Project 3 (Exploits: Buffer Overflows, Pathname Attacks, and SQL Injections)

CS356

Colorado State University

November 12, 2015
Contents

1. The Big Picture
2. Buffer Overflow
3. Pathname Attacks
4. SQL Injection
5. Deliverables
The Big Picture

The Story
You are the fictitious system admin for the company FrobozzCo (Don’t apply for a job here, you might not get a reply). Your company has recently seen some security breaches. Your boss William H. Flathead III (Has no link to the last king of Scotland) wants you to investigate these breaches and report the following in general.

- How did the breach happen? What was/were the flaws in servers systems/source codes that FrobozzoCo was using?
- How do you suggest we fix them?

In order to solve the problem, you must exploit the found vulnerabilities and come up with solutions. To do so you need to:

1. Create a Deter Experiment, like last time.
2. Log in to the Deter Node (you have only one this time). This node is the FrobozzoCo server.
3. Follow instructions on the project web-page soon to be posted at (http://www.cs.colostate.edu/~cs356/), to exploit the vulnerabilities.
4. Come up with solutions.
Buffer Overflow

General Idea

Program
#define buffer1 "Lemmy Kilmister Motorhead" /*25 characters*/
int main(int argc, char **args)
{
    char buf1[10];
    char buf2[10];
    strcpy(buf1,buffer1);
    printf("buf1 at: \%p and buf2 at \%p\n", buf1,buf2);
    printf("buf1 content: \%s\n", buf1);
    printf("buf2 content: \%s\n", buf2);
    return 0;
}

Output
buf1 at: 0x7fffe6607830 and buf2 at 0x7fffe6607840
buf1 content: Lemmy Kilmister Motorhead
buf2 content: Motorhead

Example 1

1. Our compiler allocated memory addresses 0x7fffe6607830 and 0x7fffe6607840 to our buffers in the stack.
2. Since stack grows upwards but buffers grow downwards, in buf1 we could only write [0x7fffe6607840 - 0x7fffe6607830] which is 16 bytes safely without disturbing buf2 i.e from ‘L’ in “Lemmy Kilmister Motorhead” to the space before “Motorhead” (Fig. 1).
3. We ended up writing 25 bytes, i.e. 9 extra bytes for “Motorhead”. This started writing from 0x7fffe6607840 i.e. beginning of buf2 and upwards ending with a \0 null byte at 0x7fffe6607848.
4. When we printed, buf2 it printed “Motorhead” as that is what was written in there, although we never wanted to write anything in there.

Figure 1: Stack
Buffer Overflow

General Idea contd

Program
#define buffer1 "Lemmy Kilmister Motorhead" /*25 characters*/
void do_nothing(char *data)
{
    char buf2[10];
    printf("buf2 at \%p\n",buf2);
    strcpy(buf2,data);
}
int main(int argc, char **args)
{
do_nothing(buffer1);
    printf("Test Print\n");
    return 0;
}

Output

buf2 at 0x7fffc686b0f80
*** stack smashing detected ***: ./buffer_overflow terminated
Aborted (core dumped)

Example 2

1. In this case, function do_nothing is called from main. After do_nothing has completed its execution it is supposed to return control to main, which will then execute the printf statement.
2. Thus, before do_nothing starts doing anything its return address (the one which will lead it to main) is written on the stack.
3. Since stack grows upwards but buffers grow downwards, buf2 is allocated next in the frame of do_nothing. Assume it can safely write 16 bytes [0x7fffffff6607880 - 0x7fffffff6607870] before overwriting the return address (Fig. 2).
4. Unfortunately, the string written to buf2 is larger than 16 bytes and hence overwrites the return address. As a result, when do_nothing tries to return control to main, it encounters a garbage memory address and hence fails to proceed further-crashes as shown in output of Example 2.

Figure 2: Stack
Web-browsers and chat applications like Telnet/Netcat can be used to generate HTTP request packets.

The most important fields for this project is the Get field which denotes the file you want to get from the web-server.

The web-server then processes this Get request and returns the file (most often HTML files) to you for viewing.

The web-browser then displays this file in a formatted way. Telnet will merely show you the entire HTML file.
Buffer Overflow

What you need to do

1. Follow instructions on the project web-page to locate the web-server code on the Deter Machines.
2. Start it following instructions.
3. Use Telnet to check if it replies to correct HTTP requests.
4. Observe how the server processes the request once it receives it (A good understanding of C programming language is required here).
5. Find a flaw in the server code which can be exploited to cause buffer overflow. This would be similar to the Example 2 shown in the previous slides. Remember we are looking for Stack Buffer Overflows here.
6. Send a crafted HTTP request to the server using Telnet/Netcat (Read these documentations on the project page).
7. Crash the server and then send your exploit script to us along with how you would correct the server code.
Pathname Attacks

CGI scripts

- Usually written in Perl.
- Performs OS level tasks for the web-server.
- Returns response for to the server which is then served back to the client.
- A typical cgi query looks like this:

  ```
  www.whatever.org/some_path../cgi - bin/some.cgi.cgi? some_key = some_value & some_key = some_value
  ```

- Your Frobozzo CGI reads *memos* written by users from your organization. These memos are kept under directories which are created by regular company employees via ssh-ing into the server machine. The memo file name to be read is passed via a parameter of the CGI query string.
Pathname Attacks

What else do you need to know

Pathnames

- You can move into new directories in two ways:
  - By specifying the absolute pathname “cd /home/user/path”.
  - By specifying the relative path name from a folder. Eg. “cd path” from “/home/user”.

- You can go one level back from a directory using “..”. Eg. to back to “/home/user” from “/home/user/path” type “cd ..”. Multiple “..”s can be used to step back multiple levels.

File Permissions

- Discretionary access control.
- File permissions are generally: RWX - Read, Write, Execute.
- Three subjects are generally attributed the permissions: owner, group, others.
- Eg. “-rwxr-xr-x 2 <owner_name> <group_name> 4096 Nov 12 00:13 file.sh”
  Here owner has read,write and execute permission. A particular group has read and execute permission and others have read and execute permissions on the file “file.sh”.

Setuid

- Special permission bit which allows a random user to execute a file as the owner.
- Expressed using ‘s’ instead of ‘x’ in the permission.
- If permission on “<file.sh>” is “-rwxr-xr-x”, then if others execute this file, it will run with owner’s permission and is allowed to perform any action that the owner can perform.
Pathname Attacks

What you need to do

- You have noticed that someone is reading the privileged “/etc/shadow” which only root is allowed to read.
- The question is if no user has root privilege on this server machine, then how are they reading it?
- Something must be wrong with the CGI script. Maybe it somehow ends up reading the “/etc/shadow” file and showing it to regular users. But HOW?
- Follow the instructions in the project web-page to locate the cgi code.
- Find a vulnerability in the code using the above learned steps.
- Exploit the vulnerability and send us a report of how you read the “/etc/shadow” file using the cgi-script, what allowed you to read this, and how you would fix it.
**SQL injection**

SQL

![Database Table](image)

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jim</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>Peter</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>Jack</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>43</td>
</tr>
</tbody>
</table>

*Figure 4: Database Table (Name_Table)*

- **SQL = Structured Query Language**
- Used to query tables/relations stored in databases.
- Processed by database engines and answers returned to user.
- *Select * from Name_Table where Id=1 and Name = ”Jim”;
- Returned Answer: Row 1 from Name_Table (Fig. 4).
SQL injection

SQL and Php Login Scripts

In this example, since no records are found pertaining to the username and password the database returns 0 records. The Php backend knows this is an improper login attempt and lets the user know of it.
The evil user now sends an evil username: “Bon Scott' OR 1=1 –”. The database unaware of the malicious attempt returns all 4 rows and the PHP backend happily allows the user because it found matches in the database for the evil users username and password. But what trick did the user play? He entered a cleverly crafted query and here is what the query means to the database:

- The single quote (') at the end of “Bon Scott” means the attacker closed the quotation (Fig. 5). This makes the part of the query as username = 'Bon Scott'.
- The OR tells database either evaluate condition 1 which is username='Bon Scott' OR evaluate condition 2 which is 1=1.
- The 1=1 is true for all records. Thus the database returns success on all occasions.
- The “–” at the end tells the database ignore the rest of the query. Essentially the “passwords='backinblack’ condition is never even evaluated.

Because of this clever query the database returns all rows and the because there is atleast one match the user is allowed entry. CLEVER and EVIL!!.
SQL injection

What you need to do

- FrobozzoCo has been subjected to some sort of SQL injection.
- Follow the project description to find the php script which is used for login purposes.
- Cleverly craft SQL injections to solve the requirements as stated in the project description.
- Write a memo describing how you performed the attacks and how you would want to solve it.
Deliverables

**Delivery Date**
5pm on Friday, December 4th, 2015.

**Deliverables**
Check project page for deliverables

**What can go wrong**
Drink some good quality Espresso and listen to Lynyrd Skynyrd if you feel lost. Otherwise you can contact ME (Subhojeet) for any help even outside my office hours.

**Grading Policy** You will be graded on the correctness of your answers. The exploits you mention should work for you to fetch points. The solutions should be robust and should not succumb to clever attacks that your evil TA launches.

Thank You. Keep on rocking in the free world.