Homework 1
CS 577-Reverse Engineering and Application Analysis

Jason Wong

October 12, 2021

1 File Systems

The first thing to do is download the firmware that was shared, and run the `binwalk` command within REMnux. On the first attempt of running `binwalk` with the command `binwalk -e 10452057.bin`, I received the following error:

```
remnux@ubuntu:~/Documents$ binwalk -e 10452057.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
16027         0x3E9B          Copyright string: "Copyright (c) 2006 Polycom"
18290         0x4772          gzip compressed data, maximum compression, has original f
ile name: "vmlinux.bin", from Unix, last modified: 2018-09-12 12:50:59

WARNING: Extractor.execute failed to run external extractor 'jefferson -d 'jffs2-root'
'%e'': [Errno 2] No such file or directory: 'jefferson', 'jefferson -d 'jffs2-root' '%e
'' might not be installed correctly
1147332       0x1181C4        JFFS2 filesystem, big endian
```

This means that there is a JFFS2 file system within the firmware, and the jffs2 module dependency would need to be installed. The instructions for the dependency was found at the link:
`https://github.com/ReFirmLabs/binwalk/blob/master/INSTALL.md#dependencies`.
Following the instructions on the page, which is to execute these commands:

```
sudo pip install cstruct
git clone https://github.com/sviehb/jefferson
(cd jefferson && sudo python3 setup.py install)
```

Before running `binwalk -e 10452057.bin` again though, I had to delete the old extracted directory since it contained the incomplete file systems. After deleting though, I continued with the extraction. The `binwalk` was run again, and this time there were no errors, meaning that the file systems were successfully extracted. To confirm that the file systems were successfully extracted, change

directory into the root folder.

```
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$ ls
bin  etc   lib    linuxrc  opt   root  sys  usr
dev  home  lib64  mnt      proc  sbin  tmp  var
```

2 ELF Files

Before doing anything, I changed directory back to The goal of the assignment is to grab all the ELF files in the file system. One way to grab all of them is by executing `find . -exec file {} \; | grep ELF > elf.txt`, which will find all files in the current directory and sub-directories and run the `file` command on all of them, and then search for the "ELF" string and write all matching results to a text file called `elf.txt`.

```
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$ cat elf.txt
./usr/bin/mkfs.jffs: ELF 32-bit MSB executable, Motorola m68k, version 1 (SYSV), dynami
cally linked, interpreter /lib/ld.so.1, for GNU/Linux 2.6.10, not stripped
./usr/bin/nftl_format: ELF 32-bit MSB executable, Motorola m68k, version 1 (SYSV), dyna
mically linked, interpreter /lib/ld.so.1, for GNU/Linux 2.6.10, not stripped
./usr/bin/pwutil: ELF 32-bit MSB executable, Motorola m68k, version 1 (SYSV), dynamical
ly linked, interpreter /lib/ld.so.1, for GNU/Linux 2.6.10, not stripped
./usr/bin/gcp: ELF 32-bit MSB executable, Motorola m68k, version 1 (SYSV), dynamically
linked, interpreter /lib/ld.so.1, for GNU/Linux 2.6.10, not stripped
./usr/bin/pgatest: ELF 32-bit MSB executable, Motorola m68k, version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld.so.1, for GNU/Linux 2.6.10, not stripped
```

Since all I need are the paths to each of the ELF files in the file system, I wrote a small python script that I called `elf.py` to grab only the ELF file paths and write it to a separate temp file, as shown below:

```
def elf_search():
    os.system("find . -exec file {} \; | grep ELF > temp.txt")
    elf_list = open("elf_list.txt", "w")
    with open('temp.txt') as f:
        for line in f:
            elf_list.write(line.split(":")[0]+"\n")
    f.close()
    os.remove("temp.txt")
    elf_list.close()
```

The function called the same command as before as well as only grab the path to the ELF files and write them to `elf_list.txt`. Calling the function `elf_search()` in the python program and running `python3 elf.py` produces the file `elf_list.txt`.

A quick `wc -l elf_list.txt` shows that there are 121 binaries in the file system. The result text file after doing a `cat` on `elf_list.txt` will look like this showing only the first 5 binaries found:

```
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$ cat elf_list.txt
./usr/bin/mkfs.jffs
./usr/bin/nftl_format
./usr/bin/pwutil
./usr/bin/gcp
./usr/bin/pgatest
```

3 Stack Canaries

To test whether a binary has stack smashing protection, one method is to check whether the binary contains the `__stack_chk_fail` function in the disassembly [1]. I wrote a short python function to do that within the same python file as before, as shown below:

```
def stack_canaries_check():
    canary_list = open("canary_list.txt", "w")
    with open('elf_list.txt') as f:
        for line in f:
            os.system("readelf -Ws " + line[:-1] +
                      " | grep __stack_chk_fail > canary.txt")
            if os.stat("canary.txt").st_size != 0:
                canary_list.write(line)
    os.remove("canary.txt")
    canary_list.close()
```

The function will do a `readelf -Ws` on every file in the `elf_list.txt` and then check if the string `__stack_chk_fail` is in that result and write it to a temporary text file called `canary.txt`. If there is a matching result, meaning that there is data in `canary.txt` and the size is not 0, then that means `grep` was able to find a match on that file. In other words, there is stack smashing protection. The path to that file is then written to `canary_list.txt`.

Calling the function `stack_canaries_check()` in the python program and running `python3 elf.py` produces the file `canary_list.txt`. The result of this execution shows that there are no binaries that have stack smashing protection after doing a `cat` on `canary_list.txt`.

```
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$ cat canary_list.txt
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$
```

4 Fortify Source

The list of functions that `FORTIFY_SOURCE` checks for are:

`memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf, gets`

[2]. Similar to what I did to check for stack canaries, I wrote another python function to check for specific strings, with the source code shown below:

```
def fortify_source_check():
    commands_list = ["__memcpy_chk", "__mempcpy_chk", "__memmove_chk",
    "__memset_chk", "__strcpy_chk", "__stpcpy_chk", "__strncpy_chk",
    "__strcat_chk", "__strncat_chk", "__sprintf_chk","__vsprintf_chk",
    "__snprintf_chk", "__vsnprintf_chk", "__gets_chk"]
    commands_file = open("commands.txt", "w")
    for command in commands_list:
        commands_file.write(command+"\n")
    commands_file.close()

    fortify_list = open("fortify_list.txt", "w")
    with open('elf_list.txt') as f:
        for line in f:
            os.system("readelf -Ws " + line[:-1] +
                      " | grep -f commands.txt > fortify.txt")
            if os.stat("fortify.txt").st_size != 0:
                fortify_list.write(line)
    os.remove("commands.txt")
    os.remove("fortify.txt")
    fortify_list.close()
```

Similar to the `stack_canaries_check()` function, while iterating through every file in `elf_list.txt`, this will instead check if any of the strings in `commands.txt` are found in the result of `readelf -Ws`, and then write that result to a temporary file called `fortify.txt`. Again, if there is a matching result, that means the binary was compiled with the `FORTIFY_SOURCE` macro.

Calling the function `fortify_source_check()` in the python program and running `python3 elf.py` produces the file `fortify_list.txt`. The result of this execution shows that there is only one file that has `FORTIFY_SOURCE` checks after doing a `cat` on `fortify_list.txt`.

```
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$ cat fortify_list.txt
./lib/libc-2.3.6.so
remnux@ubuntu:~/Documents/_10452057.bin.extracted/jffs2-root/fs_1$
```

To show that the one file has `FORTIFY_SOURCE` checks, here is a line in the `readelf -Ws` that calls `__memcpy_chk`:

```
 4985: 000af624   192 FUNC    GLOBAL DEFAULT   11 __mempcpy_chk
```

5 Conclusion

After doing both hardening checks (`Stack Smashing Protection & FORTIFY_SOURCE`) and seeing that there is only one file that has `FORTIFY_SOURCE` checks. My assumption is that this firmware came from a device that is really old, most likely before StackGuard was implemented as well as before and before buffer overflow prevention was a concern.

6 Sources

[1] Stack Canaries: https://access.redhat.com/blogs/766093/posts/3548631
[2] Fortify Source: https://access.redhat.com/blogs/766093/posts/1976213

7 Source Code (elf.py)

```
import os

def elf_search():
   os.system("find . -exec file {} \; | grep ELF > temp.txt")
   elf_list = open("elf_list.txt", "w")
   with open('temp.txt') as f:
       for line in f:
           elf_list.write(line.split(":")[0]+"\n")
   f.close()
   os.remove("temp.txt")
   elf_list.close()

def stack_canaries_check():
   canary_list = open("canary_list.txt", "w")
   with open('elf_list.txt') as f:
       for line in f:
           os.system("readelf -Ws " + line[:-1] +
                       " | grep __stack_chk_fail > canary.txt")
           if os.stat("canary.txt").st_size != 0:
               canary_list.write(line)
   os.remove("canary.txt")
   canary_list.close()

def fortify_source_check():
   commands_list = ["__memcpy_chk", "__mempcpy_chk", "__memmove_chk",
   "__memset_chk", "__strcpy_chk", "__stpcpy_chk", "__strncpy_chk",
   "__strcat_chk", "__strncat_chk", "__sprintf_chk","__vsprintf_chk",
   "__snprintf_chk", "__vsnprintf_chk", "__gets_chk"]
   commands_file = open("commands.txt", "w")
   for command in commands_list:
       commands_file.write(command+"\n")
   commands_file.close()

   fortify_list = open("fortify_list.txt", "w")
   with open('elf_list.txt') as f:
       for line in f:
           os.system("readelf -Ws " + line[:-1] +
                       " | grep -f commands.txt > fortify.txt")
```

```
            if os.stat("fortify.txt").st_size != 0:
                fortify_list.write(line)
    os.remove("commands.txt")
    os.remove("fortify.txt")
    fortify_list.close()

elf_search()
stack_canaries_check()
fortify_source_check()
```