Fuzzing WTF?

what fuzzing was, is, and soon will be

CODENOMICON

Mikko – Co-Founder, Codenomicon

wurldtech

dr.n8 – Co-Founder, Wurldtech

Premiere - BIOs

- Mikko

Mikko Varpiola is the test suite developer n:o 1 at Codenomicon and Director of Special Operations. His area of world class expertise is in anomaly design. Mikko knows what to feed into software to make it fail. Prior to Co-founding Codenomicon, Mikko worked as a researcher at the Oulu University Secure Programming Group (OUSPG). He is the author of the ASN,1 encoding anomalies first deployed in widely recognized PROTOS LDAP and SNMP test suites.

- Nate

Dr. Nate Kube is Co-founder and CTO of Wurldtech Security Technologies, Inc. where he oversees the development of advanced security technologies for the SCADA and process control domains. He is a expert in formal test methods, embedded systems testing, functional and declarative languages, and fault-tolerant computing. Dr. Kube has co-authored a number of best practices for the Industrial Automation Security sector, is a voting member of ISA SP99, and his research efforts have been heavily funded by Canadian, American, and International Government agencies. Dr. Kube holds a BSc in Mathematics and PhD in Computer Science.

Thank god it's Friday again - Outline

- Software Test (and fuzzing's role in it all)
- Short history of fuzzing (I am therefore I fuzz)
- Key challenges
- Typical misconceptions
- The future is non deterministic (it's evolution baby)
- Demonstration
- Wrap-up and Q/A

Software Test

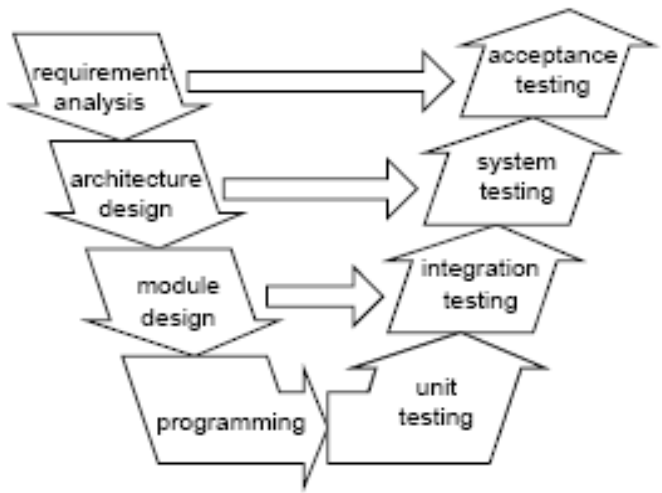
Crackers Don't Matter - Faults, Failures, and Verification

- Verification: task of showing that a software system is free from errors
- Useful to divide bugs into 2 categories: faults and failures
    - failure: incorrect system behaviour (incorrect output)
    - fault: static defect in source code
- Two complementary approaches to verification: source code review uses manual analysis to find faults and testing uses program execution to find failures.
- The dual nature of testing and review can be summarized as:
    - **Testing**
      Pro - Many tasks are automatable, Results are guaranteed
      Con - Results are specific, Must trace from failure to fault (debugging), Applies to executable products only
    - **Review**
      Pro - Automation is difficult, Results are suspect
      Con - Results are general, No fault-to-failure tracing required, Executability not required.

Through the looking glass – Test Automation

- Test automation is heavily influenced by two factors: Controllability and Observability.
- *Controllability* refers to the ease with which inputs may be supplied to the device-under-test (DUT) whereas *Observability* refers to the ease with which outputs from the DUT may be observed.
- Controllability and Observability are useful in two important ways:
  - *Controllability and Observability help testers to predict test automation problems*
  - *Controllability and Observability help testers identify opportunities for cost reduction*
- C&O necessary but not sufficient for successful automation, correlation b/w input and output required (automated test oracles).

La Bomba – Fault What?

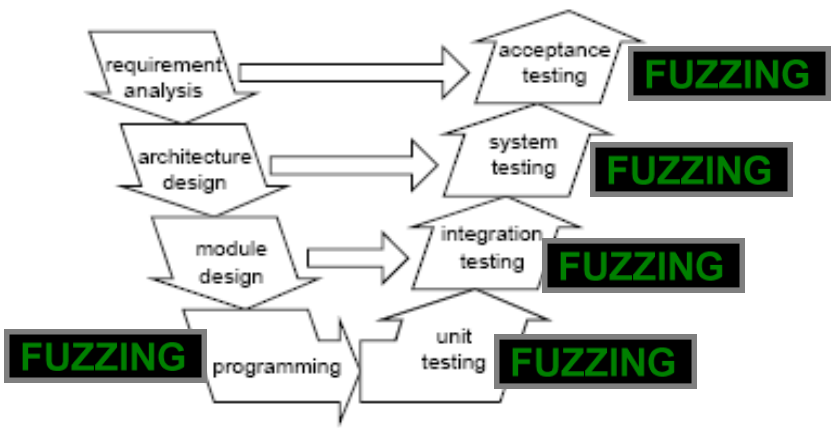


- More important than “when” is “type of fault”
- Software artefact dictates test method which dictates type of faults discovered
  - Infamous Pentium bug (1994) (unit test)
  - Ariane 5 rocket (1996) (system test)
  - Mars lander (1999) (integration test)

They've got a secret - Fuzzing is...

SW QUALITY / QA

When done during SW development



Done By:

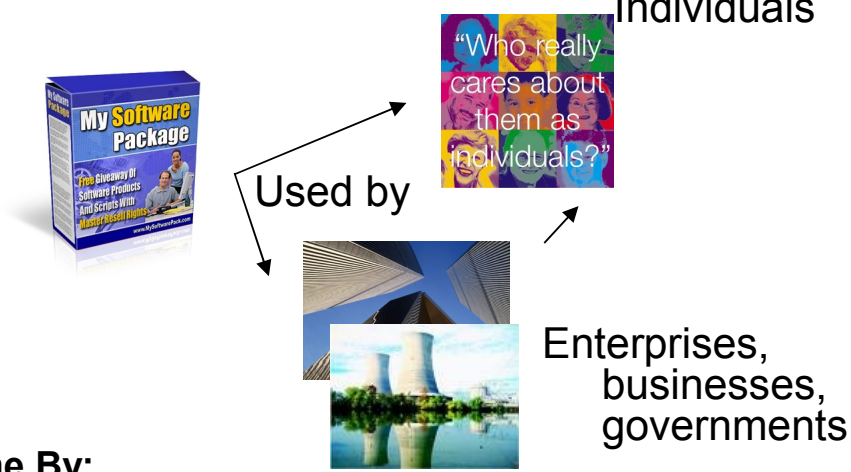
In-house test teams, developers themselves, QA teams and sometimes third party consultants

Results Are:

Usually ***just Bugs***! They can be handled as part of normal SDLC. Almost never published – some challenges when large amounts of SW already out.

SW RELIABILITY / SECURITY

After it is delivered / shipped / used



Done By:

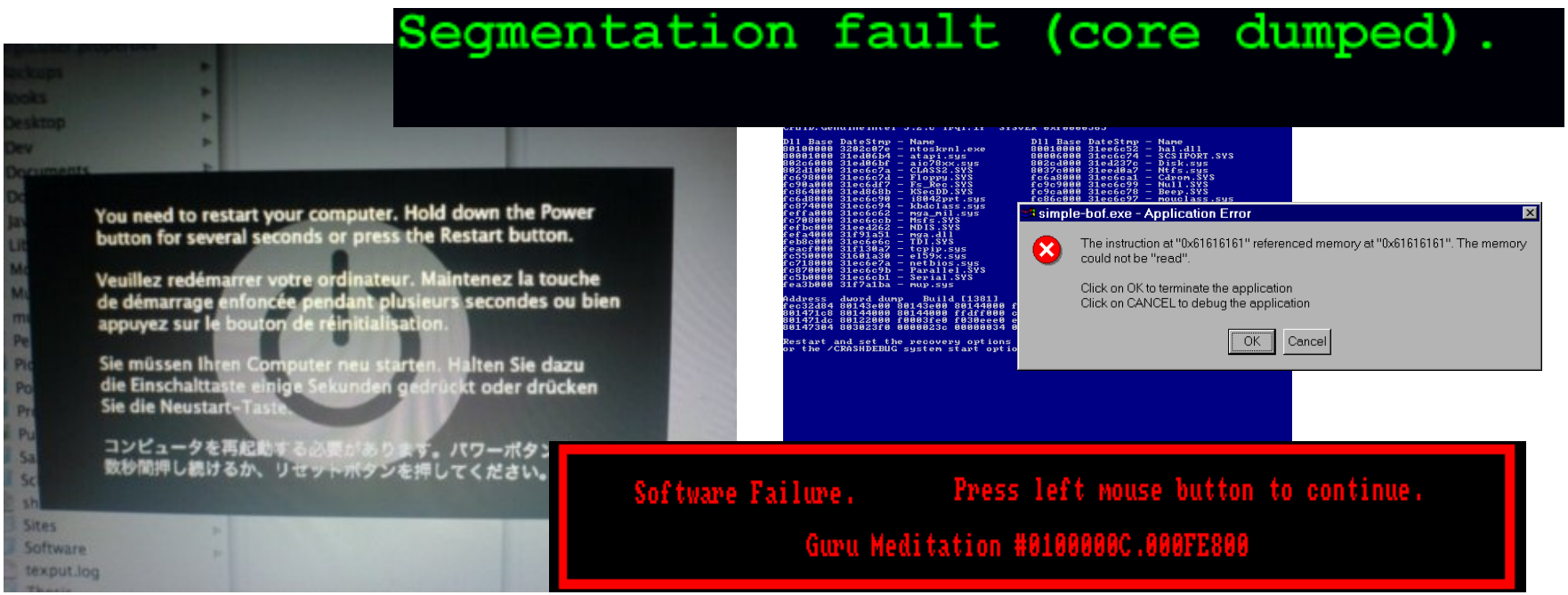
- **Organizations to verify their networks and systems,**
- Individuals, security professionals and non professionals for fun and **profit**
- **Security houses and consultants**

Results Are:

- **Found issues usually cant be fixed by who found them**
- **They have to be reported, publicly or discreetly**
- **Patches has to be issues, tested, distributed,...**

Home on the remains - ...or bluntly put!

Often equivalent of using a napalm strike to kill an insect. Or being in the business of breaking software and systems. Which is very, very FUN by the way!!!



[Images from images.google.com]

The ugly truth - Flaw Densities (known)

- Flaw: deviation from intent
- <1 flaw per kLoC is world class (like our code of course)
- 0.1 per kLoC for Shuttle code
- 30-100 per kLoC for commercial software
- Sometimes easy sometimes hard to correlate to failure rate
- 70-85% of flaws found post unit test are requirements errors

Short History of Fuzzing

The hidden memory – Definition

- Wikipedia: http://en.wikipedia.org/wiki/Fuzz_testing
  - Fuzz testing or fuzzing is a software testing technique that provides random data ("fuzz") to the inputs of a program. If the program fails (for example, by crashing, or by failing built-in code assertions), the defects can be noted.
  - Bit of a shallow...is it?
- Standard Glossary of Software Engineering Terminology, IEEE: “The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.”
- IETF: Security: “The condition of system resources being free from unauthorized access and from unauthorized or accidental change, destruction or loss.” [RFC2828]
- Bottom line:
  - Fuzzing is a software testing method which uses various methods and technologies to perform negative testing on the system (usually not requiring source code). Found issues are implementation level (security or quality) issues, e.g. How someone has implemented the system – not usually how it was designed.

That Old Black Magic – old skool fuzzing

- Barton Miller, University of Wisconsin-Madison (1990)
  - VLSI folks call this fault injection, much older
- What, when, how, whom
  - http://www.owasp.org/index.php/Fuzzing
- Does not concern state, or message structure, or oracles
  - Only repeatable part is random seed used to start random sequence
  - 32bit message == 2^32 test cases
- NOTE: Frameworks vs. Purpose build...

PK Tech Girl – advanced fuzzing

- Understands protocol structure
- Simple rule libraries
- Some support for multi message / stateful protocols
- A lot of effort here – this is currently where scene is mostly
  - It is very effective – oh we know
- Examples
  - Frameworks: Peach 1.0, GPF, Autodafe, most others
  - Application specific: PROTOS, 802.11 fuzzers, ...

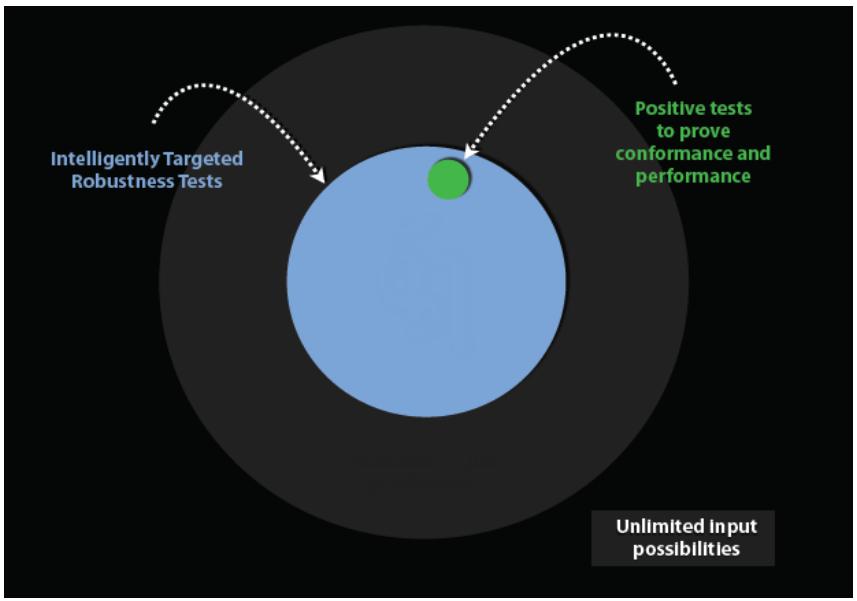
I Do, I Think - What are real fuzzers made of

- High degree of C&O with automated oracle generation
- Model/grammar based
- Understand protocol semantics, extensive rule library
  - Determine and calculate CRCs, lengths, protocol structures...
- Understand protocol states, functionality
  - Test message in right/incorrect state
- Monitoring capabilities to monitor SUT, DUT and IUT
- Feedback mechanisms to optimize test design and test execution (source/binary analysis, protocol behaviour analysis, vulnerability sources, historic data...)

Key Challenges

Infinite possibilities – fuzzing challenge

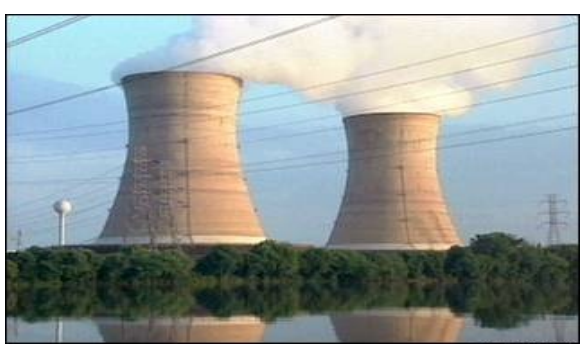
- We are testing how the tested systems are handling unexpected and malformed input
- In most practical applications the input space is roughly Infinity
- It must be somehow limited to finite and reasonable Size
- Difficulties defining coverage, or what constitutes enough?
- Testing Clients (controllability)



Incubator - "Simple" protocol example; combinatorial explosion

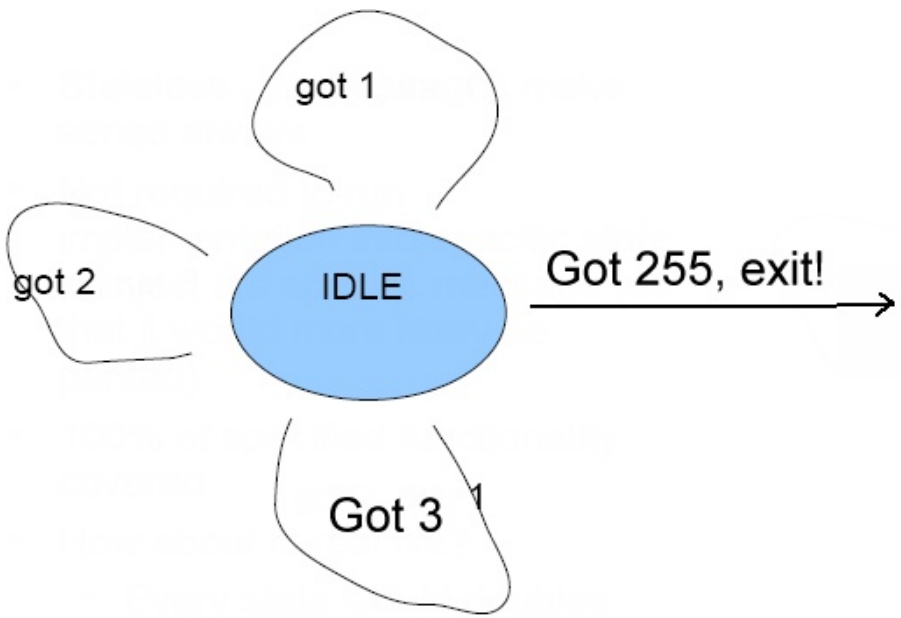
- Imaginary control system, with some constraints
  - Stateless
  - IP/UDP based; uses UDP port 31337
- Protocol is defined as:

```
<protocol> = <message1> | <message2> | <message3> | <messageX>

<message1> = 0x01 # increase power
<message2> = 0x02 # decrease power
<message3> = 0x03 # release gas
<messageX> = 0xff # self destruct
```



Picture if you will - Resulting ”design” could look like...

State machine



Pseudo code:

```
in = integer;
while (in=recv()) {
  switch (in) {
    case 1: increase(); break;
    case 2: decrease(); break;
    case 3: release(); break;
    case 255: selfdestruct(); break;
  }
}
```

So it would seem that there is 5 meaningful and maximum 256 (2^8) test cases? Not quite!!!

A not so simple plan - How Come?

- UDP can carry up to 64k of data (instead of 1 byte defined here)
- What if messages are delivered too quickly?
- Or too slowly?
- There may be no data inside UDP packets...
- State machine does not exist, but if it would...
- Now – if protocol would use 32bit integers instead of 8bit integers – basic value range would already be 2^32 to start with!

We're so screwed – fuzzing two or more fields at a time

- Fuzzing multiple fields/elements explodes input space
  - All possible combinations of two 8bit fields == 2^8 * 2^8 tests == 65535 test cases
  - With 32bit fields, this would be 18446744065119617025 test cases! :D
- However sometimes useful!
  - You have to know what to do!
  - Human is good in telling which fields are interrelated. Machine is usually not.
- Does this really find many more problems?
  - Yes it does
  - Many? Not really. Depends on protocol, implementation, and so on.

Season of death - two fields fuzzed - Code example #1

```
bool buffer_overflow (char &string1 , char &string2) {
    /* Buffer */
    char buffer [128];

    /* Check limits */
    if ( strlen (string1) > 63) return FALSE;
    if ( strlen (string2) > 63) return FALSE;
    .
    .
    sprintf (buffer ," string1: %s string2: %s ",string1 , string2 );
    return TRUE;
}
```

- Where are the errors, anyone?

...Different destinations - Two fields fuzzed - Code example #2

PART OF GIF file format specification

c. Syntax.

	7 6 5 4 3 2 1 0	Field Name	Type
0	| |	Image Separator	Byte
1	| |	Image Left Position	Unsigned
2	| |		
3	| |	Image Top Position	Unsigned
4	| |		
5	| |	Image Width	Unsigned
6	| |		
7	| |	Image Height	Unsigned
8	| |		
9	| | | | | |	<Packed Fields>	See below

<Packed Fields> =	Local Color Table Flag	1 Bit
	Interlace Flag	1 Bit
	Sort Flag	1 Bit
	Reserved	2 Bits
	Size of Local Color Table	3 Bits

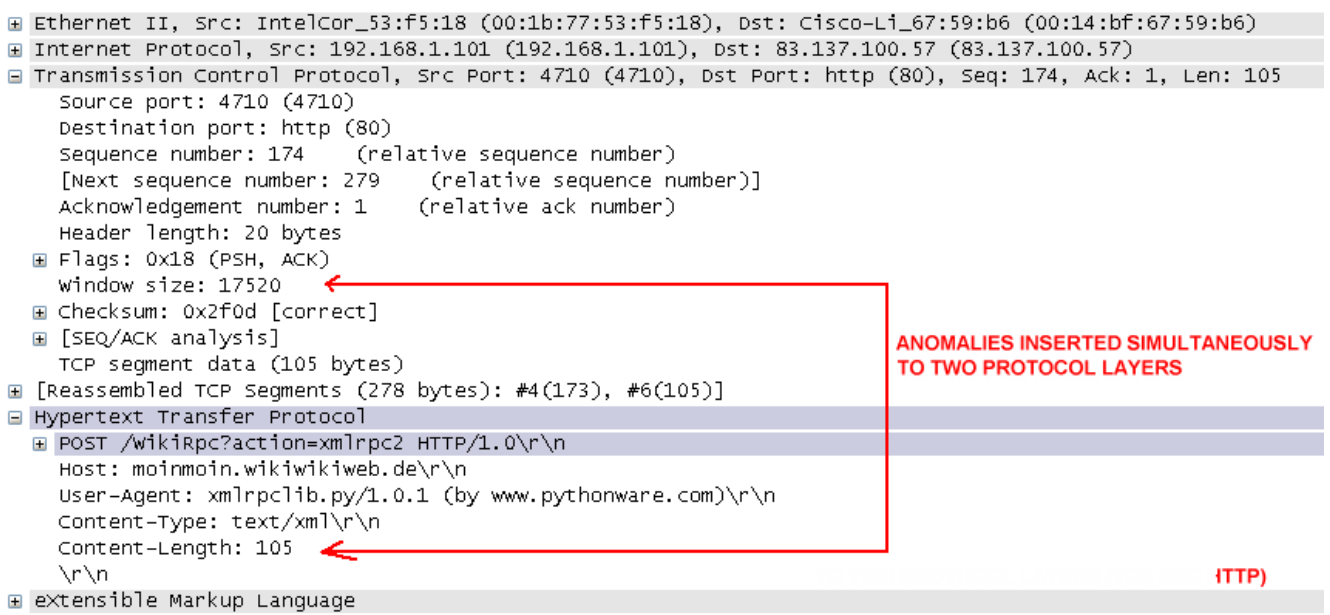
What if we have following pseudo code:

```
function load_GIF (GIFFileHandle bmfh) {
    // .. Parse GIFFileHeader

    // Reserve data for image
    data = alloc (Width * Height * BitDepth);

    // .. read data from file
    // .. decompress (if required) into *data

    if (success) return 0:
}
```

Width and Height are 16bits ==> Max width or height is 64k.Attacking both can increase malloc to 0xffff x 0xffff == 0xFFFE0001 (4294836225) bytes. Attacking bit depth at the same time (in different GIF block) will increase the amount of reserved memory. Now malloc may actually fail!! And if return value of malloc is not checked, then you are in trouble!

Losing time - Multiple levels of freedom

- Another area of debate is whether fuzzing simultaneously on multiple protocol layers makes sense..again..causes input space explosion!



- Will this increase test efficiency? Generally speaking NO (but it may...)
- It may have potential of finding something missed due to bad test case design

Typical misconceptions

Dream a Little Dream – typical misconceptions

- We have CRCs or other message checksums – so we are protected
- We use cryptography so we are completely protected (and out key exchange is ultra secure)
- I have AV, firewalls and IPS/IDS – I am protected
- Our networks are not accessible to outsiders
  - ...but large number of security breaches are due to evil insiders
  - ...or anyone ever had a broken NIC or switch causing network to melt down

Prayer – typical misconceptions

- **We have CRCs or other message checksums – so we are protected**

INCORRECT - The checksums protect against transmission errors.

Attacker can re-calculate the checksums to match the payload.
Or someone may read the spec and implement it differently while checksums still validate!

- ...or anyone ever had a broken NIC or switch causing network to melt down

Relativity – typical misconceptions

- We have CRCs or other message checksums – so we are protected
- **We use cryptography so we are completely protected (and out key exchange is ultra secure)**

INCORRECT - encryption protects against eavesdropping and alteration of message data (while in transit).

Attacker can establish cryptographic tunnel and run the attacks through the tunnel!

PLUS! Cryptographic protocol and key exchange may be attacked by themselves.

The ugly truth – typical misconceptions

INCORRECT - targeted attacks created based on 0day (fuzzing) exploits are practically impossible to be detected by any existing technology. They protect you against the script kiddies and after the fact (ok...to be fair - there are some interesting technologies here too).

Infact AV, Firewalls, IPS/IDS will add an additional layer of components that may become security threats by themselves (as protocol implementations, they need to "peek" inside data that is sent and received, making them possible weakest links...)

- **I have AV, firewalls and IPS/IDS – I am protected**
- Our networks are not accessible to outsiders
  - ...but large number of security breaches are due to evil insiders
  - ...or anyone ever had a broken NIC or switch causing network to melt down

Self inflicted wounds – typical misconceptions

FAIR ENOUGH! Have you though of likelyhood of evil insiders? Passports info for US presidential candidates were not stolen by hackers...

And even if security breach is not of concern (??) - how about system stability? There is no guarantees that some implementation would not send a given packet or data. When functioning properly, but specifically when malfunctioning for what ever reason.

- **Our networks are not accessible to outsiders**
  - ...but large number of security breaches are due to evil insiders
  - ...or anyone ever had a broken NIC or switch causing network to melt down

The Future

Bone to be wild - Put the “ART” in Fuzzing SMART

- Implementations are not created equal so why use the same test cases for each?
- Step away from deterministic methods (and no, inserting a “random” value does not give you the necessary non-determinism)
- Input space HUGE, so intelligent selection of test points required.
- The implementation is speaking, why is no one listening? Here is where we can ++intelligence
- Real-time directed fuzzing based on DUT feedback. Get smart, use much more feed back from previous tests to direct test case GENERATION.
- Hurdles
  - How do you modify the generative model based on DUT feedback?
  - How do you control the data domains based on DUT feedback?
  - How do you collect feedback?
- Data Domain control – control with mixed-strength covering arrays
- Structure control – heavily attributed syntactic generator
- Feedback collection – all based on monitoring baby

The Choice - Domain Control: Covering Arrays

- A covering array algorithm takes a list of domains and generates a subset of the test space containing all parameter combinations of a certain type

- **Consider the following 2-covering e**

- Call ::= CallerOS ServerOS CalleeOS;
- CallerOS ::= 'Macintosh';
- CallerOS ::= 'Windows';
- ServerOS ::= 'Linux';
- ServerOS ::= 'Macintosh';
- ServerOS ::= 'Windows';
- CalleeOS ::= 'Macintosh';
- CalleeOS ::= 'Windows';

	CallerOS	ServerOS	CalleeOS
1	Macintosh	Linux	Macintosh
2	Macintosh	Linux	Windows
3	Macintosh	SunOS	Macintosh
4	Macintosh	SunOS	Windows
5	Macintosh	Windows	Macintosh
6	Macintosh	Windows	Windows
7	Windows	Linux	Macintosh
8	Windows	Linux	Windows
9	Windows	SunOS	Macintosh
10	Windows	SunOS	Windows
11	Windows	Windows	Macintosh
12	Windows	Windows	Windows

Language of Call Grammar

CallerOS	CalleeOS
Macintosh	Macintosh
Macintosh	Windows
Windows	Macintosh
Windows	Windows

CallerOS	ServerOS
Macintosh	Linux
Macintosh	SunOS
Macintosh	Windows
Windows	Windows
Windows	SunOS
Windows	Windows

CalleeOS	ServerOS
Macintosh	Linux
Macintosh	SunOS
Macintosh	Windows
Windows	Windows
Windows	SunOS
Windows	Windows

Fractures - Domain Control: Mixed-Strength Covering Arrays

- Traditionally, covering arrays apply a single strength to all test domains
  - The previous example applied strength two across all three domains
- Mixed-strength covering arrays allow multiple specifications, with each specification having a different strength that is applied to a subset of the domains
- Notation: If we have N domains D0,D1,…,Dn-1:
  - ({j,k,…,l}, m) represents an m-strength cover of the domains j,k,…,l ε {0,1,..n-1}
- In the previous example, the 2-cover of the 3 domains has specification({0,1,2},2)
- Example:

CallerOS	ServerOS	CalleeOS
Macintosh	Linux	Macintosh
Macintosh	SunOS	Windows
Windows	Windows	Macintosh
Windows	Windows	Windows
Satisfies ({0,2},2) and ({1},1)		

CallerOS	CalleeOS
Macintosh	Macintosh
Macintosh	Windows
Windows	Macintosh
Windows	Windows
({0,2},2)	

CallerOS	CalleeOS
Macintosh	Macintosh
Macintosh	Windows
Windows	Macintosh
Windows	Windows
({0,2},2)	

Mental as anything - Structural Control: Tagged Syntactic Generator

- Attribute grammars to describe a generic language

- Zeros Grammar:
  Zeros ::= '0';
  Zeros ::= '0' Zeros;

Sample Derivation: Zeros → '0' Zeros → '00' Zeros → '000'
Language Size: Infinite

- Automated way of modifying the test language is to use "tags"

- Tagged Zeros Grammar:
  {rDepth 3} Zeros ::= '0';
  Zeros ::= '0' Zeros;
  Language: {'0','00', '000', '0000'}

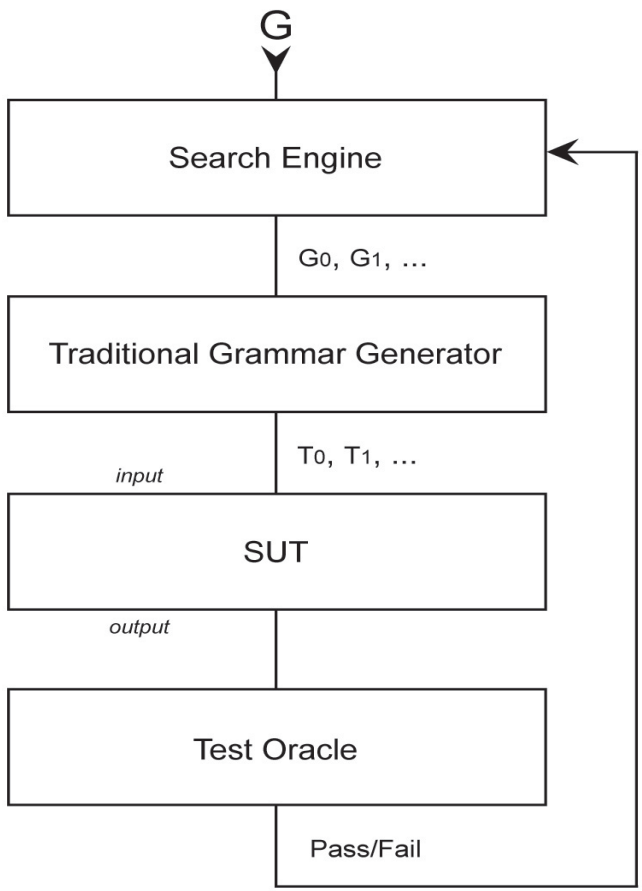
- In practice, the elements generated are protocol test sequence (semantically meaningful sequence of PDU's) TEMPLATES

Nerve - Structural Control: Tagged Syntactic Generator

Common Tags Employed:

- *Depth limit:* limits the depth of the language tree
- *Recursion depth*: limits the number of times a recursive rule can call itself
- *Count limit:* limits the number of terminals strings returned by a rule
- *Balance control*: limits the difference between path lengths in the language tree.
- *Rule weights*: allow statistical control over the selection of rules with the same left hand side.
- *Covering arrays*: combinatorial selection from the combinations generated by the right hand side of a rule.
- *General rule guards*: allow activation/deactivation of a rule based on a boolean condition.
- *Embedded code*: allows insertion of arbitrary software code which is executed during generation.

Till the blood runs clear - Monitoring – Generic Flow Model

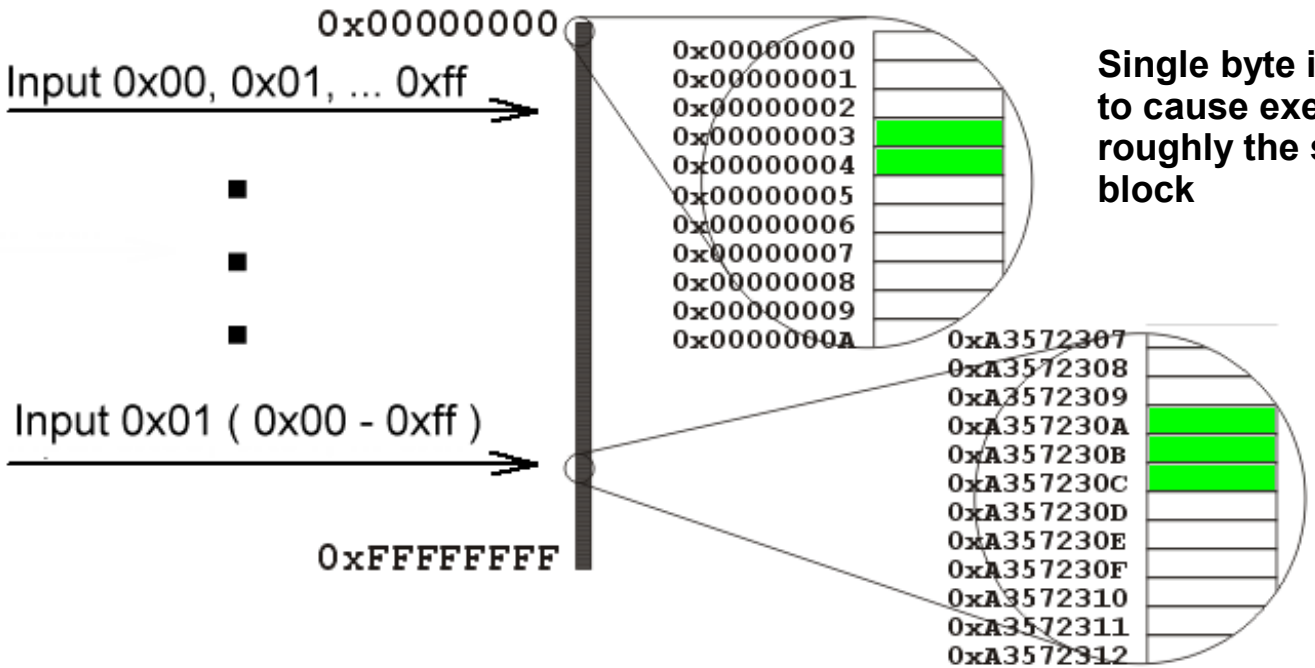


Unrealized reality – behavioural analysis

- Trying to answer some usual fuzzing challenges
  - Optimizing input space to run most optimal tests first
  - Detect and fuzz unknown code blocks (==unspecified (protocol) functionality)
- Monitor binary (or source code)
  - Start from random
  - By monitoring binary, modify inputs to gain different execution paths; become aware of protocol structure
  - NOTE: Binary and source are different

Scratch 'n sniff – behavioural analysis

Monitor process and what parts of its code are executed



Single byte inputs seem to cause execution of roughly the same code block

Here however then at minimum 2 byte input with 0x01 as a leading byte will cause execution to branch to different section of code

Continue to modify inputs based on feedback from processes behaviour to map inputs into those that get us to parts of code we haven't yet been able to get (*to boldly go where no execution has gone ever before* ;)

Promises – behavioural analysis

- “Theoretically” perfect
- For now not a holy grail - works for simple protocols
  - CRCs and cryptography are challenges
  - Complex state machines another...
  - Input space size and injection speed limits usability
- Manually used a very powerful tool – however to reap real benefits – automation is the key

Natural election – into the future with genetic pattern matching

- Alternative to and step away from model based methods
- Learn protocol structure and model to optimize fuzzing from (large) sample of example traffic/data

SEE

the structural similarities - *genes* - shared between currently used protocols and file formats!

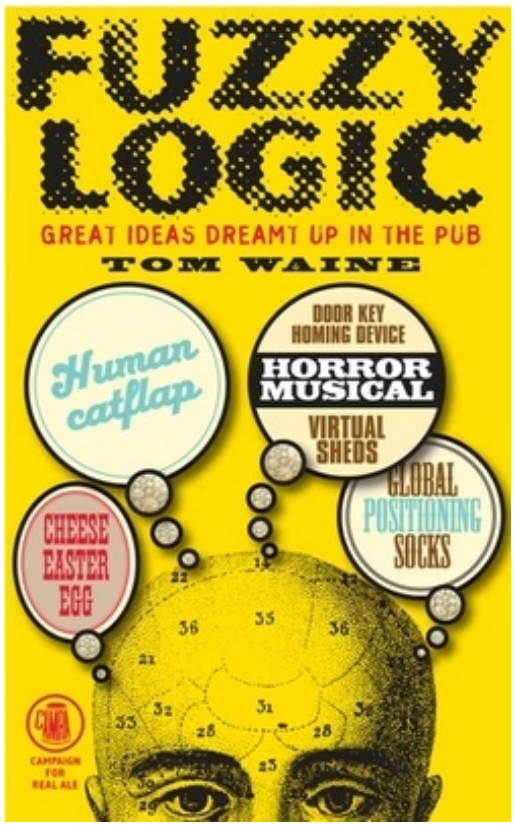
SEARCH

for these similarities to create simplified structural representations - *genomes* - from known or unknown raw data!

EXPLOIT

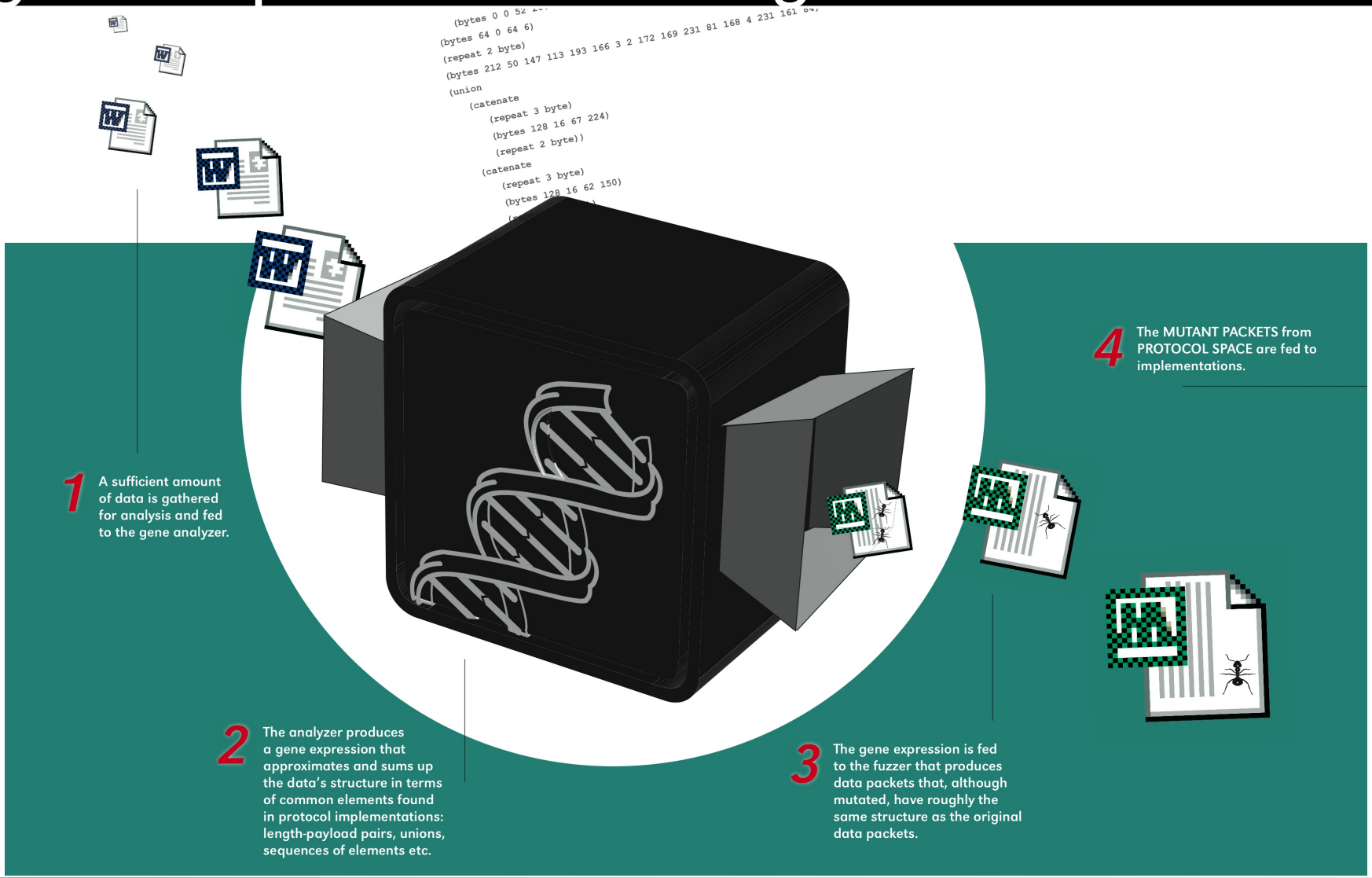
the extracted genomes in several horrifying* ways:

- Get fresh perspective to the protocol pandemonium!
- Automate structural mutation!
- Identify weak genes - constructs prone to implementation flaws!
- Identify data using its genome - a form of structural IDS!
- Create new data packets with the genome as a blueprint!
- <insert your own evil idea here>



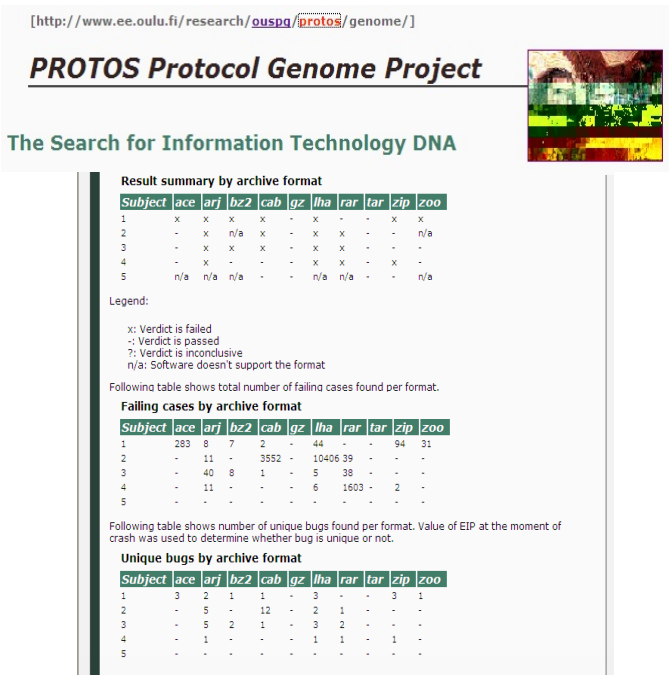
[Image from images.google.com]

DNA mad scientist – into the future with genetic pattern matching



Green eyed monster – into the future with genetic pattern matching

- OUPSG PROTOS Genome has been researching this since 2004- (Codenomicon research partner at University of Oulu)
- Very good results on static content
- Just out: PROTOS Genome c10 - Archive files
- Like 2001 – almost everything breaks – again :)



[http://www.ee.oulu.fi/research/ouspg/protos/genome/]

PROTOS Protocol Genome Project

The Search for Information Technology DNA

Result summary by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	x	x	x	x	-	x	-	-	x	x
2	-	x	n/a	x	-	x	x	-	-	n/a
3	-	x	x	x	-	x	x	-	-	-
4	-	x	-	-	-	x	x	-	x	-
5	n/a	n/a	n/a	-	-	n/a	n/a	-	-	n/a

Legend:

x: Verdict is failed
-: Verdict is passed
?: Verdict is inconclusive
n/a: Software doesn't support the format

Following table shows total number of failing cases found per format.

Failing cases by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	283	8	7	2	-	44	-	-	94	31
2	-	11	-	3552	-	10406	39	-	-	-
3	-	40	8	1	-	5	38	-	-	-
4	-	11	-	-	-	6	1603	-	2	-
5	-	-	-	-	-	-	-	-	-	-

Following table shows number of unique bugs found per format. Value of EIP at the moment of crash was used to determine whether bug is unique or not.

Unique bugs by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	3	2	1	1	-	3	-	-	3	1
2	-	5	-	12	-	2	1	-	-	-
3	-	5	2	1	-	3	2	-	-	-
4	-	1	-	-	-	1	1	-	1	-
5	-	-	-	-	-	-	-	-	-	-

Meltdown – into the future with genetic pattern matching

Result summary by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	x	x	x	x	-	x	-	-	x	x
2	-	x	n/a	x	-	x	x	-	-	n/a
3	-	x	x	x	-	x	x	-	-	-
4	-	x	-	-	-	x	x	-	x	-
5	n/a	n/a	n/a	-	-	n/a	n/a	-	-	n/a

Legend:

x: Verdict is failed
-: Verdict is passed
?: Verdict is inconclusive
n/a: Software doesn't support the format

Following table shows total number of failing cases found per format.

Failing cases by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	283	8	7	2	-	44	-	-	94	31
2	-	11	-	3552	-	10406	39	-	-	-
3	-	40	8	1	-	5	38	-	-	-
4	-	11	-	-	-	6	1603	-	2	-
5	-	-	-	-	-	-	-	-	-	-

Following table shows number of unique bugs found per format. Value of EIP at the moment of crash was used to determine whether bug is unique or not.

Unique bugs by archive format

Subject	ace	arj	bz2	cab	gz	lha	rar	tar	zip	zoo
1	3	2	1	1	-	3	-	-	3	1
2	-	5	-	12	-	2	1	-	-	-
3	-	5	2	1	-	3	2	-	-	-
4	-	1	-	-	-	1	1	-	1	-
5	-	-	-	-	-	-	-	-	-	-

They've got a secret - Oh by the way...

- Its kind of obvious, but...
- All that was presented above is good alone...
- BUT what if you would combine them?
  - Granular feedback directs model and partitions input domain
  - Genetic pattern matching to learn and direct fuzzing from existing traffic/data
  - Models and rule library to aid in/augment both
- Start and stand back, spawns like an infection, learning, adapting, crippling – the “perfect” fuzz
- Not today, but stay tuned...

Demonstration

Constellation of doubt - demo

- Oulu University Secure Programming Group (OUSPG) extension of Codenomicon research labs. OUPSG lead by co-founder of Codenomicon
- Demonstrate the effects of OUSPG PROTOS Genome c10 archives test suite
- Test material release coordinated through CERT/FI (Ficora) and CPNI (public release 2008-03-17)
- Vendor statements available at
  URL: https://www.cert.fi/haavoittuvuudet/joint-advisory-archive-formats.html
- Test suite available at
  URL: http://www.ee.oulu.fi/research/ouspg/protos/testing/c10/archive/

Thanks for sharing - Questions?

Thank you!

http://www.codenomicon.com/

http://www.wurldtech.com/

Fuzzing WTF?

what fuzzing was, is, and soon will be

CODENOMICON

Mikko – Co-Founder, Codenomicon

wurldtech

dr.n8 – Co-Founder, Wurldtech