Security vulnerabilities on OS X

SeungJin Lee (aka beist)
그레이해쉬 대표
http://grayhash.com

About me

- SeungJin Lee (aka beist) – Interested in offensive security researcher
  - beist@grayhash.com / http://twitter.com/beist
- Founder of Grayhash Inc (http://grayhash.com)
- Ms-Phd course at at Korea University (A member of SANE Lab, CIST: Professor. SeungJoo Kim)
- Wins at hacking competitions
- Running Codegate/Secuinside CTF
- Speaking at security conferences: BH Vegas, CANSECWEST, BREAKPOINT, SYSCAN, AVTOKYO, HITCON, TROOPERS, EDSC, SECUINSIDE
- Tech advisor for SAMSUNG SDS Security Center

About this talk

- Not a very technical talk
  - Not a talk by OS X/iOS super l33ts like Stefan Esser
- But focused on how to approach find security vulnerabilities on OS X
  - And some bugs could work on iOS too as the codebase
  - Hobby research
- Could be useful for those who want to start research on Apple OS
- Will share my experience and demo some 0 days on the stage

About OS X

- It is called XNU and XNU is one of unix-like operating systems
- The core components can be divided
  - Mach
  - BSD
  - libkern
  - I/O Kit

Mach

- Developed at CMU originally
- To create a lightweight and efficient platform for OSes
  - Process and thread abstractions
  - Virtual memory management
  - Task scheduling
  - Interprocess communication and messaging

BSD

- There is the BSD layer on top of Mach
- Most important thing is it supports POSIX compatibility
  - The UNIX Process Model/Thread model
  - UNIX users and groups
  - Network stack
  - File system, device access

libkern

- libkern is C++ library
- To support C++ runtime
- Most drivers can be written in C++

IOKit

- Device driver framework
- Probably one of most interesting things in OS X/iOS for bug hunters
- For developers, it's really convenient if they can code in C++
  - Much more simplified and dev-time can be reduced
- For bug hunters, as making good code in C++ is hard,
  it can be their gold mine

The first step: find attack surfaces

- I used to have (maybe still) any idea on OS X/iOS
  - Was thinking like “Oh, wow! C++ in kernel?”

- But if you do security research on a new platform, the first step to find security bugs quickly, try to understand what attack surfaces are there

Attack surfaces

- I didn’t want to spend time on reading huge books, articles about OS X to bug hunting since this is my hobby time done
- I was quickly looking at the OS
- All of operating systems have huge attack surfaces
  - Think like Apache versus Windows

Attack surfaces

- Before figuring out attack surfaces, you should make sure yourself what types of bugs you want to hunt
- Local privilege escalation
  - IOKit, Mach/Posix system calls, Frameworks, DYLD
- Remote code execution
  - Documentation, popular software, Web browsers
  - RCE bug class is pretty much same as other OS

Bug classes

- Typical bug classes you can find
  - Logical bugs
  - Memory trespass
  - Cryptography
  - Authentication
  - etc
- I was focusing on memory trespass and logical bugs

Attack surfaces

- Naive questions (Discussion time)
  - Is there any network deamon run by ‘root’?
  - How OS runs processes as ‘root’ when users request?
  - How setuid binaries work?
  - What’s the interface between kernel and user levels?
  - Is there any chance if my bug works on both OS X and iOS?
  - What bugs make huge impacts for normal users?
  - What are the popular 3rd applications on our target OS?

Attack surfaces

- Let me pick up some questions and try to explain my experience
  - Others will be covered later
- How setuid binaries works?
  - I firstly googled it and realized DYLD is partly responsible on it
  - (Not only for setuid binaries)
- Downloaded DYLD code from opensource.apple.com

DYLD

- I was just browsing and reading code
- And some weird things have been found

```
uintptr_t
_main(const macho_header* mainExecutableMH, uintptr_t mainExecutableSlide,
                int argc, const char* argv[], const char* envp[], const char* apple[],
                uintptr_t* startGlue)
{
        uintptr_t result = 0;
        sMainExecutableMachHeader = mainExecutableMH;
#if !TARGET_IPHONE_SIMULATOR
        const char* loggingPath = _simple_getenv(envp, "DYLD_PRINT_TO_FILE");
        if ( loggingPath != NULL ) {
                int fd = open(loggingPath, O_WRONLY | O_CREAT | O_APPEND, 0644);
                if ( fd != -1 ) {
                        sLogfile = fd;
                        sLogToFile = true;
                }
                else {
                        dyld::log("dyld: could not open DYLD_PRINT_TO_FILE='%s', errno=%d\n", loggingPath, errno);
                }
        }
#endif
#if __MAC_OS_X_VERSION_MIN_REQUIRED
        // if this is host dyld, check to see if iOS simulator is being run
        const char* rootPath = _simple_getenv(envp, "DYLD_ROOT_PATH");
        if ( rootPath != NULL ) {
                // look to see if simulator has its own dyld
                char simDyldPath[PATH_MAX];
                strlcpy(simDyldPath, rootPath, PATH_MAX);
                strlcat(simDyldPath, "/usr/lib/dyld_sim", PATH_MAX);
                int fd = my_open(simDyldPath, O_RDONLY, 0);
                if ( fd != -1 ) {
                        result = useSimulatorDyld(fd, mainExecutableMH, simDyldPath, argc, argv, envp, apple, startGlue);
                        if ( !result && (*startGlue == 0) )
                                halt("problem loading iOS simulator dyld");
                        return result;
                }
        }
#endif
```

bug #1

bug #2

DYLD bug #1

- Of course there is pruneEnvironmentVariables() function that deletes every environment starting with DYLD_ (also LD_LIBRARY_PATH) but previous code gets executed before it

- Using Bug 1, you can make an arbitrary file in any directory within ‘root’
  - filename can be controlled too
  - you can abuse this through the ‘sudo’ file check for sudo’ed users

DYLD bug #2

- In Bug 2, there is a sanity check unfortunately
  - provided files must be ‘root’ owned files

```
__attribute__((noinline))
static uintptr_t useSimulatorDyld(int fd, const macho_header* mainExecutableMH,
                                  const char* dyldPath, int argc, const char* argv[],
                                  const char* envp[], const char* apple[], uintptr_t* startGlue)
{
        *startGlue = 0;

        // verify simulator dyld file is owned by root
        struct stat sb;
        if ( fstat(fd, &sb) == -1 )
                return 0;
        if ( sb.st_uid != 0 )
                return 0;
```

DYLD bug #2

- But, what if there is any single ‘root’ owned file but has o+w?

```
$ ls -al /Library/Caches/com.apple.DiagnosticReporting.Networks.New.plist
-rw-rw-rw-  1 root  admin  152  2 20 00:00 /Library/Caches/com.apple.DiagnosticReporting.Networks.New.plist
```

- Then you can provide any dyld file and the rest of DYLD code parses your dyld file and jumps to the entry point

- This is 100% reliable, too

IOKit

- For these 2 questions
  - What's the interface between kernel and user levels?
    - Definitely IOKit is the hacker's choice
- Is there any chance if my bug works on both OS X and iOS?
  - iOS has IOKit too

IOKit bug #3

- Found more than 5 bugs in IOKit and they're exploitable
- How?: Downloaded the IOKit code again and found some points
- Bug #3 (Patched now)

```
IOHIDEventQueue * IOHIDEventQueue::withEntries( UInt32 numEntries,
                                                UInt32 entrySize )
{
    IOHIDEventQueue * queue = new IOHIDEventQueue;

    if ( queue && !queue->initWithEntries(numEntries, entrySize) )
    {
        queue->release();
        queue = 0;
    }

    queue->_state               = 0;
    queue->_lock                = IOLockAlloc();
    queue->_numEntries          = numEntries;
    queue->_currentEntrySize    = DEFAULT_HID_ENTRY_SIZE;
    queue->_maxEntrySize        = DEFAULT_HID_ENTRY_SIZE;

    return queue;
}
```

IOKit bug #3

- Apple gave me a credit on their website
  - Could work on OSX, iOS, Apple TV

- **IOHIDFamily**

  Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5, OS X Yosemite v10.10 and v10.10.1

  Impact: A malicious application may be able to execute arbitrary code with system privileges

  Description: A null pointer dereference existed in IOHIDFamily's handling of event queues. This issue was addressed through improved validation of IOHIDFamily event queue initialization.

  CVE-ID

  CVE-2014-4489 : @beist

IOKit bug #3

- You: “Wait a second, how do you map at NULL?”
- Me: “Thanks to Ian Beer.”
  - https://code.google.com/p/google-security-research/issues/detail?id=20

```
**************
mach_loader.c
load_segment(
        boolean_t               prohibit_pagezero_mapping = FALSE;
...

                /* XXX (4596982) this interferes with Rosetta, so limit to 64-bit tasks */
                if (scp->cmd == LC_SEGMENT_64) {
                        prohibit_pagezero_mapping = TRUE;
                }

                if (prohibit_pagezero_mapping) {
...
                        ret = vm_map_raise_min_offset(map, seg_size); //only place this is called
```

IOKit bug #4

- I was reporting Bug #3 to apple and they fixed it
- But I also knew there is the same pattern bug, if you scroll up (Bug #4)
  - Still alive

```
IOHIDEventQueue * IOHIDEventQueue::withCapacity( UInt32 size )
{
    IOHIDEventQueue * queue = new IOHIDEventQueue;

    if ( queue && !queue->initWithCapacity(size) )
    {
        queue->release();
        queue = 0;
    }

    queue->_state                = 0;
    queue->_lock                 = IOLockAlloc();
    queue->_numEntries           = size / DEFAULT_HID_ENTRY_SIZE;
    queue->_currentEntrySize     = DEFAULT_HID_ENTRY_SIZE;
    queue->_maxEntrySize         = DEFAULT_HID_ENTRY_SIZE;

    return queue;
}

IOHIDEventQueue * IOHIDEventQueue::withEntries( UInt32 numEntries,
                                                UInt32 entrySize )
{
```

IOKit bug #5

- You can talk to your graphic driver through IOKit
  - Intel3000, 4000, 5000 graphic drivers
- AppleIntelHD3000Graphics is available for 17’ Macbook
  - Memory corruption bugs can be found easily
  - For this, I wanted to fuzz to find bugs

IOKit bug #5

- There is an easy to fuzz on IOKit calls
  - Hooking IOServiceMatching and IOConnectCallMethod
  - Recording the service name and saving your mutated payload
  - Putting DYLD_INSERT_LIBRARIES environment before executing a target process
  - But make sure that your target process calls those APIs

IOKit bug #5

- There is delete_texture_internal(IOIntelAccelerator *, IOGen575Shared*, Gen575TextureBuffer *) function in the driver and you can partly control the Gen575TextureBuffer

- You can control PC register and get LPE in kernel level

3rd parties problem

- To answer
  - What are the popular 3rd applications on our target OS?
- Experts say most of 3rd parties software on OS X are not so secure
- I wanted to find some bugs in virtualization program
  - I picked Parallels but not VMware
  - Since I'm using Parallels and already found some bugs in VMware a long time ago (Via io fuzzing)

3rd parties problem

- Parallels is a virtualization program that you can run Windows, Linux, even OS X on OS X
- Bug type
  - Local privilege escalation on Host OS (bug #6)
  - Local privilege escalation on Guest OS (bug #7)
  - VM jailbreak from unprivileged guest OS to host OS (bug #8)

3rd parties problem

- I’ve managed to find all of bug types quickly
- Tips:
  - There are setuid binaries on Host OS
    - https://beistlab.wordpress.com/2015/01/08/0day_race_condition_parallels_desktop/ (pw: gr4y)
  - There are parallels drivers on Guest OS, too
    - Target Linux Guest OS first since there are source code of drivers (Easier to understand how they work)

Things not covered here

- POSIX/Mach system call auditing/fuzzing should be worth
- Frameworks are good targets
  - https://truesecdev.wordpress.com/2015/04/09/hidden-backdoor-api-to-root-privileges-in-apple-os-x/
- Filesystem bugs
  - .DMG file can be loaded when users click (Massive attacks possible)
- Learn from published 0 days, they save your time!

DEMO

- OS X LPE Demo on the latest version (bug #2)

- 100% reliable to get ‘root’

Conclusion

- People tend to think that Apple OS is more secure than Windows
  - But it is not
- Hackers just spend more time on Windows and Linux than Apple OS
  - Since they’re more popular and they can make more money
- We also should investigate Korean major software on OS X to figure out how they are strong against attacks

Conclusion

- This talk was mainly about how to approach Apple OS to find bugs for skilled hackers but no experience on it
- This could work on every new platforms
- However, this might be working for hobby Apple OS hackers but if you want to be a serious jailbreaker, you should get much deeper like the people in jailbreak scene

References

- Mac OS X and iOS Internals
- Countless Apple 0 days by Ian Beer
- Stenfan Esser’s awesome articles
- Find your own iOS Kernel bug by Chen Xiaobo and Xu Hao

Thank you!

- Question?