MONDAY, JANUARY 2, 2012

Details on Corona

Now that Corona was released by the iPhone Dev Team and the Chronic Dev Team, I can give details about how it works.

1. the user land exploit

Apple has fixed all previous known ways of executing unsigned binaries in iOS 5.0. Corona does it another way.

By the past, the trick security researchers used was to include the untethering payload as a data page (as opposed to a code page) in the Mach-O binary. The advantage of a data page was that the Macho-O loader didn't check its authenticity. ROP is used so that code execution happens without writing executable code but rather by utilizing existing signed code in the `dyld` cache. To have the ROP started by the Mach-O loader, they relied on different technics found by @comex, either :
- the interposition exploit
- the initializer exploit

Here is a detailed explanation of incomplete code sign tricks used before 5.0 : the iPhone wiki

In iOS 5.0, data pages need also to be signed by Apple for the loader to authenticate the binary. @i0n1c seems to be able to pass through these verifications though (he twitted). We may see this in the 5.1 jailbreak.

Thus, for Corona, I searched for a way to start unsigned code at boot without using the Mach-O loader. That's why I looked for vulnerabilities in existing Apple binaries that I could call using standard `launchd` plist mechanisms.

Using a fuzzer, I found after some hours of work that there's a format string vulnerability in the `racoon` configuration parsing code! `racoon` is the IPsec IKE daemon (http://ipsec-tools.sourceforge.net/). It comes by default with iOS and is started when you setup an IPsec connection.

Now you got it, Corona is an anagram of `racoon` :-) .

By the way, the exploitation of the format string vulnerability is different than what was done in 2001, check it out if you're interested !

For the jailbreak to be applied at boot, `racoon` is started by a `launchd` plist file, executing the command : `racoon -f racoon-exploit.conf`

`racoon-exploit.conf` is a large configuration file exploiting the format string bug to get the unsigned code started.

The format string bug is utilized to copy the ROP bootstrap payload to the memory and to execute it by overwriting a saved LR in the `racoon` stack by a stack pivot gadget.

The ROP bootstrap payload copies the ROP exploit payload from the `payload` file which is distributed with Corona then stack pivot to it. The idea is to escape from format strings as fast as possible, because they are CPU time consuming.

The ROP exploit payload triggers the kernel exploit.

2. the kernel exploit

The kernel exploit relies on an HFS heap overflow bug I found earlier. I don't know exactly what happens in the kernel code, I never figured it out exactly, I found it by fuzzing the HFS btree parser.

I just realized that it is a heap overflow in the zone allocator, so I started to try to mount clean, overflowed and payload images in a Heap Feng Shui way :-) And hey, that worked :p Thanks to @i0n1c for his papers on this subject. This helped me a lot. I may have given up without them.

The kernel heap overflow exploit copies 0x200 bytes from the `vnimage.payload` file to the kernel sysent replacing a syscall to a write anywhere gadget. Some syscalls (first 0xA0 bytes and the last 0x6 bytes) are trashed in the operation because I needed to respect the HFS protocol.

Thus, I restore them as fast as possible to get a stable exploit, then the write anywhere is used to copy the kernel exploit and jump to it.

The kernel exploit just patches the kernel security features, as usual. Nothing interesting there.

Happy New Year 2012 to you all, thanks a lot for the donations.

~pod2g

Source: http://www.pod2g.org/2012/01/details-on-corona.html