



Network Protocol Reverse Engineering

Eavesdropping on the Machines

Tim Estell – BAE Systems

Katea Murray – Leidos

What is this talk about?

- Eavesdropping on the machines
 - Machines are going to have a communications protocol
 - We may not have seen it and they probably won't tell us
 - We need to break down their protocol
- Providing a repeatable process for reverse engineering protocols on your networks (there are many)
- Giving you an approach for hacking the ICS Village



What we'll cover

- Overview
 - What is protocol reverse engineering
 - Why you should care
 - How hard can it be?
- Process
 - Walk through the process steps
- Wrap Up
 - Tips and Tools
 - Staying Motivated



What is NPRE?

NPRE = Network Protocol Reverse Engineering

It's an Approach or a Process

Figuring out how machines are talking to each other so you can

- Listen in
- Control the conversation

Analysis of network data captures

- Understanding the protocols
- Breaking them down to something you can interpret



Wait, aren't there tools for that?

Yes, there are!

- libpcap and tcpdump – Available for Windows, Linux, Mac
- Wireshark – Available for Windows, Linux, Mac
- Scapy – Python based, extensible
- Fuzzing – <http://tools.kali.org/tag/fuzzing>
- IDA Pro/OllyDbg – Good for API's
- Hex editors – for modifications to packets

Unknown Protocols – Tool Limitations – Breakage



Motivation?

- Pentest – Because hexdumps won't convince the customer
- Home – Because you want to know what leaves your network
- Testing – Because developers are optimistic and/or wrong
- Monitoring – Because node forgery and impersonation are so easy
- Curiosity – You'd just like to know



How Hard Can It Be?

- People design protocols – and people are predictable



- But there are a lot of variations to pick from (such as checksums)
- Sometimes designers know they need to make it hard



Why Bother?



- Because this could be you ...
- Prior DEF CON talks (see the conference CD)
 - DC 22 – Molina; McDonal; Hoffman & Kinsey
 - DC 23 – Shipely & Gooler
- Literature Search – between 2000 and 2010 a lot of work on classification algorithms (see the conference CD)

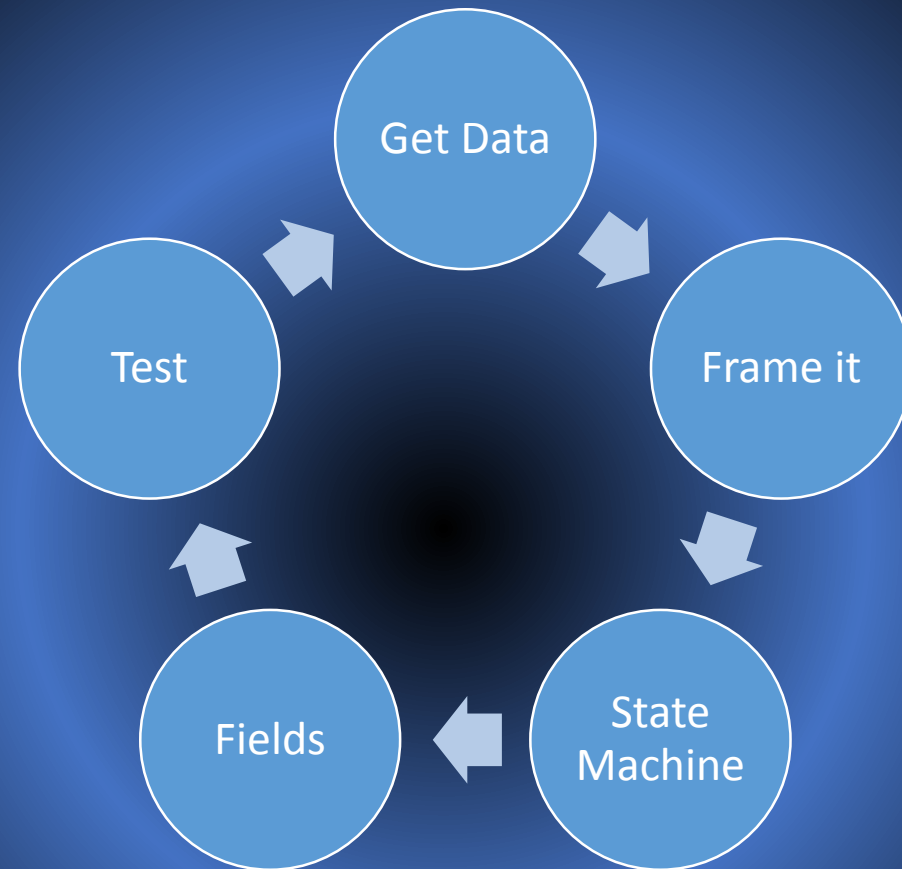


Assumptions

- Framed network protocol data
- We don't have and won't derive encryption keys
- Legal authority (only try this at home)
- “Don't be evil”



Workflow



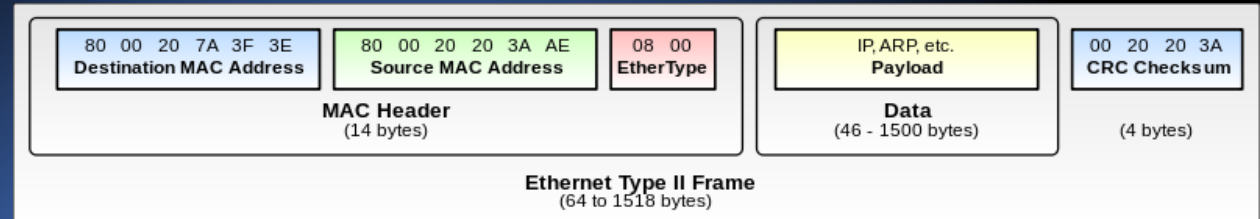
Packet Collection



- “Clean” lab environment
- Switch vs. Hub – and why span ports fail
- Cable cutting [<https://www.dgonzalez.net/papers/roc/roc.pdf>]
- Cold boot and reboot
- All-weather captures – “sunny day” to “bad weather event”
- Device management interfaces
- Setup, then test, and test, and test ...



Framing



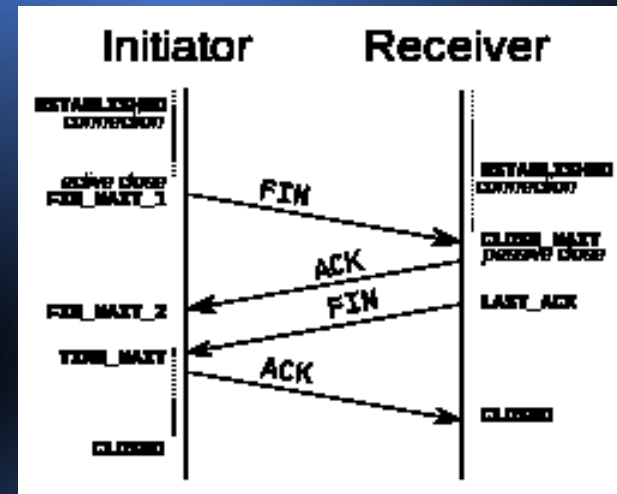
- Where the packet starts and stops – sometimes this isn't so easy
- HTML framing – how to quickly make an ugly protocol
- Fun with proprietary system bus protocols
- But – we assumed we started with framed data
 - At home it's IPv4 (or IPv6 if you're really hard core)

• Fig – <https://commons.wikimedia.org/w/index.php?curid=1546835>

State Machine

- What is a state machine?
- Figure out the message types
- Look for patterns
- Create the state chart

• Fig – https://en.wikipedia.org/wiki/File:TCP_CLOSE.svg



FitBit State Machine

- Figures from <http://arxiv.org/pdf/1304.5672v1.pdf> – “Fit and Vulnerable” by Rahman, Carbunar, Banik

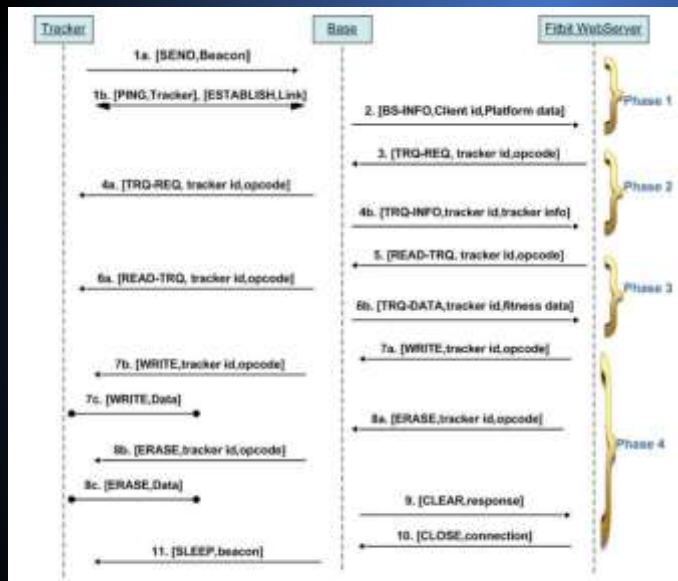
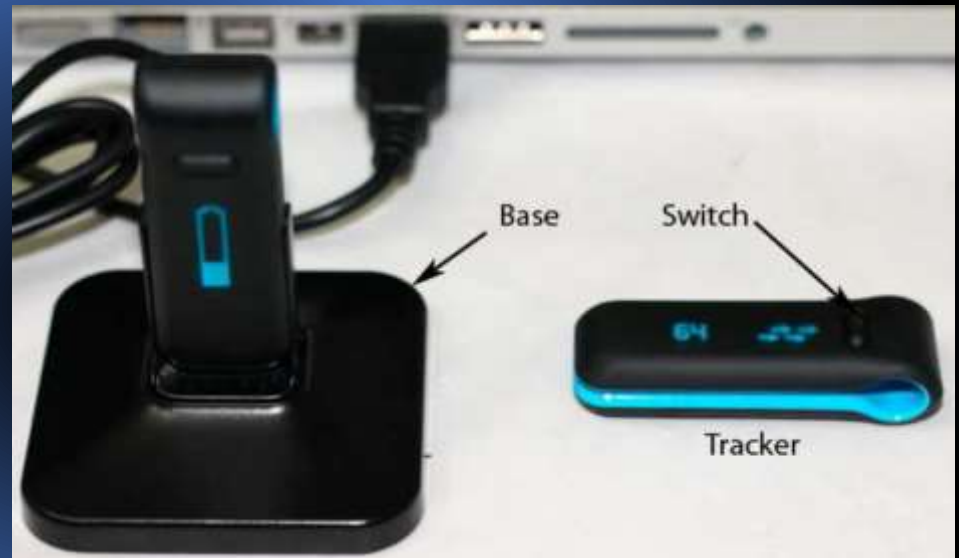
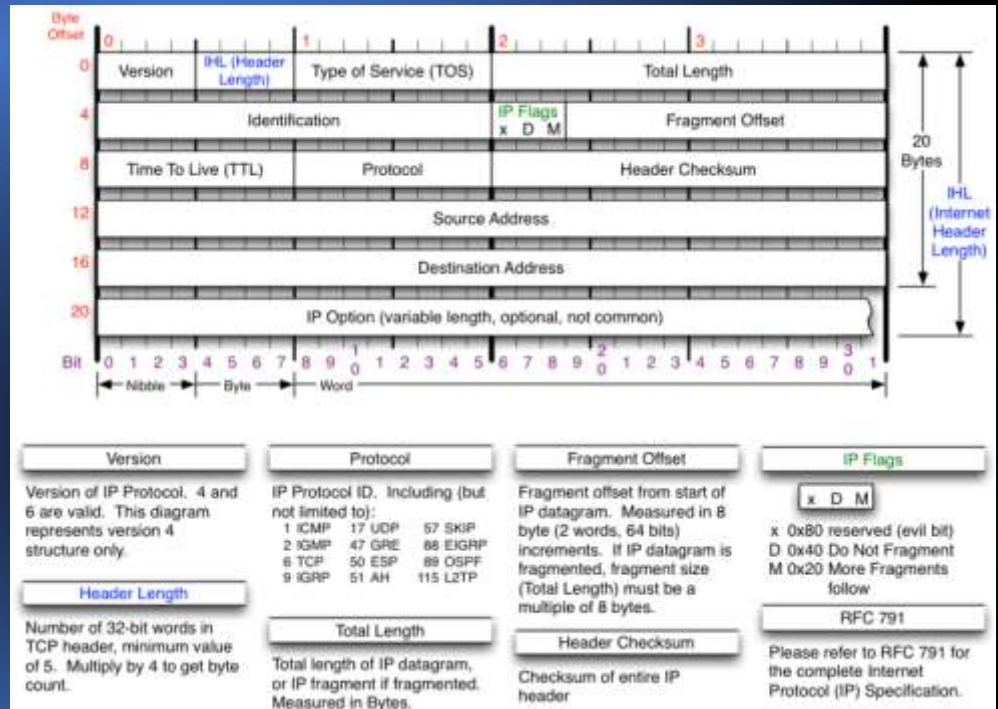


Fig. 3. Fitbit protocol between the tracker, base and the Fitbit webserver



Fields – this is where it get's fun

- String fields
- Almost string fields
- Bit fields
- Checksums
- Command values
- Everything else



Source of Figure: <https://nmap.org/book/images/hdr/MJB-IP-Header-800x576.png>
From: <https://nmap.org/book/tcpip-ref.html>

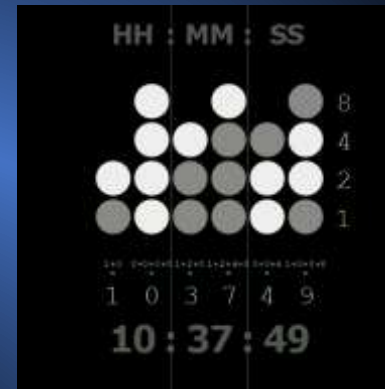
String Fields

- Easy to see in Wireshark
- Common data types:
 - XML
 - SOAP
 - HTML
 - json
- Example: ICS web interface



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help																
Apply a display filter ... <Ctrl-/>																
No.	Time		Source				Destination				Protocol		Length			
15	133.807337		PhoenixC_8c:9f:65				CadmusCo_f6:6f:d1				ARP		60			
16	156.847468		fe80::159f:c67d:78c...				ff02::c				UDP		718			
17	156.848964		169.254.236.195				239.255.255.250				UDP		698			
<p>> Frame 16: 718 bytes on wire (5744 bits), 718 bytes captured (5744 bits) on 1</p> <p>> Ethernet II, Src: BizlinkK_1f:07:ae (9c:eb:e8:1f:07:ae), Dst: IPv6mcast_0c</p> <p>> Internet Protocol Version 6, Src: fe80::159f:c67d:78ce:ecc3, Dst: ff02::c</p> <p>> User Datagram Protocol, Src Port: 50708 (50708), Dst Port: 3702 (3702)</p> <p>> Data (656 bytes)</p>																
0000	33	33	00	00	00	0c	9c	eb	e8	1f	07	ae	86	dd	60 00	33.....`.
0010	00	00	02	98	11	01	fe	80	00	00	00	00	00	00	15 9f
0020	c6	7d	78	ce	ec	c3	ff	02	00	00	00	00	00	00	00 00	.}x.....
0030	00	00	00	00	00	0c	c6	14	0e	76	02	98	71	3a	3c 3fv..q:<?
0040	78	6d	6c	20	76	65	72	73	69	6f	6e	3d	22	31	2e 30	xml vers ion="1.0
0050	22	20	65	6e	63	6f	64	69	6e	67	3d	22	75	74	66 2d	" encodi ng="utf-
0060	38	22	3f	3e	3c	73	6f	61	70	3a	45	6e	76	65	6c 6f	8"><soa p:Envelo
0070	70	65	20	78	6d	6c	6e	73	3a	73	6f	61	70	3d	22 68	pe xmlns :soap="h
0080	74	74	70	3a	2f	2f	77	77	77	2e	77	33	2e	6f	72 67	ttp://ww w.w3.org
0090	2f	32	30	30	33	2f	30	35	2f	73	6f	61	70	2d	65 6e	/2003/05 /soap-en
00a0	76	65	6c	6f	70	65	22	20	78	6d	6c	6e	73	3a	77 73	velope" xmlns:ws
00b0	61	3d	22	68	74	74	70	3a	2f	2f	73	63	68	65	6d 61	a="http: //schema
00c0	73	2e	78	6d	6c	73	6f	61	70	2e	6f	72	67	2f	77 73	s.xmlsoa p.org/ws
00d0	2f	32	30	30	34	2f	30	38	2f	61	64	64	72	65	73 73	/2004/08 /address
00e0	69	6e	67	22	20	78	6d	6c	6e	73	3a	77	73	64	3d 22	ing" xml ns:wsd="
00f0	68	74	74	70	3a	2f	2f	73	63	68	65	6d	61	73	2e 78	http://s chemas.x
0100	6d	6c	73	6f	61	70	2e	6f	72	67	2f	77	73	2f	32 30	mlsoap.o rg/ws/20
0110	30	35	2f	30	34	2f	64	69	73	63	6f	76	65	72	79 22	05/04/di scovery"
0120	3e	3c	73	6f	61	70	3a	48	65	61	64	65	72	3e	3c 77	><soap:H eader><w
0130	73	61	3a	54	6f	3e	75	72	6e	3a	73	63	68	65	6d 61	sa:To>ur n:schema

Almost String Fields



- Binary Coded Decimal (BCD)
 - Buy your own!
(<https://www.scientificsonline.com/product/powers-of-two-clock-crystal-blue-chrome-version>)
- History repeats itself
 - Fig – <https://commons.wikimedia.org/w/index.php?curid=1274824>
 - Code – https://en.wikipedia.org/wiki/Binary-coded_decimal

```
uint32_t BCDadd(uint32_t a, uint32_t b)
{
    uint32_t t1, t2;    // unsigned 32-bit intermediate values

    t1 = a + 0x06666666;
    t2 = t1 ^ b;          // sum without carry propagation
    t1 = t1 + b;          // provisional sum
    t2 = t1 ^ t2;         // all the binary carry bits
    t2 = ~t2 & 0x11111110; // just the BCD carry bits
    t2 = (t2 >> 2) | (t2 >> 3); // correction
    return t1 - t2;       // corrected BCD sum
}
```



Old Protocol Encapsulation

ICS Example:
MODBUS

The image shows a Wireshark packet capture window titled 'write_register.pcapng'. The packet list on the left shows three packets: a DHCPv6 Solicit (No. 33), a DHCP Discover (No. 34), and a Modbus query (No. 35). The packet details pane for packet 35 shows the following structure:

- Frame 35: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
- Ethernet II, Src: CadmusCo_f6:f6:d1 (08:00:27:f6:6f:d1), Dst: PhoenixC_8c:9f:65 (00:a0:45:8c:9f:65)
- Internet Protocol Version 4, Src: 192.168.0.5, Dst: 192.168.0.2
- Transmission Control Protocol, Src Port: 58158 (58158), Dst Port: 502 (502), Seq: 2, Ack: 3, Len: 12
- Modbus/TCP
 - Transaction Identifier: 9
 - Protocol Identifier: 0
 - Length: 6
 - Unit Identifier: 0
- Modbus
 - Function Code: Read Holding Registers (3)
 - Reference Number: 0
 - Word Count: 10

The packet bytes pane shows the raw data in hexadecimal and ASCII. The Modbus transaction is highlighted in blue, showing the Transaction Identifier (00 09) and the Function Code (00 03).

No.	Time	Source	Destination	Protocol	Length	Info
33	198.073079	fe80::159f:c67d:78c...	ff02::1:2	DHCPv6	154	Solicit XID: 0xd4c9d1 CID: 0001000...
34	210.197777	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7...
35	219.978011	192.168.0.5	192.168.0.2	Modbus...	66	Query: Trans: 9; Unit: 0,...

Transaction Identifier (mbtcp.trans_id), 2 bytes

Packets: 155 · Displayed: 155 (100.0%) · Load time: 0:0.33 | Profile: Default

Bit Fields and Checksums

- Fixed field values – such as IPv4 headers
- Checksums – random values (high entropy)
- Typical field sizes: 8, 16, 32
- Odd checksum calculation example – IPv4 (RFC 793):
 - Take a few fields from the IP header (Source and destination IP address, protocol, and TCP length)
 - Create a pseudo header
 - Attach this to the TCP header
 - Zero out the checksum field
 - Then calculate the checksum over the pseudo header, header, and data



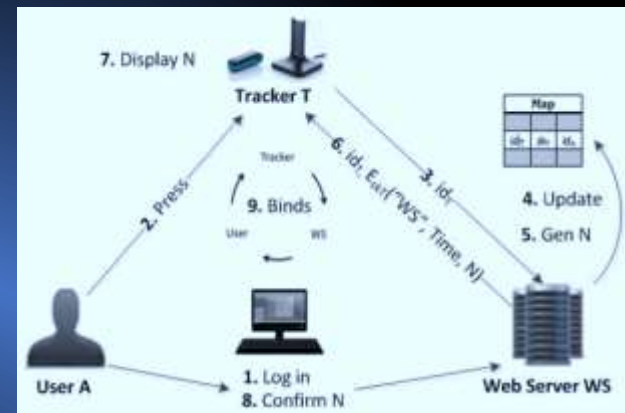
Command Values

- Bit fields with a sense of purpose
- Could be one-up values:
 - 1 = request status; 2 = status response; 3 = request temperature; 4 = current temperature; ...
- Could be constants based on a Hamming distance:
 - 0x001; 0x010; 0x100; 0x111 or 1,2,4,7
- Could be encoded (base64 or BCD)



All the rest

- The “all others” category
- Understanding what state the device was in
- Making assumptions about what the device is sending
 - Example: FitBit sends activity data base64 encoded in clear text HTTP. Understanding if the device is checking in or uploading activity values helps sort out what fields should be found.
 - [Source: Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device – <http://arxiv.org/pdf/1304.5672v1.pdf>]



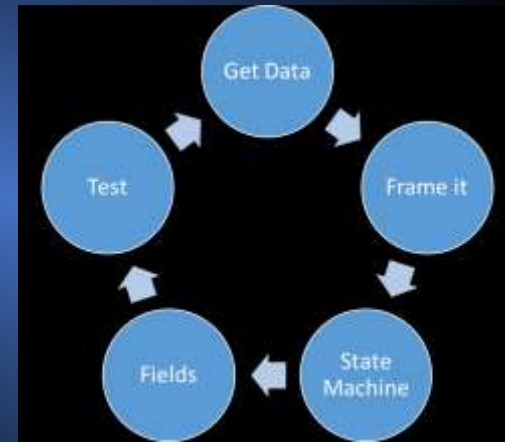
Now test

- See if you're on the right track
- Test your assumptions by spoofing communication
- Good Python tool for this is scapy
 - Workflow for Modbus hacking in ICS Village
 - Start scapy session
 - Capture a Modbus packet
 - Change the register value
 - Send the modified packet
 - See the light change – Woot!



Iterate

- Wash, rinse, repeat
- Know when you're "done enough"
- Keep refining state machine and field knowledge until:
 - there are no unknowns (good luck); or
 - you've figured out enough to do the job at hand (more feasible).



Tips

Tricks of the trade

- Find the reset switch
- Legacy modes are often weaker
- Replay
- Fuzz
- Observe where you fail
- Device discovery and management
- Status reporting



Tools

- Don't force a tool to fit a task, but leverage them when they make sense
- NetZob – <https://www.netzob.org/>. Available for Linux and Windows

“Netzob is an open source tool for reverse engineering, traffic generation and fuzzing of communication protocols. It allows to infer the message format and the state machine of a protocol through passive and active processes. The model can afterward be used to simulate realistic and controllable traffic.” Version 1.0 was released in January, 2016.



Don't Panic! (or forget your towel)

Avoiding the death march

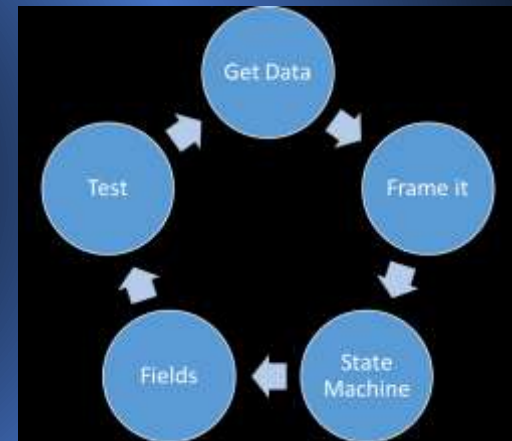
- Talk to others
 - Like the people in this room
- Don't give up!
 - looking for other projects that have solved similar challenges
- This is a game
 - SuperBetter, by Jane McGonigal



Network Protocol RE

What we covered:

- What is protocol reverse engineering
- Why you should care
- A process for NPRE
- Walk through our process steps
 - Collect data – Get some packets
 - Frame it – Figure out where the data is
 - State Machine – Understand sessions
 - Fields – Derive packet fields
 - Test – Try it out
 - Iterate – Make it better
- Tips, Tools, Don't Panic





Network Protocol Reverse Engineering: Eavesdropping on the Machines

Tim Estell – tim.stell@baesystems.com

Katea Murray – murrayka@leidos.com