



Realizing the Fuzzing Potential: Precision and Accuracy vs. Coverage

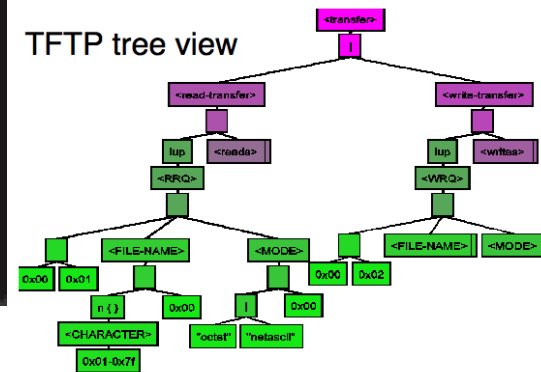
<http://www.codenomicon.com/products/coverage.shtml>

Ari Takanen
CTO
Codonomicon

Mikko Varpiola
Fuzzing Specialist
Codonomicon



- **Founded in 2001, after five years of research in product security at University of Oulu (1996-2001)**
- **Customers include:**
 - Manufacturers
 - Telco Service Providers
 - Defense
 - Finance and Leading Enterprises
- **CROSS (IPv6, xml,..., FTP)**



Feds, Industry, Battle the Biggest Bug

Kevin Poulsen, SecurityFocus 2002-05-12

A security hole in implementations of Abstract Syntax Notation One may threaten some of America's most crucial networks. Relax, the President's been briefed.

So severe are the potential ramifications of widespread ASN.1 security holes, that President Bush was personally briefed on the matter, according to cyber security czar Richard Clarke, speaking at a meeting of the National Security Telecommunications Advisory Committee (NSTAC) last March. "When Howard [Schmidt] and I briefed the President on the ASN.1 vulnerability, he said to us, 'Don't wait for somebody to tell you that there's intelligence, or that there's a hacker group out there about to exploit the vulnerability because it will be too late then to fix it,'" said Clarke, according to a transcript of the meeting.



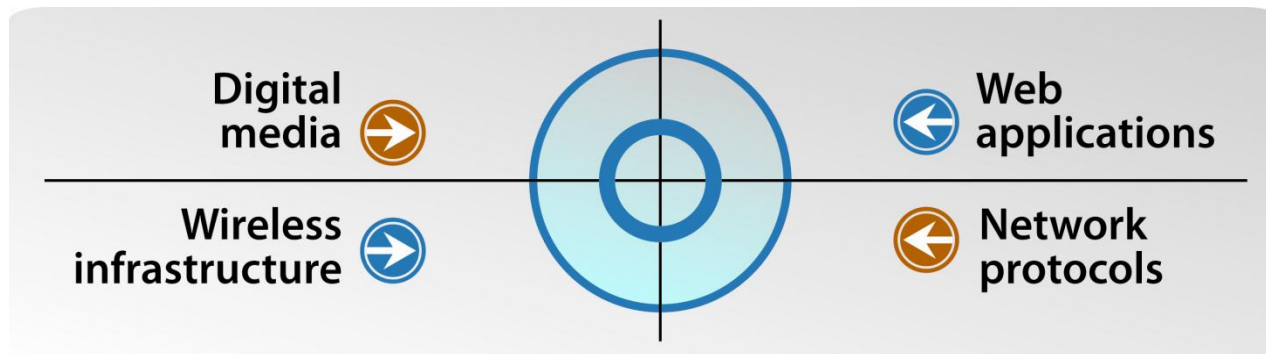
About Mikko Varpiola

- **One of the original PROTOS team members at University of Oulu**
 - WAP, LDAP, SNMP, ...
 - Fuzzing (and otherwise breaking SW) since 1996 or so
- **Co-founder of Codenomicon, key customer services specialist in USA market**
 - Need any weird protocols extensively fuzzed: contact Mikko ;-)



What is Codenomicon DEFENSICS?

- **Product line of model-based fuzzers for over 200 protocols and interfaces**
- **Both Client/Server**
- **20-30% of tools are customer proprietary**
- **Only solution covering whole application stack**
 - Wireless / Layer 2
 - Network protocols
 - File formats
 - Application / XML
 - API





Some Helpful Definitions

- **Vulnerability** – a weakness in software, a bug
- **Threat/Attack** – exploit/worm/virus against a specific vulnerability
- **Protocol Modeling** – Technique for explaining interface message sequences and message structures
- **Fuzzing** – process and technique for security testing
- **Anomaly** – abnormal or unexpected input
- **Failure** – crash, busy-loop, memory corruption, or other indication of a bug in software

Fuzzing is a NEW way of doing QA. Its robustness, its security! Its here, today. And it works! Lets make sure it is used!



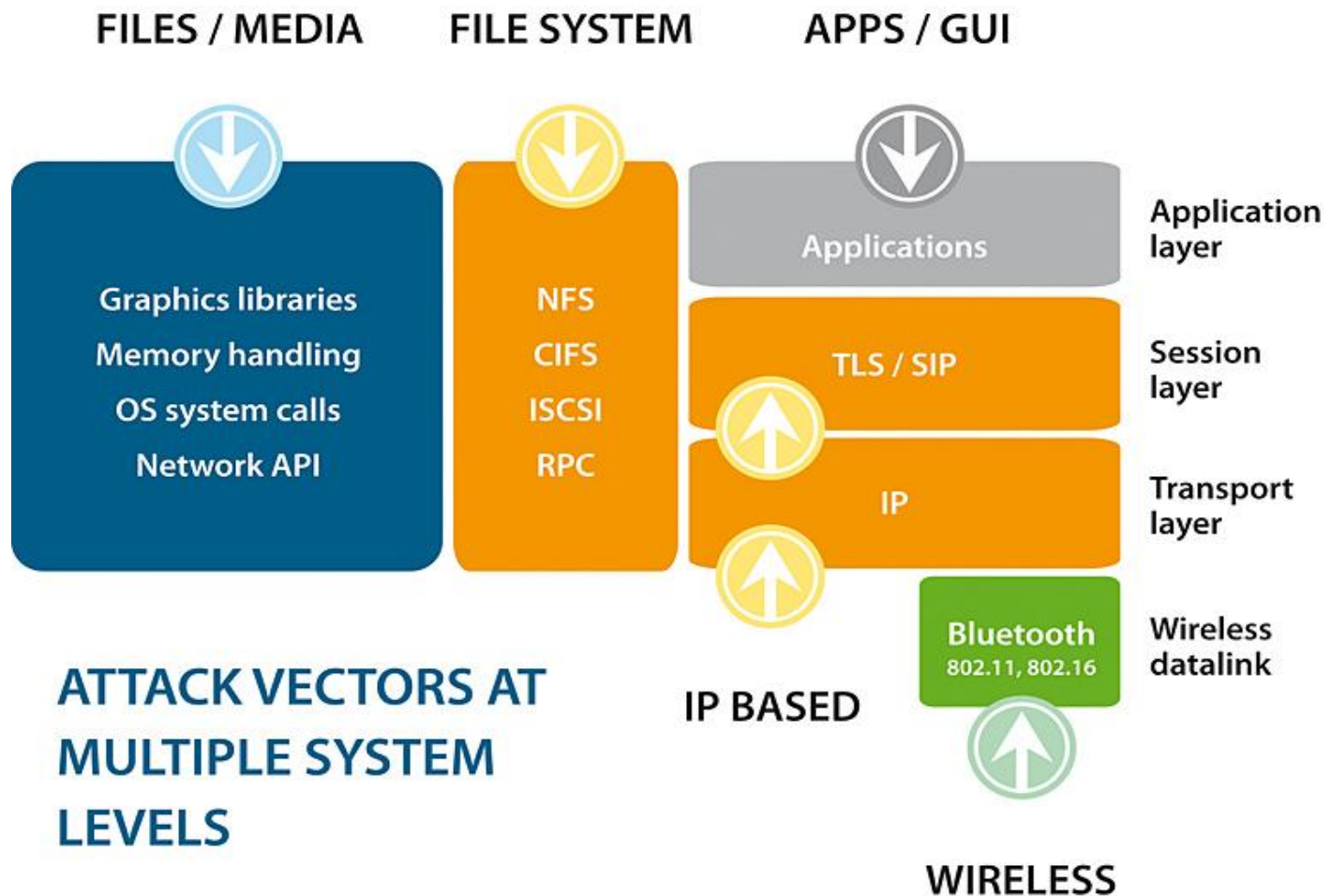
From: <http://www.google.com/googlebooks/chrome/>

- **All fuzzing approaches find vulnerabilities and failures. Some more than others.**
- **More fuzzing is usually the better**
- **There is no shortage of tools today – both FOSS and commercial**
- **Fuzzing is right thing to do – and its cheap and cost effective!**
- **[almost] Everyone SHOULD fuzz!**



A bug trying to hide from us

Example: Multitude of Attack Vectors / Attack Surface

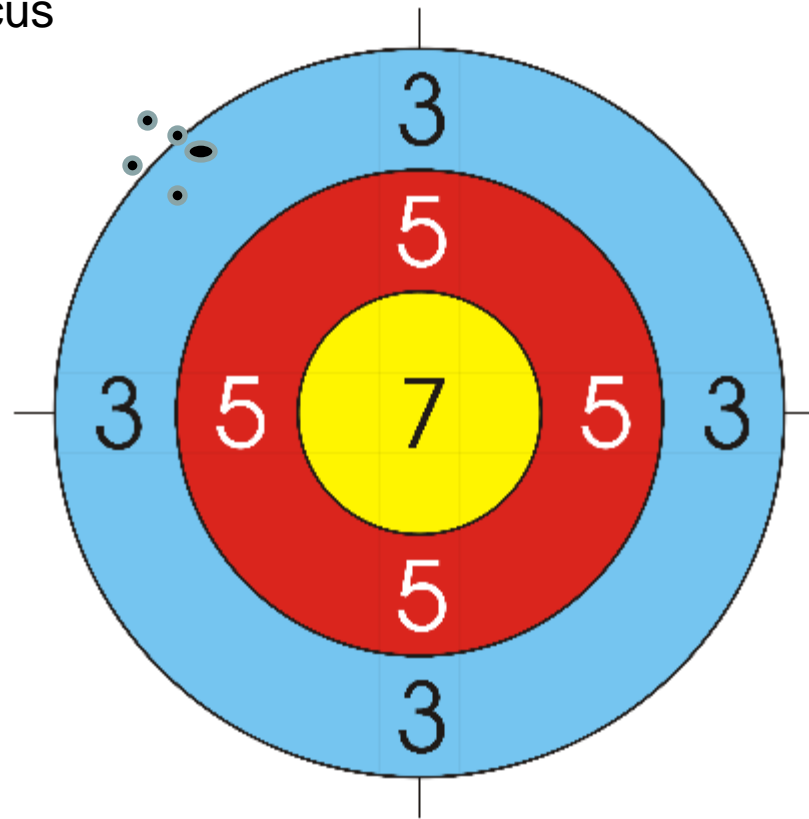




**The Greatest Challenge in Fuzzing?
Test Coverage, or how:
Just Getting IT done – affects it?**

PRECISION

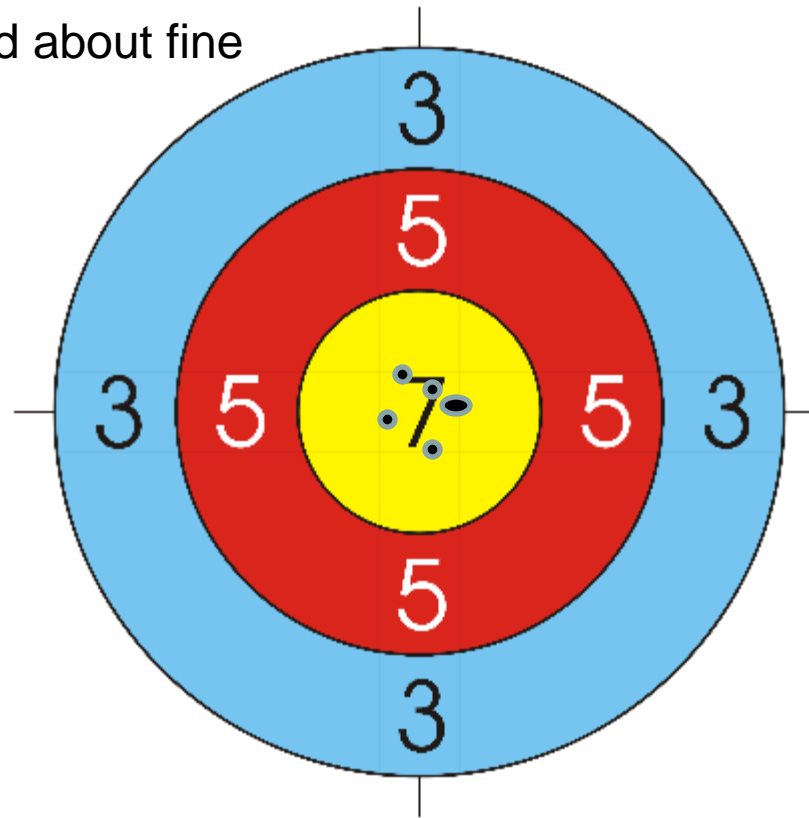
Precision is about focus



Attack surface
Protocol layers
Protocol use cases
...

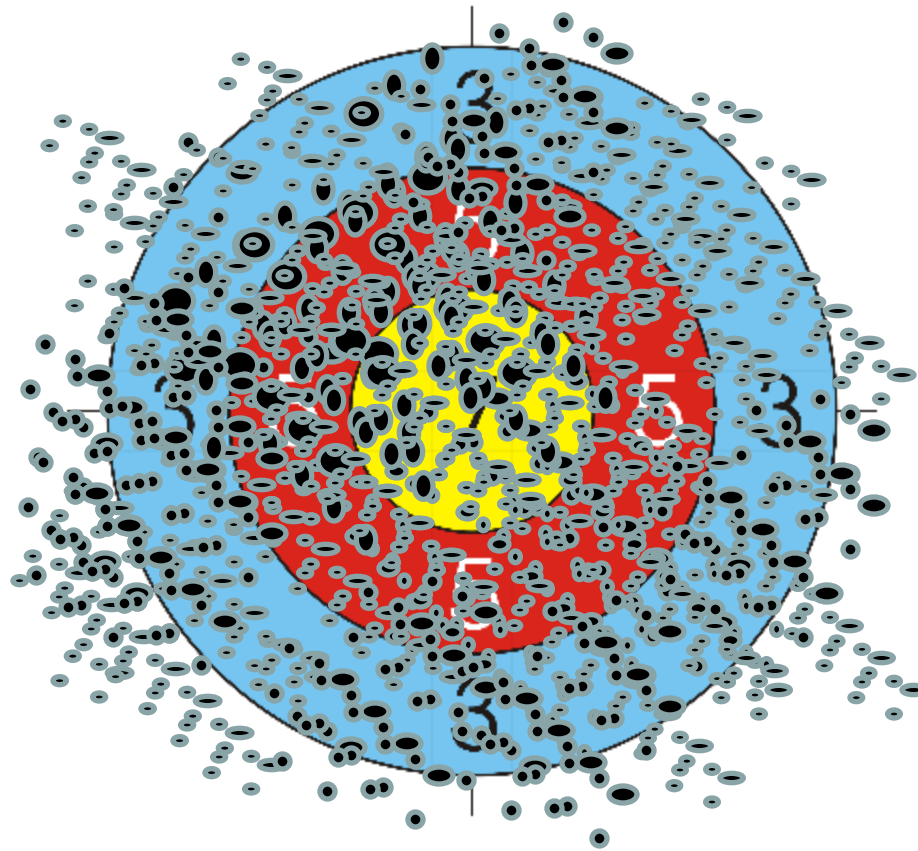
ACCURACY

Accuracy is skills and about fine tuning your “tools”



SQL anomalies
ASN.1 anomalies
XML anomalies
Integer anomalies
Structural anomalies
...
Monitoring
...

ROBUSTNESS TESTING - FUZZING



So whats the problem?

- **What we a trying to find is usually microscopically "small" – so {shot/mini}gunning is usually not the optimum solution**
- **Various methods are needed to optimize the fuzz... This is what we and others have been doing past 10 years++**
- **This translates to coverage. Coverage seems to be function of accuracy and precision!
(in part anyway)**





What is Fuzzing Today?

- **Fuzzing tests SW/HW by providing anomalous input to communication protocols/interfaces**
- **Attempts to cause the software to fail or behave in unexpected ways**
- **Models communication protocols, APIs, or file formats**
 - Pretty much all fuzzing today is model based (only variable is what the model is based on and how complex it is)
- **Creates large amounts of anomalous input automatically (which leads to some problems debated later on)**
- **Interacts with the software under test and observes the behavior (!!!!)**



Some technical (traditional?) methods to assess the coverage

- **By observing the test suite**
 - **Specification coverage of the test suite**
 - **Functional coverage of the test suite**
 - **Engine capabilities** (How the model is build, Evolving, Traffic capture, Specification, How test cases are generated, Systematic, Mutation, Anomaly libraries, Bit-flips,...)
- **By observing the test target**
 - **Code coverage (depth, breadth?) (!!!)**
 - **Binary coverage (e.g. PaiMei, IDAPRo...)**
 - **Number actual unique bugs observed**
 - **(Resource utilization)**
 - **CPU, memory leaks, file handles, hidden exceptions,...**

Couple of observations on coverage (accuracy, precision)

- **Affected by several factors outside (often) direct control of the [fuzzing] test suite**
 - **Configuration of DUT**
 - **Execution environment (ripple effect)**
 - **Configuration and capabilities of the test suite** (people just run the default settings)
- **[better, automatic, easier] observation facilities increase the accuracy of the tests**
 - **E.g. use Valgrind to catch memory leaks**
 - **E.g. use OllyDbg, windbg to catch bad use of try {} catch {} to hide exceptions...**
 - **E.g. monitoring /var/log/messages :-S**



Reality check on technical coverage

- **depth-first might be sexier but breadth-first seems to help in getting better overall results**
- **Fuzzing multiple layers or fields at the same time may sound fancy, but usually results in significantly poorer results!**



Accuracy, precision, coverage – beyond technical means – socio-economical coverage

- **It's not only technical challenges that we are facing when trying to get best out of fuzzing**
- **I assert that organizational, educational, processual, usability, etc. aspects of fuzzing ecosystem are bigger challenge than technical issues**
 - These challenges in part also prevent fuzzing being able to realize its potential



Accuracy, precision == Coverage == function of usage

- **All fuzzing finds flaws**
 - IF users know HOW to use the tools
 - AND how to read and use the results (big issue with commercial fuzzers)
- **Some reasons for IF/HOW/AND(s) above**
 - As fuzzing goes more main stream, people don't do fuzzing because of its inherent benefits, BUT because they are told to. Which means they don't get it the way we do.
 - Fuzzing is not exact "science" – e.g. results need interpretation, its not always clear what to look for or what happened?
 - Something needs to be done with results – which is not always obvious!

Commercial deployments of fuzzing suffer from non technical challenges...

- **For commercial deployments there are few key challenges**
 - Fuzzing is NOT recognized as integral part of modern SDLC
 - Even when it is, there is big confusion on how/where to deploy it (security vs. QA vs. choose)
 - There are NO fuzzing career paths or certifications
 - There are NO compliance requirements that fuzzing would fulfill (CC EAL4?)
 - Fuzzing is NOT taught in colleges, universities,... (at least not in India, China, Indonesia,... where the testing is done)
- **As a result fuzzing is not realizing its full potential!**
(and that has almost nothing to do with technical capabilities of fuzzers)

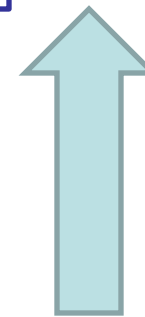
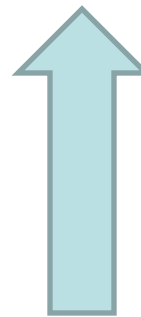
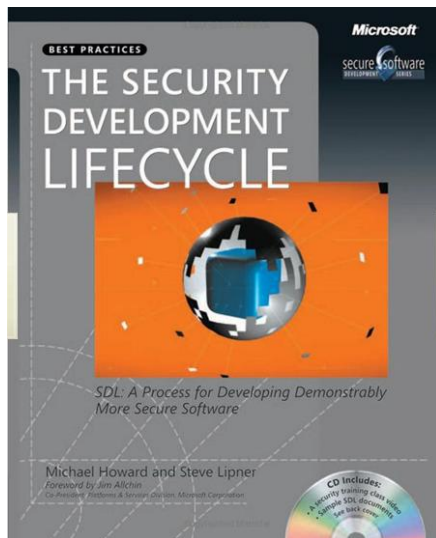
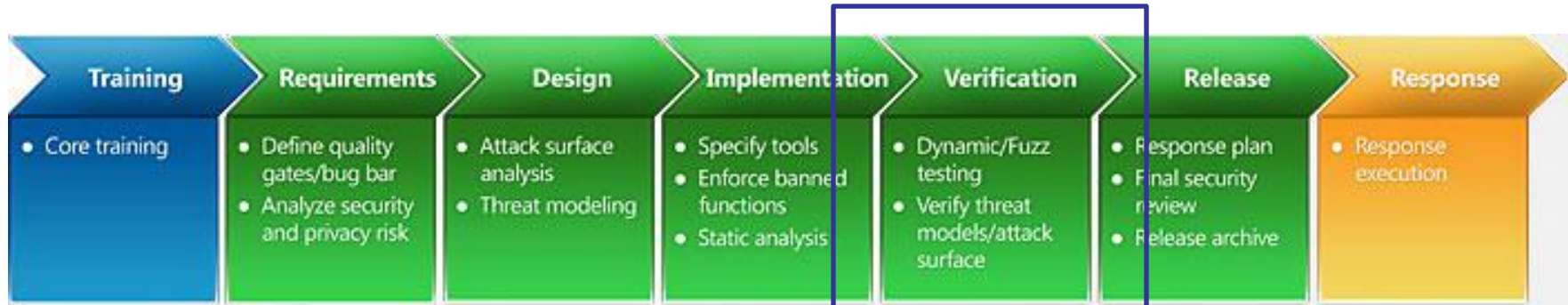


Increasing the coverage of commercial fuzzers (in non technical way)

- **Fuzzing added as a compliancy requirement of its own**
 - Likely decreases the actual "quality", BUT overall outcome is likely light years ahead of where we are today!
 - Solves the problem that fuzzing is skipped because people are worried about their job security
 - Career paths, certifications (!!!!)
(you need a permit to hold a gun in most countries, should same be applied to fuzzers? ☺)
- **Education, training, support!!!**
 - What to do with the vulnerability after its found? Fix it, report it (whom, where?), mitigate it, forget about it?
 - To make people aware of fuzzing...

Microsoft SDL Example: Fuzz Here?

<http://msdn.microsoft.com/en-us/security/dd219581.aspx>



Many organizations choose to deploy fuzzing in other parts of the SDL as well.



Microsoft SDL helps in so many ways

- **Well defined SDL with fuzzing presented as an integral part of it**
- **SDL how ever does not define HOW, hence we need:**
 - Provide tools / test suites / instructions / guides / encouragement
 - Provide the baseline test plans, information,..
 - Support with monitoring facilities (Core dumps, Memory leaks, CPU, Other (...))
- **The key question is how fuzzing is done (and what/when is enough)?**
 - **This is both technical and non-technical question!**

- **Fuzzing may be ready for the prime time, but in order to get there we need *support***
- **We need vendor independent**
 - Best practises and recommendations
 - Fuzzing added as a compliancy requirement
 - Education on fuzzing as part of QA / testing curriculums
- **Fuzzing to be made even easier and more accessible to the end users**



**Case 1: XML Fuzzing – more coverage by
fuzzing more layers**

XML Introduction, XML Vulnerabilities, XML Fuzzers

<http://www.codenomicon.com/defensics/xml>



Case: XML

- **Early 2009, Codenomicon developed fuzzers for XML-based telecommunication protocols**
- **All open source XML parser libraries failed**
 - All applications using these libraries are vulnerable for the very same issue
- ***Note that this is very similar to the PROTOS/OUSPG ASN.1/SNMP discovery... Covering potentially thousands of bugs... ASN.1/SNMP bugs were never really fixed either... Codenomicon SNMP suite still today crashes all commercial SNMP implementations***



Wait a Sec! What is XML?

- XML is not a protocol
- XML is a syntax for messages, similar to ASN.1 in binary messages

```
<?xml version='1.0'?>
<methodCall>
  <methodName>AttachFile</methodName>
  <params>
    <param>
      <value><string>FrontPage</string></value>
    </param>
    <param>
      <value><string></string></value>
    </param>
    <param>
      <value><string>list</string></value>
    </param>
    <param>
      <value><string></string></value>
    </param>
    <param>
      <value><boolean>0</boolean></value>
    </param>
  </params>
</methodCall>
```



XML Is Used In

- **XML used by IETF, 3GPP, W3C and commercial vendors**
- **XML-based standard protocols**
 - SOAP, XMPP, CWMP (TR-069), UPnP, NETCONF, SIP/IMS, SyncML
- **XML file formats**
 - HTML/XHTML, XSLT, RSS/Atom, WAP/WML, SAML, SMIL, Office applications
- **Proprietary protocols and file formats**
 - IBM, Microsoft, SAP, Oracle, BEA etc.
- **Custom application logic built on XML messaging using Web Services platforms (Various vendors)**



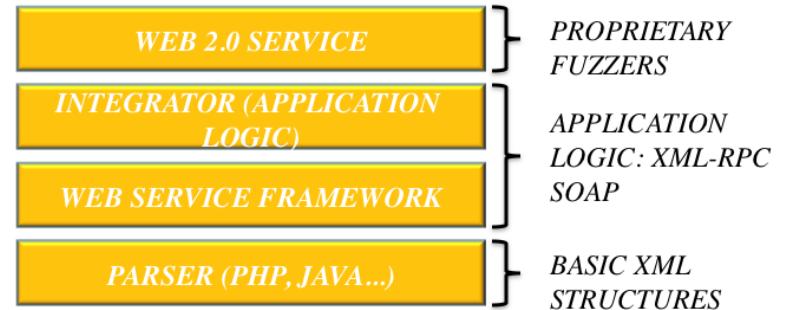
XML: The Discovery

- **A single exploit can be used to attack:**
 - Security products such as Firewalls
 - Application platforms
 - Programming languages: PHP, Ruby, Python, C, Java
- **Repair process:**
 - Fix the parser libraries
 - Deploy updates to operating systems and platforms
 - Urge application developers to re-build if needed

XML: A Look At The Test Coverage

- **XML-based systems**
 - Complex and highly critical applications
 - Built in “layers”
 - Parsers, SOAP and XML-RPC applications
- **Fuzzers for XML**
 - Advanced model-based fuzzing capability required for testing anything above parser-level
 - Customized fuzzers needed for application layer

XML DOMAIN REFERENCE MODEL



```

Test case
#8641
Group    xml.xml_file.xml_document.xml_prolog.xml_doctypedecl.xml_intSubset.xml_markupdecl
Anomaly  UTF-8: PSP
Category CWE-ID-1 CWE-ID-137 CWE-ID-138 CWE-ID-156 CWE-ID-17 CWE-ID-18 CWE-ID-19
Case hash 0x57F8B733F0CDC8F8
<?xml version='1.0' ?>\r\n
<!DOCTYPE a[<!ENTITY a PUBLIC \x226\x128\x169>]><!--><methodCall>\r\n
  <<methodName>AttachFile</methodName>\r\n
  <<params>\r\n
    <<param>\r\n
      <<value><string>FrontPage</string></value>\r\n
    <</param>\r\n
  <</param>\r\n
  <<param>\r\n

```



Coverage

- **Precision?**
- **All messages tested?**
- **All message structures tested?**
- **All data definitions tested?**
- **All “tags” tested?**
- **Precision seems like it is about protocol coverage**
- **Accuracy?**
- **Anomaly categories? SQL? Buffer overflow?**
- **All values: 0..65k, a..z, 0x00..0x255 ?**
- **Combinations of anomalies?**
- **Accuracy seems like it is about anomaly coverage**



Templates-Based General Purpose Fuzzing

-- more coverage via less precision and accuracy

Traffic Capture Fuzzing (==fuzzing for the masses?)

<http://www.codenomicon.com/defensics/traffic-capture-fuzzer>



Traffic Capture Fuzzing

- **All fuzzing needs a model of correct behavior**
- **The easiest method for acquiring default functionality is from templates**
 - Files
 - PCAP traffic flows
- **The model is easily built by e.g. Wireshark protocol dissectors**
 - Open source has had this a while (autodafe, peach,...)
 - Commercial fuzzers following, with a twist...



Acquire traffic capture from analyzers, vulnerability feeds or bug reports

A screenshot of the Defensics 3 software interface. The main window is titled 'Defensics 3 Early Access - sip1'. On the left, there is a sidebar with a tree view containing 'Basic configuration', 'Test cases', 'Scoring', 'Capture', 'Instrumentation', and 'Suite configuration'. The 'Basic configuration' section is active, showing fields for 'Sequence' (print2), 'Target address' (192.168.255.255), and 'Timeout for received messages' (1000). A dialog box titled 'Packet capture import and suite launch - Step 2 / 2' is open in the foreground. It has a 'Test sequence name' field with 'Printer Test' entered. Below that is a 'Test suite' dropdown menu set to 'Generic Test Suite 0.2'. There are two checkboxes: one checked, 'Load the selected suite with the created sequence', and one unchecked, 'Export into a file with custom location and/or custom name'. The 'Export to' field contains the path 'Z:\Programs\Codonomicon\suites-code\d3-generic-0.2\testtooluser\Printer Test.seq'. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The background interface shows a 'Test group' section with a 'Generic' test case and a summary of '3296 cases' and '6 cases'.



Protocol (and protocol layer) selection from a set of protocol captures

The screenshot shows the Defensics 3 Early Access - sip1 interface. The main window displays configuration for a test group named "Generic". The configuration includes:

- Sequence: print2
- Target address: 192.168.255.255
- Timeout for received messages: 1000
- TCP and UDP transport: Server port: 631, Local port: 631
- Ethernet transport: Virtual Ethernet MAC address: 0a:c0:d0:e4:cc:c0, Name of network device: eth0 : VMware Virtual Ethernet
- Loop test cases: Valid cases and failed cases
- Logging level: Valid cases and failed cases
- Instrumentation valid-case: Generic

A dialog box titled "Packet capture import and suite launch - Step 1 / 2" is open, showing the process of selecting a protocol to test from a pcap file. The dialog includes the following information:

- Import file: Z:\Temp\print2.pcap
- Protocol to test: Pick the protocol exchange to test from the exchanges identified from the pcap file.
- Message exchanges list:
 - SSL over tcp 188.2.235.221:443 >> 192.168.2.96:4773 (4 messages sent, 2 messages received)
 - NBDGM-SMB-MAILSLOT-SMB_NETLOGON over udp 192.168.2.76:138 >> 192.168.255.255:138 (1 messages sent, 0 messages received)
 - CUPS over udp 192.168.1.11:631 >> 192.168.255.255:631 (1 messages sent, 0 messages received)
 - DNS over udp 192.168.3.98:5353 >> 224.0.0.251:5353 (1 messages sent, 0 messages received)
 - DNS over udp 192.168.3.10:5353 >> 224.0.0.251:5353 (1 messages sent, 0 messages received)
 - UNKNOWN over udp 192.168.2.66:3830 >> 192.168.255.255:1947 (1 messages sent, 0 messages received)
 - LLC-STP over eth 00:11:43:f3:e9:78 >> 01:80:c2:00:00:00 (6 messages sent, 0 messages received)
 - SSL over eth 00:50:56:3f:00:0c >> 00:24:e8:ab:48:2f (6 messages sent, 5 messages received)
 - NBDGM-SMB-MAILSLOT-SMB_NETLOGON over eth 00:1c:23:11:bc:35 >> ff:ff:ff:ff:ff:ff (1 messages sent, 0 messages received)
 - CUPS over eth 00:1c:23:d7:11:2c >> ff:ff:ff:ff:ff:ff (1 messages sent, 0 messages received)
 - DNS over eth 00:0c:29:94:65:45 >> 01:00:5e:00:00:fb (1 messages sent, 0 messages received)
 - DNS over eth 00:0c:29:88:4f:e1 >> 01:00:5e:00:00:fb (1 messages sent, 0 messages received)
 - UDP over eth 00:18:8b:d9:bf:80 >> ff:ff:ff:ff:ff:ff (1 messages sent, 0 messages received)
- Selection options:
 - How to pick the right message exchange? (radio button)
 - Select if should test the server side entity or the client side entity: >> Server (right) (radio button selected)
 - Select if messages are binary data or if messages are textual: Textual (radio button selected)

The dialog also shows a summary of cases: 3296 cases and 6 cases.

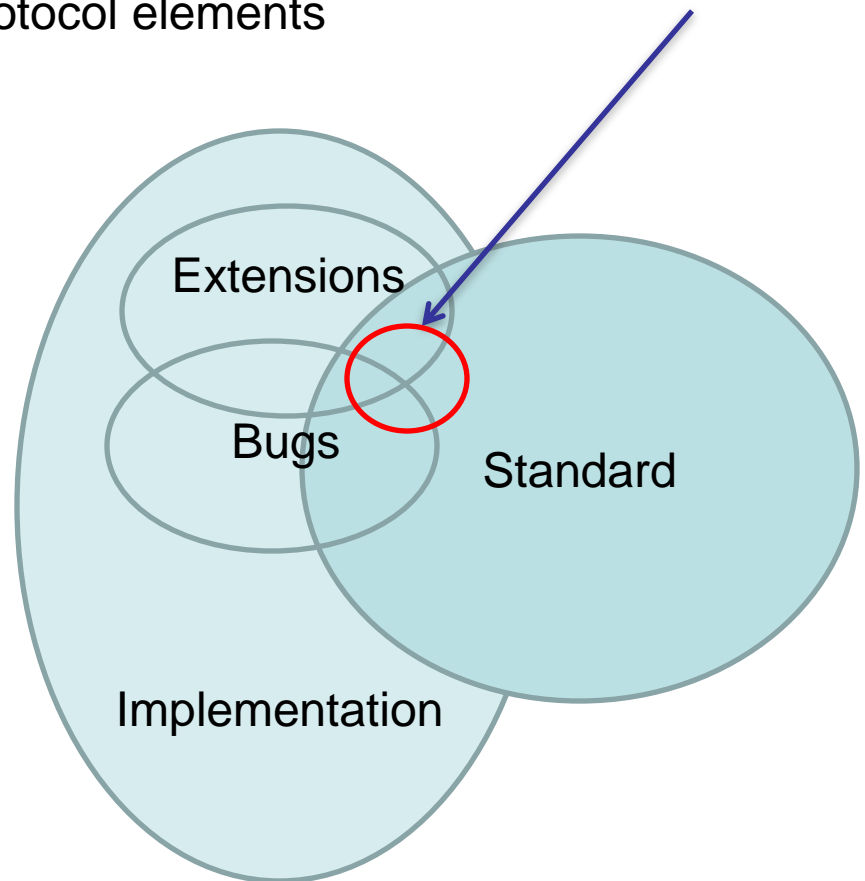


Model-based vs Template-based

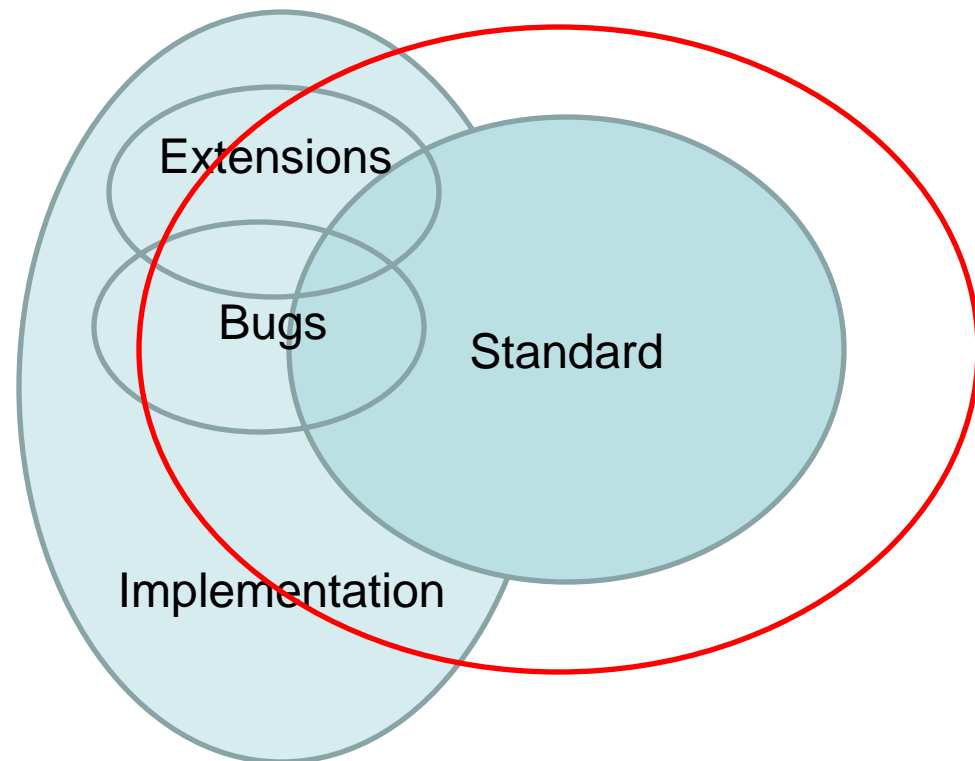
- **Benefits of Model Based Fuzzing**
 - Full test coverage (all elements, all anomaly categories)
 - Short test cycle
 - More optimized tests
 - Easy to edit and add tests to an easy to understand model
- **Template Based Fuzzing**
 - Quality of tests is based on the used seed
 - Covers only visible protocol elements
 - Blind sets of anomalies (if no meta-data on fields)
 - Very quick to develop, but slow to run
 - Editing requires deep protocol know-how

CAPTURE-BASED TESTING

Claims in the industry:
Traffic capture fuzzing tests all used protocol elements



Claims in the industry:
Specification-based fuzzing tests all used protocol elements

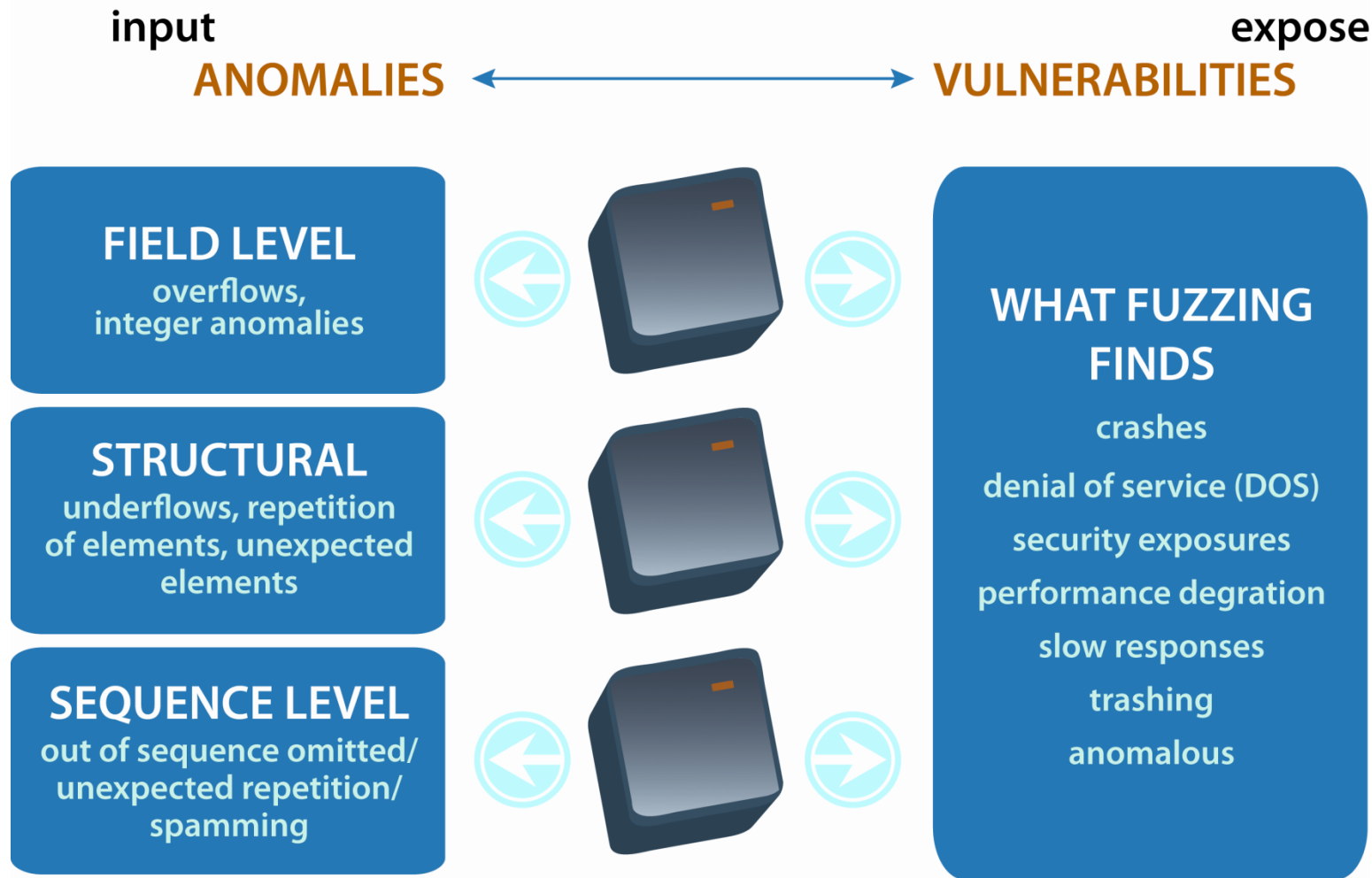


- **Capture fuzzing is not substitute for (specification) model based testing**
 - Valuable as entry level solution when model-based fuzzer is not available
 - Model based fuzzer can be adapted to proprietary extensions
 - Capture based fuzzer can't be taught rarely used elements specified by standard (*)

(Security problems are often found from the attack surface parts which are not usually covered in day-to-day traffic. Bugs are there because those parts of the code are usually less tested and reflect rarely needed portions of a protocol.*

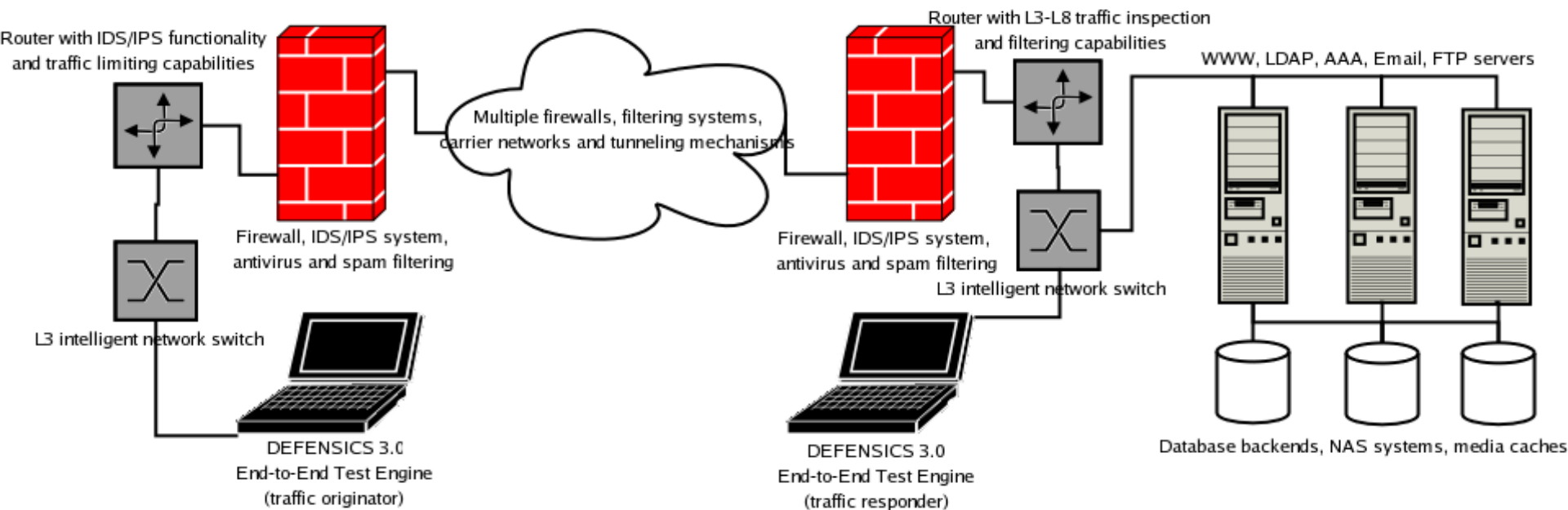
However, it does not matter if a vulnerability is in unusual interface surface, the system is still 100% vulnerable.

Challenge: Fuzzing: The Anomalies



Fuzz the Entire System (E2E)

- **Fuzz Testing needs to be conducted for the entire system, not just one layer on one interface**



THANK YOU – QUESTIONS?

“Thrill to the excitement of the chase!
Stalk bugs with care, methodology,
and reason. Build traps for them.

....

Testers!

Break that software (as you must) and
drive it to the ultimate
- but don't enjoy the programmer's
pain.”

[from Boris Beizer]



CODENOMICON

PROACTIVE SECURITY AND ROBUSTNESS SOLUTIONS