
.text:00000000167DD7E7    int     3
```

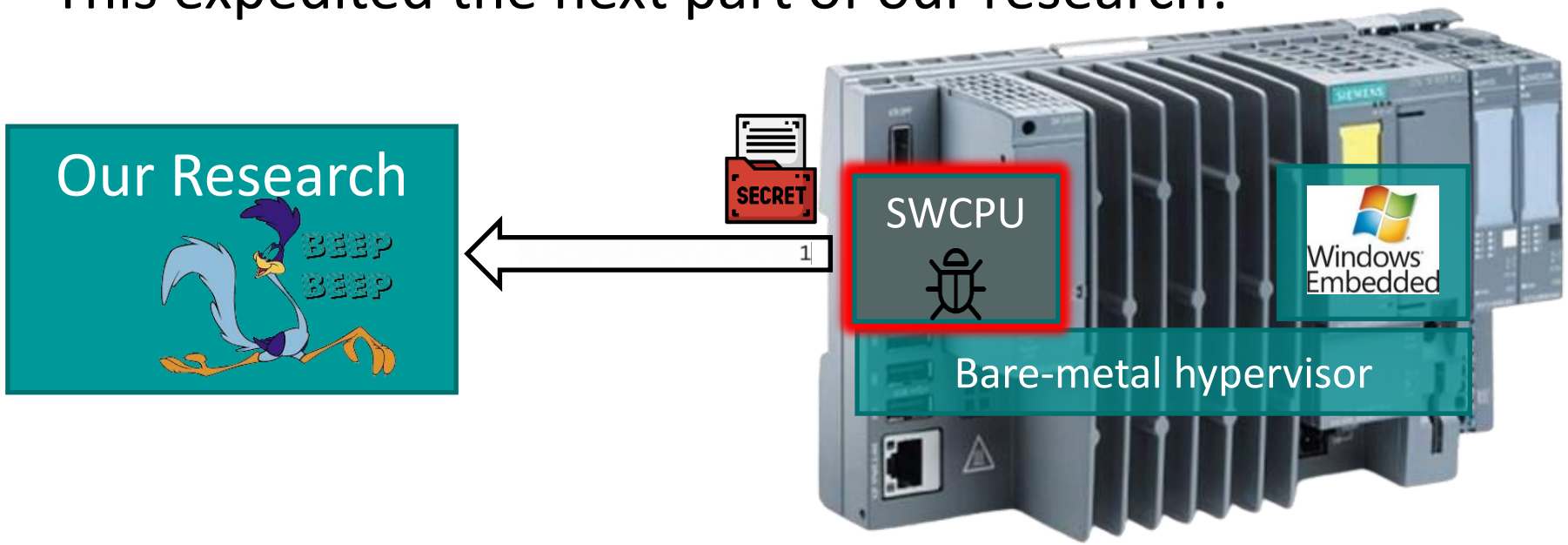
Now it looks like this

```
.text:00000000167DD7DB    mov     eax, [ebx]
.text:00000000167DD7DE    mov     eax, [eax+18h]
text:00000000167DD7E2     mov     edx, 0DEADBEEFh
text:00000000167DD7E7     vmcall
```

Level 1 – Gather Intel (Research the firmware)

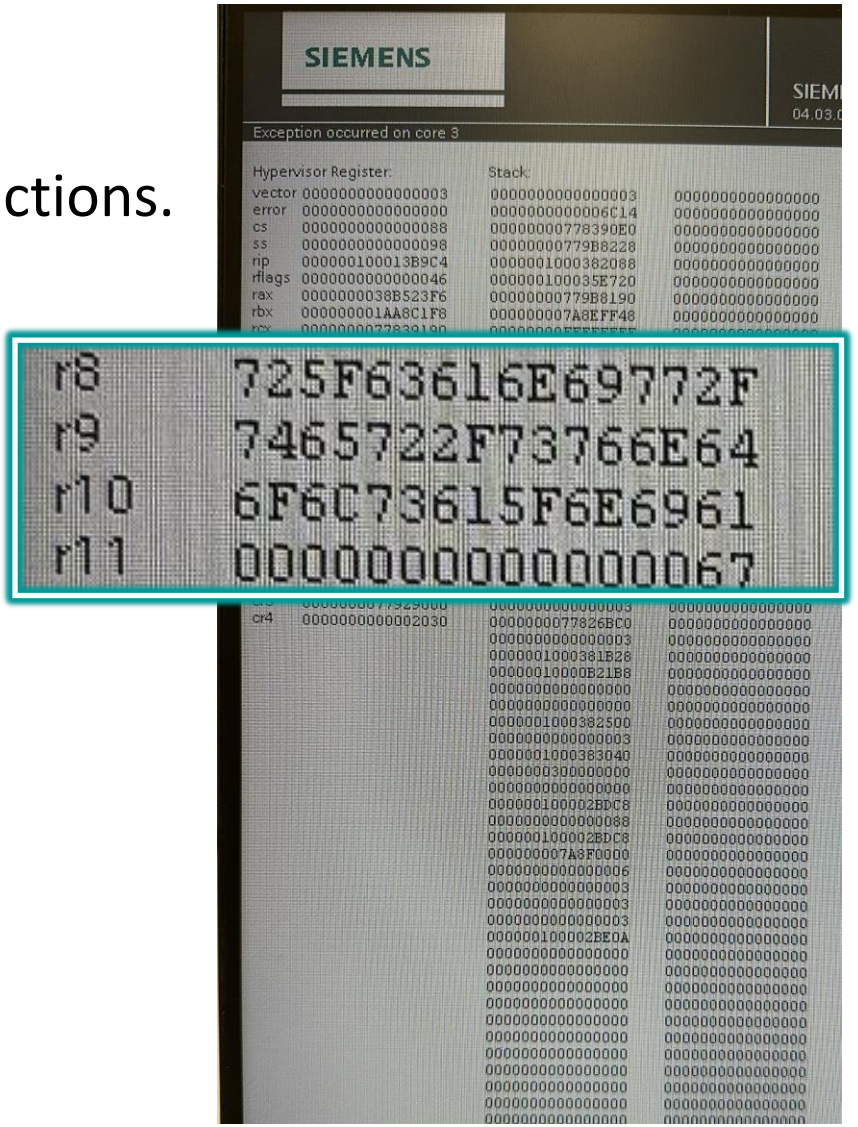
A Quick Recap

- The original SWCPU file can be easily replaced with any ELF file.
  - Using the “drag and drop” method
- By modifying the SWCPU and the hypervisor, we were able to extract runtime information from the SWCPU.
- This expedited the next part of our research!



So How Did This Help Us

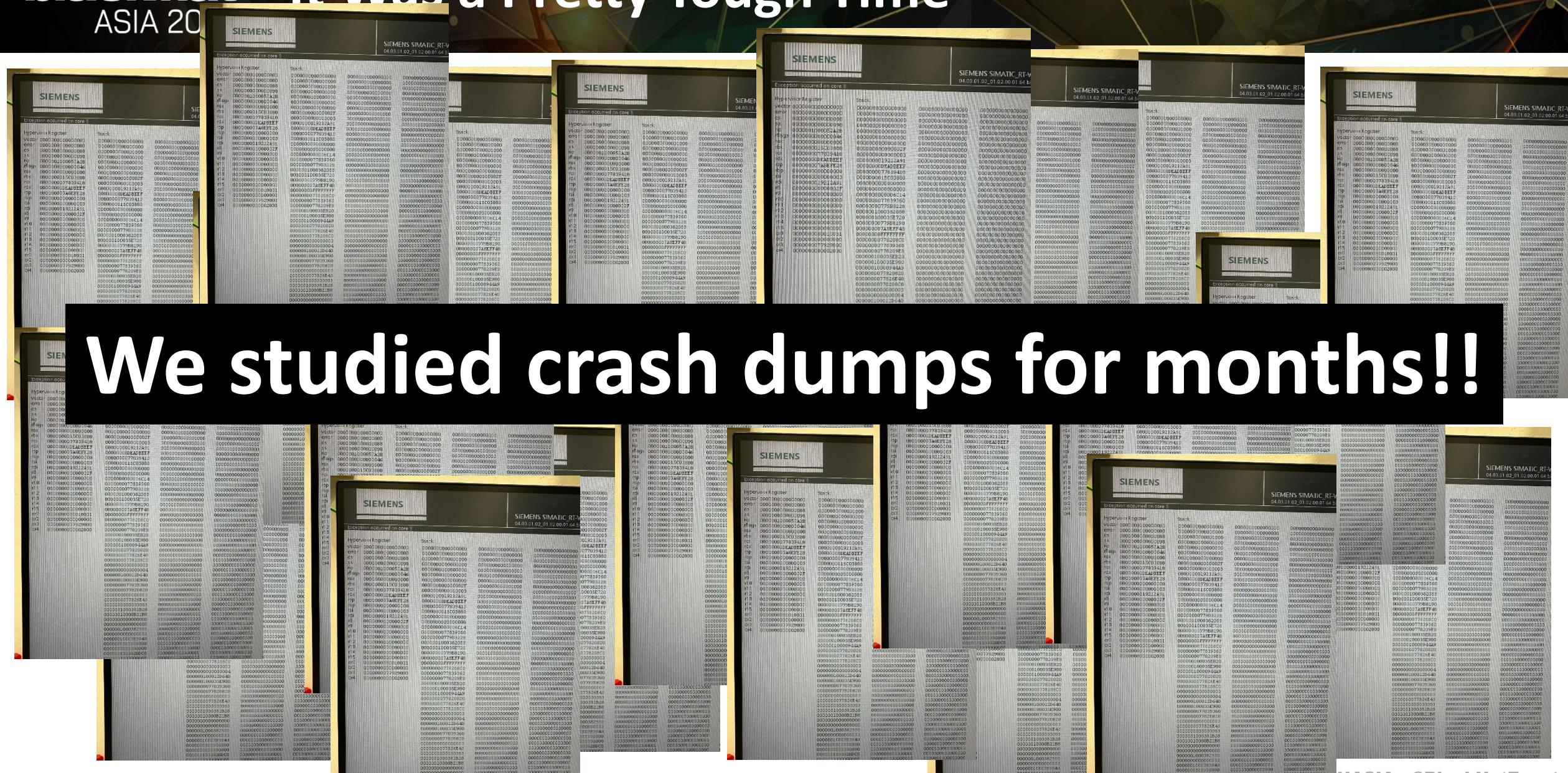
- The SWCPU's system call table is obfuscated.
  - Using static analysis, we found it and mapped several functions.
- After locating the "open" system call:
  - We copied the filename to the registers.
- For example:
  - r8-r11 contain *"/winac_rdnvs/retain_aslog"* in ASCII.



The System Call Table

```
.rodata_kernel:10C01380 syscall_table   dd offset sub_10C0C6B0
.rodata_kernel:10C01380
.rodata_kernel:10C01384 off_10C01384    dd offset sub_10C0C870
.rodata_kernel:10C01384
.rodata_kernel:10C01388 off_10C01388    dd offset sub_10C0CF30
.rodata_kernel:10C01388
.rodata_kernel:10C0138C off_10C0138C    dd offset kernel_close
.rodata_kernel:10C0138C
.rodata_kernel:10C01390 off_10C01390    dd offset kernel_ioctl
.rodata_kernel:10C01390
.rodata_kernel:10C01394 off_10C01394    dd offset sub_10C189D0
.rodata_kernel:10C01394
.rodata_kernel:10C01398 off_10C01398    dd offset sub_10C18C50
.rodata_kernel:10C01398
.rodata_kernel:10C0139C off_10C0139C    dd offset sub_10C18D30
.rodata_kernel:10C0139C
.rodata_kernel:10C013A0 off_10C013A0    dd offset kernel_open
.rodata_kernel:10C013A0
.rodata_kernel:10C013A4 off_10C013A4    dd offset kernel_read
.rodata_kernel:10C013A4
.rodata_kernel:10C013A8 off_10C013A8    dd offset kernel_write
.rodata_kernel:10C013A8
.rodata_kernel:10C013AC off_10C013AC    dd offset sub_10C1A320
.rodata_kernel:10C013AC
.rodata_kernel:10C013B0 off_10C013B0    dd offset sub_10C1A330
.rodata_kernel:10C013B0
.rodata_kernel:10C013B4 off_10C013B4    dd offset sub_10C1A3E0
```

It Was a Pretty Tough Time



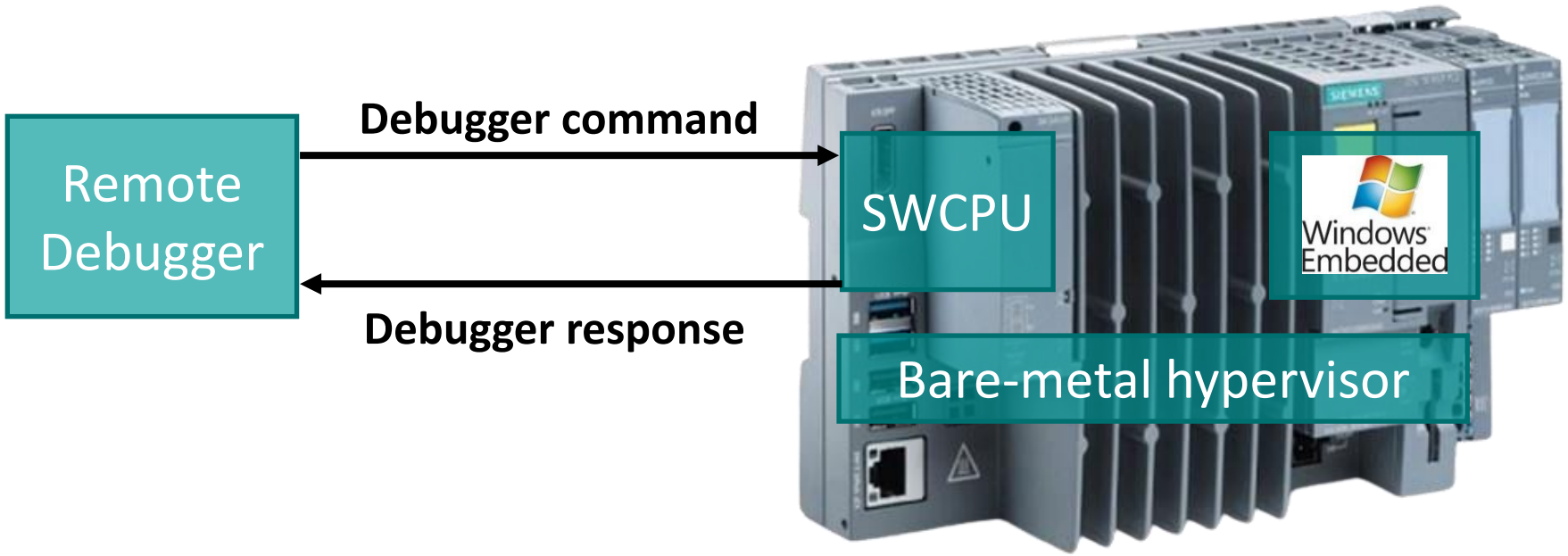
We studied crash dumps for months!!

Talk Topics

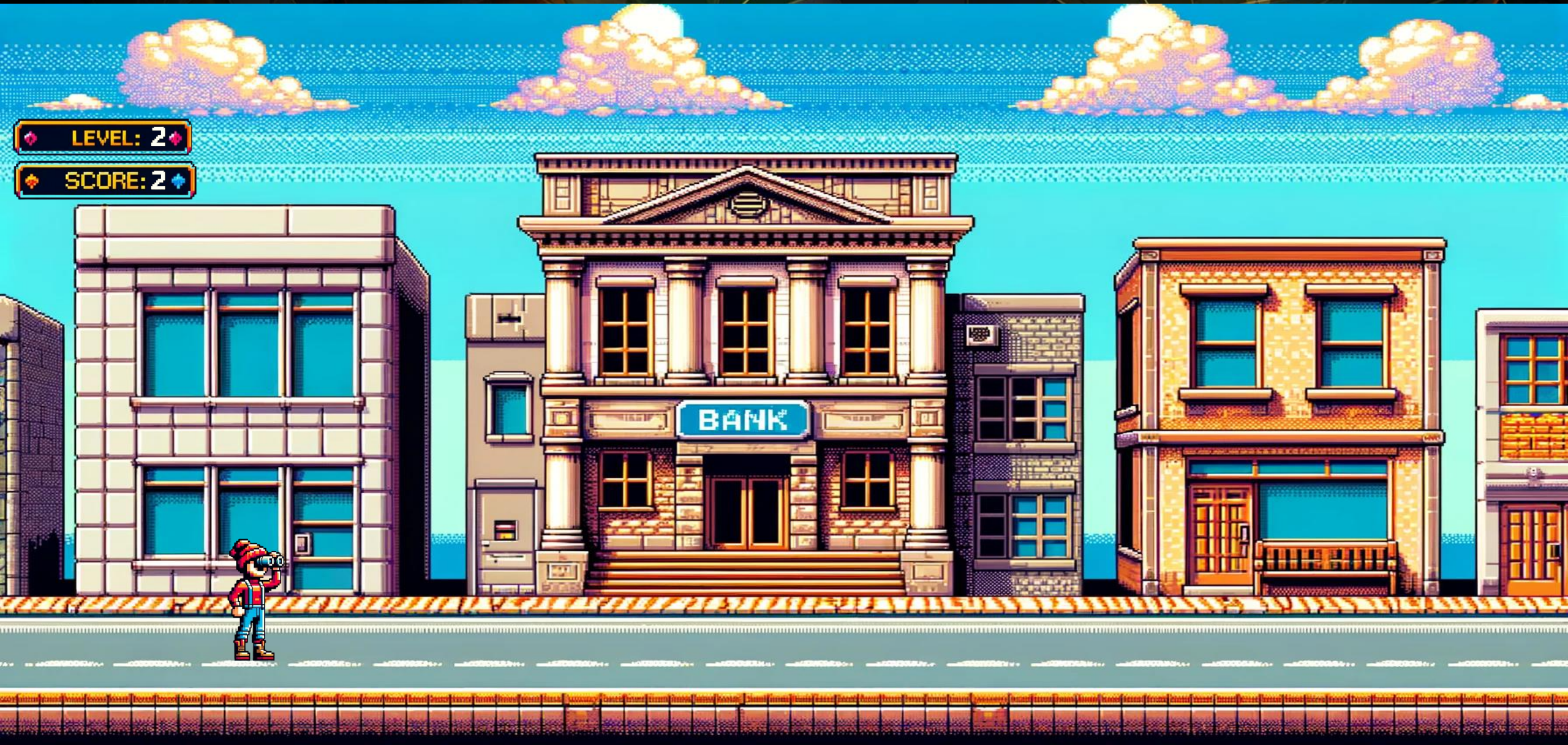
- Introduction and Previous Research
- Runtime Manipulation of Siemens S7 PLCs Firmware
- Implementation of Debug 7 - a Remote Debugger for Siemens S7 PLCs
- Debugger Video Demo
- Conclusions

Implementation of Debug7

- To build the remote debugger we needed to:
  - Inject debugger commands into the SWCPU.
  - Exfiltrate data from the SWCPU to the debugger.
- Effectively, we needed C2 capabilities.

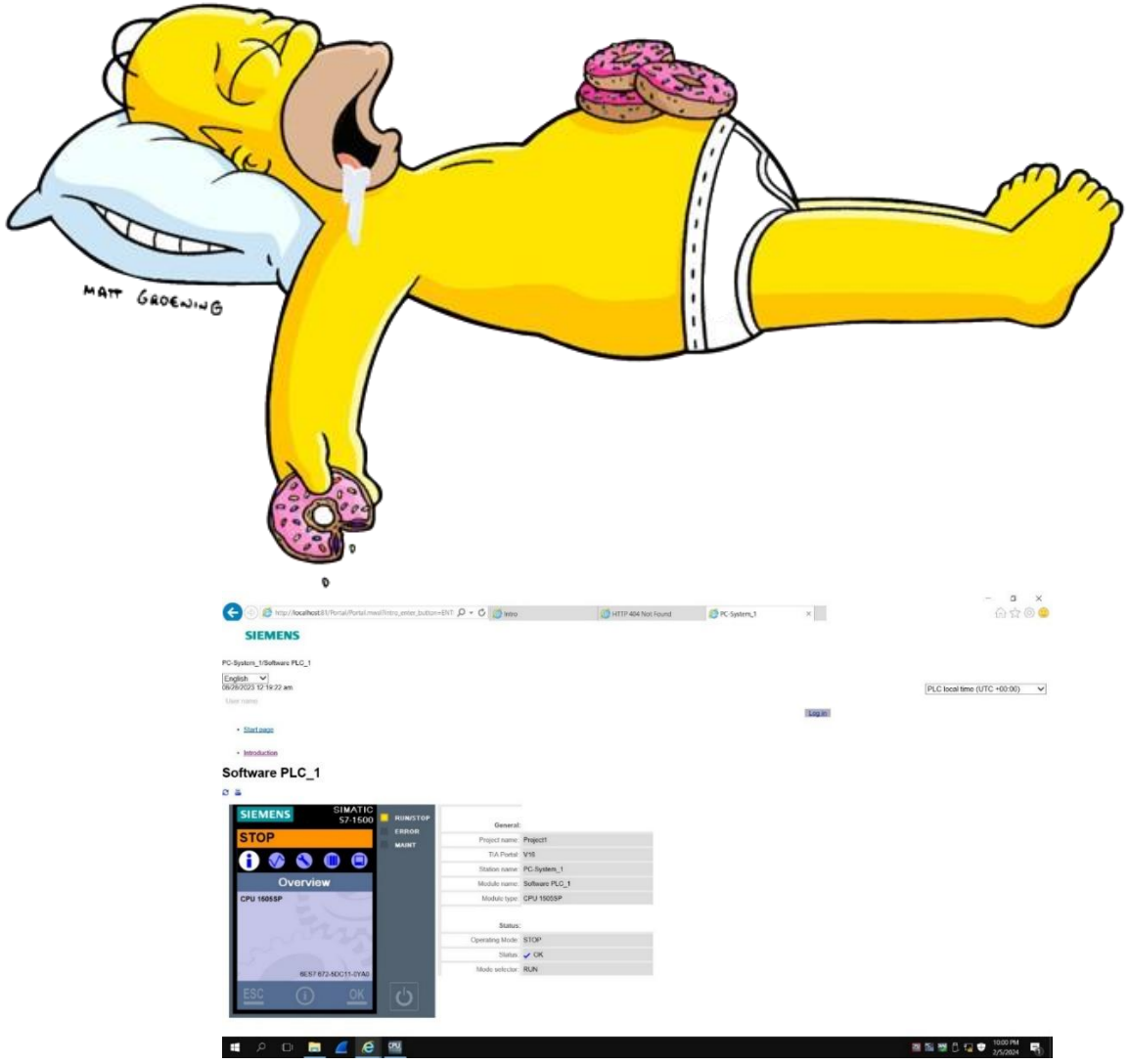


Level 2 – Breach the Vault (Inject Data)



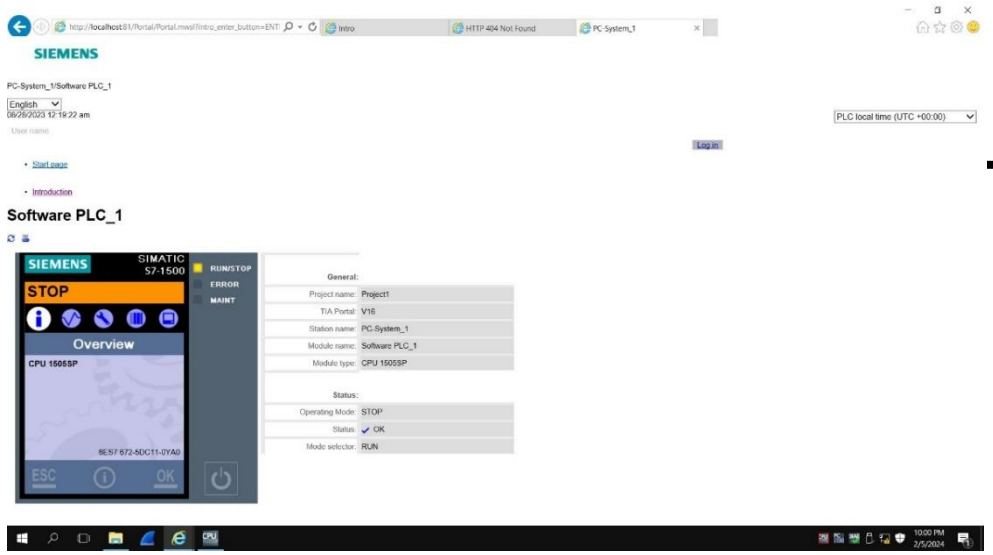
Its OK to be Lazy

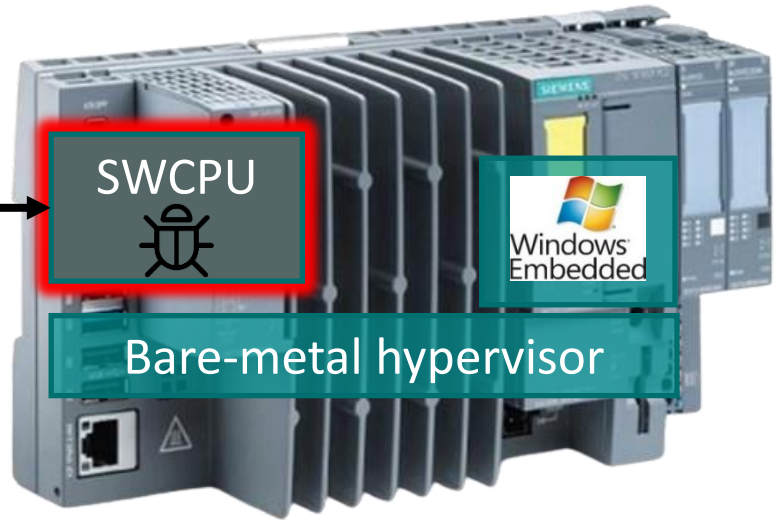
- We are very lazy!!!
  - Developing a C2 server is hard ☹
- What if we told you that a C2 platform already exists?
  - Its already installed for us on the PLC.
  - It can save us weeks of development time.
  - It also has a nice GUI.
- Let me introduce you to the PLC's web server.



Can we Inject Data via the Web Server?

- The PLC exposes a web server via the SWCPU.
  - For example, a technician can connect to it via his cell phone during maintenance.
- Can we inject data into the SWCPU via the URL?





Can we Inject data via the Web Server?

Request a non-existing page from the web server → The SWCPU attempts to open the file → Hook the "open" syscall → If the filename starts with "dbg7" save it to the memory. → Otherwise, continue the normal "open" flow

SWCPU

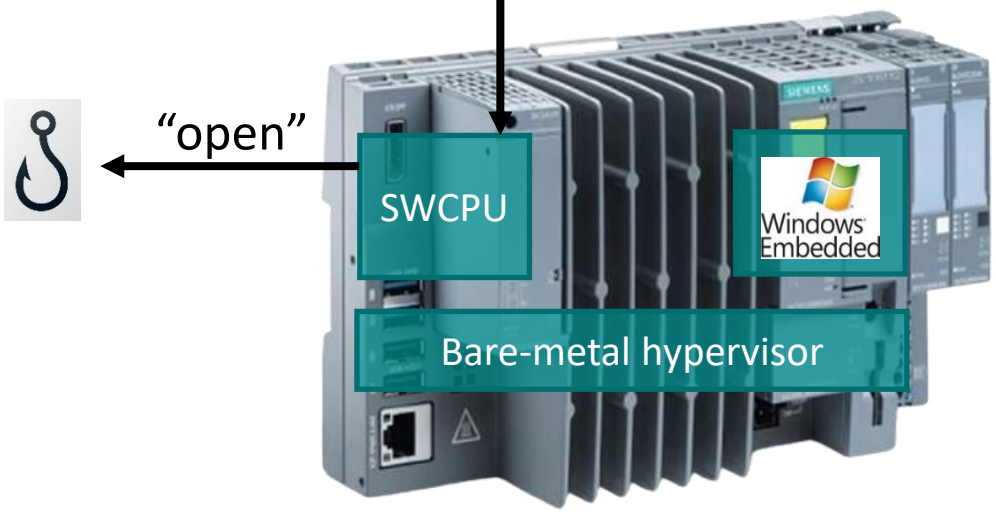
Hook Code

```
open_hook:
if (filename starts with dbg7)
{
   save filename in memory
}
else
{
   resume normal "open" flow
}
```

Memory

HTTP Client

http://localhost:81/ **dbg7_hello**



Level 2 – Breach the Vault (Inject Data)

Level 3 – Escape (Exfiltrate Data)

Can We Extract data via the Web Server?

Request the home page from the web server → The SWCPU attempts to read a css page from memory → Hook the "read" system call → The read hook injects the saved string into the css page → The web server now displays our data

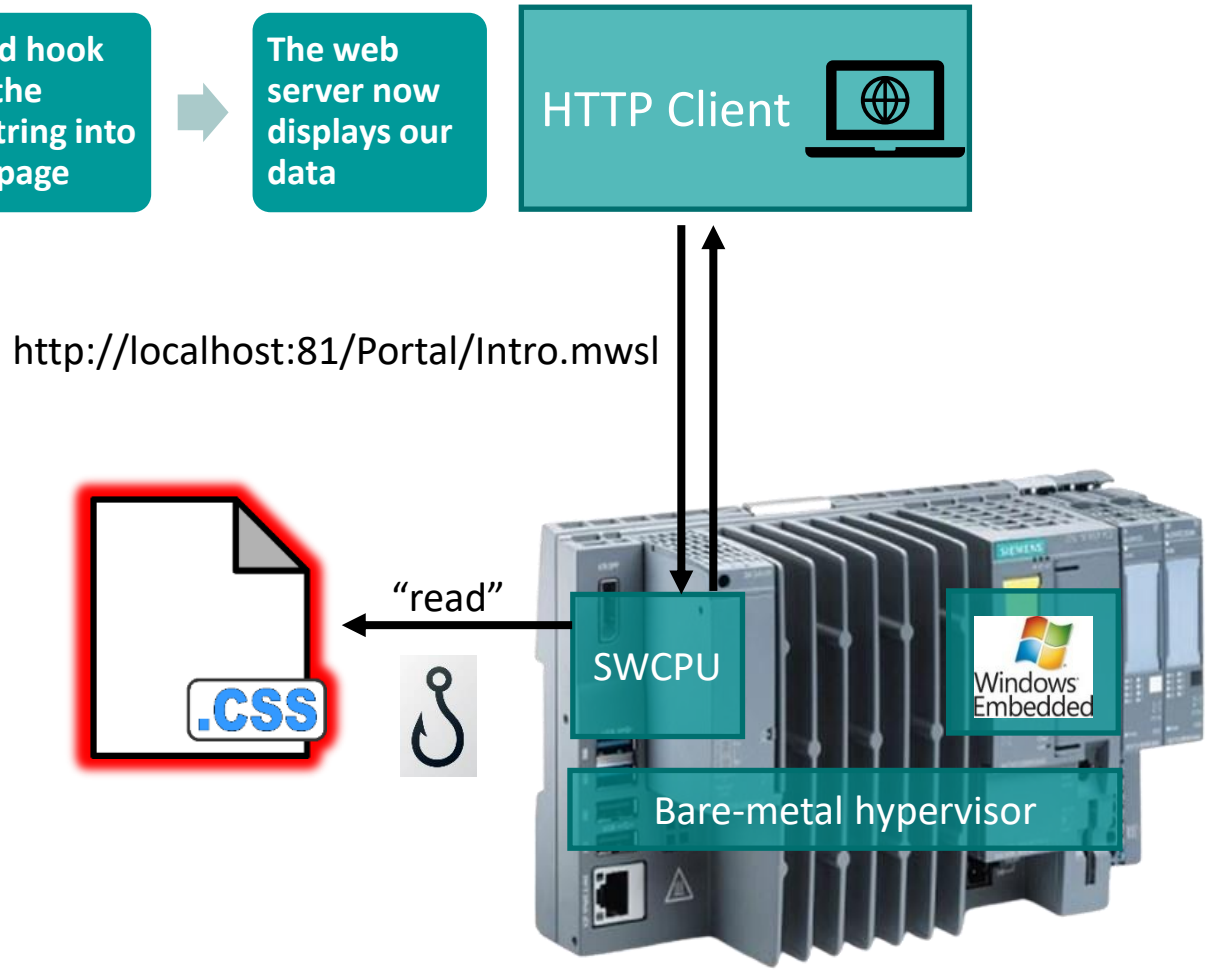
SWCPU

Hook Code

```
read_hook:
if (filename == "MiniWeb.css")
{
   return data from memory
}
else
{
   resume normal "read" flow
}
```

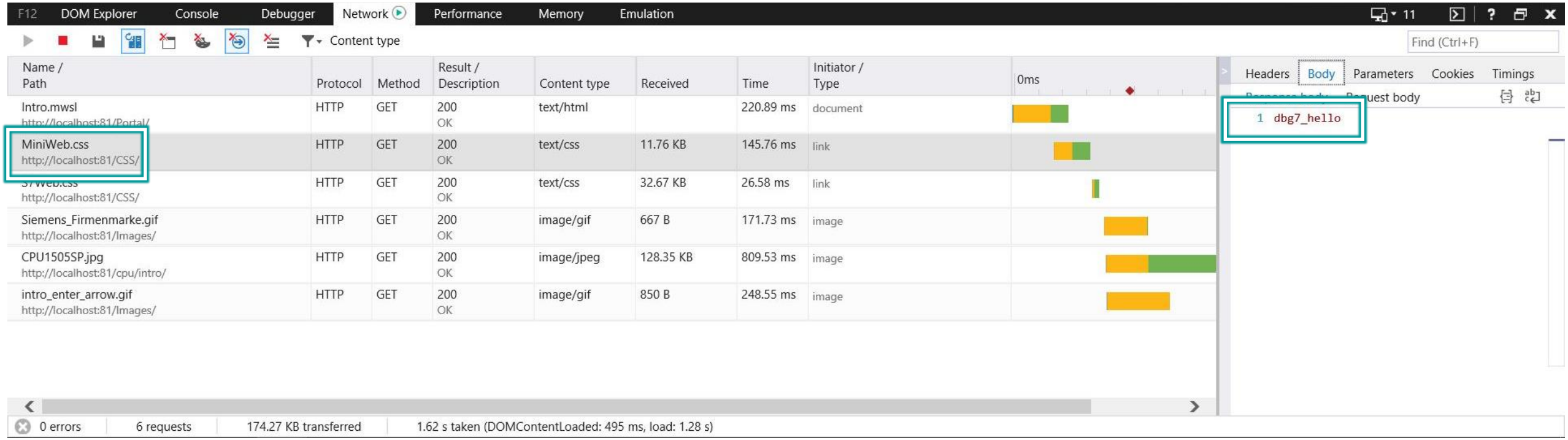
Memory

dbg7_hello



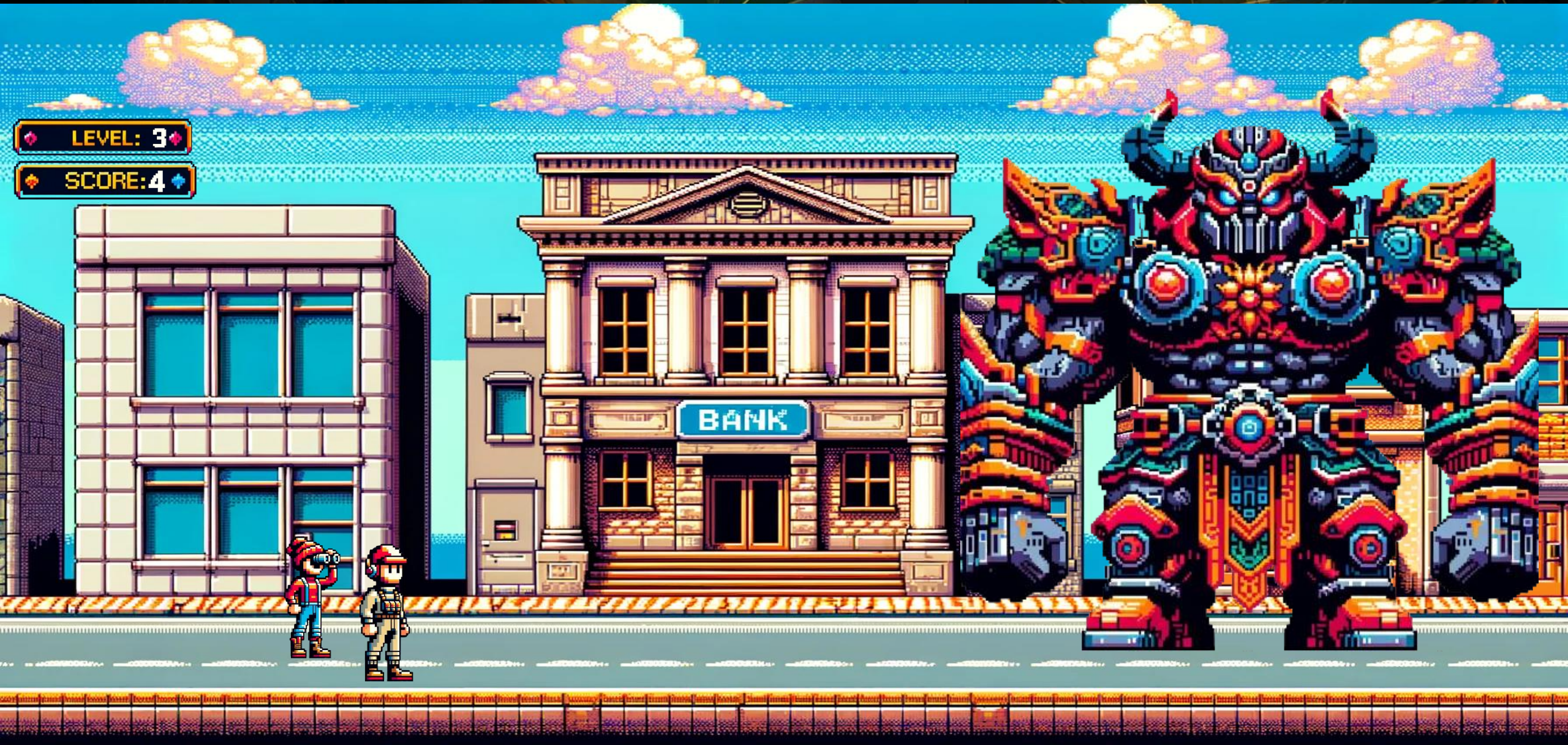
Results

- We exfiltrated data using the “MiniWeb.css” file.
  - After refreshing the home page, our data appeared on the web page.



Level 3 – Escape (Exfiltrate Data)

Boss – Build the Debugger



How to Modify the SWCPU Firmware File

- We needed to add code to the SWCPU without interrupting its normal flow.

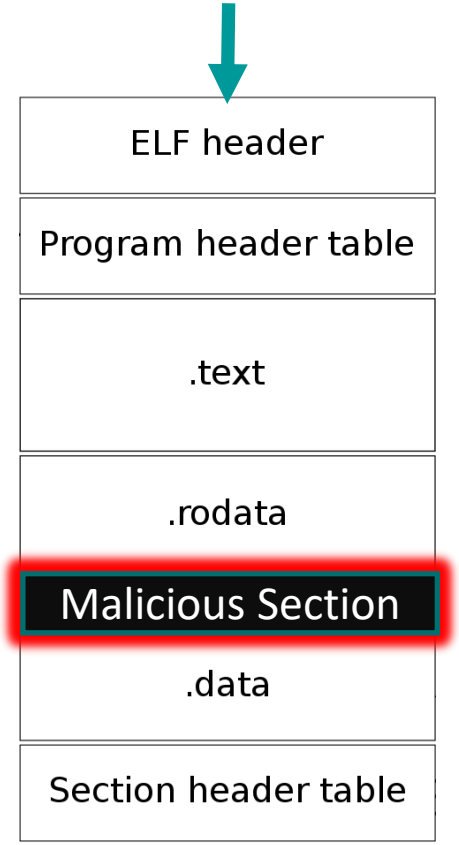
- We added a whole new section to the decrypted SWCPU file.

Our added section

Loads as RWE

```
There are 32 section headers, starting at offset 0x8708118:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .adonis_memory_ta PROGBITS        10000000 001000 000060 00  WA  0   0  1
  [ 2] .ipc_mem          NOBITS          10e00000 03e000 081640 00  WA  0   0 4096
  [ 3] .posix_shm        NOBITS          10e81640 03e000 2800000 00  WA  0   0 64
  [ 4] .jump             PROGBITS        10c00000 004940 00002c 00  WA  0   0  1
  [ 5] .rodata_kernel    PROGBITS        10c00040 004980 001a2c 00   A  0   0 32
  [ 6] .text_kernel      PROGBITS        10c01a70 0063b0 036e86 00  AX  0   0 16
  [ 7] .data_kernel      PROGBITS        10000400 001400 00353c 00  WA  0   0 1024
  [ 8] .bss_kernel       NOBITS          10004000 00493c 07f050 00  WA  0   0 4096
  [ 9] .ctors_kernel     PROGBITS        10c388f6 03d236 000008 00  WA  0   0  1
  [10] .dtors_kernel     PROGBITS        10c388fe 03d23e 000008 00  WA  0   0  1
  [11] .data             PROGBITS        18400000 2c78c40 c57b90 00  WA  0   0 32
  [12] .data_smp         PROGBITS        19057b90 38d07d0 000010 00  WA  0   0  8
  [13] .rodata           PROGBITS        19057ba0 38d07e0 ab3454 00   A  0   0 32
  [14] .hwdb_product_nam PROGBITS        19b0b000 4383c40 000020 00   A  0   0 32
  [15] .hwdb_order_id    PROGBITS        19b0b020 4383c60 000014 00   A  0   0 16
  [16] .adn_heap         NOBITS          19b0b040 4383c74 1f000000 00  WA  0   0 32
  [17] .bss              NOBITS          38b0b040 4383c74 56d5684 00  WA  0   0 64
  [18] .code_untrst      PROGBITS        13e00000 03d250 0010a1 00  AX  0   0 16
  [19] .data_untrst      PROGBITS        14000000 03f000 29ea80 00  WA  0   0 4096
  [20] .text             PROGBITS        14400000 2ddb00 24164aa 00  AX  0   0 256
  [21] .cpu1500_version  PROGBITS        168164aa 26f3faa 000015 00  WA  0   0  1
  [22] .ctors            PROGBITS        168164c0 26f3fc0 002198 00  WA  0   0  4
  [23] .time             PROGBITS        16818658 26f6158 000014 00  WA  0   0  1
  [24] .eh_frame         PROGBITS        1681866c 26f616c 581c48 00   A  0   0  4
  [25] .eh_frame_hdr     PROGBITS        16d9a2b4 2c77db4 000058 00   A  0   0  4
  [26] .section_info     PROGBITS        16d9a30c 2c77e0c 000400 00  WA  0   0  1
  [27] .gpio_section     PROGBITS        16d9a70c 2c7820c 000a00 00  WA  0   0  1
  [28] .comment          PROGBITS        00000000 4383c74 000022 01  MS  0   0  1
  [ 0] .hooks            PROGBITS        3e1e06c4 383c96 000028 00      0   0  1
                         STRTAB          00000000 707fb2 000165 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)

Elf file type is EXEC (Executable file)
Entry point 0x10c01cec
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x001000 0x10000000 0x10000000 0x0393c 0x83050 RW  0x1000
  LOAD           0x004940 0x10c00000 0x10c00000 0x38906 0x38906 RWE 0x20
  LOAD           0x03e000 0x10e00000 0x10e00000 0x00000 0x2881640 RW  0x1000
  LOAD           0x03d250 0x13e00000 0x13e00000 0x010a1 0x010a1 R E 0x10
  LOAD           0x03f000 0x14000000 0x14000000 0x29ea80 0x29ea80 RW  0x1000
  LOAD           0x2ddb00 0x14400000 0x14400000 0x299b10c 0x299b10c RWE 0x100
  LOAD           0x2c78c40 0x18400000 0x18400000 0x170b034 0x25de06c4 RW  0x40
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x100
  LOAD           0x4383   0x3e1e06c4 0x3e1e06c4 0x1300000 0x15   0 RWE G 100

 Section to Segment mapping:
  Segment Sections...
   00     .adonis_memory_table .data_kernel .bss_kernel
   01     .jump .rodata_kernel .text_kernel .ctors_kernel .dtors_kernel
   02     .ipc_mem .posix_shm
   03     .code_untrst
   04     .data_untrst
   05     .text .cpu1500_version .ctors .time .eh_frame .eh_frame_hdr .section_info .gpio_section
   06     .data .data_smp .rodata .hwdb_product_name .hwdb_order_id .adn_heap .bss
   07
   08
```

The SWCPU firmware file



The Debugger Architecture

.hooks (our added section)

Command Parser

Prefix Type Params

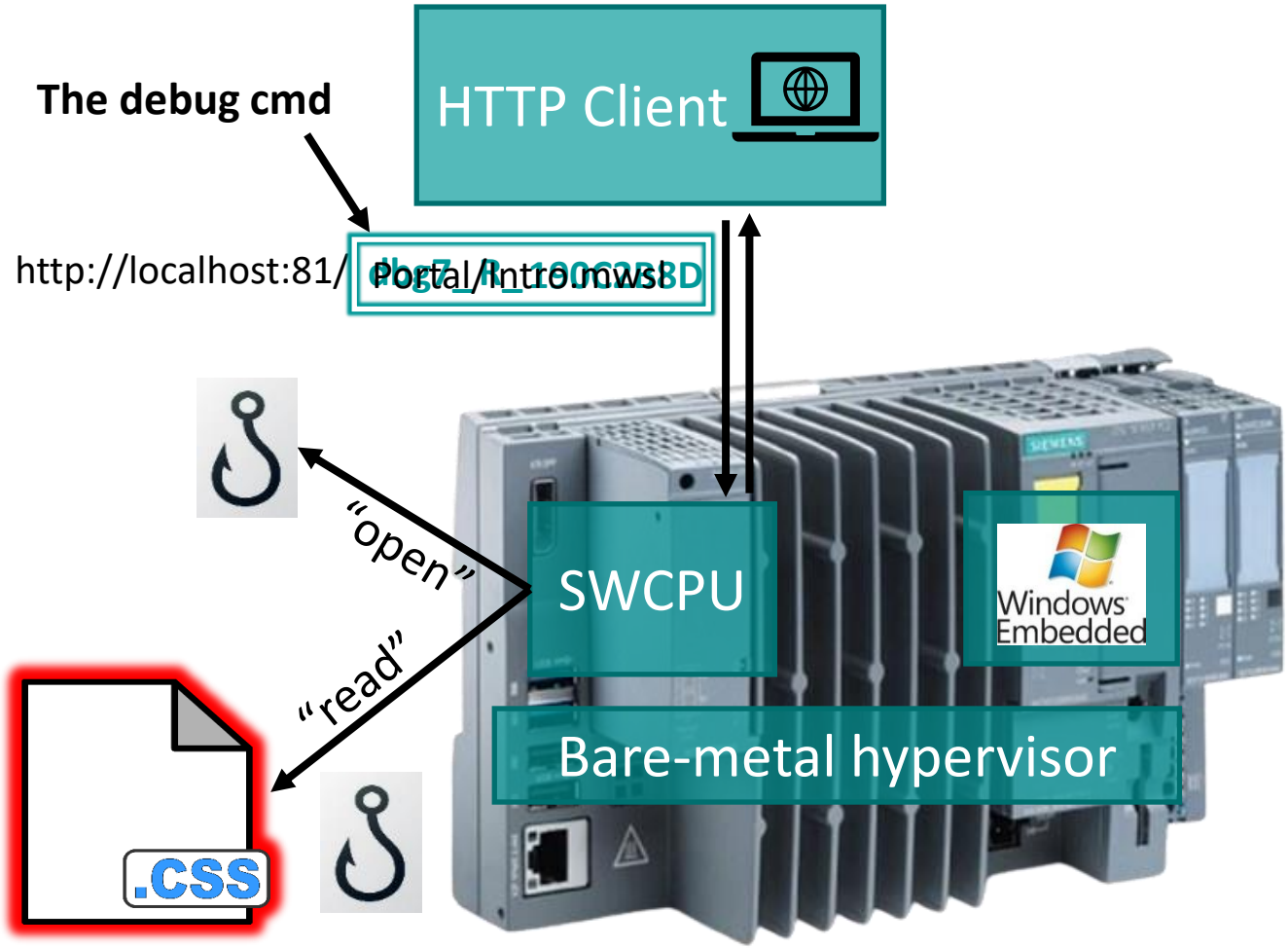
Memory

TEST_STRING

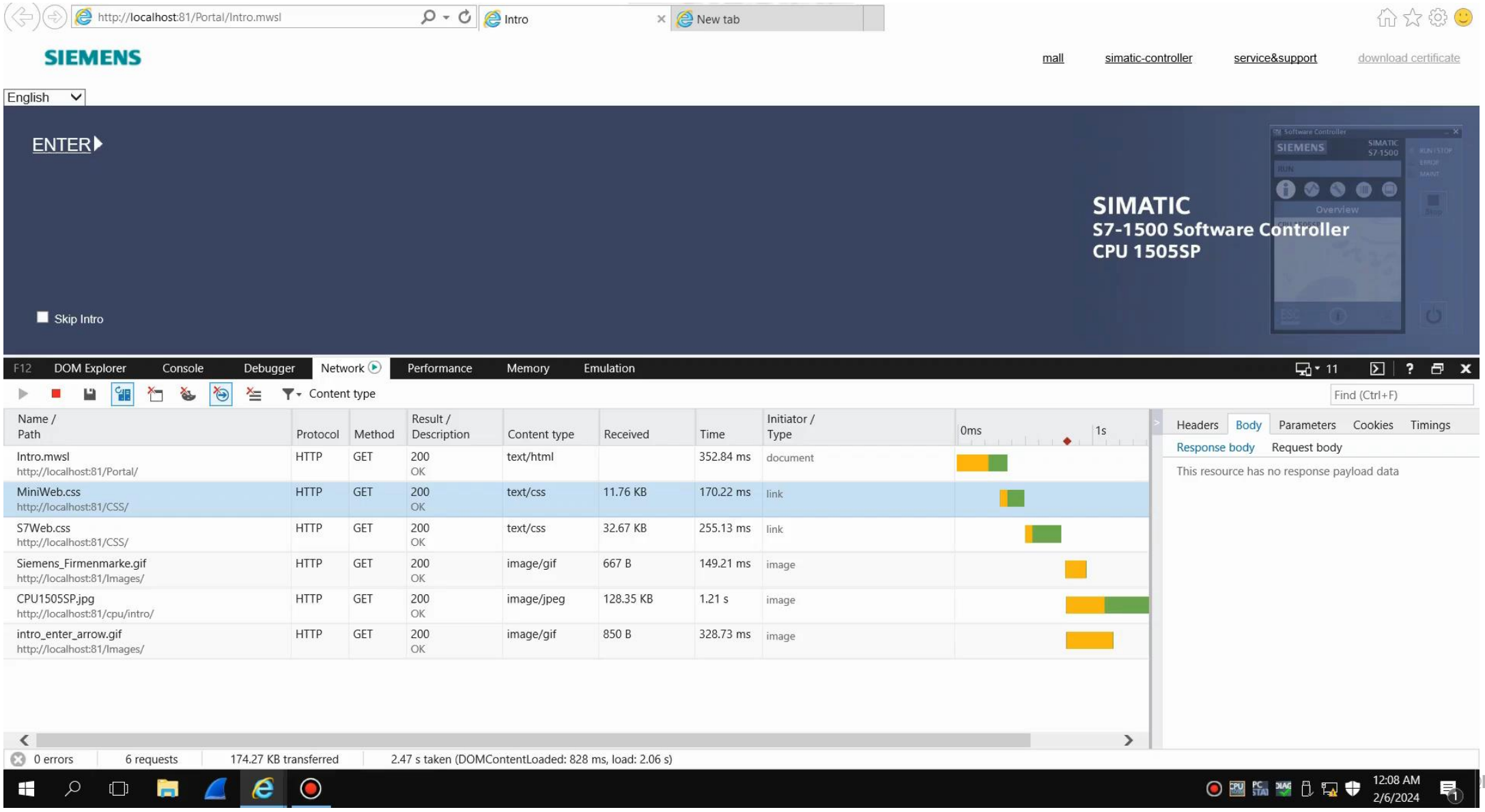
Breakpoint Handler

Write Handler

Read Handler



A Video Demonstration of Debug7



Boss – Build the Debugger

Mitigations

- Common security mitigations:
  - ASLR
  - .text section should not be writeable.
- The hypervisor should be an iron wall between guest OSs.
  - Siemens gave us an elegant backdoor to modify the SWCPU via the Windows OS.
  - This feature must be removed!
- Secure boot chain!!!
  - This is the only way to really protect the PLC from firmware modification attacks.

Research Impact

Our novel debugger is the only known method to dynamically analyze the SWCPU.

- Researchers can now thoroughly research the SWCPU firmware.

The code is shared between all of Siemens' Simatic S7 product line.

- All the Siemens' S7 PLCs are now much more vulnerable.

Security Impact

We showed you how a **persistent** trojan horse could be developed for one of the leading PLCs in the market:

- The malware is persistent and can withstand a reboot.
- It can’t be easily detected.
- We know for a fact that bad actors are looking for these capabilities, for example the Triton attack.

The longer the industry fails to properly secure PLCs, the greater the risk.

- Patching the existing PLCs in the wild is almost impossible.

How Safe Do You Really Feel?!

Demand Secure Products

Contact

Please feel free to contact us for any questions:

 thesemelbros@gmail.com

 sarab@cycloak.com

 dankner@cs.technion.ac.il

 biham@cs.technion.ac.il