**Local Privilege Escalation for macOS 10.12.2 via mach\_voucher and XNU port Feng Shui**

Author: Min (Spark) Zheng @ Team OverSky

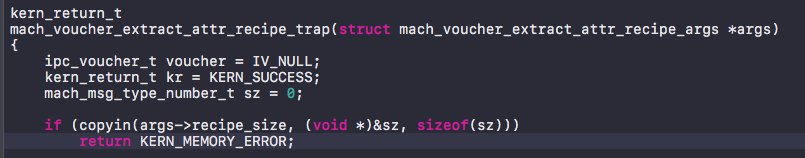
0x00 Introduction

From yalu 10.2, we could learn lots of new exploit techniques especially the XNU port feng shui and kpp bypass. In this article, we discuss the technique detail about XNU port feng shui and transform this technique to macOS to gain root privilege. Last but not least, the exploit source code could be downloaded at:

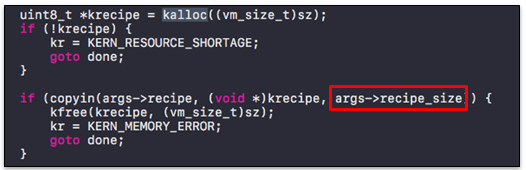
<https://github.com/zhengmin1989/macOS-10.12.2-Exp-via-mach_voucher>

0x01 Kernel heap overflow

To gain root privilege in macOS 10.12 through XNU port feng shui, we need a kernel vulnerability. In this article, we choose mach\_voucher heap overflow as an example. Mach\_voucher\_extract\_attr\_recipe\_trap() is a mach trap which can be called inside the sandbox. It's a new function added in iOS 10 and macOS 10.12. But, it has a terrible buffer overflow vulnerability.

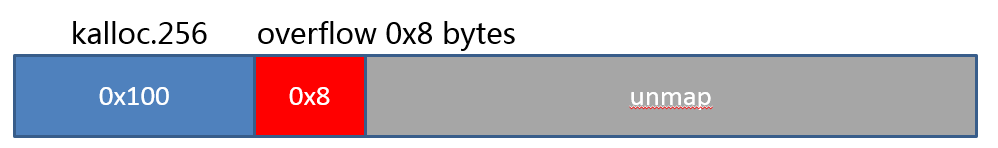


In the function, args->recipe\_size is a user mode pointer which points to an integer number. So mach\_voucher\_extract\_attr\_recipe\_trap() copies the size value from user mode through copyin().



The function then uses the sz value to allocate a memory block on the kernel heap. However, the developer forgot args->recipe\_size was a user mode pointer and then used it as a size value in copyin(). We know that user mode pointer could be larger than the sz value which will cause a buffer overflow in kernel heap.

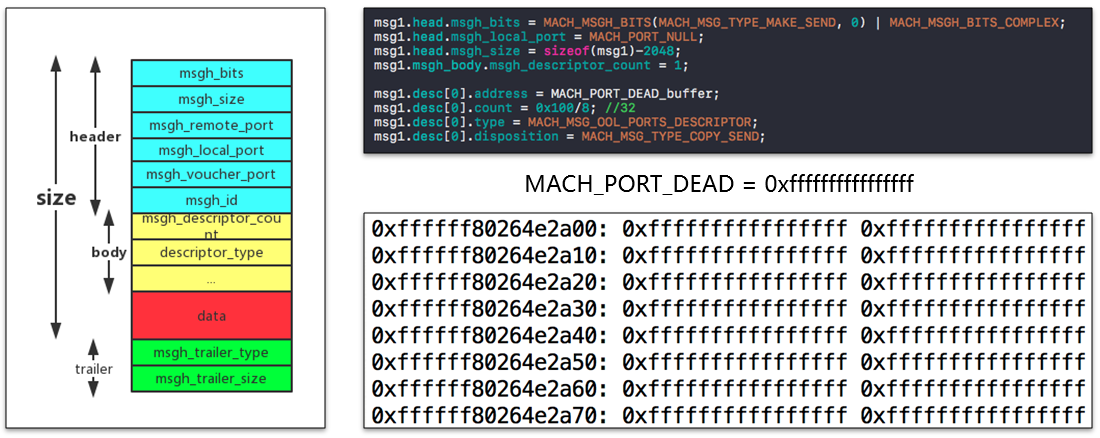
Note that we may not be able to control the address of the user mode pointer if we want to allocate a memory block on that address. But it’s not a problem. Copyin() function will automatically stop if it meets unmapped memory. So we could allocate a memory block on a high address and then unmap the rest memory block to control the overflow data.



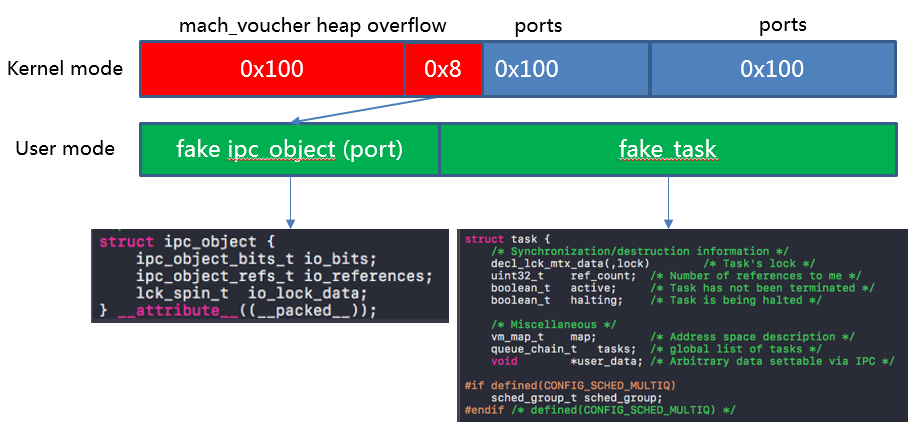
0x02 Heap feng shui through port

In iOS 10 and macOS 10.12, Apple added a new mitigation mechanism to check the freeing into the wrong zone attack, so we cannot use the classic vm\_map\_copy (changing vm\_map\_size) technique to do heap feng shui. In addition, Apple added a freelist randomization mechanism in iOS 9.2 and macOS 10.11, we cannot easily predict the location of reallocated memory block. To solve those problems, we need a new heap feng shui technique. In Yalu 10.2, qwertyoruiop used OOL\_PORTS to get the kernel task port which can be used to do arbitrary kernel read and write. This technique bypassed all mitigation mechanisms in XNU heap. In the rest of this section, we will discuss the detail of this technique.

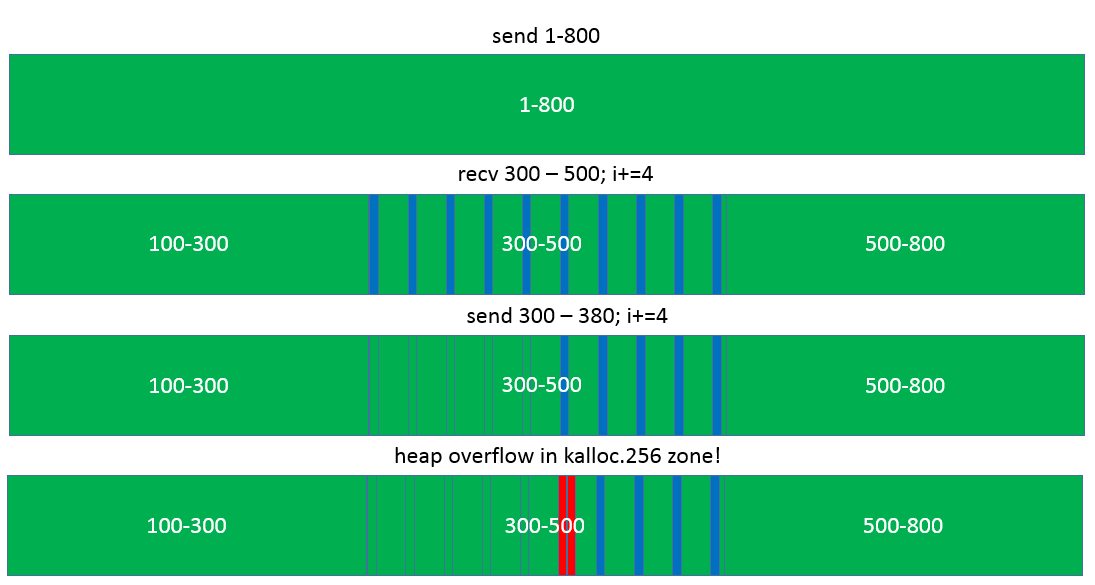
Mach msg is the most frequently used IPC mechanism in XNU and a lot of messages are transmitted through “complicated message”. Through the “complicated message” of MACH\_MSG\_OOL\_PORTS\_DESCRIPTOR msg\_type, we can transmit out-of-line ports to the kernel. For example, we sent 32 MACH\_PORT\_DEAD ool ports (32\*8 bytes=0x100 bytes) to the kalloc.256 zone of kernel.



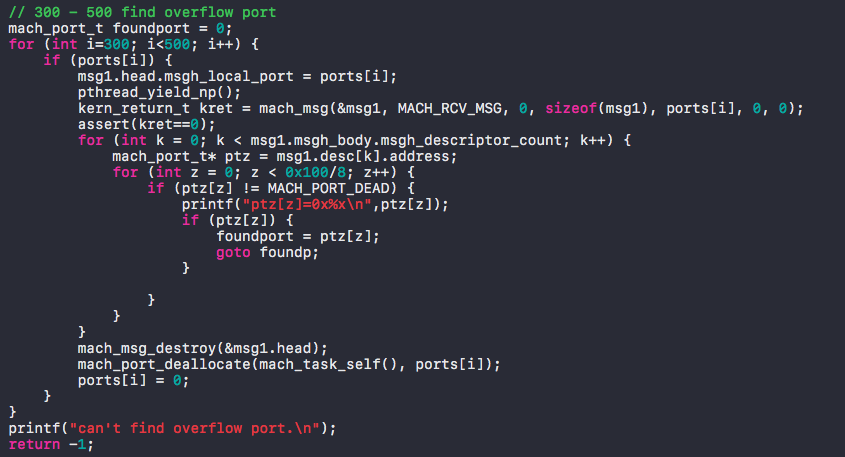
The ool ports saved in mach msg are ipc\_object pointers and the pointer can point to a user mode address. Therefore, we can use mach\_voucher vulnerability to overflow those pointers and modify one ipc\_object pointer to point to a fake ipc\_object in user mode. In addition, we could create a fake task in user mode for the fake port as well.



To insure overflowing the right ipc\_object pointer, we need to do some heap feng shui. Firstly, we send lots of ool ports messages to the kernel to insure the new allocated memory blocks are continuous. Then we receive some messages in the middle to dig some slots. Then we send some messages again to make the overflow point at the middle of the slots. At last, we use mach\_voucher to trigger the heap overflow at the overflow point.

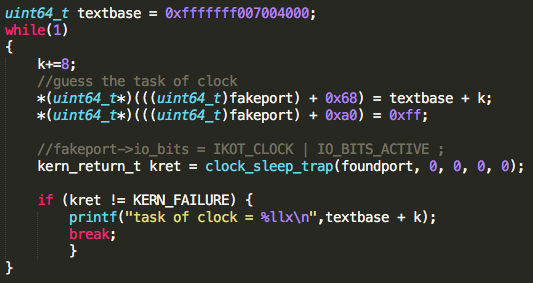


After overflow, we can receive the rest of mach messages to find the corrupted port (not MACH\_PORT\_DEAD port).

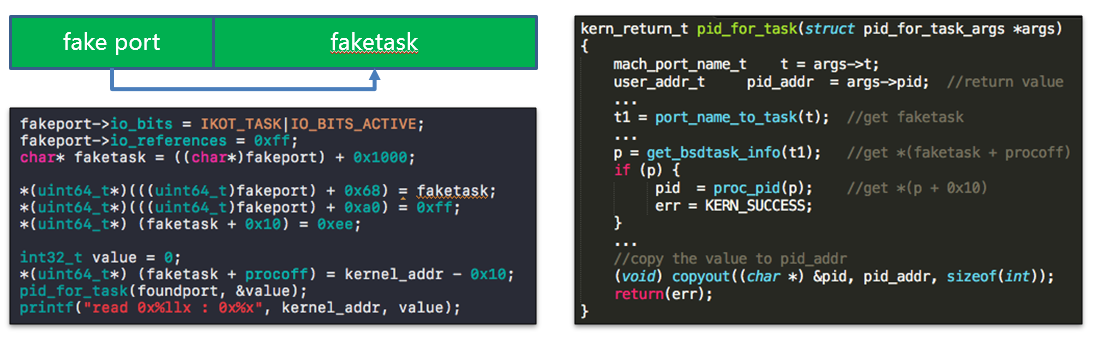


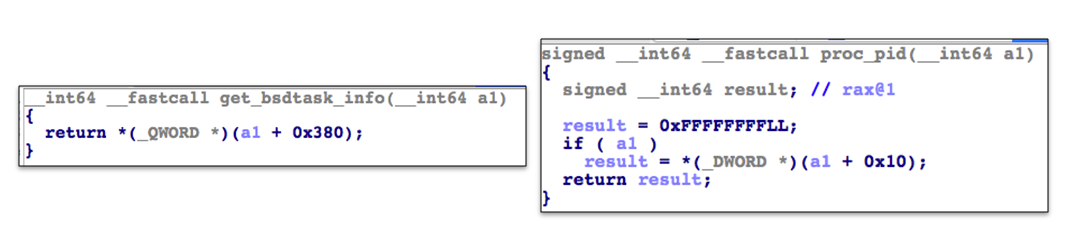
0x02 Arbitrary kernel read and write

Firstly, we set the io\_bits of the fake ipc\_object to IKOT\_CLOCK. So we can use clock\_sleep\_trap() to traverse the kernel to get the address of clock task in the kernel. This address will help us to find the kernel text base later.

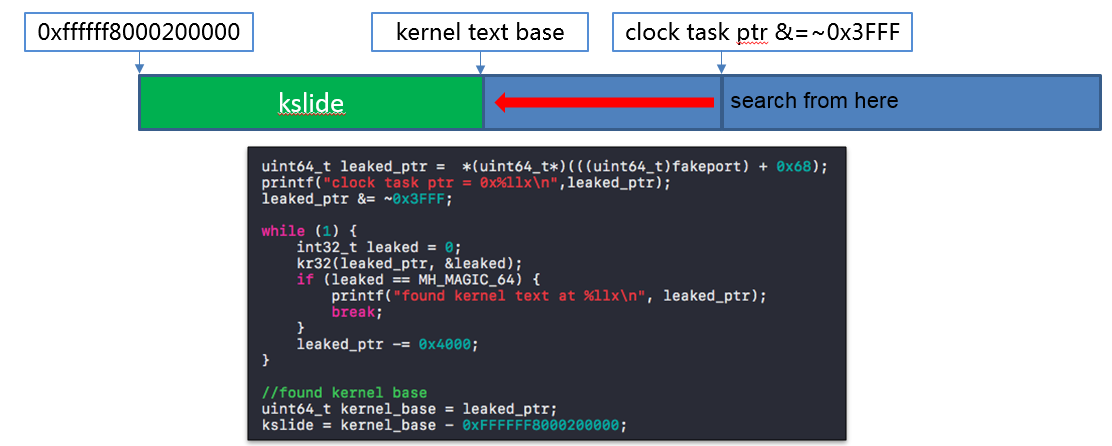


Then we set io\_bits of the fake ipc\_object to IKOT\_TASK and craft a fake task for the fake port. By setting the value at the faketask+0x380 (in arm64, it’s 0x360), we could read arbitrary 32 bits kernel memory through pid\_for\_task(). That’s amazing because the function doesn’t check the validity of the task, and just return the value of \*(\*(faketask + 0x380) + 0x10). So we could gain arability kernel read without any gadgets and ROP.

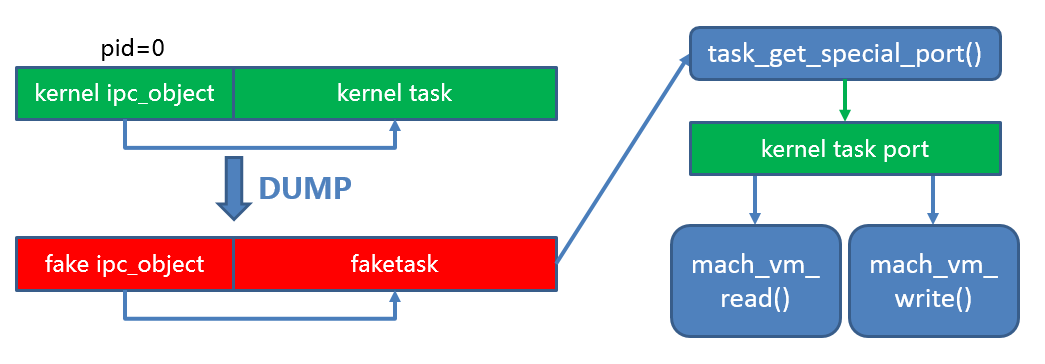




By using the leaked kernel address of clock task, we could search the magic number of kernel image in the memory to find the kernel slide.

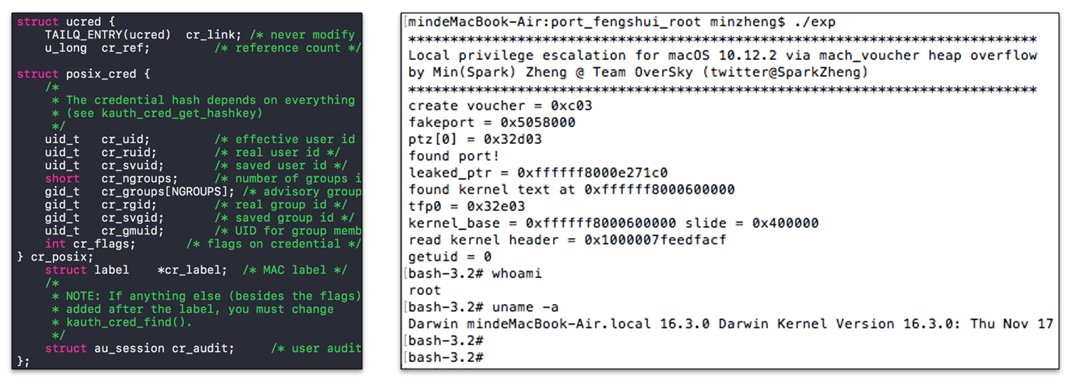


After getting the kernel base, we can traverse all the processes to find the kernel ipc\_object and kernel task. Then we dump this information to our fake ipc\_object and fake task. By using task\_get\_special\_port() to our fake ipc\_object and task, we could get the kernel task port. Note that the kernel task port is very powerful. It can be used to do arbitrary kernel read and write through mach\_vm\_read() and mach\_vm\_write().



0x03 Getting root

The credential information, the posix\_cred structure, for each process is stored in kernel memory. We need to find our process information first (from kernel base + allproc) and then find the posix\_cred structure data of our exploit process. After that, we use mach\_vm\_write() through kernel\_task\_port to set the cr\_ruid (real user id) value to 0 (which means root). Last but not least, we can use system(“/bin/bash”) to get the root shell!



0x04 Summary

In this article, we introduced how to use mach\_voucher heap overflow and port feng shui to achieve local privilege escalation on macOS 10.12.2. And the exploit source code can be downloaded at:

<https://github.com/zhengmin1989/macOS-10.12.2-Exp-via-mach_voucher>

0x05 Reference

1. Yalu 102: <https://github.com/kpwn/yalu102>

2. <https://bugs.chromium.org/p/project-zero/issues/detail?id=1004>