

Malware and Exploit Enabling Code

Information Assurance

Fall 2006

Reading Material

- In Computer Security: Art and Science
 - Vulnerability Analysis Section 23.4.1 Malicious Logic – Chapter 22
- Sony DRM article
 - <http://blogs.technet.com/markrussinovich/archiv>
- Worm Anatomy and Model
 - <http://portal.acm.org/citation.cfm?id=948196>
- Smashing the Stack for Fun and Profit
 - <http://phrack.org/phrack/49/P49-14>

Outline

- Malware
 - Trojans, Virus, Worms, etc.
- Exploitable Code Issues
 - Configuration Management
 - Buffer Overview
 - Format String
 - Input Checking
 - Time-of-use to Time-of-check
- Ethics of hacking

Why Do We Care?

- SANS Top 20 Internet Security Vulnerabilities
 - <http://www.sans.org/top20/>
- Broad issues very similar year in and year out

Top 20 Vulnerabilities - Windows

- W1 Web Servers & Services
- W2 Workstation Service
- W3 Windows Remote Access Services
- W4 Microsoft SQL Server (MSSQL)
- W5 Windows Authentication
- W6 Web Browsers
- W7 File-Sharing Applications
- W8 LSAS Exposures
- W9 Mail Client
- W10 Instant Messaging

Top 20 Vulnerabilities - Unix

- U1 BIND Domain Name System
- U2 Web Server
- U3 Authentication
- U4 Version Control Systems
- U5 Mail Transport Service
- U6 Simple Network Management Protocol (SNMP)
- U7 Open Secure Sockets Layer (SSL)
- U8 Misconfiguration of Enterprise Services NIS/NFS
- U9 Databases
- U10 Kernel

Windows Meta File Exploit

- Exploit flaws in the Windows rendering engine enable remote code execution
 - Memory corruptions
 - Visiting web site with “bad image” causes attack
 - Attack sold for \$4,000
 - <http://www.eweek.com/article2/0,1895,1918198,00.asp>
- Bugtraq post in December.
 - Probably lingering earlier
 - 0 day exploit
- Microsoft’s response in early January
 - <http://www.microsoft.com/technet/security/bulletin/ms06-001.ms>

Malicious Logic

- Set of instructions that cause a site's security policy to be violated
- Often leveraging an inadvertent flaw (design or implementation)
 - To propagate/install on target
 - To cause harm on target

Malicious Code Taxonomy

- Virus: A program that attaches itself to non-malicious programs and propagates itself to other programs.
- Worm: Propagates copies of itself through a network
- Trojan Horse: Malicious code that in addition to its primary non-malicious effect, has a non-obvious malicious effect
- A logic bomb: A class of malicious code that “detonates” when a specified condition occurs
- Trapdoor: Allows unauthorized access to undocumented functionality
- Rabbit: Replicates without limit to exhaust memory
- Rootkits: Tools to misrepresent what is on the system
- Keylogger/spyware: Code that observes and reports actions on the computer
- Netbots: Programs controlled through a communication channel (originally IRC). Can be used for DDoS

Trojan Horses

- Seemingly useful program that contains code that does harmful things
 - Perform both overt and covert actions
- Frequently embedded in applets or games, email attachments
- Trojan horse logins, spoof authentication or webpage forms
- Thompson's early login/compiler example

Key Loggers and Spyware

- Gather information from computer
 - Send back to the central office
- From key loggers can gather
 - Passwords
 - Confidential communication
 - Keep track of your kids/employees
- From spyware can gather
 - Web browsing habits
 - Gather marketing information

Rootkits

- Insert file filters to cause files or directories disappear from normal listings
 - Can replace Windows API pointers (user mode)
 - Can also replace syscall table pointers
- Both require privilege, but with Windows most installs require privilege anyway
 - The power of extensibility used for the dark side
- Techniques apply equally well to Linux and Mac

Sony Player DRM and Rootkits

- Bad press for Sony last year
 - Mark Russinovich's original observations
<http://blogs.technet.com/markrussinovich/archiv>
 - A timeline
http://www.boingboing.net/2005/11/14/sony_an
- To ensure that copy protection is not evaded install rootkit to hide the protection code
 - Available for other attackers to use
 - Uninstallable
 - Uses CPU and memory
 - Not adequately noted in EULA

Virus Operation

- Virus Phases:
 - Dormant: Waiting on trigger event
 - Propagation: Replicating to programs/disks
 - Triggering: By event to execute payload
 - Execution: Executing payload
- Details usually Machine/OS specific
 - Exploits different features or weaknesses

Virus Pseudocode

- beginvirus:
- If spread-condition then begin
 - For some set of target files do begin
 - If target is not infected then begin
 - Determine where to place virus instructions
 - Copy instructions from beginvirus to endvirus into target
 - Alter target to execute new instructions
- Perform some actions
- Goto beginning of infected program
- endvirus:

Virus Attachment

- A Virus can attach itself to a program or to data by
 - Appending itself to either the beginning or end of either source code or assembly, so it is activated when the program is run
 - Integrate itself into the program, spread out code
 - Integrate into data: executable text macro, scripting
 - Macros and email attachments
- An activated virus may:
 - Cause direct or immediate harm
 - Run as a memory resident program (TSR, daemon, or service)
 - Replace or relocate boot sector programs, start at system start-up

Homographic Phishing

- Design error
 - Inconsistent parameter validation
- Trying to solve problem of displaying International Domain Names (IDN) in Roman alphabet browsers
 - E.g., Maps Russian 'а' to Roman 'a'
 - Enables creation of URLs and certificate names that are indistinguishable
- More details
 - Bad links <http://www.shmoo.com/idn/>
 - <http://db.tidbits.com/getbits.acgi?tbart=07983>

Macros Viruses

- Macro code attached to some data file
 - Interpreted rather than compiled
 - Platform independent
- Interpreted by program using the file
 - E.g., Word/Excel macros
 - Esp. using auto command and command macros
 - Often automatically invoked
- Blurs distinction between data and program files making task of detection much harder
- Classic trade-off: "ease of use" vs "security"

Email Viruses

- Spread using email with attachment containing a macro virus
 - Melissa, LoveBug
- Triggered when user opens or executes attachment
 - Also when mail viewed by using scripting features in mail agent
 - Usually targeted at Microsoft Outlook mail agent and Word/Excel documents, Microsoft IIS

Basic Precautions

- Don't import untrusted programs
 - Who can you trust?
 - Viruses have been found in commercial shrink-wrap software
 - Standard download sites have been corrupted
 - Check MD5 sigs
- Scan for viruses, install anti-virus software
- Update anti-virus software regularly

Signature Scanning

- Early viruses had characteristic code patterns known as signatures
- Create a database of patterns, search files for patterns (McAfee)
- Use data-mining, learning, feature extraction etc. to look for disguised or obfuscated patterns
- Can only scan for known signatures

Signature Avoiding Viruses

- **Polymorphic Virus** produces varying but operational copies of itself
 - Use alternative but equivalent instructions
 - Gets around signature scanners. Whale virus, 32 variants
- **Stealth Virus** actively tries to hide all signs of its presence
 - A virus can intercept calls to read a file and return correct values about file sizes etc. Brain Virus

Another Signature Avoiding Virus

- **Encrypted Virus** stores bulk of self encrypted
 - Small decrypt routine in clear
 - Key stored in clear

Worms

- Propagate from one computer to another
- Viruses use email/infected media to propagate to so differentiation is fuzzy

The Morris Worm Incident

- How 99 lines of code brought down the Internet (ARPANET actually) in November 1988.
- Robert Morris Jr. Ph.D student, Cornell, wrote a program that could:
 - Connect to another computer, and find and use one of several vulnerabilities (buffer overflow in fingerd, password cracking etc.) to copy itself to that second computer.
 - Begin to run the copy of itself at the new location.
 - Both the original code and the copy would then repeat these actions in an infinite loop to other computers on the ARPANET (mistake!)
- Morris was sentenced to three years of probation, 400 hours of community service, and a fine of \$10,050. He is now a Professor at MIT.
- Worms have gotten bigger and more aggressive

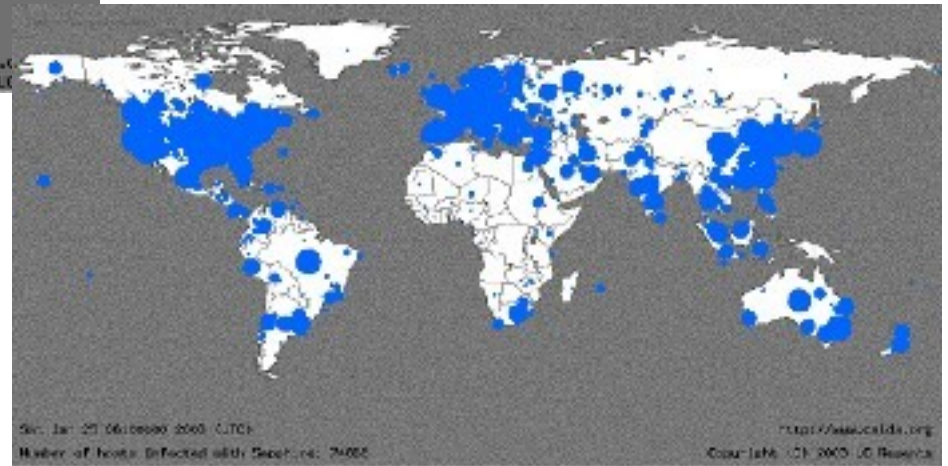
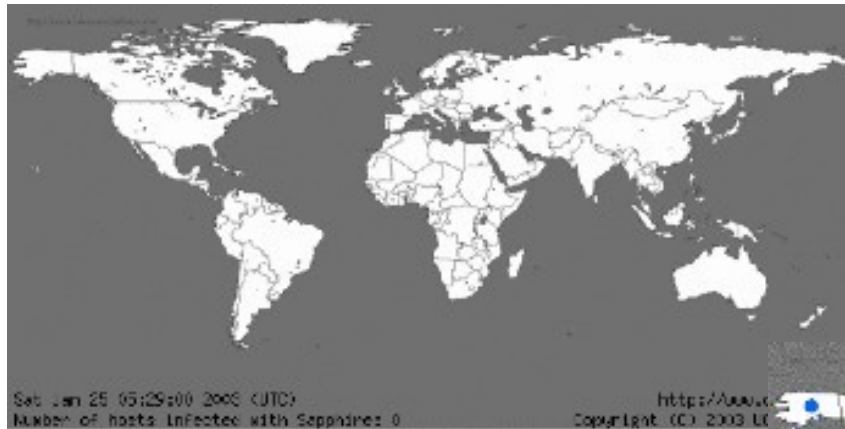
Worm Phases

- Dormant
- Propagation
 - Search for other systems to infect
 - Establish connection to target remote system
 - Replicate self onto remote system
- Triggering
- Execution

Who to target?

- Scanning
 - Currently generally used
 - Select random addresses
 - Mix of addresses in current network (local computers probably have similar vulnerabilities) and remote networks
 - No longer feasible in IPv6
 - 32 bit vs 128 bit address space

Viruses and Worms in IPv4



- Slammer infected most of the IPv4 Internet in 10 minutes (75,000 hosts infected in one-half hour)

Source [caida.org](http://www.caida.org)

Worms in IPv6

- Address space is 2^{128} instead of 2^{32}
 - Random address selection will not work
- Say $\frac{1}{4}$ of address in IP4 network run Windows
 - 1 in 4 chance of finding a target with each probe
- Spread that among 2^{128} addresses
 - 1 in 2^{98} chances of finding a viable target

Viruses and Worms in IPv6

- Pure Viruses don't change in IPv6 but hybrid and pure worms do.
 - Hybrids and pure worms today rely in Internet scanning to infect other hosts, this isn't feasible as shown earlier in this presentation.
 - At 1 million packets per second on a IPv6 subnet with 10,000 hosts it would take over 28 years to find the *first* host to infect
 - Let's take a look at the same animation this time simulating how slammer might fare in an all IPv6 Internet:



- Worm developers will adapt to IPv6 but pure random scanning worms will be much more problematic for the attacker. Best practices around worm detection and mitigation from IPv4 remain.

Other Techniques to Find Targets

- Interesting Papers
 - How to Own the Internet...
<http://www.icir.org/vern/papers/cdc-usenix-sec02/>
 - Top speed of flash worms <http://vividmachines.com/papers/topspeed.pdf>
- Hitlist Scanning
 - Stealthy scans (randomized, over months), distributed scanning,
- DNS searches, Spiders (Code red, crawls for high connectivity), listening on P2P networks, public lists
- Permutation scanning (divide up IP address space)
- Warhol worm- Hit list + permutation

Network Propagation

- Send small number of packets to reduce detection
- UDP packets
 - No ACK needed, so can spoof source address
- Connect to vulnerable network services
 - Generally exercise buffer overflow
 - Launch shell
 - Running at high privilege (ideal)
 - Or use as foothold to mount other attacks to gain privilege
 - Or use as attack launch point

Worm Examples

- Morris Worm
- Code Red
 - Exploited bug in MS IIS to penetrate and spread
 - Probes random IPs for systems running IIS
 - Had trigger time for denial-of-service attack
 - 2nd wave infected 360000 servers in 14 hours
- Code Red 2 - trapdoor, for remote control
- Nimda - used multiple infection mechanisms, email, file-sharing, web-client, IIS, Code Red 2 backdoor

NetBots

- Install on compromised machines
- Master sends commands to netbots
 - Originally communicate through IRC
 - Cause DDoS
- Stable framework to create your own netbots
 - <http://www.egghelp.org/>
 - <http://www.energymech.net/>

General Defenses Against Malware

- User education
- Detect program changes
 - Trip wire
- Scanning programs
 - Virus scans
 - Rootkit revealers
- Intrusion detectors
 - NIDS to detect worm probes
 - HIDS to detect odd behaviors on infected systems
- Keep system patches up to date
- Quarantine Systems
 - Detect systems where version is out of spec and force off network until further investigation

Taxonomy of Program Security Flaws

- Malicious vs non-malicious flaws
 - Malicious flaws introduced by programmers deliberately, possibly by exploiting a non-malicious vulnerability. e.g., Worms, Trapdoors, Logic Bombs
 - Non malicious flaws are oversight. e.g., Buffer overflow, TOCTTU flaws etc.
- Can divide flaws into seven categories (RIOS):
 - Incomplete parameter validation
 - Inconsistent parameter validation
 - Implicit sharing of privileged/confidential data
 - Asynchronous validation/inadequate serialization
 - Inadequate identification/authentication/authorization
 - Violable prohibiting/limit
 - Exploiting logic error

Buffer Overflow

- Most common pen-test security vulnerability 2003 (SANS/FBI)
- One of the most serious classes of security threats
 - An attacker can gain partial or complete control of a host
- A buffer (array or string) is a space in which data can be held
- A buffer's capacity is finite:
 - `char sample[10];`
 - `sample[10] = 'A';`
- Buffer sizes do not have to be predefined. Out-of-bounds error

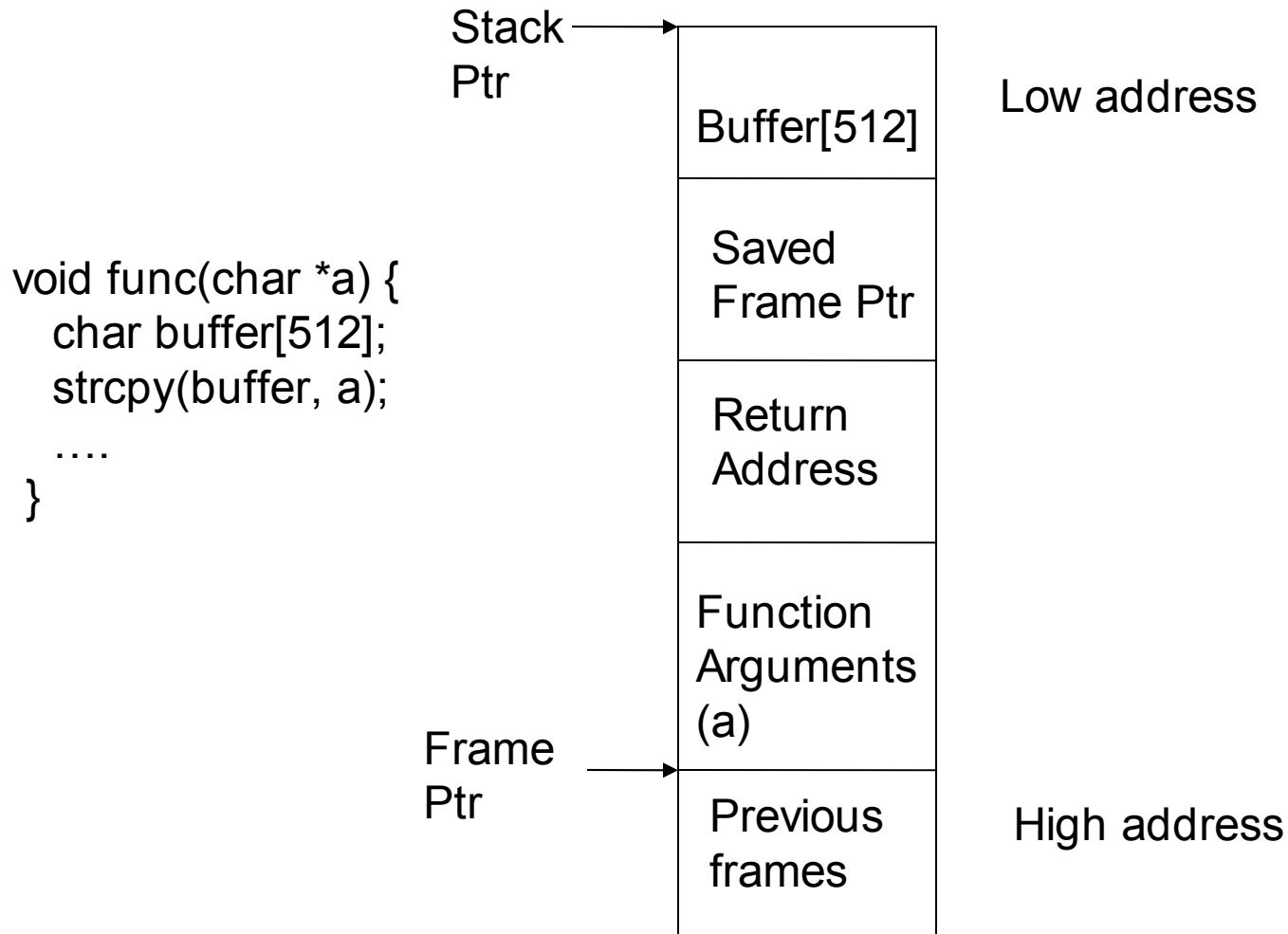
What Happens When A Buffer Overflows?

- A program that fails to check a buffer overflow may allow vital code or data to be overwritten
- A buffer may overflow into and change:
 - User's own data structures
 - User's program code
 - System data structures
 - System program code
- Most common attack is to subvert the function of a privileged program and take control of the host

Stack Smashing

- Attacker overflows automatic variable to corrupt the return address in the AR
- Also called Stack Smashing Attack.
 - Most common buffer-overflow attack
- Rewrite return address or frame pointer with attack code, or rewrite pointer to address to “attack” code in user memory
- On return executing code in stack buffer at original program privilege
 - Typically attackers exec a shell

Stack Structure



Shell Code

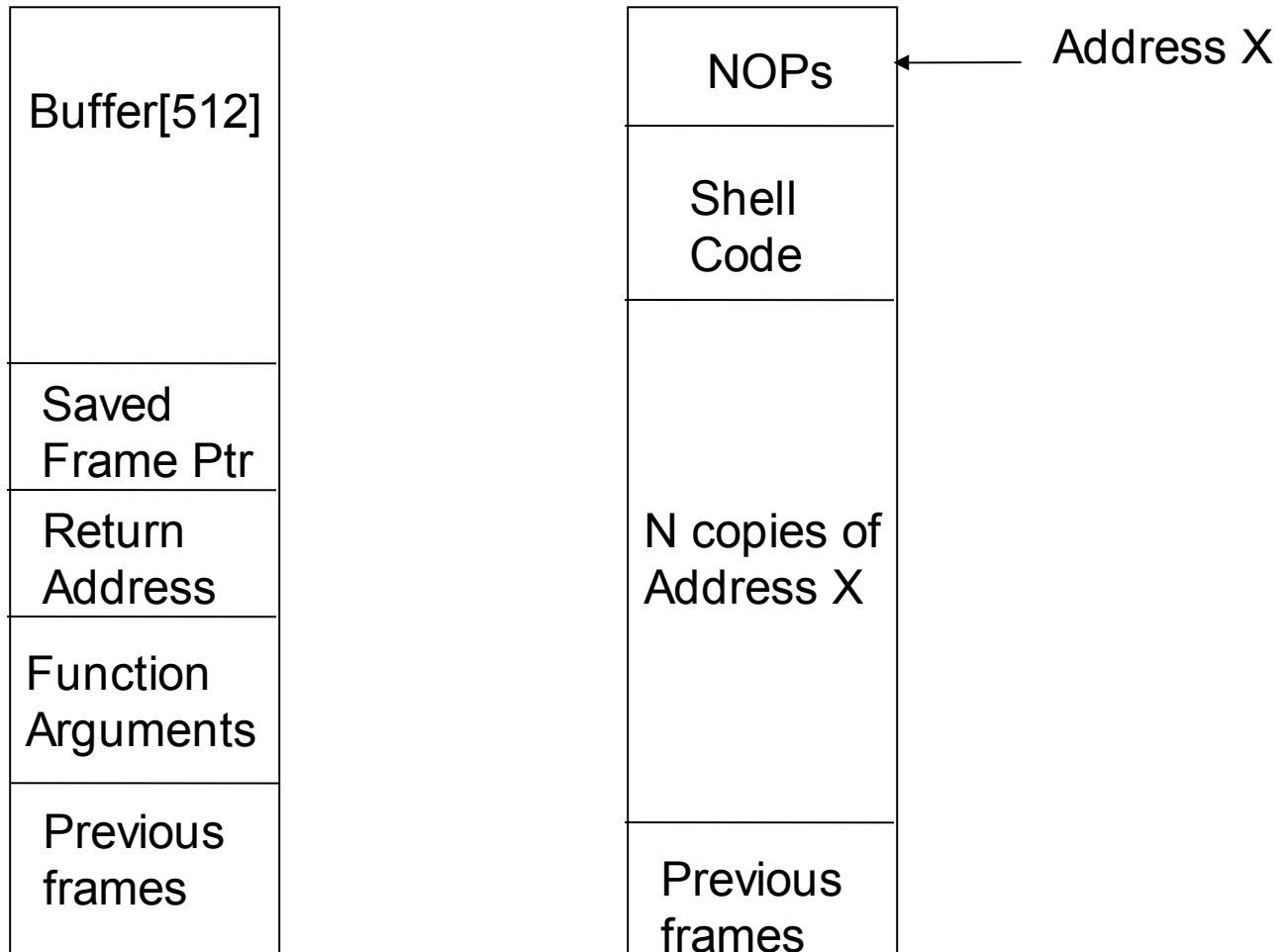
- Insert code to spawn a shell
- Phrack article discusses how to do this from first principles
 - Create assembly code to exec /bin/sh
 - Use GDB to get hex of binary code
 - Rework assembly as necessary to avoid internal 0's
 - Could break attack if strcpy is used by attack target
- Will result in a hex string like:
 - “\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40xcd\x80\xe8\xdc\xff\xff\xff/bin/sh”

Attack Buffer

- Buffer more than 512 bytes will replace other information on the stack (like return address)
- Problem is determining absolute address in buffer to jump to and ensuring you replace the return address
 - Pad with leading NOPs and trailing return addresses
 - Then your guesses on the stack structure do not need to be exact

NOPs	Shell Code	Return Address Replacements
------	------------	-----------------------------

Copied Stack



Buffer Overflow Defenses

- Write correct code
- Use appropriate languages
- Use tools to analyze problems
- Address Space Randomization
- Make buffers non-executable
 - Should never need to execute code on the stack or on the heap

Writing Correct Code

- Simple solution, but expensive!
 - Performance vs. correctness
 - Software industry practices
- Automatic source-code analysis (limited scope)
 - Super greps like RATS and FlawFinder
 - Embedded compiler analysis
- Audit teams, code review

Use Appropriate Language

- Languages that are type-safe and enforce bound checks
 - E.g., Java, ML, Smalltalk
 - Perl and Taint-mode
- Subsections of language and/or code standards
 - C++ using only smart pointers, `std::strings`, and STL containers
 - Managed Code and the Common Runtime Library (CRL)

Tools for Buffer Overflow Protection

- LibSafe
 - <http://www.research.avayalabs.com/project/libsafe/>
 - Intercept calls to functions with known problems and perform extra checks
 - Source is not necessary
- StackGuard and SSP/ProPolice
 - Place “canary” values at key places on stack
 - Terminator (fixed) or random values
 - ProPolice patch to gcc

Address Space Randomization

- Vary the base stack address with each execution
 - Stack smashing must have absolute address to overright function return address
 - Enabled by default in some linuxes (e.g., FC3)
- Wastes some address space
 - Less of an issue once we have 64 bit address space
- Not absolute
 - Try many times and get lucky

Incomplete Parameter Validation

- Failure to perform “sanity checks” or “range checks” on data
- Filling wrong values in correct format
- Example: USS Yorktown
 - “Smart ship” with Aegis missiles and on-board control system on Windows NT LAN
 - Caused a database overload when someone entered a zero in a data field—the action that triggered the Yorktown’s LAN crash Sept. 21, 1997.
 - Had to be towed into Norfolk, VA

Time-of-Check to Time-of-Use Attacks

- A delay between checking permission to perform certain operations and using this permission. Lazy binding
- Example: Separate file access check from file open
 1. If `access(file_path, "w") == allowed`
 2. `file_id = open(file_path, "w")`
 3. `return file_id`
- Say `file_path="/usr/tom/X"`
 - For step1, this is a simple file in tom's directory
 - For step2, `/usr/Tom/x` is a symlink to `/etc/passwd`
- Asynchronous validation flaw

Configuration Management

- Control of changes made in the system's hardware, software, firmware, documentation, testing through system life cycle
- Requirements
 - Version control and tracking
 - Often involves version control system like Subversion or ClearCase
 - Change authorization
 - Integration procedures
 - Use of branches or labels to select versions to build
 - Tools for product generation
 - Makefiles and build scripts

Defense Through Attack

- Ethical hacking
 - You too can become a certified ethical hacker
 - <http://www.wired.com/news/infostructure/0,1377,640>
 - <http://www.vigilar.com/training/ceh/index.html?gclid=C>
- Sexy term for
 - Penetration Testing
 - Vulnerability analysis
 - Vulnerability researching

Penetration Testing

- Bring in outside team to “attack” system
 - Well-defined rules of engagement, e.g.,
 - no DOS but social engineering is allowed
 - Specified target of attack
 - Cause no permanent damage
 - Amount of inside knowledge
- Benefits
 - Ability to think outside the box may reveal new issues
- Concerns
 - All discovered flaws reported?
 - Probably not systematic

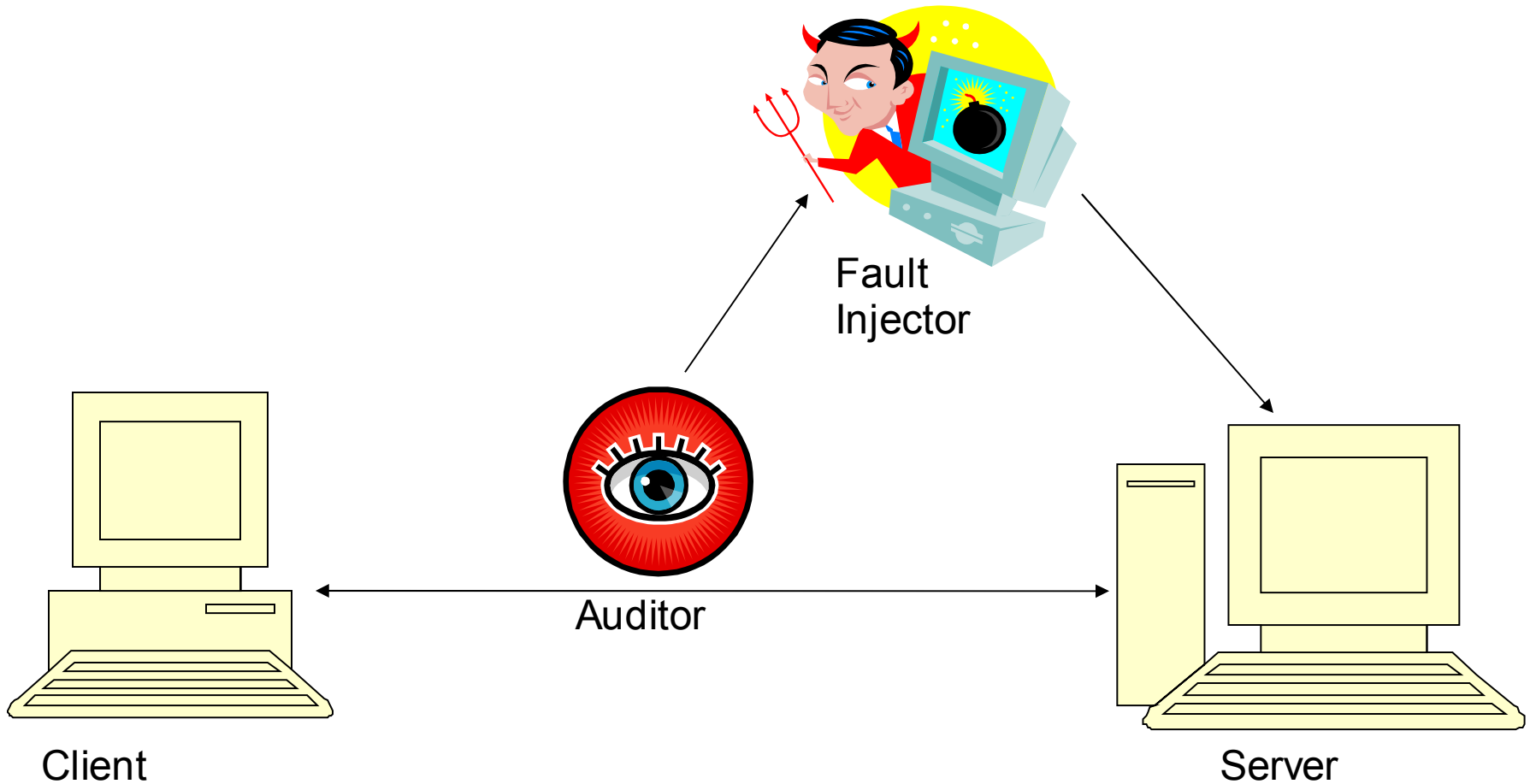
Vulnerability Research

- Find exploits in deployed software
 - Zero Day exploit – Exploit that is released before fix is available
- Ethical issues once exploit is found
 - How soon to reveal exploit after giving vendor heads up?
 - Can you protect your customers in the mean time?

Software Fault Injection

- Hardware fault injection well used and understood
 - Software fault injection still emerging
 - Active research area at CSL
- Identify input areas
 - Generally network, but could also be files, environment variables, command line
- Inject bad inputs and see what happens

Fault Injection Model



Fuzzing

- A variant of the fault injection model
 - Create “fuzzed” input to cause errors
- ShareFuzz
 - Intercept all `getenv()` calls to return very, very long strings

More Fuzz - SPIKE

- An input language for creating variant network packets
- From ethereal output, make it easy to express new packets
 - `a_binary("00 01 02 03")`
Data: <00 01 02 03>
 - `a_block_size_big-endian_word("Blockname");`
Data: <00 01 02 03 00 00 00 00>
 - `a_block_start("Blockname")`
`a_binary("05 06 07 08")`
Data: <00 01 02 03 00 00 00 00 05 06 07 08>
 - `a_block_end("Blockname");`
Data: <00 01 02 03 00 00 00 04 05 06 07 08>

Program Tracing

- Run target program in debugger
 - Get first chance at all exceptions
- Instrument target program to concentrate on expected vulnerability
 - Hook functions
- ltrace/strace
 - Lists library and system calls

Exploit Frameworks

- Metasploit
 - <http://www.metasploit.com/index.html>
- Canvas
 - <http://www.immunitysec.com>
- Core Impact
 - <http://www.coresecurity.com/products/coreimpact/ind>

Key points

- Malware is real
 - Propagation aspects
 - Attack aspects
- Implementation flaws directly enable system attacks
- Defense mechanisms
- Ethical Hacking