



Software Considered Harmful: Why Software is Insecure

Juha Röning
Infotech Oulu and Department of
Electrical Engineering
P.O.Box 4500
FIN-90014 University of Oulu
Finland

ouspg@ee.oulu.fi

<http://www.ee.oulu.fi/research/ouspg>



Content © by OUSPG 2002

Art © by Origion 1999



Motivation

- 📄 Software vulnerabilities prevail:

“Fragile and insecure software continues to be a major threat to a society increasingly reliant on complex software systems.”

- Anup Ghosh [Risks Digest 21.30]

- 📄 A focal problem area is software implementation, which may introduce potential for unanticipated and undesired program behaviour
- 📄 We have made some rather strong claims:
 - 🖱 (A) Secure programming errors are systematic!
 - 🖱 (B) Many vulnerabilities could be eliminated with low cost!
 - 🖱 (C) Dynamic black-box testing would be a decent first-aid!



Presentation outline

- 📄 Background and context
 - 🖱️ OUSPG
 - 🖱️ Security versus vulnerabilities
- 📄 Implementation level vulnerabilities
 - 🖱️ Specifically for embedded systems
 - 🖱️ Typical examples
 - 🖱️ Impact & Elimination
- 📄 Systematic approach to eliminate implementation level vulnerabilities: **PROTOS**
 - 🖱️ PROTOS test suites
- 📄 PROTOS test suite: c06-snmpv1
- 📄 Conclusions: Lessons learned



OUSPG

- ☞ Active as an independent and academic research group in the Computer Engineering Laboratory since summer 1996

- ☞ Our purpose:

*“To study, evaluate and develop methods of implementing and testing application and system software in order to prevent, discover and eliminate implementation level security vulnerabilities in a **pro-active** fashion.*

*Our focus is on **implementation level** security issues and software security testing.”*



Implementation & testing

- 📄 The total security of the release is the product of the specification, design, implementation and testing performed in the software process

1. Specification

2. Design

3. Implementation

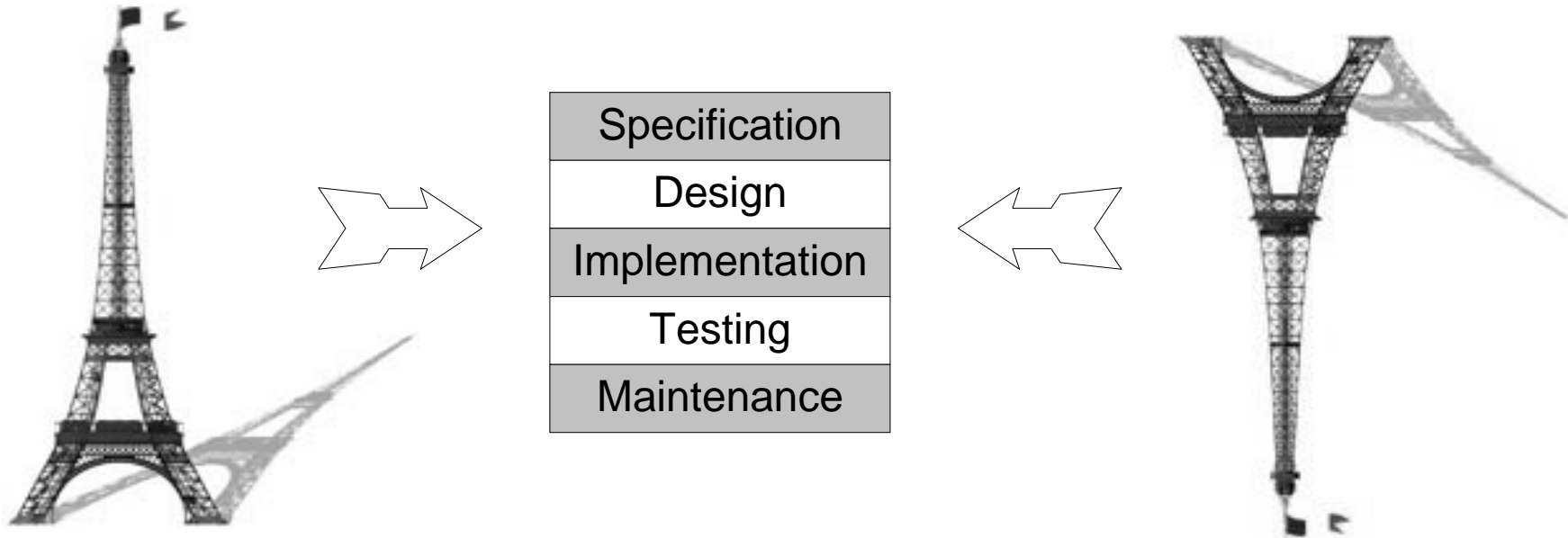
4. Testing

5. Maintenance/Use



Security Development

- 🖥️ Distribution of effort in development





Security Endangered by Vulnerabilities

- 📄 InfoSec vulnerabilities endanger (CIA):
 - 🖱 Confidentiality, Integrity and/or Availability of information
- 📄 Security may have Safety implications
- 📄 InfoSec vulnerability could be caused by:
 - 🖱 Software failure
 - 🖱 Misconfiguration
 - 🖱 Human or procedural error
- 📄 What threatens our InfoSec:
 - 🖱 Spontaneous combustion
 - 🖱 Hardware and software reliability
 - 🖱 Natural disasters
 - 🖱 Malicious activity (who we prepare for)
 - 🖱 Pranksters, Script kiddies, Terrorists, Professionals ...



Vulnerability Reality Check

$$\text{THREAT} * \text{VULNERABILITY} = \text{RISK}$$

- 📄 Security is not the Holy Grail:
 - 🖱 Address and understand risks first
 - 🖱 Risk arithmetics [$T * V = R$]:
 - 🖱 $0 * V = 0$ (no threats equals no risks)
 - 🖱 $T * 0 = 0$ (no vulnerabilities equals no risks)
- 📄 Risk is impossible to assess without possibility of measuring the vulnerability and threat
- 📄 Reactive or Proactive approach to the risk

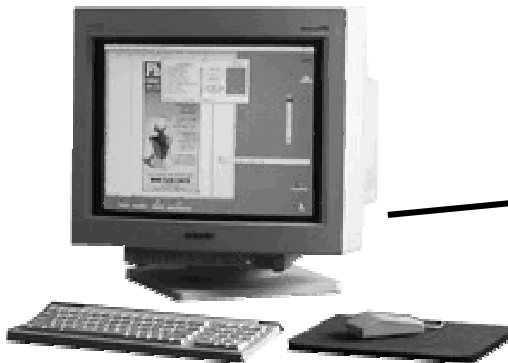


Embedded devices: new ground for vulnerabilities

- Features from workstation computers and low-end wireless terminals are integrated in embedded systems
 - Modern embedded systems have operating systems and open interfaces

FreeBSD
Linux-variants
Palm
Symbian
EPOC
Cisco IOS

...





Nature of embedded systems

- ☐ Differ from traditional computer systems
 - ☞ No hard disk (EEPROM, flash memory)
 - ☞ Weak software development tools
 - ☞ Some differences between Low-end embedded devices, PDAs and Workstation computers

Type	RAM	Data transfer rate	UI	MIPS
Low-end	> 256 kB	1kbit/s - 43.2kbit/s	Fixed	30
PDA	8MB - 64MB	9.6kbit/s - 11Mbit/s	Semi-windowed	240
WS	128MB - 64GB	56kbit/s - 1Gbit/s	Fully windowed	> 1000



Implementation level vulnerabilities

- 🖥 Embedded systems have a new problem of very limited amount of memory
 - 🖱 Primitive memory management
- 🖥 Components that process external input can have implementation level vulnerabilities
 - 🖱 Cause system to fail, either by crashing, entering a forever loop or just stopping to accept input
- 🖥 Programmers incorrectly assume that the data to be processed is of a certain form and it meets certain requirements
 - 🖱 The length of strings is always limited to pre-defined value, length values are always correct and the data is always well-formed



Flaws 1/2

Buffer overflows

- ☞ The memory reserved for a buffer or a variable can be exceeded and the program can write outside this memory (in the stack or in the heap)
- ☞ Most common vulnerabilities, very common in low-level languages

Memory allocation bombs

- ☞ Embedded systems usually have very light weight memory management in their operating systems: Processes may share the same memory space and the memory allocation function of the operating system could wait until a block big enough is free or fail instantly if a memory block big enough is not available.

Recursive parsers

- ☞ Embedded systems usually have a rather small stack, where registers and function call return addresses are saved. This will cause recursive parsers to fail, if it is possible to force it to do a deep level of recursion.



Flaws 2/2

- 📖 Signed indices or lengths
 - 🖱 Values used in table look ups and length comparisons should be unsigned. However, programmers are often using signed data types. Negative indices and lengths are sure to cause problems.
- 📖 Format string vulnerabilities
 - 🖱 Caused by incorrect usage of printf()-style functions
 - 🖱 Fairly common and serious flaw
- 📖 Missing checks for missing elements
 - 🖱 An application receiving information from the parser might crash due to a missing mandatory element it expects to be always present. This might also result in accessing illegal memory areas.
- 📖 Too small data types might cause an infinite loop due to roll-over
- 📖 Missing integer boundary value checks
 - 🖱 Missing a check could result in eg. reading data located after the end of a table, thus resulting in access violation (denial of service)



Failures

- As a flawed system receives faulty input, a failure might occur

The most serious failures for flaw types

Type	Failure mode
Buffer overflow	Execute arbitrary code
Malloc bomb	Wait forever / Write arbitrary data to memory starting from address 0x00000000
Recursive parser	Corrupt memory outside reserved stack
Missing integer boundary value check	Read from illegal memory space
Signed index or length	Execute arbitrary code (if causes buffer overflow)
Format string	Execute arbitrary code
Missing check from missing elements	Denial of service
Too small data types	Denial of service



Impact for embedded system 1/2

- 🖥️ Currently no publicly available exploits that would execute arbitrary code in any embedded system with wireless connectivity targeted to consumers
- 🖥️ To write a clean stack-based buffer overflow exploit that makes the victim system to execute arbitrary code requires at least the following knowledge:
 1. Instruction set of the target system with encoding rules
 2. Knowledge on how to crash the target system due to buffer overflow
 3. Decent knowledge of memory map of the target system. This requires knowledge of where arbitrary code will be inserted and how to be able to jump there.



Impact for embedded system 2/2



Patch deployment

- ☞ It is nearly impossible for the consumers to update the devices themselves
- ☞ If an embedded product gets mass-exploited, software upgrades to fix the vulnerability are required
- ☞ The only feasible way to upgrade the software is to go to the nearest service point which offers upgrading service
- ☞ The cost of mass-upgrading consumer electronics is high







Another approach is to add a filter to the network infrastructure which drops malicious PDUs

- ☞ Extra cost for service providers or operators
- ☞ Might decrease the network performance



Methods for vulnerability elimination 1/2

-  Vulnerability elimination is a process where vulnerabilities are searched from a software component using testing or other activities and the problems found are removed
-  Manual reading of the source code
-  Static analysis
-  Manual testing



Methods for vulnerability elimination 2/2

- 🖥️ **Manual reading of the source code**
 - 🖱️ Inspection
 - 🖱️ Code audits
- 🖥️ **Static analysis**
 - 🖱️ Tools to examine the source code for security weaknesses: Flawfinder, RATS, ITS4
 - 🖱️ Tools produce false positives (and do not cover all flaws)
- 🖥️ **Manual testing**
 - 🖱️ In vulnerability elimination, test cases are used for robustness testing
 - 🖱️ Typical test cases for robustness testing try to prove that the system does not tolerate exceptional input
 - 🖱️ Test cases can be grouped to testgroups and further to test suites



Searching for the process Grail to reduce vulnerability

- 📄 Bug prevention and elimination methods in the software development process (by B. Beizer)
 - 👉 Thorough analysis
 - 👉 Prototypes
 - 👉 Analytical models
 - 👉 Formal methods
 - 👉 Inspections
- 📄 Awareness: skills in secure programming and safety engineering
- 📄 Testing is the means for discovering the bugs that persist after these



Searching for the technical Grail to reduce vulnerability

Alternatives for educating the engineers:

- ☞ Safer libraries
- ☞ Better compilers and languages (e.g. Java)
- ☞ Operating System (kernel) solutions

Methods behind them:

- ☞ Bounds checking / strong typing (run/compile time)
- ☞ Non-executable stack, stack guarding techniques
- ☞ Sandboxing and managed code
- ☞ Code signing (You will know who to blame? ;)

Deployment? Adaptation? Completeness?

- ☞ There will still be room for a safety net provided by testing



PROTOS

Testing the Security of Protocol Implementations

- 📄 Protocols are used for communication between software functions, modules, components and packages, or even between the software and the user.
- 📄 Information security is constantly endangered by errors in the contemporary protocol implementations.
- 📄 The PROTOS project will research different approaches of testing implementations of protocols using black-box (i.e. functional) testing methods.
 - 🔗 The goal is to support pro-active elimination of faults with information security implications.
 - 🔗 Awareness in these issues is promoted.
 - 🔗 Methods are developed to support customer driven evaluation and acceptance testing of implementations.
 - 🔗 Improving the security robustness of products is attempted through supporting the development process.
 - 🔗 Vendors are informed of found vulnerabilities.
- 📄 Results are public, except for the bug reports and demonstration exploits



PROTOS - "the goal"

- 📄 Despite existence of TTCN and others, vulnerabilities were constantly found
- 📄 Testing framework
 - 🖱 *a skeletal structure designed to support or enclose something - Webster*
- 📄 Testing platform (a.k.a. scripting platform)
 - 🖱 *(Mil.) (a) solid ground on which artillery pieces are mounted ... (b) a metal stand or base attached to certain type of artillery pieces - Webster*
- 📄 At least we learn the protocols ... ;)

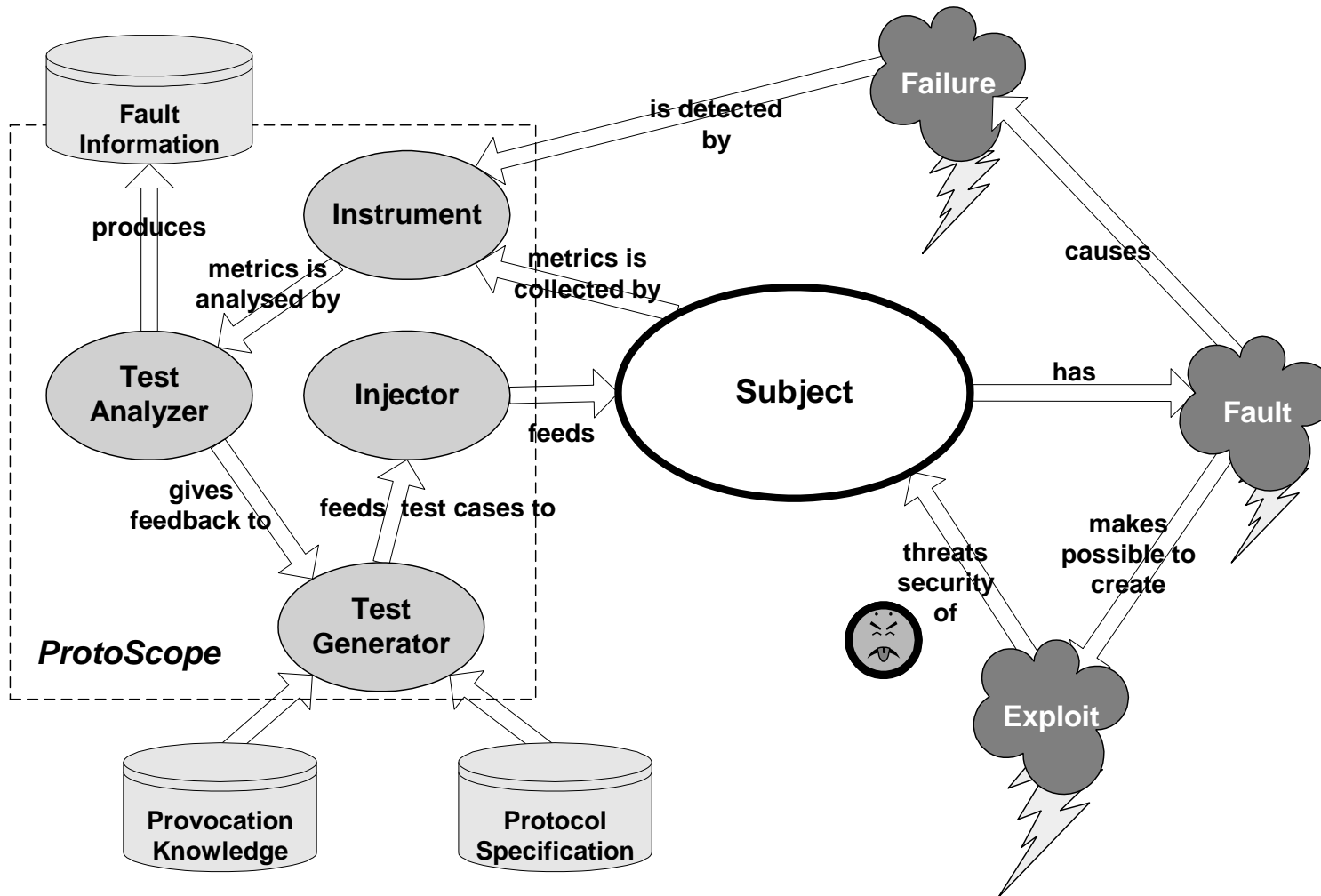


PROTOS

- 🖥️ Security Testing of Protocol Implementations
- 🖥️ Three-year (1998-2001) project in close cooperation with VTT
- 🖥️ Results:
 - 🖱️ A novel (mini-simulation) vulnerability testing method developed
 - 🖱️ Several papers and test suites published
 - 🖱️ Spin-off company Codenomicon Ltd



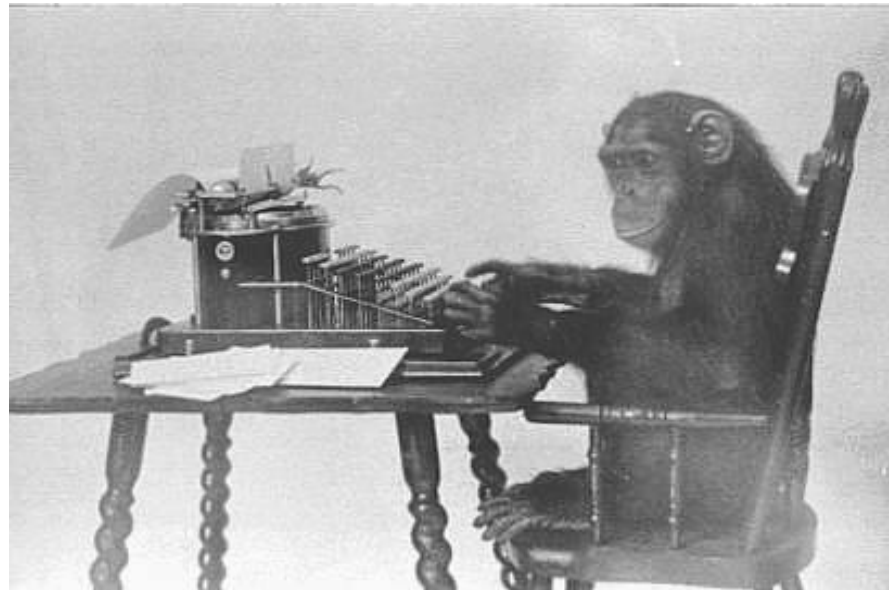
PROTOS - Framework & Platform





Our approach - in a nutshell

Today, thousands of gifted and patient, but uncoordinated monkeys are pounding different products in order to reveal vulnerabilities.

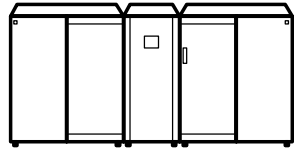


Visual by
<http://www.PDIImages.com>

Think of us as rather dumb monkeys using a monkey-machine and systematic methodology to eliminate the most trivial ones.

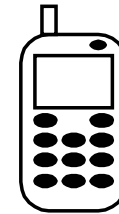


[Server & Telephony]



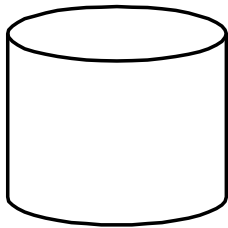
WAP Gateway

[Embedded & Telephony]

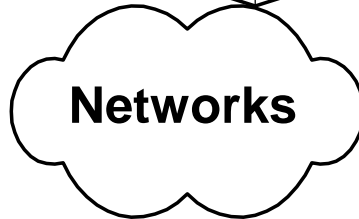


WAP Terminal

[Server & Infrastructure]

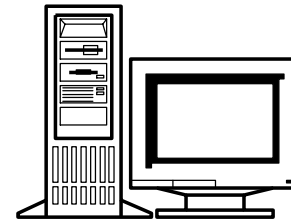


LDAP Database



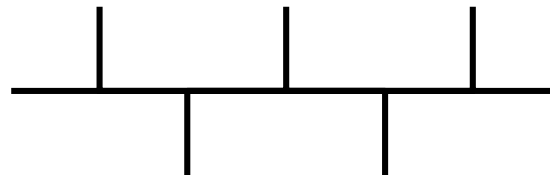
Networks

[Home & Desktop]



HTTP Client

[Infrastructure & Management]



SNMP



Test-suite summary

Test-suite	Test groups	Test cases	Failed products
wap-wsp-request	39	4236	7 (7 tested)
wap-wmlc	84	1033	10 (10 tested)
http-reply	115	3966	5 (12 tested)
ldapv3	93	12649	6 (8 tested)
snmpv1	118 / 100	29516 / 24100	12 (12 tested)



Recent PROTOS Test-Suite: c06-snmpv1

CERT® Advisory CA-2002-03 Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP)

- 🖱 <http://www.cert.org/advisories/CA-2002-03.html>
- 🖱 Couple of man months to develop
- 🖱 Several man months to coordinate
- 🖱 As of May 2002:
 - 🖱 **Over 200 vendors informed**
 - 🖱 **~140 vendors have responded publicly**
 - 🖱 **~100 vendors had affected (vulnerable) products**
- 🖱 New vendor statements keep pouring into the advisory



c06-snmpv1: Impact

- 📁 The Simple Network Management Protocol (SNMP) is the most popular protocol in use to manage networked devices. SNMP runs on a multitude of devices and operating systems, including, but not limited to:
 - 🔗 Core Network Devices (Routers, Switches, Hubs, Bridges, and Wireless Network Access Points)
 - 🔗 Consumer Broadband Network Devices (Cable Modems and DSL Modems)
 - 🔗 Consumer Electronic Devices (Cameras and Image Scanners)
 - 🔗 Networked Office Equipment (Printers, Copiers, and FAX Machines)
 - 🔗 Network and Systems Management/Diagnostic Frameworks (Network Sniffers and Network Analyzers)
 - 🔗 Networked Medical Equipment (Imaging Units and Oscilloscopes)
 - 🔗 Manufacturing and Processing Equipment
- 📁 Affected vendors include Cisco, Novell, Sun, IBM, Microsoft, 3Com, Nokia, Stonesoft, Xerox, Compaq and Dell

[<http://www.cert.org/advisories/CA-2002-03.html>]



Quotes on c06-snmpv1

- 🖥️ *"If the box is still running, and still responding to SNMP after the test suite completes, you pass. If the box crashes, reboots, or bursts into flames, you fail."* [snmp-forum -mailinglist]
- 🖥️ *"The University of Oulu [...] SNMP vulnerability warning. My advice: Find out what the university's security team will investigate next and turn it off in your environment before the team releases its next report."* [Computerworld]



Test-suite vs. bug hunters

- 🖥️ *"... Imagine if FTP was assumed to be free of exploits and somebody dumped a tool on the Internet that demonstrated all the discovered vulnerabilities all at once." [snmpv1]*
- 🖥️ *"X will deliver the patch (fixed 19 bugs) ..." [ldapv3]*
- 🖥️ The test-suites do not discover just one, but a set of vulnerabilities in the interface



Surprising findings in sub-components

- 📄 *"Very interesting. We were extremely careful, but there was a deeply embedded support routine that was not doing proper bounds checking on the host portion of the URL." [wap-wsp-request]*
- 📄 Even with careful software development, portions that were outside the process can contain failures
- 📄 Also software implemented in Java were shown to have buffer overflows in the native code sections



Surprising return packets

- 🖥️ *"The most serious problem (from a security point of view) might cause the gateway to transmit some of its memory contents as an HTTP header name to the HTTP server, though you may not have noticed it doing that." [wap-wsp-request]*
- 🖥️ Sometimes the software does not fail in noticeable form, but just returns some (confidential?) data or even memory structures to the requester



Test reproduction

- 🖥️ *"It is always good to receive reports on the performance of our products, especially when they provide details on how to reproduce problems." [wap-wsp-request]*
- 🖥️ Test-suites provide the vendor the means for assessing the quality of the product themselves



Regression testing

- 🖥️ *"I believe this alert will do wonders for improving general security in LDAP implementations."* [ldapv3]
- 🖥️ If integrated to the software development process, the test-suites have a chance of 'raising the bar' in the software products
- 🖥️ The most trivial errors are easily discovered and eliminated



Code reuse: bugs are in the hiding

- 📄 "[...] I loaded the oldest backup tape I could find and read, which was from early 1991, and some of these vulnerabilities were present then [...] these vulnerabilities have been silently present for over a decade and they are ubiquitous [...]" [snmpv1]
- 📄 A bug in software can be in the hiding, and be copied into new instances and versions of the software



Motivation for quality improvement

- 📄 *"I am disappointed in X for not even testing for these vulnerabilities until pressure was put on them through resellers and for not publicly announcing it so that administrators are made aware." [ldapv3]*
- 📄 Public pressure to reliability and security issues increases



Product comparisons

- 📄 *"I was wondering if you are going to post your results anywhere for us to look through? We would be interested to see how we compared to the other products you have been testing." [wap-wsp-request]*
- 📄 Both the vendors themselves and the customers lack the means of comparing product quality between products of different vendors



Conclusions: Lessons learned

- 📄 Security should be inheritant in software
- 📄 Security is, by its nature, risk analysis
- 📄 Death-Zones were apparent in PROTOS test-suites
 - 🖱 Several products had problems in exactly same categories
 - 📄 E.g. String table index handling in wap-wmlc and proxy authentication in http-reply were rather error prone
- 📄 Decoder problems are abundant
- 📄 Real diversity is not produced by developing different implementation with same toolkits, but by using different tools and paradigms
- 📄 Embedded systems are becoming similar to conventional WS computers
 - 🖱 Formerly closed systems will be used in open context (IP networks etc.)
 - 🖱 Vulnerabilities will emerge due to immature software culture
 - 🖱 Consumer electronics will have security problems