

Hacking Your Way To More Secure Code

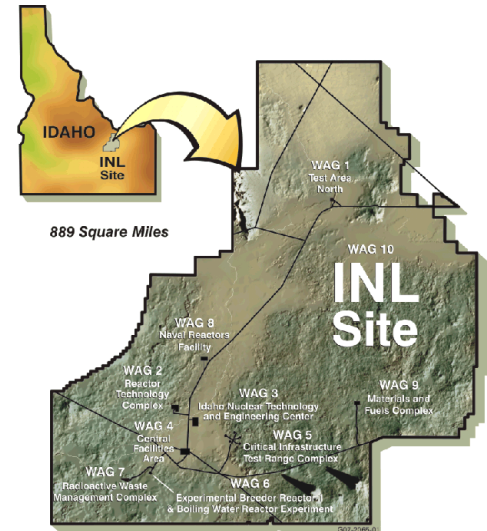
Robert Erbes, Kenneth Rohde

INL

2 December 2009

Where/What is the INL?

- Department of Energy (DOE) National Laboratory located in Idaho Falls, ID
- Primary mission is “sustainable energy systems”
- Cyber security research team working to secure critical infrastructure since 2003



Who are we?

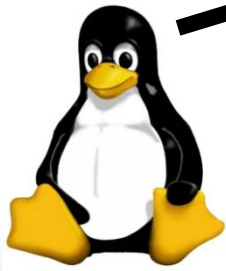
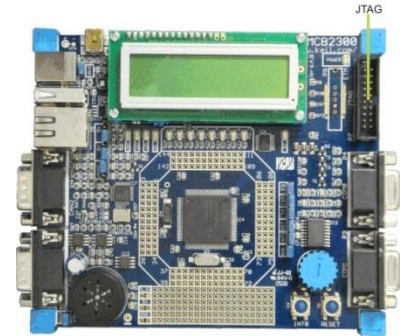
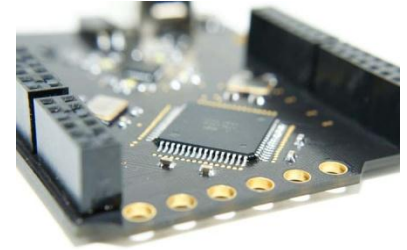
- **Cyber security researchers**
 - **Support the DOE National SCADA Test Bed (NSTB)**
 - **Support the DHS Industrial Control Systems CERT (ICS-CERT)**
 - **Majority are Computer Scientists/Engineers by education**
 - **Hackers by hobby and trade**
 - **Around 20 full-time researchers**



Outline and Objectives

- **A little motivation**
- **Common vulnerability locations**
 - **Where, why, and how**
- **SQL Injection**
- **Fuzzing**
 - **OPC UA**

Where are today's hackers looking?



Motivation


- **Network perimeter defenses will never be adequate**
 - **Cannot rely upon firewalls**
 - **Cannot rely upon IDS/IPS**
- **Our software systems have the same problem**
 - **Hard and crunchy on the outside (sometimes)**
 - **Soft and chewy in the center (almost always)**
- **Start with the outside and work your way in...**

Common Vulnerability Locations

- **Web applications**
- **Custom applications**
- **SCADA software**
- **Protocols**



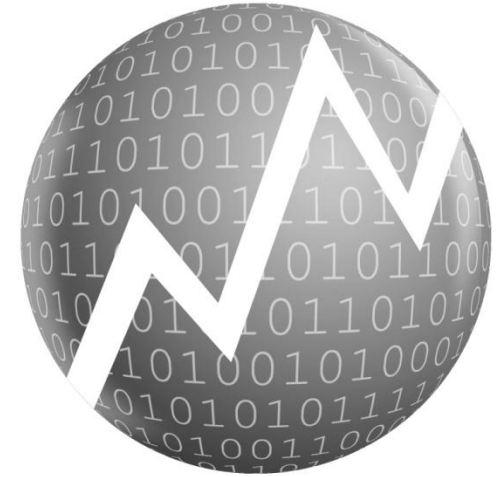
Web Applications

- **Where**
 - **DMZs**
 - **Corporate Networks**
 - **Why**
 - **Usually not created by experienced developers**
 - **Use SCADA vendor plugins, SDKs, APIs, protocols**
 - **How**
 - **SQL injection**
 - **XSS**
- 



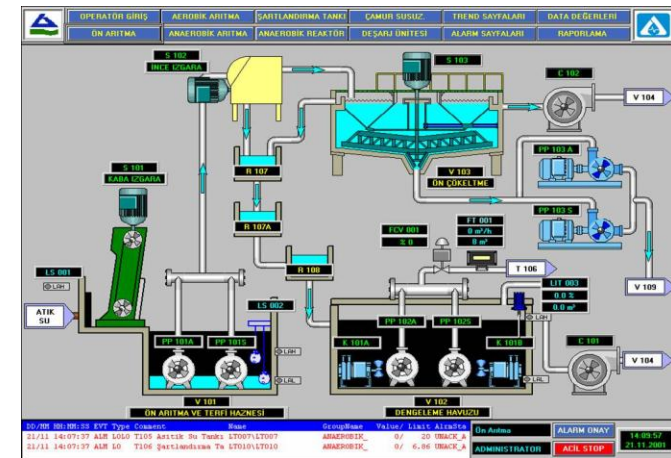
Custom Applications

- **Where**
 - **DMZs**
 - **“Outside” locations**
- **Why**
 - **Often implemented in C/C++**
 - **Generally very old (software and hardware)**
- **How**
 - **Memory corruption issues (buffer overflows)**
 - **Design (logic) problems**

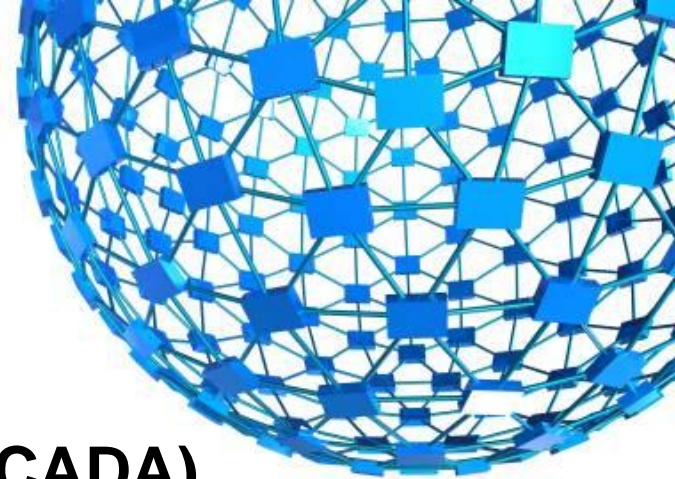


SCADA Software

- **Where**
 - **Critical infrastructure networks**
 - **Technically everywhere...**
- **Why**
 - **Coollest (and worst) 0-day ever**
- **How**
 - **Binary reverse engineering**
 - **Sometimes access to source code**

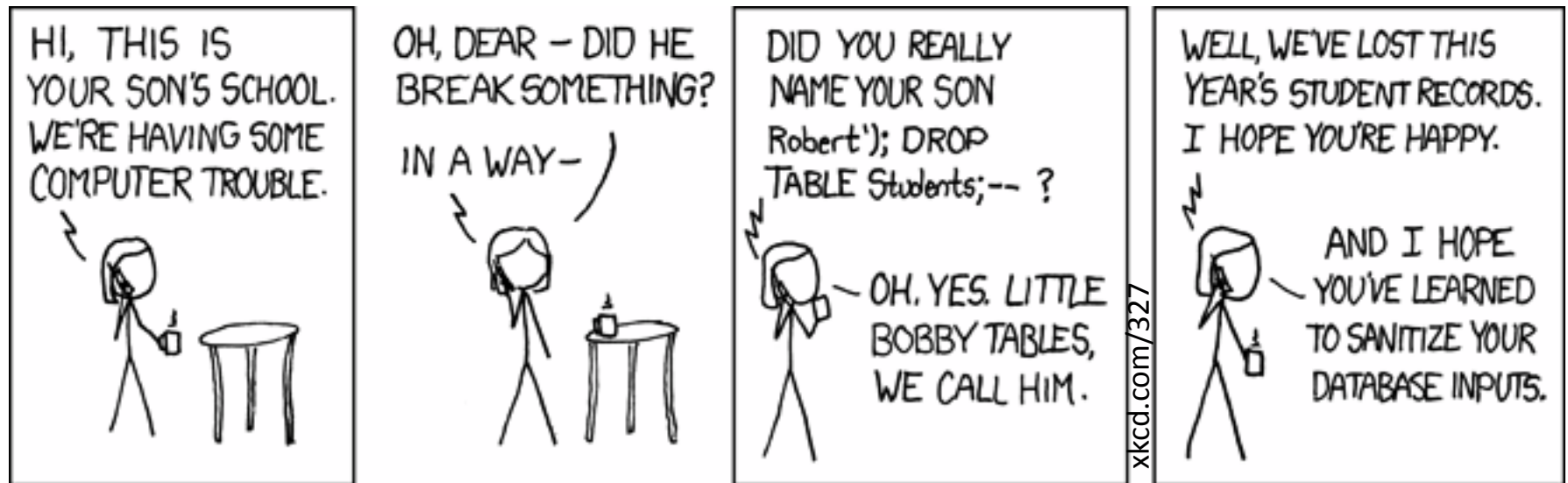


Protocols



- **Where**
 - **Almost all networks (not just SCADA)**
- **Why**
 - **Often cross network boundaries**
 - **Proprietary == no IDS or firewall support**
- **How**
 - **Lots of patience and decoding**
 - **Fuzzing**

SQL Injection



Classic SQL Injection

SELECT * WHERE student = '[studentName]';

+

studentName = "Robert'; DROP TABLE Students;--"

=

SELECT * WHERE student = 'Robert'; DROP TABLE Students;--';

Protection from SQL Injection

- **Sanitization**
 - **Escaping**
 - **White lists**
- **Principle of least privilege**
- **Let someone else worry about it**
 - **i.e., use methods that are safe**

Fuzzing in Detail

First Things First: Time for a Demo

Fuzzing defined

“The original work was inspired by being logged on to a modem during a storm with lots of line noise. And the line noise was generating junk characters that seemingly was causing programs to crash. The noise suggested the term ‘fuzz’.” – Barton Miller, 2005



www.bioweb.uncc.edu

What can/should be fuzzed

- Network protocols
 - Remote Services
- File formats
- Option switches
- APIs
- et cetera



More Generally: any input that crosses a “security boundary”

Why Fuzz?

- **Makes more robust applications.**
- **Makes more secure applications.**
- **Microsoft does it.**
- **“Hackers” do it.**

Buying vs. Building

Buying

- \$\$\$
- Some one else does it for you (good)
- Some one else does it for you (bad)
- Is it even possible to buy a fuzzer for your super secret protocol?

Building

- Time == \$\$\$
- Easier to customize
- Easier to integrate into SDL
- Are you sneaky enough?

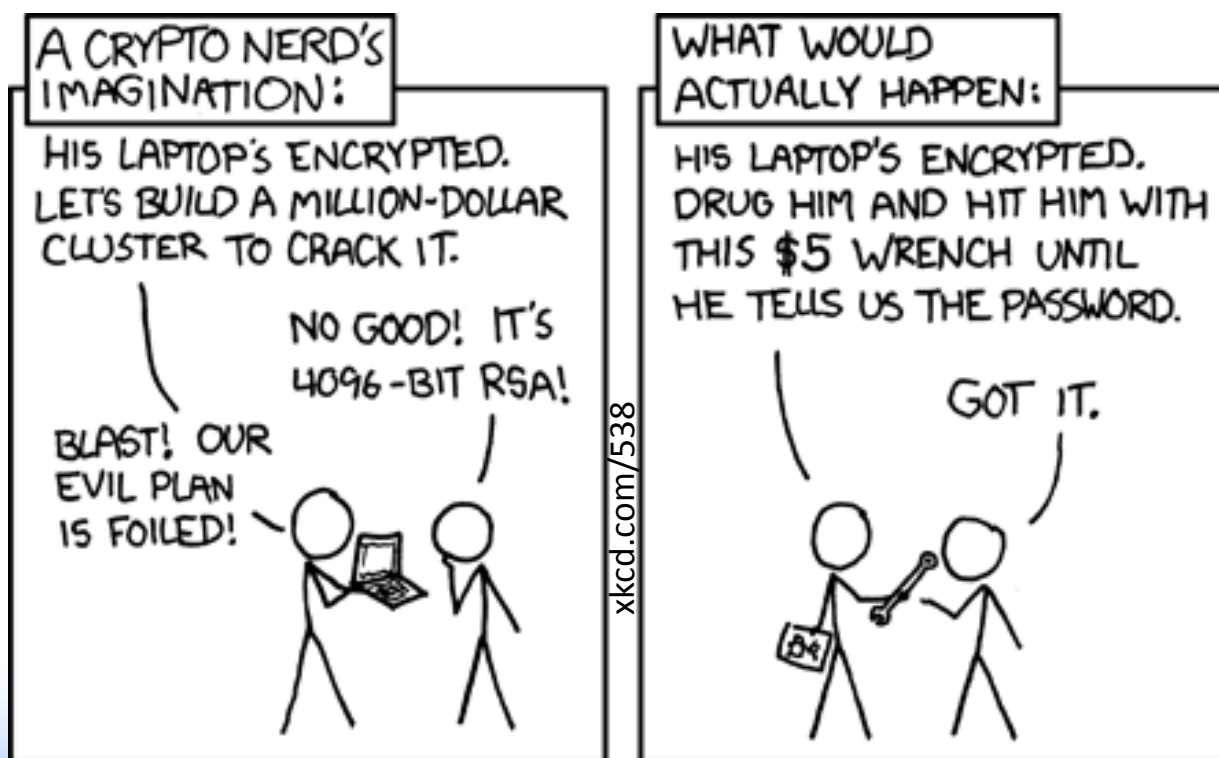
When building...

- **Speak the Language. Uh... I mean... protocol.**



When building...

- Be malicious. Break ***all*** assumptions. Don't ***just*** interact with the target.



When building...

- **Be careful what you reuse**
 - **Layers**
- **Be careful of your assumptions**
 - **Maybe someone else should fuzz your code?**
 - **QA team**
- **Be mindful of the targets expectations**

Types of Fuzzing

Dumb ☹️

- **Simple injections/manipulations**
 - 3 Million “A”s (0x41 ftw)
 - DWORD slide
 - Bitflips
- **Easily foiled by simple CRC**
- **Run while developing Smart fuzzer**

Smart 😊

- **Can account for CRC/other checks**
- **Aware of structure**
- **Aware of state**
- **Aware of relationships**

What you'll likely find

- Depending on what you're fuzzing...
 - Buffer overflows
 - Access violations
 - Other memory management problems
 - Pointer errors
 - Arithmetic
 - Null
 - Type conversion errors
 - State machine/Logic problems
 - Resource consumption (DoS)
 - More general parsing errors, crashes, and hangs
 - 2nd order vulnerabilities
 - Information disclosure

Running the Fuzzer

- **Require some sort of event/anomaly detection**
 - **Debuggers (duh)**
 - **Memory analysis**
 - **Watchdog programs**
 - **ping / netcat**
- **One test case, one test**

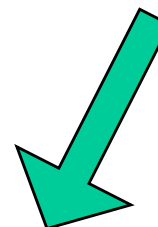
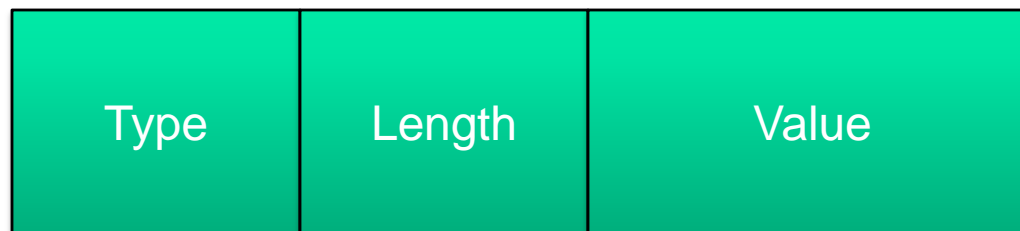
Other Important Things

- **Randomization and Repeatability**
- **Connections / Layers**
- **Failures and the continuation of testing**

FUZZING HOW-TO

The network protocol example

- Type
- Length
- Value



OPC UA Binary Hello Message

HelloMsg.bin - Data																			
Len: \$0000004B					Type/Creator:					/					Sel: \$0000004B:0000004B / \$00000000				
00000000:	48	45	4C	46	4B	00	00	00	00	00	00	00	FF	FF	00	00	HELFK.....		
00000010:	FF	FF	00	00	00	00	80	02	00	00	00	00	2B	00	00	00+...		
00000020:	6F	70	63	2E	74	63	70	3A	2F	2F	37	34	2E	32	31	37	opc.tcp://74.217		
00000030:	2E	31	30	31	2E	32	31	31	3A	39	30	30	31	2F	55	41	.101.211:9001/UA		
00000040:	2F	50	49	55	41	53	65	72	76	65	72						/PIUAServer		

Types

- Most often an explicit indication of what's to come.



OPC UA Binary Example

HelloMsgOSI.bin - Data									
Len: \$0000004B		Type/Creator:		/		Sel:		\$00000000:00000003 / \$00000003	
00000000:	48 45 4C 46	4B 00 00 00	00 00 00 00	FF FF 00 00	HEL	FK.....			
00000010:	FF FF 00 00	00 00 80 02	00 00 00 00	2B 00 00 00+	...			
00000020:	6F 70 63 2E	74 63 70 3A	2F 2F 31 37	32 2E 31 36	opc.tcp://172.16				
00000030:	2E 31 30 37	2E 31 30 30	3A 39 30 30	31 2F 55 41	.107.100:9001/UA				
00000040:	2F 50 49 55	41 53 65 72	76 65 72		/PIUAServer				

Message Type field: 0x48 0x45 0x4C == "HEL"

Chunk Type field: 0x46 == "F"

Version field: 0x00000000 == "0"

Breaking Types

- **Blatantly invalid**
 - **“0x00 0x0A 0x0D”**
- **Mismatched**
 - **“HEL” type/header followed by “OPN” Message**
- **Missing**

Lengths

- **Length, count-of, offset, delimiter, array index**
- **Explicit vs. Implicit**
- **Multi-layer length relationships**
- **Variable length length fields**

OPC UA Binary Example

HelloMsgOSI.bin - Data									
Len: \$0000004B		Type/Creator:		/		Sel: \$00000000:00000000 / \$00000000			
00000000:	48 45 4C 46	4B 00 00 00	00 00 00 00	FF FF 00 00	HELFK.....				
00000010:	FF FF 00 00	00 00 80 02	00 00 00 00	2B 00 00 00+...				
00000020:	6F 70 63 2E	74 63 70 3A	2F 2F 31 37	32 2E 31 36	opc.tcp://172.16				
00000030:	2E 31 30 37	2E 31 30 30	3A 39 30 30	31 2F 55 41	.107.100:9001/UA				
00000040:	2F 50 49 55	41 53 65 72	76 65 72		/PIUAServer				

Message Length: 0x4B000000 (little endian) == 75 bytes

Receive Buffer Size: 0xFFFF0000 == 65535 bytes

Send Buffer Size: 0xFFFF0000 == 65535 bytes

Max Message Size: 0x00008002 == 41943040 bytes

Max Chunk Count: 0x00000000 == 0

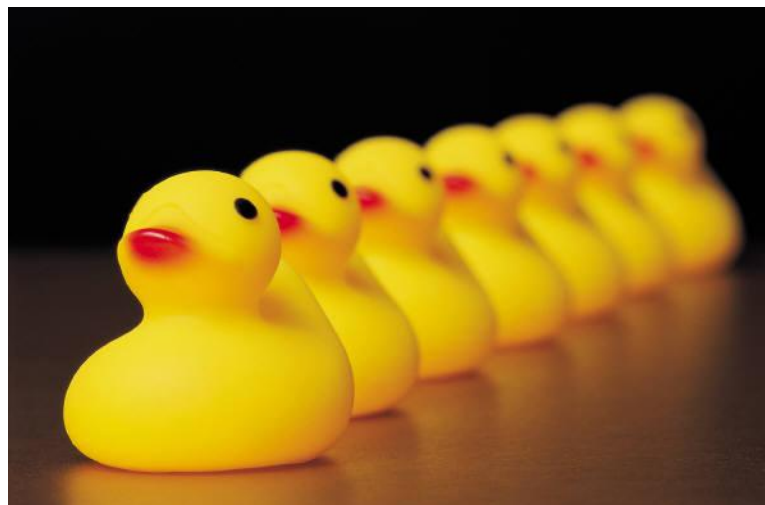
String Length: 0x2B000000 == 43 bytes

Breaking Lengths

- Invalidate relationships
- Number boundaries
 - 16-bit number has at least five
- Common buffer sizes
 - Powers of 2 for small → Powers of 10 for big
- Strings representing numbers / lengths
- Excessive use / manipulation of delimiters
- Combo Length + Delimiter relationships

Values

- **Anything.**
 - **Numbers, Strings/Text, Blobs, Tokens... clip-art**



OPC UA Binary Example

HelloMsgOSI.bin - Data									
Len: \$0000004B		Type/Creator: /		Sel: \$00000000:00000000 / \$00000000					
00000000:	48 45 4C 46	4B 00 00 00	00 00 00 00	FF FF 00 00	HELFK.....				
00000010:	FF FF 00 00	00 00 80 02	00 00 00 00	2B 00 00 00+...				
00000020:	6F 70 63 2E	74 63 70 3A	2F 2F 31 37	32 2E 31 36	opc.tcp://172.16				
00000030:	2E 31 30 37	2E 31 30 30	3A 39 30 30	31 2F 55 41	.107.100:9001/UA				
00000040:	2F 50 49 55	41 53 65 72	76 65 72		/PIUAServer				

Endpoint URL == “opc.tcp://172.16.107.100:9001/UA/PIUAServer”

Breaking Values

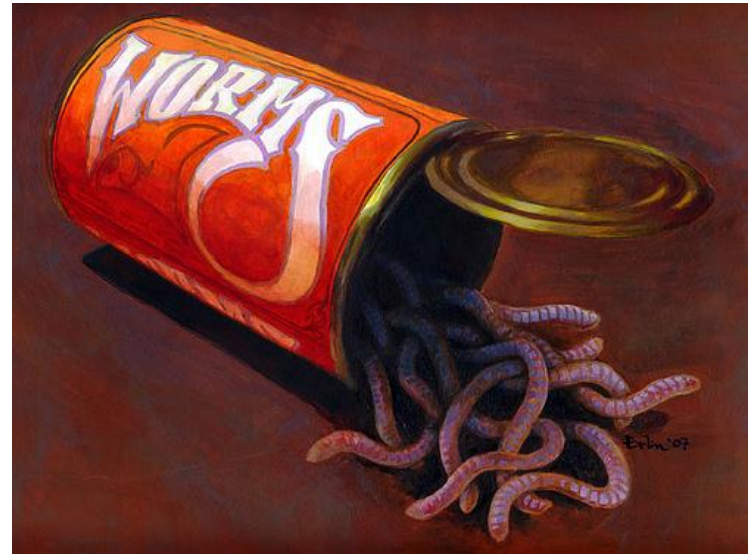
- TLV
- Numbers
- Binary blobs
 - Add/subtract bits
 - Truncation
- Strings
 - Whole new can of worms



Jeffrey Simpson

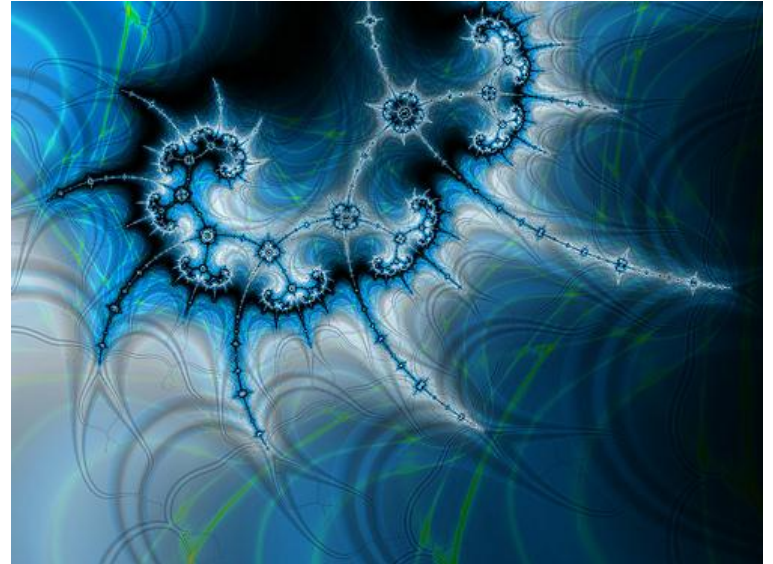
Strings

- **Encoding and character sets**
- **Compression/Expansion**
- **Escape sequences**
- **Character constraints**
- **Delimiters**



State Machine Fun

- Out of order sequences
- Sequence repetition
- Absurdly high recursion
- Selective deletion of sequence parts



Final Fuzzing Takeaway

- Simple in concept
- Relatively simple to perform
- Less bugs == Better software
 - Increase the Quality, Robustness AND Security of your application!
- Make the bad guys work!

FUZZING WITH PEACH

Thank you!

QUESTIONS / COMMENTS