

# Malware and Protections

Cyber Security Lab

2006

# Outline

- A recent exploit
  - Windows Meta File
- Exploit Protection
  - Safe Code
  - Analysis
  - Fault Injection
  - Fuzzing

# Other exploit references

- Steve Hanna's Shellcode page
  - <http://vividmachines.com/shellcode/shellcode.html>
- Non-Stack Based Exploitation of Buffer Overrun Vulnerabilities on Windows NT/2000/XP
  - <http://www.ngssoftware.com/papers/non-stack-bo-windows.pdf>
- Once Upon a Free()
  - <http://phrack.org/phrack/57/p57-0x09>
- Exploiting Format String Vulnerabilities
  - <http://doc.bughunter.net/format-string/exploit-fs.html>

# 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.msp>

# How Can We Protect Ourselves?

- Avoid creating bugs
- Change environment to detect, mitigate errors
- Find bugs ourselves
  - Ethical(?) hacking

# Write Correct Code

- Seemingly simple solution, but...
- Extra time to develop
  - Code reviews
  - Testing
  - Analysis

# 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
- Performance vs. correctness
  - Bounds checking in Pascal vs C
- What about code you inherit?

# Change the Environment

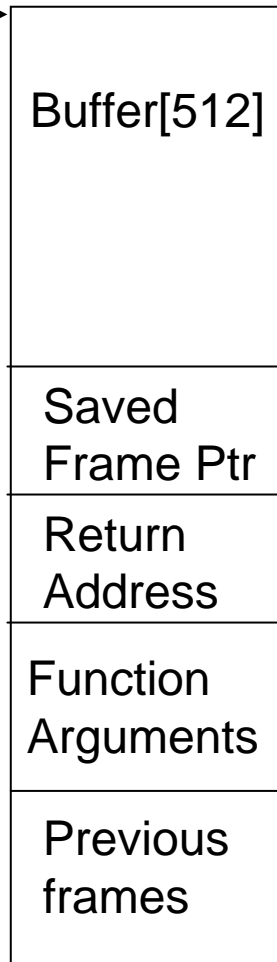
- Use tools to stop exploit effectiveness

# 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
    - [http://en.wikipedia.org/wiki/Stack-smashing\\_protection](http://en.wikipedia.org/wiki/Stack-smashing_protection)
  - Terminator (fixed) or random values
  - ProPolice patch to gcc

# LibSafe

Target  
Buffer

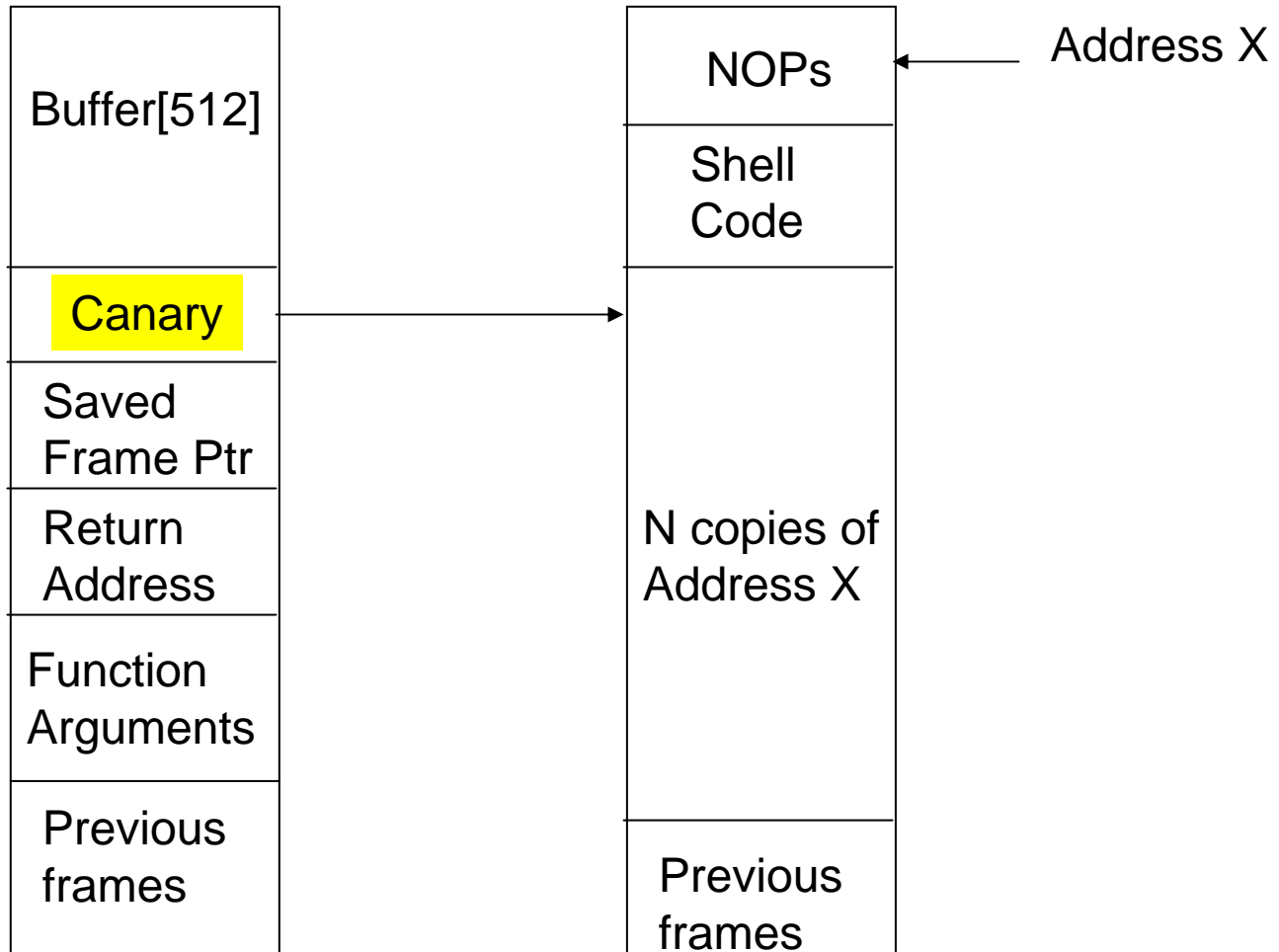


Uses LD\_PRELOAD to intercept all  
“dangerous” calls.

Use Frame pointer and buffer address to  
detect corruption of stack

Frame Pointer

# Canary Values



# Address Space Randomization

- Vary the base stack address with each execution
  - Stack smashing must have absolute address to overwrite 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
- Does not help return to libc or heap overflows

# Non-Executable Stack

- Set page as non-executable
  - Supported by newer AMD and x86 chips
  - Supported by some OS's
- Does not protect against return to libc or heap attacks.

# Find the Bugs First

- In your code
- In inherited/purchased code.

# Static Code Analysis

- Code Reviews/Audits
  - Very time consuming
- Super greps
  - Splint
  - RATS
- More sophisticated code analysis in research and commercial world
  - Still many false positives to track down
- In all cases might find problems, but may be difficult to determine if problem is really exploited

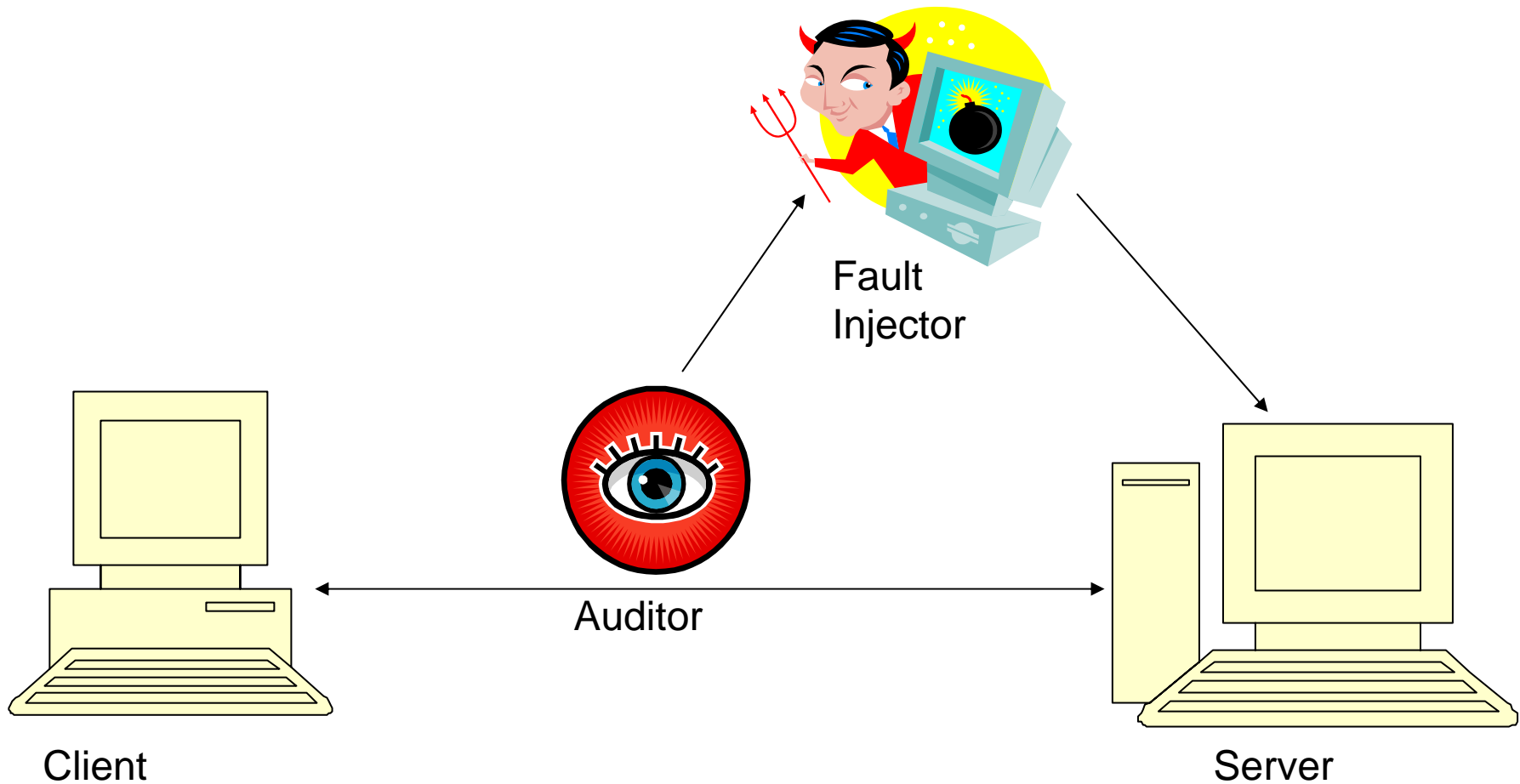
# Combine Runtime and Compile Time Analysis

- BoundsChecker and related tools
  - <http://www.compuware.com/products/devpartner/>
  - Augments code with bounds checking code
  - Coverage Analysis
- Rational Purify
  - <http://www-306.ibm.com/software/awdtools/purify/>

# 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/index.php>

# Ethical Concerns

- Resulting vulnerabilities can be used for good or bad
  - When should the public be notified?
- By educating people about exploit finding aren't you just expanding the population of potential hackers?
  - Tools for script kiddies