Executive Summary –Return Oriented Programming Exploit on NTP Daemon in MAC OS based on CVE-2014-9295

Chanakya Gaur, *cgaur1,* Sagar Wani*, swani1,* Prashanth Venkateswaran, *pvenkat7*

***Abstract*— The Network Time Protocol daemon (NTPD) is a program run by the operating system that maintains its system time by synchronizing with the time server using the Network Time Protocol. In OS X Mavericks, a buffer overflow was discovered with the code of NTPD which handles the control packets. If these control mode responses exceed the size of the buffer, they are fragmented. This vulnerability is listed as CVE-2014-9295 discovered by Stephen Röttger and has been patched by Apple. In this summary, we aim to use this vulnerability and exploit it with Return Oriented Programming(ROP) to escape the sandbox and obtain shell access.**

***Index Terms*— Buffer Overflow, Return Oriented Programming (ROP), MAC, NTPD, Vulnerability, Exploit**

I. INTRODUCTION

This summary explains how a Return Oriented Programming(ROP) exploit can be used on the NTPD vulnerability of MAC OS X Mavericks listed as CVE-2014-9295 discovered by Stephen Röttger. The Network Time Protocol is a networking protocol used to synchronize clocks between systems using packet-switched and variable-latency data networks. Network Time Protocol Daemon is the system program that uses NTP to ensure system time to be synchronized with the server time.

Return Oriented Programming(ROP) is a technique in which the attacker arbitrarily changes the control flow of the compromised program and can change the behavior of the program without the need to inject any code. This is carried out using short instruction sequences all of which end with a return statement. These instruction sequences are called "gadgets". The advantage of ROP is that it immune to the Data Execution Prevention as ROP does not inject malicious code but uses gadgets to manipulate return addresses. The addresses of gadgets are stored together in a buffer called a Chain. This Chain is replaced onto the return pointer if the vulnerable program to exploit the system. For this exploit, we are assuming that we have the NTPD authentication key and know the ASLR slide offset as well. In the next section, we will talk about the setup and requirements to execute this exploit.

II. SETUP OF EXECUTION ENVIRONMENT

For this exploit, we are using the vulnerable machine OS X Mavericks and OS X Sierra as the attacking machine. Below we describe how both systems should be configured.

A. MAC OS X Mavericks (OS X 10.9)

Firstly, do not connect Mavericks to the internet. Apple patches this vulnerability immediately without any consent from the user. Now, enable SSH server and using secure copy (SCP) command line tools to the virtual machine from the host. Install command line tools i.e. LLDB. LLDB is a software debugger which is built as a set of components which use libraries of the LLVM project such as Clang expression parser and LLVM disassembler. Now the NTPD authentication should be setup and verify if the machine can ntpq connect from Sierra to Mavericks ntpd.

B. MAC OS X Sierra (OS X 10.12)

To setup Sierra, install XCode. XCode is a development environment for MAC OS that contains a suite for developing software for Apple product's operating systems. Once XCode is installed, build OpenSSL. Also, disable Gatekeeper and SIP. Gatekeeper is a feature in MAC which checks for developer ID of the application being installed. It does not allow the application to run if the application is tampered with. Gatekeeper should hence be disabled. Session Initiation Protocol(SIP) is a communication protocol for controlling and signaling of multimedia communication sessions. With this done, Sierra is now setup.

Now that both the vulnerable machine and the attack machine are setup we will see in the next section, how the exploitation process is carried out.

III. EXPLOITATION

The exploitation process can be divided into 6 steps:

A. Authenticate
B. Determine ASLR slides
C. Remote Write

This work was performed as part of the assigned Project work in EN.650.660 Software Vulnerability Analysis at the Johns Hopkins University – Information Security Institute and was submitted on December 8, 2017

Sagar Wani, Graduate student, Johns Hopkins University (e-mail:swani1@jhu.edu).
Chanakya Gaur, Graduate student, Johns Hopkins University (e-mail:cgaur1@jhu.edu).
Prashanth Venkateswaran, Graduate student, Johns Hopkins University (e-mail:pvenkat7@jhu.edu).

D. Stack pivot to gain NTPD Execution
E. Escape the Sandbox
F. Launch Shell

A. *Authenticate*

Ntp.conf can be used to specify keys and assign IDs to them. These key IDs can be assigned to different roles such as requestkey and controlkey. The requestkey is used to authenticate private mode packets and control key is used to control mode packets. We would need a controlkey to be able to send out configuration rquests but using a requestkey in private mode would set the controlkey ID to a specific value which would be sufficient. [1]

If no key is specified, a random 31-bit key is generated. A brute force using 2^31 packets with 68-byte payload each could be used in that case. The random key, however, is generated by a random number generator which is seeded with a 32-bit value which we can get through standard time synchronization requests. [1]

However, since this project is an academic exercise, we have assumed that the keys are known.

B. *Determine ASLR slides*

On examination, we know that the system library address has two null bytes while the binary address starts with three null bytes. Thus, we could get address of the system library overwriting the GOT entry with some length field.

However, for our dummy vulnerable program, we have disable the ASLR using the -Wl and -no_pie.

C. *Remote Write*

The internal buffer is used for read and writing to NTPD. The vulnerability is that there is no size checking when memmove is performed. Additionally, the binary is compiled with the buffer located at a lower address than datapt. The current index pointer datapt is not kept locally and is reloaded from memory on each use.

To exploit this, we need two variables. The first variable will write to buffer and several locations following and the lower three bytes of datapt. The second variable write will be written to wherever the datapt points. We have successfully corrupted GOT entry.

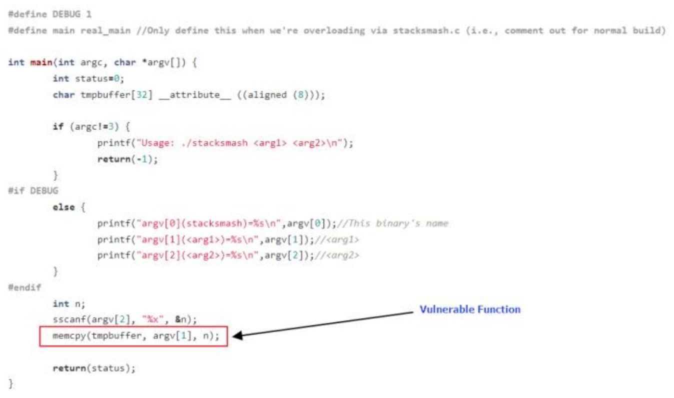
D. *Stack pivot to gain NTPD Execution*

Stack pivoting is a common technique used by vulnerability exploit to chain ROP gadgets. In stack pivoting, attackers can pivot from the real stack to a fake stack which could be an attacker-controlled buffer, such as the heap, then attackers can control the program execution. For example, this is achieved by controlling data pointed to by RSP (stack pointer registers), such that each ret instruction results in incrementing RSP and transferring execution to the next address chosen by attackers.

This is important primarily because every exploit is not a simple buffer overflow where the attacker-controlled stack is linear. Generally, the stack frame of the vulnerable function is located elsewhere. This makes it essential to be able to find gadgets that are able to control the stack pointer simultaneously keeping instruction pointer control by chaining in more gadgets. Stephen Rottger mentions that this can be achieved by a few gadgets to the buffer such that a ROP chain can be inserted. We will be demonstrating that a simple ROP chain is possible on OS X.

Learning ROP on OSX:

For the purposes of simplicity, we have created a simple vulnerable program with a basic stack overflow vulnerability.



```
#define DEBUG 1
#define main real_main //Only define this when we're overloading via stacksmash.c (i.e., comment out for normal build)

int main(int argc, char *argv[]) {
        int status=0;
        char tmpbuffer[32] __attribute__ ((aligned (8)));

        if (argc!=3) {
                printf("Usage: ./stacksmash <arg1> <arg2>\n");
                return(-1);
        }
#if DEBUG
        else {
                printf("argv[0](stacksmash)=%s\n",argv[0]);//This binary's name
                printf("argv[1](<arg1>)=%s\n",argv[1]);//<arg1>
                printf("argv[2](<arg2>)=%s\n",argv[2]);//<arg2>
        }
#endif
        int n;
        sscanf(argv[2], "%x", &n);
        memcpy(tmpbuffer, argv[1], n);

        return(status);
}
```

Tools Used that are similar to Linux Systems but for Mac OSX, LLDB is the debugging tool similar to gdb and uses similar commands for run and break. LLVM or clang is the compiler that is used to compile the code and creates a mach-o binary.

```
$cc -g -fno-stack-protector -D_FORTIFY_SOURCE=0 stacksmashtester.c stacksmash.c -Wl,-no_pie -o stacksmash
```

Once this is compiled, we can run this in lldb like this: -
(lldb)r stacksmash

Using the "(lldb)image list", we can find the base address of the libsytem_c.dylib. We also used the "#objdump -h <binary>" to find the data segment base address of the binary.

Once, these are figured out, the next step would be figure out the offsets of important gadgets. For this we tried a few open source tools, namely, ROPGadget and Ropper2. But unfortunately, due to the volatility of this field, the offsets given by these tools were incorrect or incompatible with Mach-O shared libraries (dylibs). Instead, we used IDA Pro and loaded the libsystem dylib and resorted to manually finding certain gadgets for the purpose of this demonstration. We found the gadget that Stephen Rottger used for stack pivoting and one other gadget. We created the stack such that the appropriate gadget addresses in the text segment are coinciding with the corresponding return instruction of the original program and all subsequent gadget's return instruction. We must ensure to keep the track of the stack pointer and also the values that are getting popped based on the instruction that is being run in the gadget.

Now that we have confirmed that a ROP chain is possible in MAC OS X by the demonstration, we can theorize with certainity that this can be used to exploit the ntpd daemon.

We can use this for stack pivot by placing a the first gadget while corrupting the GOT entry of strlen or free in the vulnerable ctl_putdata() function in the ntp_control.c file. Simultaneously, this gadget must be controlling the stack pointer such that we can place the remaining gadgets in our controlled global buffer to execute the remaining stages: -

1) Escape the Sandbox
2) Launch a Shell

These will be described in the following sections.

E. Escape the Sandbox

Escaping the sandbox is not in the scope of this project. However, there are a few ways to escape the sandbox such as effective_audit_token which is an XPC type confusion sandbox escape and root pipe which is a hidden backdoor API to root privileges in Apple OS X. More information on this can be found from the links provided in the presentation.

F. Launch Shell

After escaping the sandbox, the aim of the remote attacker has to be able to gain shell access. The attacker has to gain access to the vulnerable system to be able to actually exploit it. Thus, an ROP chain can be used to gain shell using the appropriate gadgets. The attack is successful when the attacker has shell access.

IV. Future Work

If the stack pivot is successsfully created to exploit the ntpd daemon, it can be used to gain the shell access. With the shell access with limited privileges, privilege escalation techniques like effective_audit_token XPC type confusion and rootpipe hidden backdoor API can be further used to escape the sandbox. This can be read in more detail using below links:

- https://bugs.chromium.org/p/project-zero/issues/detail?id=130&redir=1
- https://truesecdev.wordpress.com/2015/04/09/hidden-backdoor-api-to-root-privileges-in-apple-os-x/

V. Conclusion

Based on this experiment we were able to demonstrate the use of Return Oriented Programming to create the stack pivots in order to exploit the buffer overflow vulnerabilities in MACHO binaries. The purpose of this experiment was to create a simple model of a vulnerable program and it's exploit to show stack pivoting which can be further used as a base to create an exploit for the MacOS ntpd daemon. This research paper provides an essential description that shows mechanisms like Data Execution Prevention are not very useful when it comes to preventing buffer overflow ROP attacks.

This demonstration was done on a 64-bit machine architecture with DEP turned on. Mechanisms like ASLR and Stack Protectors have to be turned off as they are effective in preventing such attacks. With the help of this experiment, we were able to demonstrate various MacOS tools that are used for compilation and debugging MACHO binaries as well as searching the specific ROP gadgets using IDA Pro.

VI. APPENDIX

We have setup a GitLab which contains the code as well as documentation for this project. The link for the GitLab is: http://sva-gitlab.mssi-lab.isi.jhu.edu/swani1/SVA-Final-Project

VII. References

1.The Internet, “Over The Air - Vol. 2, Pt. 3: Exploiting The Wi-Fi Stack on Apple Devices”, Google Project Zero

2. Over The Air - Vol. 2, P. (2017). Over The Air - Vol. 2, Pt. 3: Exploiting The Wi-Fi Stack on Apple Devices. [online] Googleprojectzero.blogspot.com. Available at: https://googleprojectzero.blogspot.com/2017/10/over-air-vol-2-pt-3-exploiting-wi-fi.html

3. GitHub. (2017). JonathanSalwan/ROPgadget. [online] Available at: https://github.com/JonathanSalwan/ROPgadget

4. Hex-rays.com. (2017). Welcome. [online] Available at: https://www.hex-rays.com/

5. GitHub. (2017). sashs/Ropper. [online] Available at: https://github.com/sashs/Ropper

6. Levin, J. (n.d.). *OS internals.

7. Nvd.nist.gov. (2017). NVD - CVE-2014-9295. [online] Available at: https://nvd.nist.gov/vuln/detail/CVE-2014-9295

8. GitLab. (2017). Sign in. [online] Available at: https://gitlab.com/reubenajohnston/MHI/wikis/home