Unambiguous Encapsulation

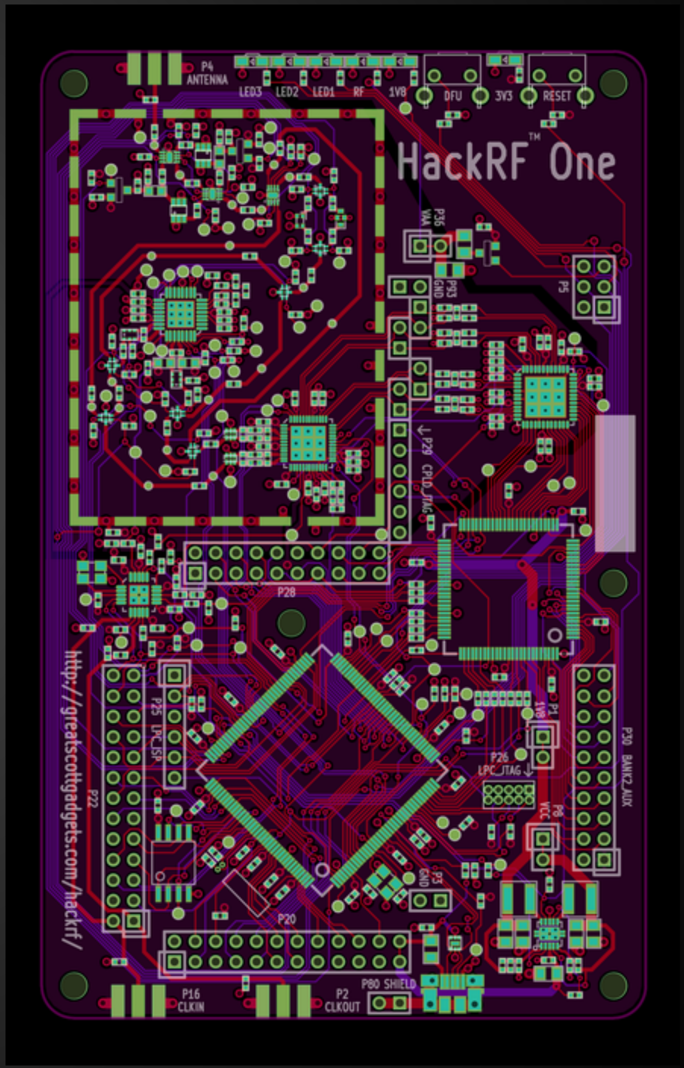
Separating Data and Signaling

Michael Ossmann

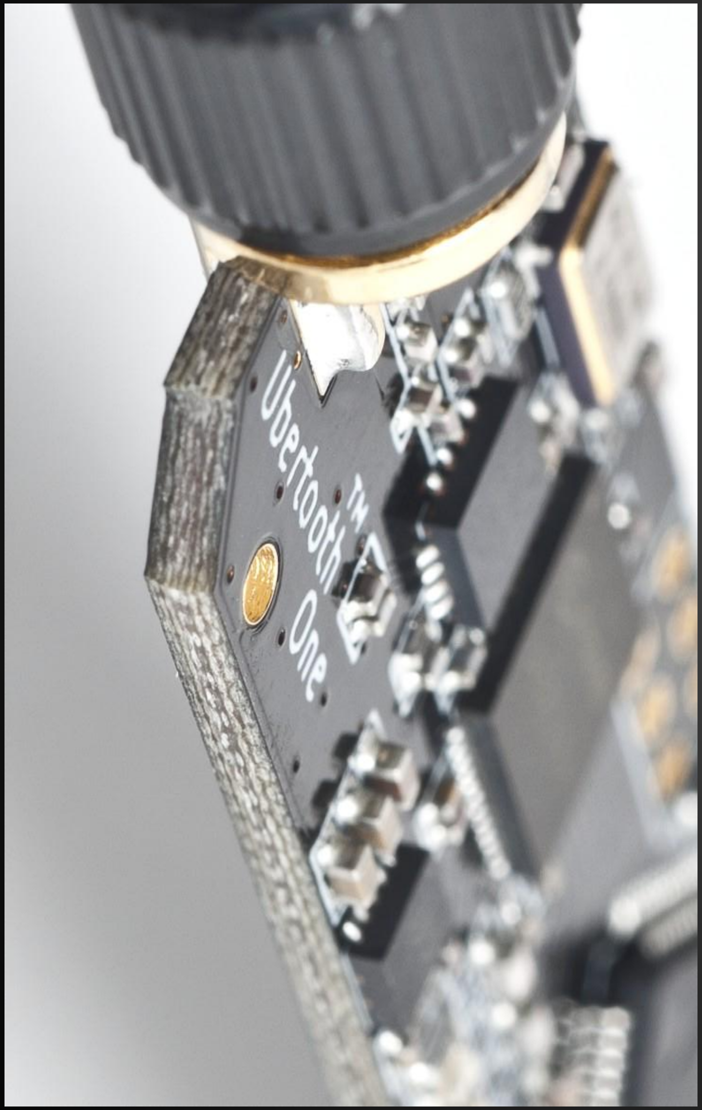
Primary on Unambiguous Encapsulation

Creator of multiple OSHW projects, Ubertooth, HackRF, Daisho, YARD Stick One

Founder of Great Scott Gadgets



Dominic Spill



Verilog for Unambiguous Encapsulation

Dev on Ubertooth, BTBB, gr-bluetooth, Daisho, USBProxy

Other projects include BeagleDancer, PS/2 tap and fcc.io

Disclaimer

The views expressed are the views of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

Outline

The Problem

Unambiguous Encapsulation

Error Control Codes

Finding Interesting Error Control Codes

Background

LANGSEC

Packets in Packets

The Problem - Packets in Packets

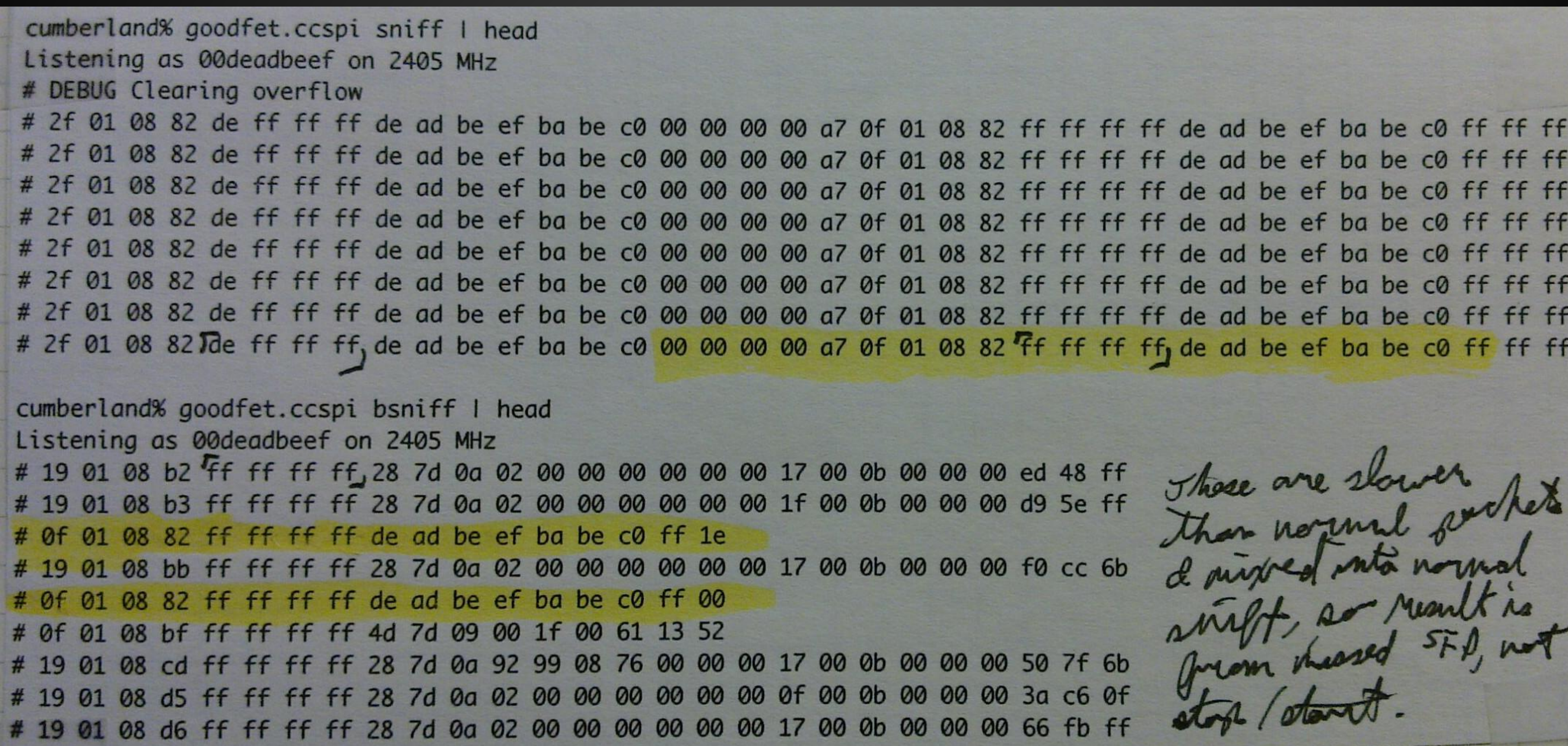
Interference or glitch obscures packet header

Second packet in payload

Receiver detects second packet

Zigbee / Ethernet susceptible

The Problem - Packets in Packets



```
cumberland% goodfet.ccspi sniff | head
Listening as 00deadbeef on 2405 MHz
# DEBUG Clearing overflow
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff
# 2f 01 08 82 de ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff ff ff

cumberland% goodfet.ccspi bsniff | head
Listening as 00deadbeef on 2405 MHz
# 19 01 08 b2 ff ff ff ff 28 7d 0a 02 00 00 00 00 00 00 17 00 0b 00 00 00 ed 48 ff
# 19 01 08 b3 ff ff ff ff 28 7d 0a 02 00 00 00 00 00 00 1f 00 0b 00 00 00 d9 5e ff
# 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff 1e
# 19 01 08 bb ff ff ff ff 28 7d 0a 02 00 00 00 00 00 00 17 00 0b 00 00 00 f0 cc 6b
# 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff 00
# 0f 01 08 bf ff ff ff ff 4d 7d 09 00 1f 00 61 13 52
# 19 01 08 cd ff ff ff ff 28 7d 0a 92 99 08 76 00 00 00 17 00 0b 00 00 00 50 7f 6b
# 19 01 08 d5 ff ff ff ff 28 7d 0a 02 00 00 00 00 00 00 0f 00 0b 00 00 00 3a c6 0f
# 19 01 08 d6 ff ff ff ff 28 7d 0a 02 00 00 00 00 00 00 17 00 0b 00 00 00 66 fb ff
```

These are slower than normal packets & mixed into normal sniff, so result is from missed SFD, not stop/start.

Credit: Travis Goodspeed

Ethernet Too!

Packet-In-Packet on wired Ethernet

```
 --------------------------------------------------------------------
| Idle | SSD | Preamble | SFD | Data | SFD | Data | FCS | ESD | Idle |
 --------------------------------------------------------------------
```

```
17:47:15.972801 00:1f:16:37:b1:3d > 00:22:6b:dc:c6:55, ethertype IPv4
(0x0800), length 1104: (tos 0x0, ttl 64, id 20574, offset 0, flags [none],
proto UDP (17), length 1090)
    192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
        0x0000:  0022 6bdc c655 001f 1637 b13d 0800 4500  ."k..U...7.=..E.
        0x0010:  0442 505e 0000 4011 62f1 c0a8 0001 c0a8  .BP^..@.b.......
        0x0020:  420a 927d 0035 0024 0000 c007 0100 0001  B..}.5.$........
        0x0030:  0000 0000 0000 0667 6f6f 676c 6503 636f  .......google.co
        0x0040:  6d00 0001 0001 0000 749c 9b85 0000 0000  m.......t.......
        0x0050:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        .....
        0x01f0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0200:  2165 c8fe 0000 0000 0000 0000 0000 0000  !e..............
        0x0210:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        .....
        0x0400:  0055 5555 5555 5555 d500 1f16 37f2 ff00  ..UUUUUU....7...
        0x0410:  1f16 37b1 3d08 0045 0000 3900 0040 0040  ..7.=..E..9..@.@
        0x0420:  0616 bb0a 0108 020a 0108 0102 9a02 9a00  ................
        0x0430:  0000 0000 0000 0050 0200 004f 5500 0000  .......P...OU...
        0x0440:  0000 0000 0000 0000 0066 6f6f 6261 7200  .........foobar.
```


Fully arbitrary 802.3 packet injection

Credit: Andrea Barisani and Daniele Bianco

The Problem - Buffer Overflow

User supplied data written to buffer

Overwrite data on stack

CPU executes data as instructions

Ambiguous Encapsulation

Given a piece of data without context, it is not possible to determine if it is meta-data or encapsulated data

CSV File Format

Widely used example of ambiguous encapsulation

Fields separated by commas

Records separated by newline

CSV File Format

How do you represent commas or newlines within fields

- Escaping?
- Quoting?

Meaning of comma or newline is ambiguous without state

CSV File Format

How do you write a parser for the CSV format?

Do you try to recognize every escaping and quoting mechanism anyone has ever used?

```
mossmann@falstaff /usr/lib/python3.2 $ grep guess csv.py
                        self._guess_quote_and_delimiter(sample, delimiters)
                delimiter, skipinitialspace = self._guess_delimiter(sample,
    def _guess_quote_and_delimiter(self, data, delimiters):
    def _guess_delimiter(self, data, delimiters):
mossmann@falstaff /usr/lib/python3.2 $ 
```

CSV Example

shmoocon,january,"washington, dc"
defcon,july,las vegas
toorcon,october,san diego

Base64 Delimited Example

c2htb29jb24=,amFudWFyeQ==,d2FzaGluZ3RvbiwgZGM=
ZGVmY29u,anVseQ==,bGFzIHZlZ2Fz
dG9vcmNvbg==,b2N0b2Jlcg==,c2FuIGRpZWdv

Base64 Delimited File Format

We can remove even more parsing variations by getting rid of newlines

A complete file format specification and example implementation for the Hammer parsing library are in our repo:

http://github.com/mossmann/unambiguous-encapsulation

Base64 Delimited File Format

Fields are base64 encoded

Field separator is ‘,’

Record separator is ‘.’

Header field separators is ‘;’

Header record separator is ‘:’

These do not appear in base64 encoded data

Similar: Interpolique

http://dankaminsky.com/interpolique/

Base64 encoding to prevent SQL injection

Ambiguous Encapsulation

Given a piece of data without context, it is not possible to determine if it is meta-data or encapsulated data

Unambiguous Encapsulation

Given a piece of data without context, it is possible to determine if it is meta-data or encapsulated data

If you haven’t found the analog medium beneath a particular bit or byte, keep digging

Error Control Codes

Error control codes are used at the boundary between analog and digital

Can we find error control codes that provide useful encapsulation properties?

Error Control Codes

Encapsulate data in codewords

Binary Linear Block Codes encode k data bits in n bit codewords with a minimum Hamming distance d

Often designated by $[n,k]$ or $[n,k,d]$

[7,4,3] Hamming Code

0101010

1000011

0000000

1101001

1011010

0110011

1110000

0011001

1001100

1100110

0001111

0100101

1111111

0111100

0010110

1010101

Each codeword is 7 bits long, $n = 7$
There are 2^4 codewords, $k = 4$
At least 3 bits differ between any two codewords, $d = 3$

[7,4,3] Hamming Code

codeword length = 7

number of codewords = 2^4

minimum Hamming distance = 3

One bit flipped: error corrected

Two bits flipped: error detected

Three bits flipped: undetected error

Implementation

[7,4,3] Hamming encoder:

look-up table: 16 * 7 bits

[7,4,3] Hamming decoder:

look-up table: 128 * 4 bits

Much of the complexity of coding theory is related to clever decoding methods, but a look-up table works for shorter (small *n*) codes

Brute Force Coding

Decoding by look-up table is sort of a brute force approach

We can also take a brute force approach to the discovery of new codes

A [5,3,2] Code

00000 01110
00011 10110
00101 11010
01001 11100

Hamming Distance = 2

Isolation

A code can be thought of as a pair of complementary sub-codes.

A [5,3,2,3] Isolated Complementary Binary Linear Block Code (ICBLBC)

codeword length = 5

number of codewords = 2^3

minimum Hamming distance = 2

minimum isolation = 3

One bit flipped: error detected

Two bits flipped: undetectable error, isolated

Three bits flipped: isolation broken

Searching for codes

icblbc.c

- C program to brute force search for codes

- Depth First Search recursive algorithm

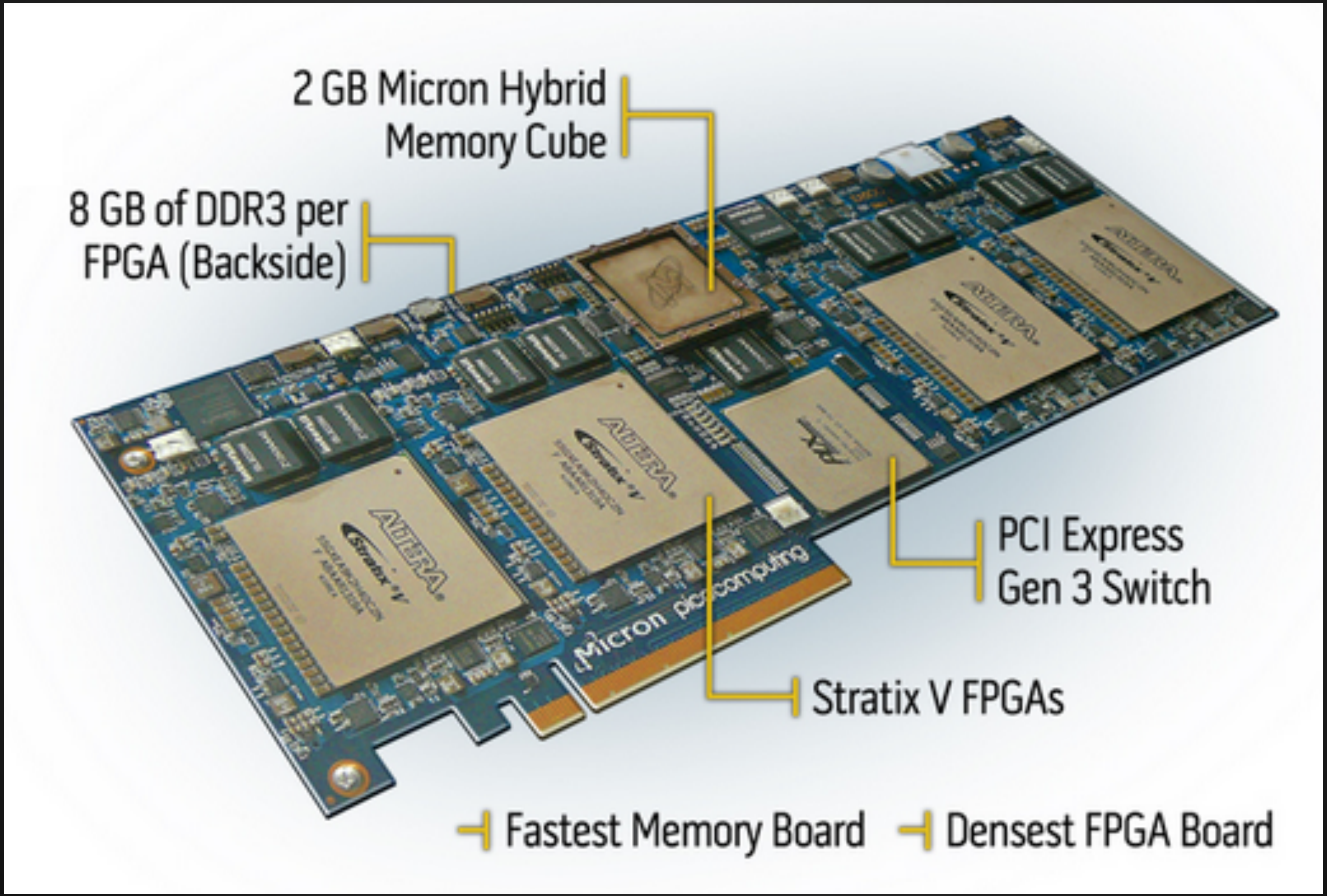
Demonstration

Searching for longer codes

FPGA

- Verilog implementation of icblbc in progress
- Recursion difficult unless we know max depth of recursion at compile time
- Modified algorithm for maximum depth of recursion

Soon . . .



2 GB Micron Hybrid
Memory Cube
8 GB of DDR3 per
FPGA (Backside)
PCI Express
Gen 3 Switch
Stratix V FPGAs
Fastest Memory Board
Densest FPGA Board

Code Selection

Error control codes are typically selected based on:

- code rate (k/n)
- complexity of decoder
- probability of undetectable error
- probability of uncorrectable error

We suggest an addition to this list:

- probability of encapsulation breakage

Unambiguous Encapsulation

Any time you encapsulate data within other data, consider unambiguous encapsulation

Thank You

LANGSEC community

DARPA Cyber Fast Track

David Hulton

Mike Kershaw

Questions?

http://github.com/mossmann/unambiguous-encapsulation

Email: mike@ossmann.com

Twitter:

@michaelossmann

@dominicgs