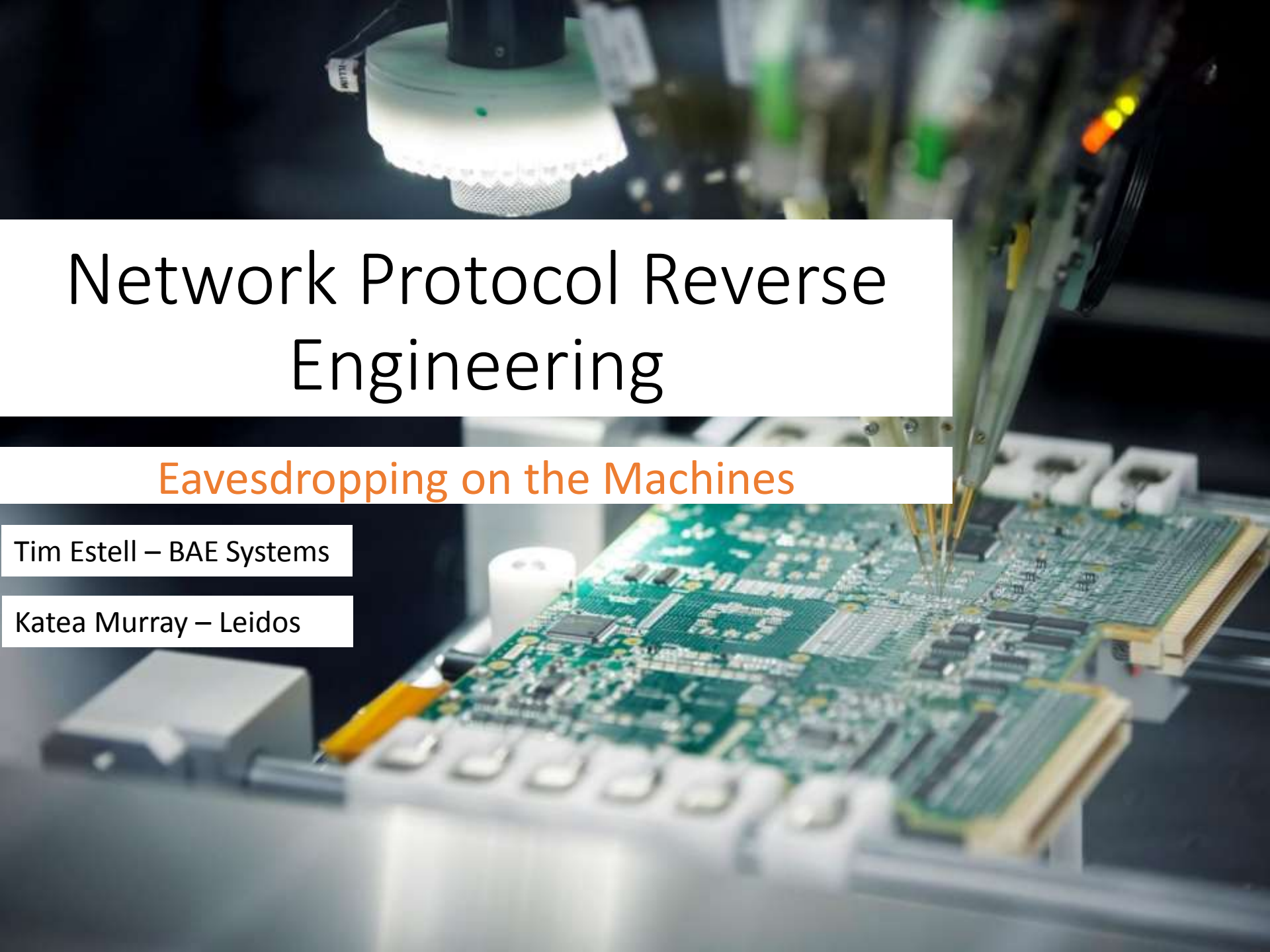


Network Protocol Reverse Engineering

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

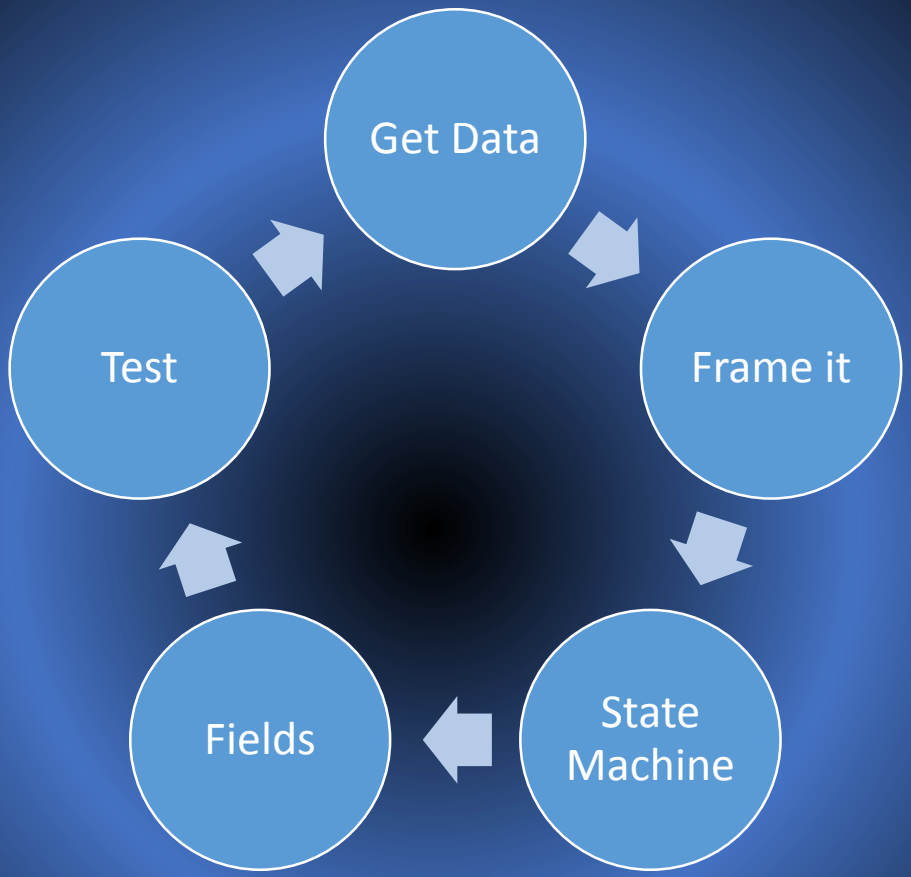
Why Bother?

- 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"

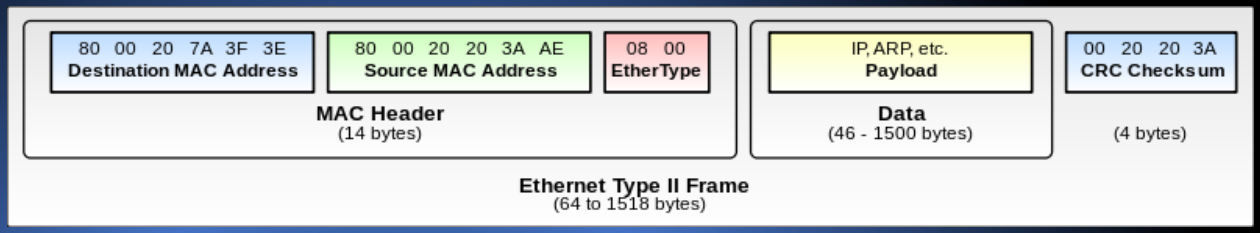
Workflow



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



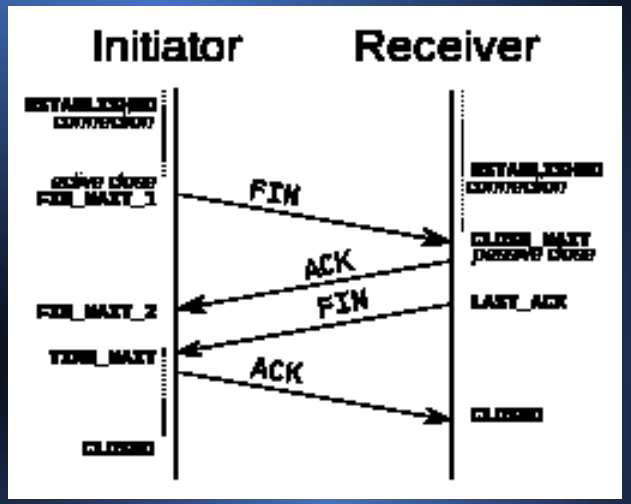
- Where the packet starts and stops – sometimes this isn't so easy
- HTML framing – how to quickly make an ugly protocol
- Fun with proprietary system bus protocols

- 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

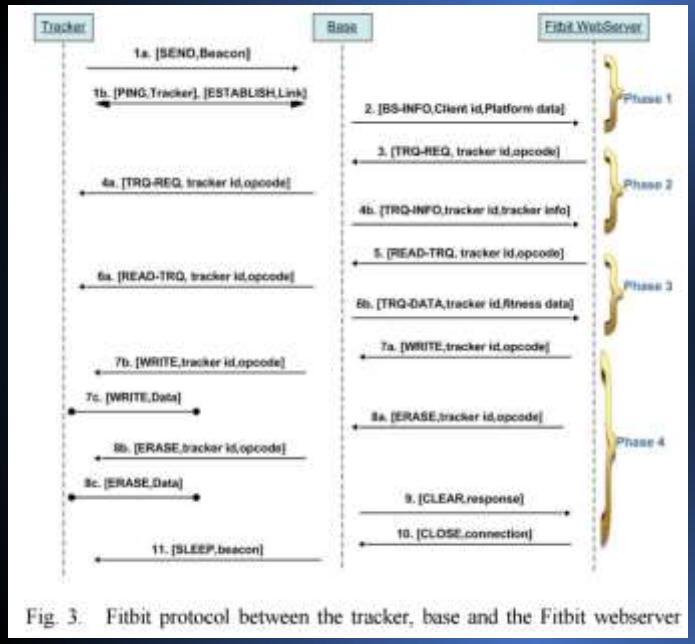
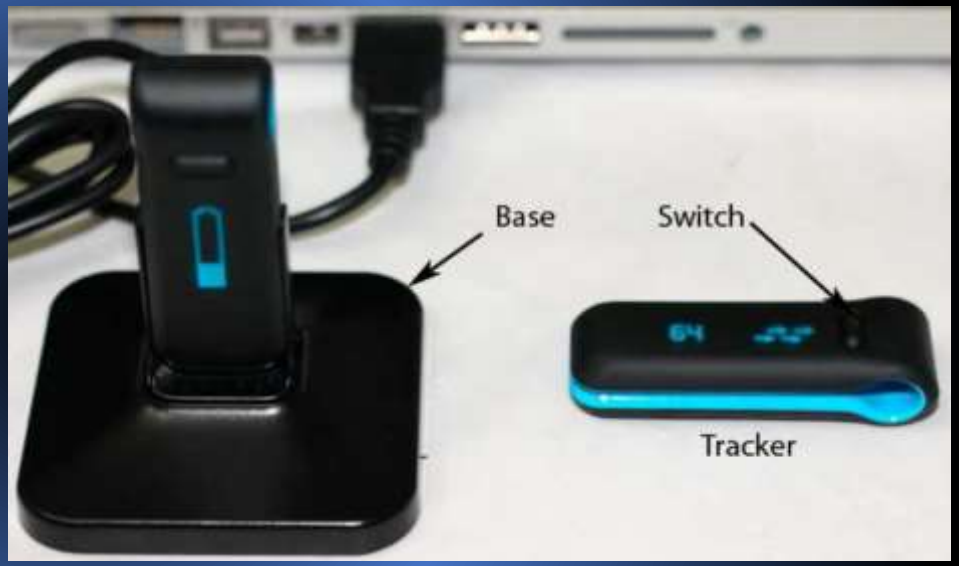
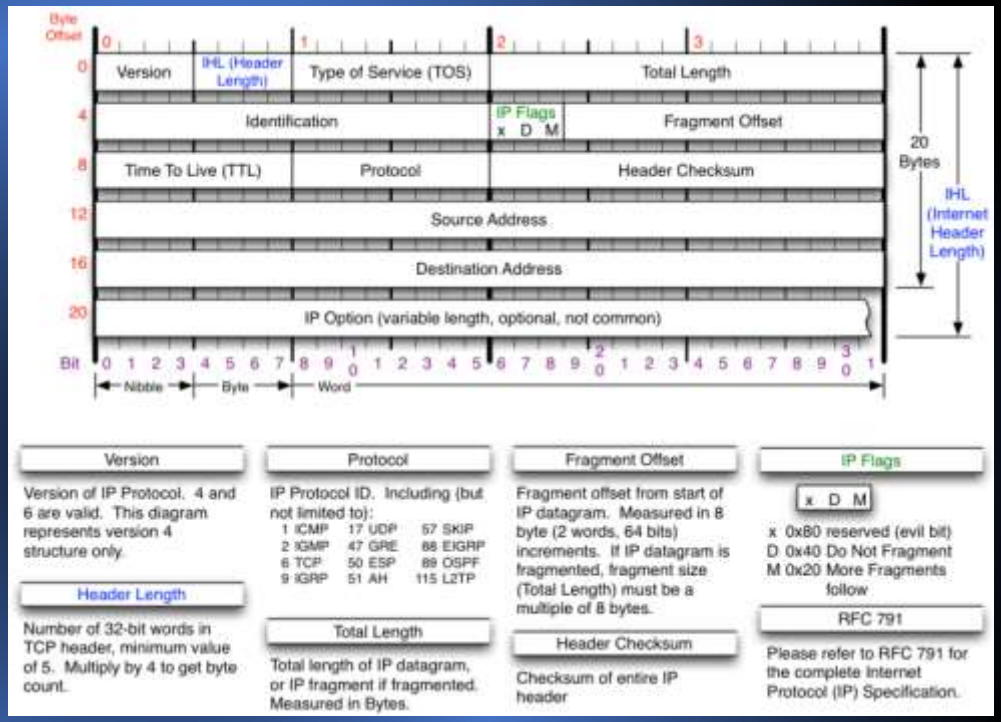


Fig. 3. Fitbit protocol between the tracker, base and the Fitbit webserver



Fields – this is where it get's fun

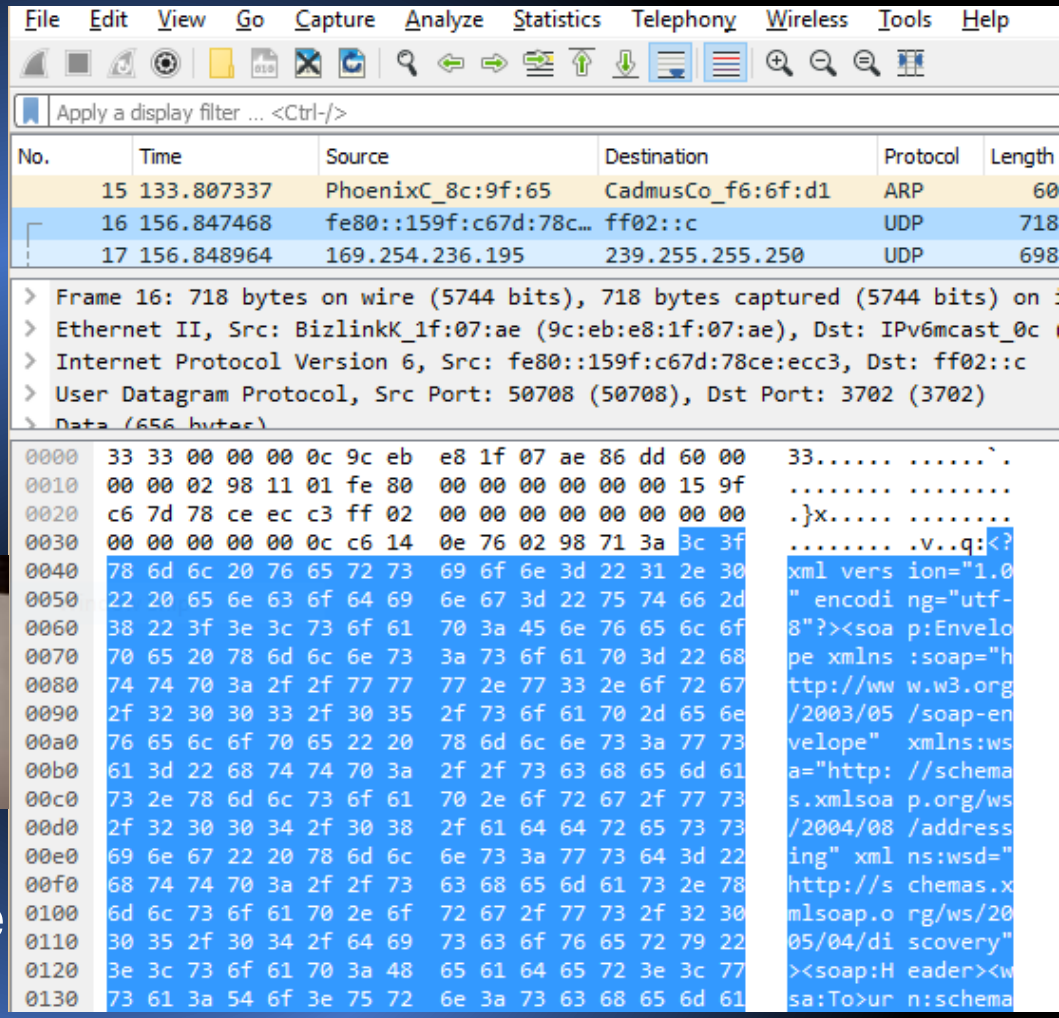
- String fields
- Almost string fields
- Bit fields
- Checksums
- Command values
- Everything else



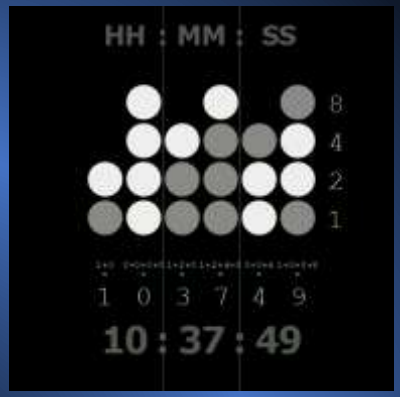
Source of Figure: https://nmap.org/book/images/hdr/MJB-IP-Header-800x576.png
From: https://nmap.org/book/tcpip-ref.html

String Fields

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



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

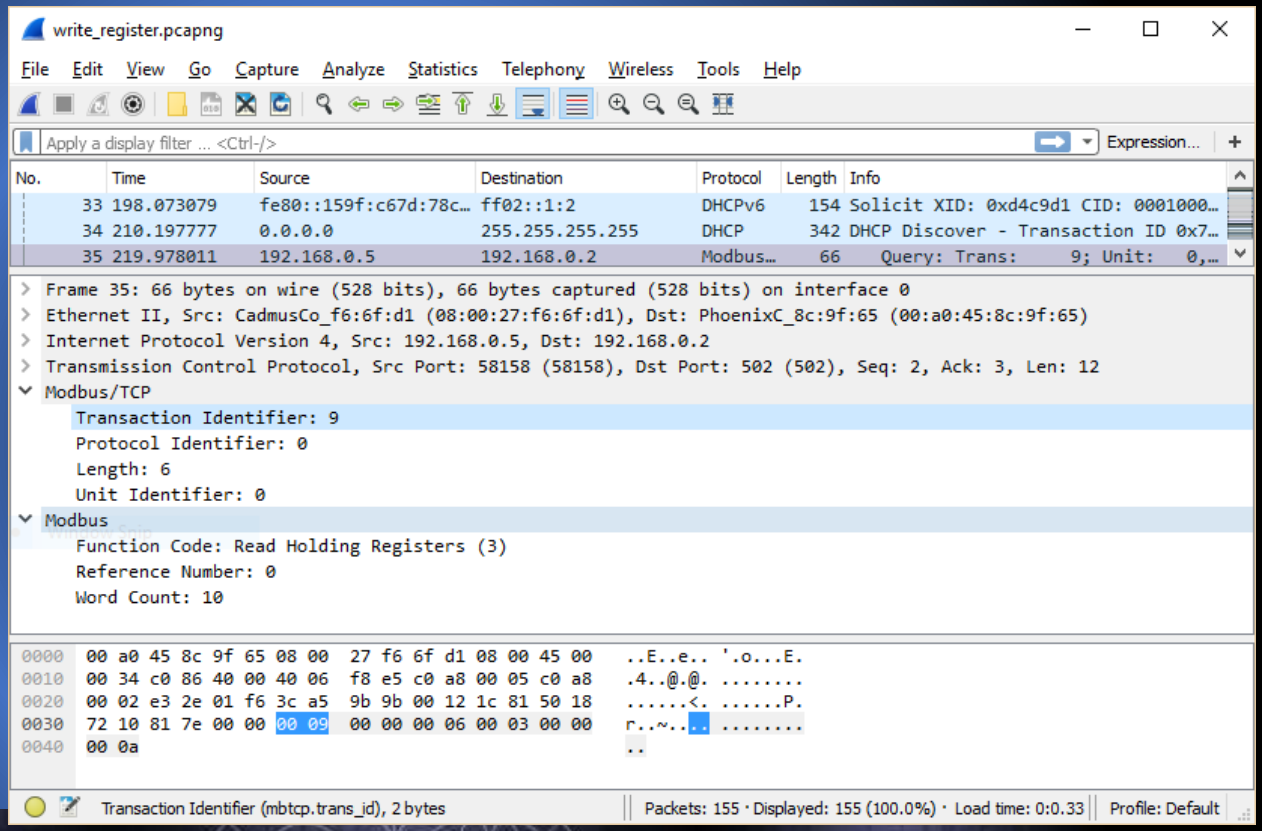
```
uint32_t BCDadd(uint32_t a,uint32_t b)
{
    uint32_t  t1, t2;    // unsigned 32-bit intermediate values

    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



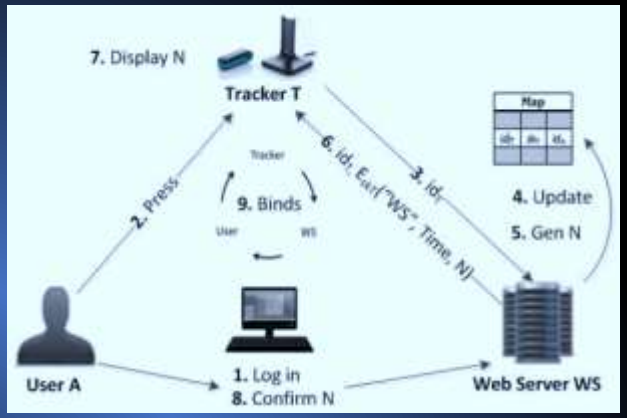
Bit Fields and Checksums

- 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 ….

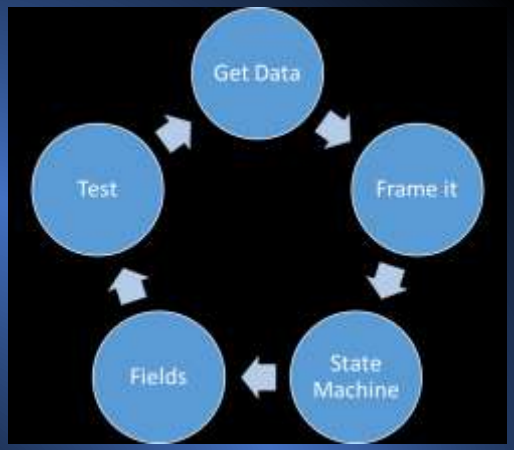


- The "all others" category
- Understanding what state the device was in
- Making assumptions about what the device is sending
  - Example: FitBit sends activity data base64 encoded in clear text HTTP. Understanding if the device is checking in or uploading activity values helps sort out what fields 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)

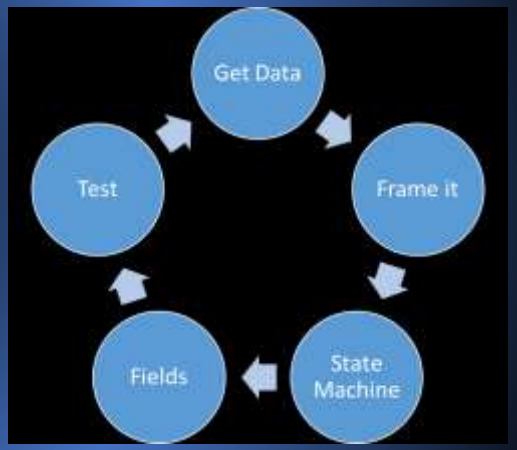
Avoiding the death march

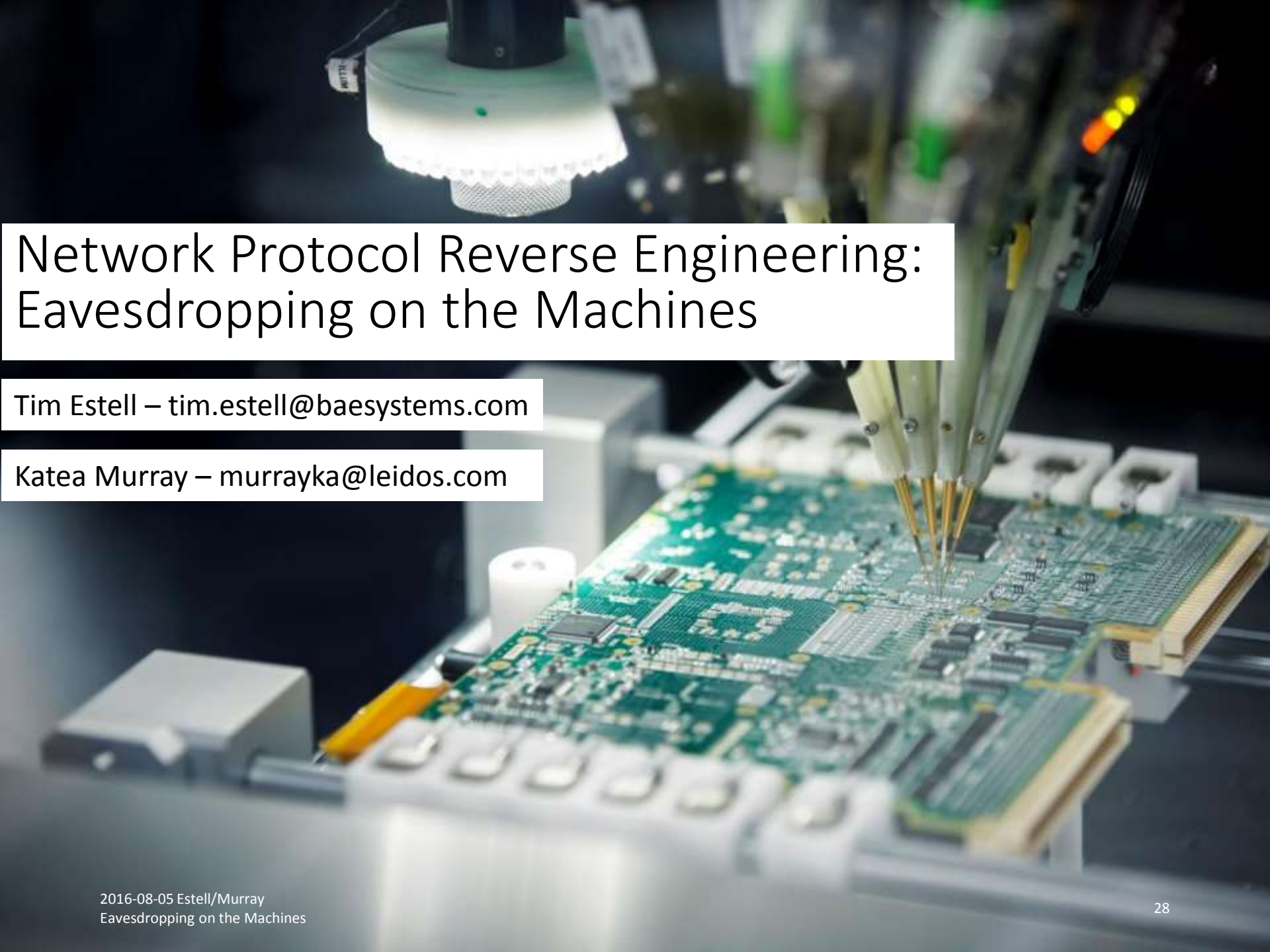
- Talk to others
  - Like the people in this room
- Don’t give up!
  - looking for other projects that have solved similar challenges
- This is a game
  - SuperBetter, by Jane McGonigal

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.estell@baesystems.com

Katea Murray – murrayka@leidos.com