Malicious Code for Fun and Profit

Mihai Christodorescu
mihai@cs.wisc.edu
23 March 2006
What is Malicious Code?

Viruses, worms, trojans, ...
Code that breaks your security policy.

Characteristics

\[
\begin{align*}
\text{Attack vector} \\
\text{Payload} \\
\text{Spreading algorithm}
\end{align*}
\]
Outline

• Attack Vectors
• Payloads
• Spreading Algorithms
• Case Studies
Attack Vectors

• Social engineering
  “Make them want to run it.”

• Vulnerability exploitation
  “Force your way into the system.”

• Piggybacking
  “Make it run when other programs run.”
Social Engineering

• Suggest to user that the executable is:
  – A game.
  – A desirable picture/movie.
  – An important document.
  – A security update from Microsoft.
  – A security update from the IT department.

• Spoofing the sender helps.
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking
• Payloads
• Spreading Algorithms
• Case Studies
Vulnerability Exploitation

• Make use of flaws in software input handling.

• Sample techniques:
  – Buffer overflow attacks.
  – Format string attacks.
  – Return-to-libc attacks.
  – SQL injection attacks.
A buffer overflow occurs when data is stored past the boundaries of an array or a string.

The additional data now overwrites nearby program variables.

Result:
Attacker controls or takes over a currently running process.
Expected input: `\hostname\path`

```c
void process_request( char * req )
{
    // Get hostname
    char host[20];
    int pos = find_char( req, '\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );
}

process_request( "\tux12\usr\foo.txt" );  ⇒  ✓ OK
process_request( "\aaabbbccccdddeeeeffffggghh\bar" );  ⇒  ✗ BAD
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );
    ...  
    return;
}
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );
    ...
    return;
}
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[20];
    int pos = find_char( req, '\\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );
    ...
    return;
}
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\', 2 );
    strcpy( host, 
        substr( req, 2, pos - 1 ) );
    ...
    return;
}
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );
    ...
    return;
}
```
A stack frame per procedure call.

```c
void process_request( char * req )
{
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\\', 2 );
    strcpy( host,
            substr( req, 2, pos - 1 ) );

    ...

    return;
}
```
void process_request( char * req ) {
    // Get host name
    char host[ 20 ];
    int pos = find_char( req, '\', 2 );
    strcpy( host,
            substr( req, 2, pos – 1 ) );
    ...
    return;
}
Normal Execution

```c
void process_request(const char * req)
{
    // Get hostname
    char host[20];
    int pos = find_char(req, '\\', 2);
    strcpy(host,
           substr(req, 2, pos - 1));
    ...
    return;
}
```

```plaintext
process_request( "\\tux12\usr\foo.txt" );
```

Diagram:
- `main()`
  - arg: `req`
  - process_request()
    - return address
    - frame pointer
    - local: `host`
      - 2 \ 0
      - t u x 1
    - local: `pos` 7
```
void process_request( char * req ) {
    // Get hostname
    char host[ 20 ];
    int pos = find_char( req, '\', 2 );
    strcpy( host, substr( req, 2, pos – 1 ) );
    ...
    return;
}
Smashing the Stack

The attacker gets one chance to gain control.

Craft an input string such that:
- The return address is overwritten with a pointer to malicious code.
- The malicious code is placed inside the input string.

Malicious code can create a root shell by executing 
"/bin/sh".
### Shell Code

#### Code for `exec("/bin/sh")`:

```
mov edx, arg2
mov ecx, arg1
mov ebx, "/bin/sh"
mov eax, 0Bhint 80h
```

#### Pointer value for overwriting the return address.

```
i n / s h \0 arg 2
```

```
arg 2 arg 1 pointer
to code
```
• Defense against stack-smashing attacks:
  – Bounds-checking.
  – Protection libraries.
  – Non-executable stack.
  – setuid()/chroot().
  – Avoid running programs as root!
  – Address randomization.
  – Behavioral monitoring.
More Info

“Smashing the Stack for Fun and Profit”
by Aleph One

StackGuard, RAD, PAX, ASLR

CERT
Format String Attacks

• Another way to illegally control program values.

• Uses flaws in the design of `printf()`:

```
printf( "%s%ld", s, x );
```
printf() Operation

```
printf("%s: %d, %x", s, x, y);
```

Format Strings

foo()

<table>
<thead>
<tr>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>s</td>
</tr>
</tbody>
</table>

format string ptr

printf()
Attack 1: Read Any Value

What the code says:
printf( str );

What the programmer meant:
printf( "%s", str );

If str = '%%x%x%x%x%x%x%x%x'

Format Strings
secret key ptr
format string ptr
Attack 2: Write to Address

What the code says:
printf(str);

If str = ‘%d%d%d%d%n’

Format Strings
return address
format string ptr
Defenses

Never use `printf()` without a format string!

`FormatGuard`. 

Format Strings
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads
• Spreading Algorithms
• Case Studies
Piggybacking

Malicious code injected into a benign program or data file.

- Host file can be:
  - An executable.
  - A document with some executable content (Word documents with macros, etc.).
Piggybacking Executables

• Modify program on disk:

Variations:

• Jump to malicious code only on certain actions.

• Spread malicious code throughout program.
Piggybacking Executables

• Modify program on disk:

Variations:

• Jump to malicious code only on certain actions.

• Spread malicious code throughout program.
Piggybacking Executables

• Modify program on disk:

```asm
jmp evil_code
```

Variations:

• Jump to malicious code only on certain actions.

• Spread malicious code throughout program.
Piggybacking Executables

- Modify program on disk:

```
jmp evil_code
```

Variations:
- Jump to malicious code only on certain actions.
- Spread malicious code throughout the program.
Piggybacking Documents

• Documents with macros:
  Microsoft Office supports documents with macros scripted in Visual Basic (VBA).

• Macro triggered on:
  – Document open
  – Document close
  – Document save
  – Send document by email
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads

• Spreading Algorithms

• Case Studies

• Defenses
2 Payload

Target the interesting data:

- Passwords ⇒ Keylogger
- Financial data ⇒ Screen scraper
- User behavior ⇒ Spyware
- User attention ⇒ Adware
Keylogger Use

Web Branch Login

Member Number: [blank]
Password: [blank]

- Not a Web Branch user? [Request access]
- Test your browser to ensure that it meets Web Branch requirements.

Phishing Alert: IRS Refund Scam

More about UWCU’s Online Security

Zero Down & No PMI HOME LOAN

- No down payment—save immediately
- No Private Mortgage Insurance—save every month
- Low 5-year adjustable rate—save every year

Learn more  Apply now
Screen Scraper Use

Welcome to ING DIRECT USA!

All 2005 paper tax forms have been mailed. If you're eligible and haven't received yours yet, login and click on the 'Tax Info' icon to find out how to get it online now!

To login to your account, please complete the following three steps.

Step 1
Customer Number: [blank]

Step 2
First 4 digits of your Social Security Number: [blank]

Step 3
Use your mouse to click the numbers on the keypad that correspond to your PIN.

OR

Use your keyboard to type the letters from the keypad that correspond to your PIN.

What is this?

PIN: [blank]

Don't remember your Customer Number or PIN?
More Payload Ideas

Victim machines are pawns in larger attack:

– Botnets.
– Distributed denial of service (DDoS).
– Spam proxies.
– Anonymous FTP sites.
– IRC servers.
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads

• Spreading Algorithms

• Case Studies

• Defenses
3 Spreading Methods

Depends on the attack vector:

Email-based
  ⇒ need email addresses

Vulnerability-based
  ⇒ need IP addresses of hosts running the vulnerable service

Piggybacking
  ⇒ need more files to infect
Spreading through Email

- HTML files (from cache)
- Windows Address Book
- Outlook Express folders
- Outlook folders

Malware

Internet
Vulnerable Target Discovery

Need to find Internet (IP) addresses.

• **Scanning:**
  - Random
  - Sequential
  - Bandwidth-limited

• **Target list:**
  - Pre-generated
  - Externally-generated $\Rightarrow$ Metaserver worms
  - Internal target list $\Rightarrow$ Topological worms

• **Passive:** Contagion worms
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads

• Spreading Algorithms

• Case Studies
Types of Malicious Code


• Virus
  Self-replicating, infects programs and documents.
  e.g.: Chernobyl/CIH, Melissa, Elkern

• Worm
  Self-replicating, spreads across a network.
  e.g.: ILoveYou, Code Red, B(e)agle, Witty
Types of Malicious Code

• Trojan
  – Malware hidden inside useful programs
    e.g.: NoUpdate, KillAV, Bookmarker

• Backdoor
  – Tool allowing unauthorized remote access
    e.g.: BackOrifice, SdBot, Subseven
Types of Malicious Code

• Spyware
  – Secretly monitors system activity
  e.g.: ISpynow, KeyLoggerPro, Look2me

• Adware
  – Monitors user activity for advertising purposes
  e.g.: WildTangent, Gator, BargainBuddy
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads

• Spreading Algorithms

• Case Studies: Sobig
The **Sobig** Worm

- Mass-mailing, network-aware worm
- Multi-stage update capabilities

<table>
<thead>
<tr>
<th></th>
<th>Launch</th>
<th>Deactivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sobig.A</td>
<td>9 Jan. 2003</td>
<td>-</td>
</tr>
<tr>
<td>Sobig.B</td>
<td>18 May 2003</td>
<td>31 May 2003</td>
</tr>
<tr>
<td>Sobig.C</td>
<td>31 May 2003</td>
<td>8 June 2003</td>
</tr>
<tr>
<td>Sobig.D</td>
<td>18 June 2003</td>
<td>2 July 2003</td>
</tr>
</tbody>
</table>
Sobig: Attack Vector

• E-mail

From: big@sobig.info

Subject: support@microsoft.com bill@microsoft.com admin@support.com support@yahoo.com

• Compressed executable attachment with renamed extension.
• Later: attachment in ZIP file.

• Network shares
Sobig: Payload

- **1st stage:** Backdoor (Lala) & keylogger
- **2nd stage:** Proxy (WinGate)

Geocities web page

Trojan web server
Sobig: Payload

Hacked DSL/cable hosts

Trojan web server
Sobig: Spreading Algorithm

- E-mail addresses extracted from files on disk.

- Network shares automatically discovered.
Sobig.F in Numbers
Outline

• Attack Vectors:
  ▪ Social Engineering
  ▪ Vulnerability Exploitation
  ▪ Piggybacking

• Payloads

• Spreading Algorithms

• Case Studies: Sobig, Blaster
The **Blaster** Worm

- Multi-stage worm exploiting Windows vulnerability

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>2003: July</th>
<th></th>
<th>August</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>17</td>
<td>25</td>
<td>31</td>
<td>11</td>
</tr>
</tbody>
</table>

- Microsoft releases patch
- CERT advisory
- LSD Research exploit released
- Metasploit refined exploit
- Blaster appears

- 13
- 15
- 17
- 19

- FRB Atlanta, MD DMV, BMW
- Scandinavian bank closes all 70 branches
- 1.2 million hosts infected
Blaster: Attack Vector

- Uses a Microsoft Windows RPC DCOM vulnerability.

- Coding flaw:
  1. The RPC service passes part of the request to function `GetMachineName()`.
  2. `GetMachineName()` copies machine name to a fixed 32-byte buffer.
Blaster: Attack Vector
Blaster: Attack Vector

Exploit

“tftp GET msblast.exe”
Blaster: Attack Vector

1. Exploit
2. “tftp GET msblast.exe”
3. “GET msblast.exe”
Blaster: Attack Vector

1. Exploit
2. "tftp GET msblast.exe"
3. "GET msblast.exe"
4. C:\__
Blaster: Attack Vector

1. Exploit
2. "tftp GET msblast.exe"
3. "GET msblast.exe"
4. "start msblast.exe"
5. "start msblast.exe"
Blaster: Payload

- Worm installs itself to start automatically.
- All infected hosts perform DDoS against windowsupdate.com.
  - SYN flood attack with spoofed source IP,
    Aug 15 → Dec 31 and after the 15th of all other months.
Blaster: Effect on Local Host

- RPC/DCOM disabled:
  - Inability to cut/paste.
  - Inability to move icons.
  - Add/Remove Programs list empty.
  - DLL errors in most Microsoft Office programs.
  - Generally slow, or unresponsive system performance.
Blaster: Spreading Algorithm

• Build IP address list:
  40% chance to start with local IP address.
  60% chance to generate random IP address.

• Probe 20 IPs at a time.

• Exploit type:
  80% Windows XP.
  20% Windows 2000.
Blaster: Infection Rate
Future Threat: Superworm

“Curious Yellow: the First Coordinated Worm Design” – Brandon Wiley

- Fast replication & adaptability:
  - Pre-scan the network for targets.
  - Worm instances communicate to coordinate infection process.
  - Attack vectors can be updated.
  - Worm code mutates.
Conclusions

- Vulnerabilities left unpatched can and will be used against you.

- Attackers are more sophisticated.

- Need to understand the attackers’ perspective.
Malicious Code
for Fun and Profit

Mihai Christodorescu
mihai@cs.wisc.edu
23 March 2006