Exploring Yosemite:
Abusing Mac OS X 10.10.

Ming-chieh Pan, Sung-ting Tsai. Team T5.

Contact: (nanika|tt)@teamt5.org

Abstract

Mac OS X 10.10 Yosemite is going to be released soon. It brings lots of new features as well as security improvements. In the fist part of the talk, we are going to review these improvements from both defensive and offensive perspectives: what problem it solved, what issues it brought up, and what tricks still work.

In the second part, we will try several ways to abuse Mac OS X 10.10, and show you running malware and even rootkit is not a problem. A number of new offensive techniques will be introduced, including kernel mode and user mode, for example, loading a unsigned kernel module without warnings, manipulating kernel objects (rootkit) to evade detection, very stealthy techniques to launch malware, etc. All of the tricks were tested on Mac OS X 10.10.

Not only the offensive side, we are going to release a security tool in this talk as well. A comprehensive rootkit and abnormality scanner, we call it SVV-X (System Virginity Verifier for Mac OS X 10.10). The tool covers not only basic checks, such as hooks on syscall table, mach trap, IDT table, critical data verification, kernel code integrity, and it also checks many user mode tricks.Attacking Mac OS X has become a trend as we see more and more malware with advanced attack techniques on Mac OS X. In order to gain persistent control and avoid detection, malware have started to adopt rootkit tricks.

Table of Contents

1. Previous Works

1.1 The rubilyn Rootkit

The rubilyn rootkit was released on full disclosure in 2012, and claimed the following capabilities:

- It works across multiple kernel versions (tested 11.0.0+)
- Give root privileges to pid.
- Hide files / folders
- Hide a process
- Hide a user from 'who'/'w'
- Hide a network port from netstat
- sysctl interface for userland control
- execute a binary with root privileges via magic ICMP ping

Although it is already 2 years old, it is still the most famous rootkit on Mac OS X.

Here is the process structure in kernel:

```
struct  proc {
        LIST_ENTRY(proc) p_list;                /* List of all processes. */

        pid_t           p_pid;                  /* Process identifier. (static)*/
        void *          task;                   /* corresponding task (static)*/
        struct  proc *  p_pptr;                 /* Pointer to parent process.(LL) */
        pid_t           p_ppid;                 /* process's parent pid number */
        pid_t           p_pgrpid;               /* process group id of the process (LL)*/

        lck_mtx_t       p_mlock;                /* mutex lock for proc */

        char            p_stat;                 /* S* process status. (PL)*/
        char            p_shutdownstate;
        char            p_kdebug;               /* P_KDEBUG eq (CC)*/
        char            p_btrace;               /* P_BTRACE eq (CC)*/

        LIST_ENTRY(proc) p_pglist;              /* List of processes in pgrp.(PGL) */
        LIST_ENTRY(proc) p_sibling;             /* List of sibling processes. (LL)*/
        LIST_HEAD(, proc) p_children;           /* Pointer to list of children. (LL)*/
        TAILQ_HEAD( , uthread) p_uthlist;       /* List of uthreads  (PL) */
```

Rubilyn uses a simple DKOM (direct kernel object modification) to hide processes. It just unlinks p_list to hide process, so it is not difficult to detect rubilyn process hiding.

1.2 Volatility and Bypass Volatility

Volatility is a well-know memory forensic tool. Volatility can detect rubilyn rootkit.

In Black Hat Asia 2014, we introduced how we can make a rootkit to bypass Volatility detection. We found that it checks p_list, p_hash, p_pglist, and task. So we can unlink p_list, p_hash, p_pglist, and task list, then Volatility cannot detect.

Demonstration video: https://www.youtube.com/watch?v=_QD5YVSZz4U

1.3 Other Mac OS X Rootkit Researches from BH ASIA 2014

Please refer to our previous slides and paper in Black Hat Asia 2014: You can't see me - A Mac OS X rookit uses the tricks you haven't known yet. Topic includes:

- An advanced Rootkit can bypass Volatiliy detection.
- A privileged normal user.
- Direct kernel memory access.
- Loading kernel module without warnings.
- A trick to gain root permission.

2. Learning OS X Rootkit

2.1 From Windows Perspectives

We believe most of security researchers are more familiar with Windows rootkits, since there are too many rootkit samples and there are already many research papers, books, and materials that help people to understand rootkit well.

We found lots of concepts are very similar, so in following chapters, we will also take windows example to help you understanding rootkit easier.

2.2 Rootkit Startup

Starting a malware/rootkit, one of the way to start programs is using:

- C:\Users\(username)\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\

And it is similar on Mac OS X, just different location:

- ~/Library/LaunchDaemons/
- ~/Library/LaunchAgents/
- /System/Library/LaunchDaemons/
- /System/Library/LaunchAgents/

2.3 DLL Pre-loading

Pre-loading a DLL file is quite common tricks. It usually use:

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\

On Mac OS X, it is similar. Editing plist file in:

- DYLD_INSERT_LIBRARIES

2.4 File Extension Trick

Manipulation file extension associated program is also a malware trick. On windows, settings could be found here:

- HKEY_CLASSES_ROOT\txtfile\shell\open\command

On OS X, it is similar. You may find something here:

~/Library/Preferences/com.apple.LaunchServices.plist

com.apple.LaunchServices-036501.csstore

And using this API - LSSetDefaultRoleHandlerForContentType(), you can update the setting.

3. Comparing Mavericks and Yosemite

3.1 Kernel Interface Changes

We have listed IDT_list, mach_kernel_api, machtrap, and syscall of 10.9 and 10.10. You may find differences in the attachment.

4. Kernel Mode Rootkit on Yosemite

4.1 Kernel Files

On Windows, the kernel file is NTOSKRNL.EXE (NTKRNLPA.EXE).

On Mac OS X, it is usually:

- /mach_kernel (Mac OS 10.7 / 10.8 / 10.9)

However, in 10.10, it changed to:

- /System/Library/Kernels/kernel

4.2 Detecting rubilyn Process Hiding

There is a corresponding task to each process, and tasks are also a linked-list like process list.

```
struct  proc {
        LIST_ENTRY(proc) p_list;                /* List of all processes. */

        pid_t           p_pid;                  /* Process identifier. (static)*/
        void *          task;                   /* corresponding task (static)*/
        struct  proc *  p_pptr;                 /* Pointer to parent process.(LL) */
        pid_t           p_ppid;                 /* process's parent pid number */
        pid_t           p_pgrpid;               /* process group id of the process (LL)*/
```

```
struct task {
        /* Synchronization/destruction information */
        decl_lck_mtx_data(,lock)                    /* Task's lock */
        uint32_t        ref_count;      /* Number of references to me */
        boolean_t       active;         /* Task has not been terminated */
        boolean_t       halting;        /* Task is being halted */

        /* Miscellaneous */
        vm_map_t        map;            /* Address space description */
        queue_chain_t   tasks;  /* global list of tasks */
        void            *user_data;     /* Arbitrary data settable via IPC */

        /* Threads in this task */
        queue_head_t            threads;
```

So we can easily detect rubilyn process hiding by listing tasks and comparing with process list. Actually rubilyn can only hide process from 'ps' command, however using Active Monitor can see process/task that hided by rubilyn.

4.3 It still works

We unlinked p_list, p_hash, p_pglist, and task list to bypass Volatility checking. The trick still works on 10.10. Volatility hasn't support 10.10 yet.

5. User Mode Rootkit on Yosemite

5.1 Launchd Magic

In previous chapters, we did lots of hard works in kernel in order to hide process. However, there is a trick that we can easily find an invisible process from user mode!

Launchd is monitoring all process creation and termination. It maintains a job list in user mode. 'launchctl' is the tool to communicate with launchd. It can easily list jobs like this:

```
Naniteki-MacBook-Air:ext_research Nani$ launchctl list
PID     Status  Label
11665   -       0x7fc8e9c3b1a0.anonymous.launchctl
11648   -       0x7fc8e9d07a00.anonymous.vmware-vmx
11511   -       [0x0-0x5ab5ab].com.SweetScape.010Editor
11483   -       0x7fc8e9e0e9b0.anonymous.Google Chrome H
11401   -       0x7fc8e9c390f0.anonymous.Google Chrome H
11305   -       0x7fc8e9e0c7c0.anonymous.Google Chrome H
11263   -       0x7fc8e9d07700.anonymous.Google Chrome H
11253   -       0x7fc8e9d06d90.anonymous.Google Chrome H
11178   -       0x7fc8e9e0cdc0.anonymous.Google Chrome H
10785   -       0x7fc8e9e0cac0.anonymous.Google Chrome H
10411   -       0x7fc8e9c3b4a0.anonymous.Google Chrome H
10341   -       0x7fc8e9c3aea0.anonymous.Google Chrome H
10312   -       0x7fc8e9d07100.anonymous.Google Chrome H
10237   -       0x7fc8e9c3aba0.anonymous.vmnet-dhcpd
10247   -       0x7fc8e9c3a390.anonymous.vmware-usbarbit
10242   -       0x7fc8e9c3a8a0.anonymous.vmnet-netifup
10240   -       0x7fc8e9c39d90.anonymous.vmnet-natd
```

5.1.1 Unlink a job in Launchd

Here are the steps to unlink a job in launchd:

- Get root permission.
- Enumerate process launchd and get launchd task.
- Read launchd memory and find data section
- Find root_jobmgr
  - Check root_jobmgr->submgrs and submgrs->parentmgr
- Enumerate jobmgr and get job
- Enumerate job and find the target job
- Unlink the job

5.2 Launchd on OS X 10.10

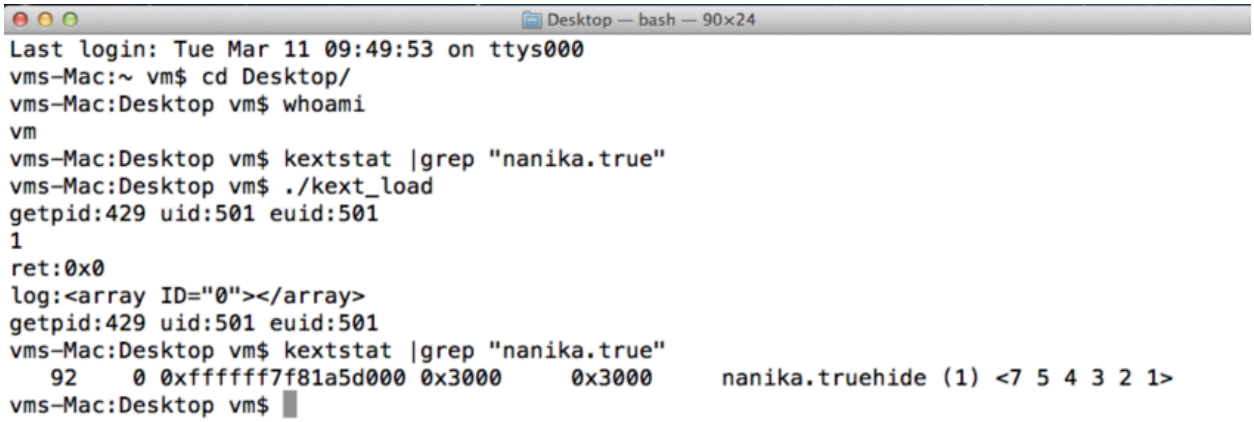
On OS X 10.10, we found there is a significant change: there is only 1 launchd now, i.e. there is a lot of changes. So the technique we introduced is not working on 10.10.

Since the source code haven't been released yet, we reverse-engineered new launchd and successfully unlink jobs in new launchd. We will have a demonstration to show this use mode rootkit.

6. Other Rootkit Tricks

6.1 Running Privileged Tasks as a Normal User

Following picture shows that we can do privileged tasks as normal user:



As a normal user vm (uid:501), we successfully loaded a kernel module 'nanika.true'. How did we do this?

6.2 Host Privilege

In Mac OS X, when a process performing a task that requires permission, it doesn't check uid of the process, instead, it checks if the task is granted the Host Privilege.

```
struct  host {
        decl_lck_mtx_data(,lock)                /* lock to protect exceptions */
        ipc_port_t special[HOST_MAX_SPECIAL_PORT + 1];
        struct exception_action exc_actions[EXC_TYPES_COUNT];
};

typedef struct host     host_data_t;

extern host_data_t      realhost;


/*
 * Always provided by kernel (cannot be set from user-space).
 */
#define HOST_PORT                           1
#define HOST_PRIV_PORT                      2
#define HOST_IO_MASTER_PORT                 3
#define HOST_MAX_SPECIAL_KERNEL_PORT        7 /* room to grow */
```

Here is a list of the things we can do with host privilege:

Host Interface

host_get_clock_service - Return a send right to a kernel clock's service port.
host_get_time - Returns the current time as seen by that host.
host_info - Return information about a host.
host_kernel_version - Return kernel version information for a host.
host_statistics - Return statistics for a host.
mach_host_self - Returns send rights to the task's host self port.

Data Structures

host_basic_info - Used to present basic information about a host.
host_load_info - Used to present a host's processor load information.
host_sched_info - - Used to present the set of scheduler limits associated with the host.
kernel_resource_sizes - Used to present the sizes of kernel's major structures.

Host Control Interface

host_adjust_time - Arranges for the time on a specified host to be gradually changed by an adjustment value.
host_default_memory_manager - Set the default memory manager.
host_get_boot_info - Return operator boot information.
host_get_clock_control - Return a send right to a kernel clock's control port.
host_processor_slots - Return a list of numbers that map processor slots to active processors.
host_processors - Return a list of send rights representing all processor ports.
host_reboot - Reboot this host.
host_set_time - Establishes the time on the specified host.

Host Security Interface

host_security_create_task_token - Create a new task with an explicit security token.
host_security_set_task_token - Change the target task's security token.

And actually it can have permission to control a tasks via these API:

- processor_set_default
- host_processor_set_priv
- processor_set_tasks

Host privilege gives a process power to do a lot of things. That's the reason why we can load a kernel module as a normal user.

6.3 How to Get Host Privilege

There are 3 ways to grant host privilege to a regular process:

- Assign host privilege to a task
  - Parse mach_kernel and find _realhost
  - Find task structure
  - Assign permission: task->itk_host = realhost->special[2]
  - Then the task/process can do privilege things.
- Hook system call (Global)
  - When process is retrieving the task information, make it return with host privilege.
  - Patch code (Global, good for rootkit)
  - Here is the code we are going to patch: (host_self_trap)

```
mach_port_name_t
host_self_trap(
        __unused struct host_self_trap_args *args)
{
        ipc_port_t sright;
        mach_port_name_t name;

        sright = ipc_port_copy_send(current_task()->itk_host);
        name = ipc_port_copyout_send(sright, current_space());
        return name;
}
```

  - Patch code:

```
                                                    _host_self_trap:
0xffffff8000225f20 55                                   push        rbp
0xffffff8000225f21 4889E5                               mov         rbp, rsp
0xffffff8000225f24 65488B042508000000                   mov         rax, qword [gs:0x8]
0xffffff8000225f2d 488B8058030000                       mov         rax, qword [ds:rax+0x358]
0xffffff8000225f34 488BB820020000                       mov         rdi, qword [ds:rax+0x220]
0xffffff8000225f3b E89034FFFF                           call        _ipc_port_copy_send
0xffffff8000225f40 65488B0C2508000000                   mov         rcx, qword [gs:0x8]
0xffffff8000225f49 488B8958030000                       mov         rcx, qword [ds:rcx+0x358]
0xffffff8000225f50 488BB168020000                       mov         rsi, qword [ds:rcx+0x268]
0xffffff8000225f57 4889C7                               mov         rdi, rax
0xffffff8000225f5a 5D                                   pop         rbp
0xffffff8000225f5b E9E034FFFF                           jmp         _ipc_port_copyout_send
```

call _host_self

mov rax, [rax+0x20]

mov rdi, rax

6.4 Access Tasks Objects in Kernel from User Mode

Since Mac OS X 10.6, it restricted task access for kernel task. According to this report:

> "task_for_pid() is not supported on the kernel task, no matter your privilege level nor what API you use.
>
> … there is no legitimate use for inspecting kernel memory."

However, we discovered a way to direct access kernel task memory. We don't use task_for_pid(), instead we use processor_set_tasks().

- processor_set_tasks(p_default_set_control, &task_list, &task_count);
- then, **task_list[0]** is the kernel task!

We can control all of tasks and read / write memory, even use thread_set_state() to inject dynamic libraries.

7. Bypass Kernel Module Verification in 10.10

7.1 Loading a Kernel Module

In Mac OS 10.9, if you want to load a kernel module you have to:

- Put the kernel module file into /System/Library/Extensions/
- Run kextload to load the file
- If the kernel module is not signed, OS will pop up a warning message.

It is just a warning message, however, in 10.10 it will be blocked.

```
▼ 上午7:58:27 com.apple.kextd:
  ERROR: invalid signature for nanika.patch-kext-request, will not load
```

We are going to introduce a way to

- Load a kernel module from any path.
- Load a kernel module on the fly, from a memory buffer, etc. File is not required.
- Load a kernel module without verification. (no warning message)
- No need to patch kextd.

7.2 kext_request()

Using kext_reqest() to load kernel module, we can bypass many verifications. Following are steps to use kext_request():

- Get kext data ready. You need to know mkext

```
typedef struct mkext2_file_entry {
    uint32_t  compressed_size;  // if zero, file is not compressed
    uint32_t  full_size;        // full size of data w/o this struct
    uint8_t   data[0];          // data is inline to this struct
} mkext2_file_entry;

typedef struct mkext2_header {
    MKEXT_HEADER_CORE
    uint32_t plist_offset;
    uint32_t plist_compressed_size;
    uint32_t plist_full_size;
} mkext2_header;
```

- Get your host privilege. It checks the privilege.

```
    if (isMkext) {
#ifdef SECURE_KERNEL
        // xxx - something tells me if we have a secure kernel we don't even
        // xxx - want to log a message here. :-)
        *op_result = KERN_NOT_SUPPORTED;
        goto finish;
#else
        // xxx - can we find out if calling task is kextd?
        // xxx - can we find the name of the calling task?
        if (hostPriv == HOST_PRIV_NULL) {
            OSKextLog(/* kext */ NULL,
                kOSKextLogErrorLevel |
                kOSKextLogLoadFlag | kOSKextLogIPCFlag,
                "Attempt by non-root process to load a kext.");
            *op_result = kOSKextReturnNotPrivileged;
            goto finish;
        }

        *op_result = OSKext::loadFromMkext((OSKextLogSpec)clientLogSpec,
            request, requestLengthIn,
            &logData, &logDataLength);
```

- Call kext_request() to load the kernel module.
- Then you won't get any problems.

8. Introducing SVV-X

SVV-X stands for System Virginity Verifier for Mac OS X. The tool will verify all critical kernel objects, interfaces. If it is modified, SVV-X will give a warning message.

This tool could help to check all existing rootkit and potential malicious modifications in kernel.

9. Conclusion

In this paper, we verified many old tricks still works on 10.10 as well as implemented new rootkit tricks on 10.10.

The purpose of the research, we would like to demonstrate the possibilities of abusing Mac OS X. It is not 100% secure. Users still have to be aware of this.

And we are going to release tools to help checking rootkit on Mac OS X, the SVV-X.