Exploring Yosemite
Abusing Mac OS X 10.10

Team T5

Ming-chieh Pan (Nanika)
Sung-ting Tsai (TT)

About Us

Team T5

CHROOT

Team T5

Who we are: Threat intelligence expert and provider

What we do: Cyber threat research

Our product: indicators, feeds, subscription for reports

Customer: intelligence firms, security vendors, ⋯

We monitor, analyze, and track cyber threats.

Sung-ting Tsai (TT)

Team T5

Team leader

Speech

Black Hat USA / Asia / Europe
Syscan 10’ / 12’
Codegate 2012
HITCON 08’

Research

Threat Intelligence
New security technology
Malicious document
Malware auto-analyzing system
(sandbox technologies)
Malware detection
System vulnerability and protection

Ming-chieh Pan (Nanika)

Team T5

Chief Researcher

Speech

Black Hat USA / Asia / Europe
Syscan 08/10
HITCON 05/06/07/09/10/12

Research

Vulnerability discovery and analysis
Exploit techniques
Malware detection
Mobile security

Acknowledgements

Research conducted along with iSIGHT Partners

Agenda

Learning Mac OS X rootkit and a little bit of history.

diff Mavericks Yosemite

User and kernel mode rootkit tricks on Yosemite.

Loading an unsigned kernel module on Mac OS X 10.10.

Releasing SVV-X (System Virginity Verifier for Mac OS X)

Previous Works

on Rootkits

The rubilyn Rootkit

rubilyn-0.0.1.tar.gz - Mac OS X rootkit

From: Levent Kayan <levon.kayan () gmail com>
Date: Sat, 06 Oct 2012 13:22:39 +0200

```
Hi FD,

we are bored and wanted to share something with you:


name
====
rubilyn

description
===========
64bit Mac OS-X kernel rootkit that uses no hardcoded address to hook the
BSD subsystem in all OS-X Lion & below. It uses a combination of syscall
hooking and DKOM to hide activity on a host. String resolution of
symbols no longer works on Mountain Lion as symtab is destroyed during
load, this code is portable on all Lion & below but requires re-working
for hooking under Mountain Lion.

currently supports:

* 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

link
====
http://www.nullsecurity.net/backdoor.html
```

Volatility for Mac OS X

Volatility

Volatility is a well-know memory forensic tool.
New version of Volatility supports Mac OS X.
It can detect rubilyn rootkit as well.

Previous Works (BH ASIA 14)

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.

Learning OS X Rootkit

From Windows perspectives

Loading rootkits (program startup)

Windows:

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

OS X:

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

Loading rootkits (DLL/library preloading)

Windows:

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

OS X:

DYLD_INSERT_LIBRARIES plist file

Loading rootkits (file extension)

Windows:

HKEY_CLASSES_ROOT\txtfile\shell\open\command

OS X:

~/Library/Preferences/com.apple.LaunchServices.plist
com.apple.LaunchServices-036501.csstore
LSSetDefaultRoleHandlerForContentType()
/private/var/folders/yf/dhhxjj316h1dsnkgsc7zjd5h0000gn/C/com.apple.LaunchServices-036501.csstore
/private/var/folders/zz/zyxvpxvq6csfxvn_n000007000001r/

DEMO

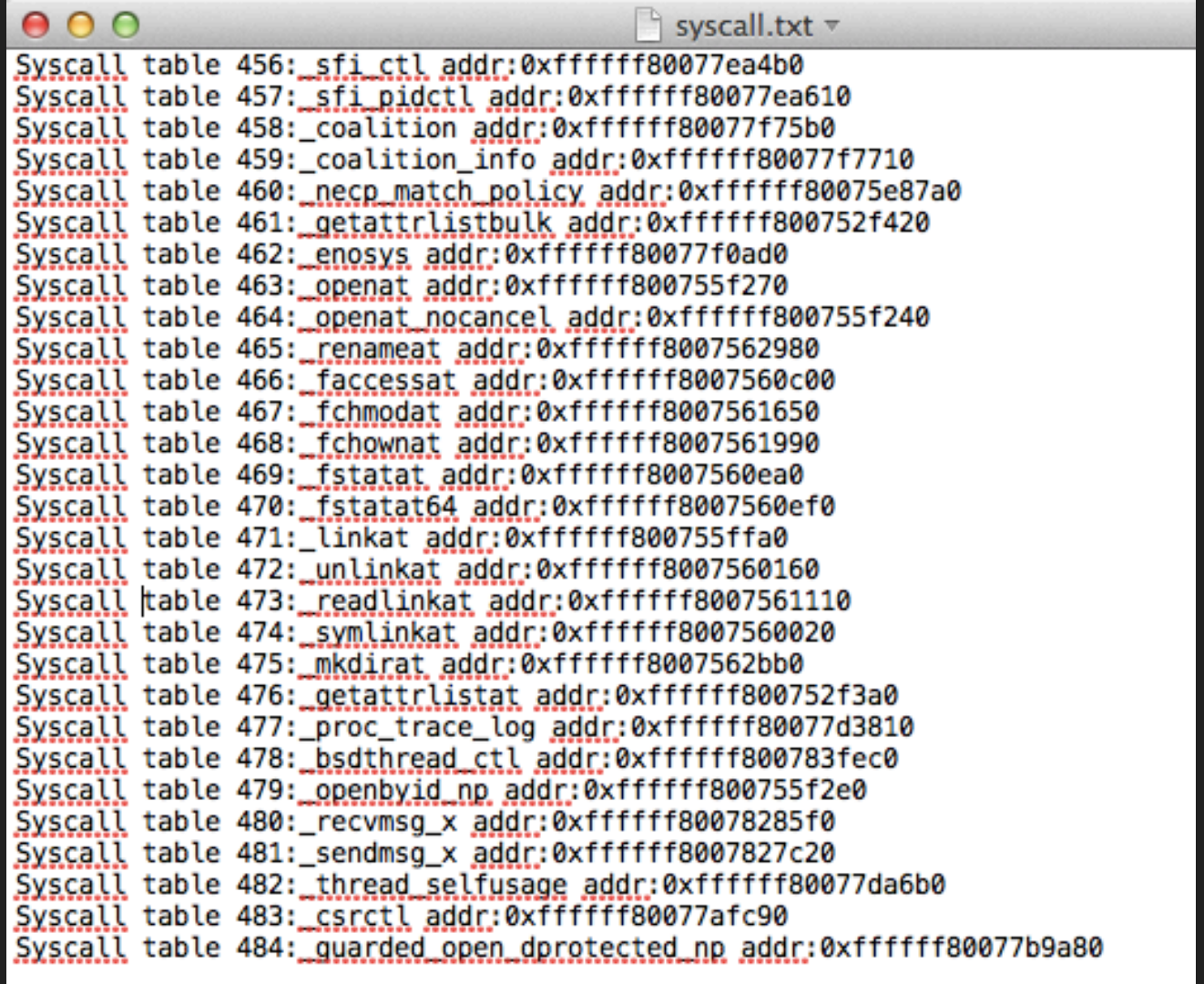

diff Mavericks Yosemite

Kernel changes

syscal

machtrap

IDT


syscall.txt
Syscall table 456:_sfi_ctl addr:0xffffff80077ea4b0
Syscall table 457:_sfi_pidctl addr:0xffffff80077ea610
Syscall table 458:_coalition addr:0xffffff80077f75b0
Syscall table 459:_coalition_info addr:0xffffff80077f7710
Syscall table 460:_necp_match_policy addr:0xffffff80075e87a0
Syscall table 461:_getattrlistbulk addr:0xffffff800752f420
Syscall table 462:_enosys addr:0xffffff80077f0ad0
Syscall table 463:_openat addr:0xffffff800755f270
Syscall table 464:_openat_nocancel addr:0xffffff800755f240
Syscall table 465:_renameat addr:0xffffff8007562980
Syscall table 466:_faccessat addr:0xffffff8007560c00
Syscall table 467:_fchmodat addr:0xffffff8007561650
Syscall table 468:_fchownat addr:0xffffff8007561990
Syscall table 469:_fstatat addr:0xffffff8007560ea0
Syscall table 470:_fstatat64 addr:0xffffff8007560ef0
Syscall table 471:_linkat addr:0xffffff800755ffa0
Syscall table 472:_unlinkat addr:0xffffff8007560160
Syscall table 473:_readlinkat addr:0xffffff8007561110
Syscall table 474:_symlinkat addr:0xffffff8007560020
Syscall table 475:_mkdirat addr:0xffffff8007562bb0
Syscall table 476:_getattrlistat addr:0xffffff800752f3a0
Syscall table 477:_proc_trace_log addr:0xffffff80077d3810
Syscall table 478:_bsdthread_ctl addr:0xffffff800783fec0
Syscall table 479:_openbyid_np addr:0xffffff800755f2e0
Syscall table 480:_recvmsg_x addr:0xffffff80078285f0
Syscall table 481:_sendmsg_x addr:0xffffff8007827c20
Syscall table 482:_thread_selfusage addr:0xffffff80077da6b0
Syscall table 483:_csrctl addr:0xffffff80077afc90
Syscall table 484:_guarded_open_dprotected_np addr:0xffffff80077b9a80


Kernel Mode Rootkit
On Mac OS X 10.10

Where is the kernel

Windows:

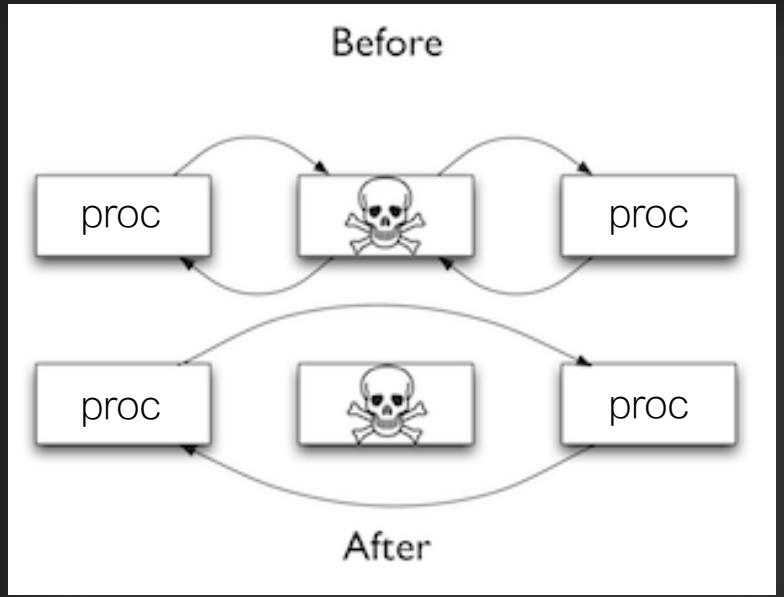
NTOSKRNL.EXE (NTKRNLPA.EXE)...

OS X:

/mach_kernel Mac OS 10.7 / 10.8 / 10.9

/System/Library/Kernels/kernel Mac OS X 10.10

Rubilyn – Using DKOM to hide process



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) */
```

```
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;
```

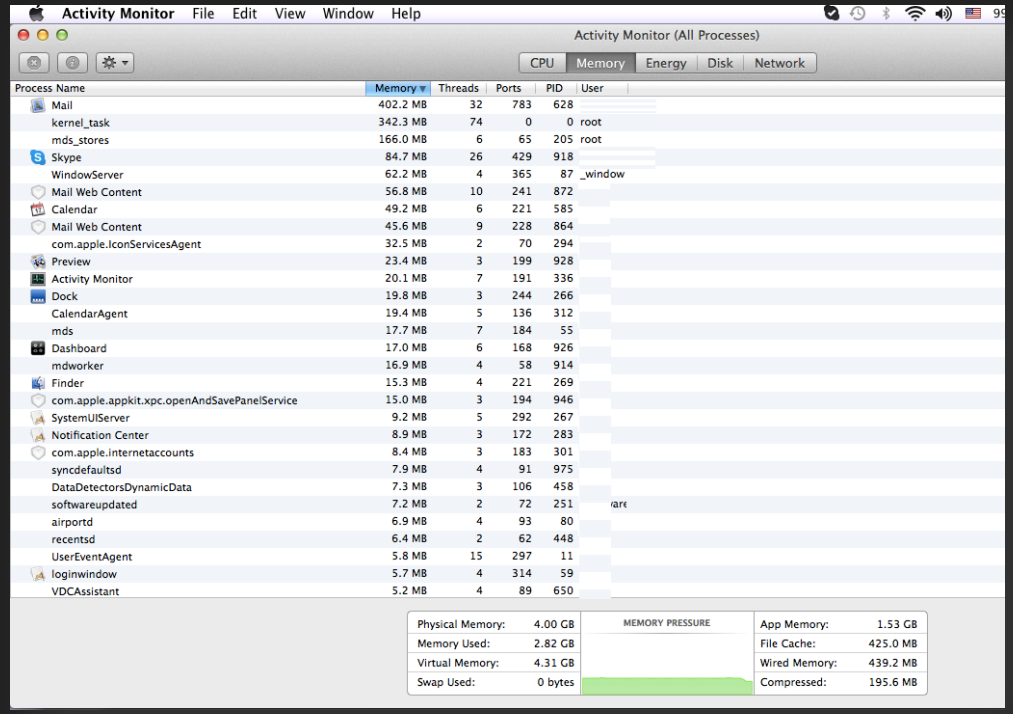
Detecting rubilyn Process Hiding

DKOM

Rubilyn uses a simple DKOM (direct kernel object modification) to hide processes. It just unlinks p_list to hide

So we can easily detect rubilyn process hiding by listing tasks and comparing with process list.

Rubilyn can NOT hide from Active Monitor. (Mac OS X 10.7)



An advanced Rootkit on Mac OS X 10.10

Unlink p_list.

Unlink p_hash

Unlink p_pglist

Unlink task

Bypass volatility (on 10.9)

demo

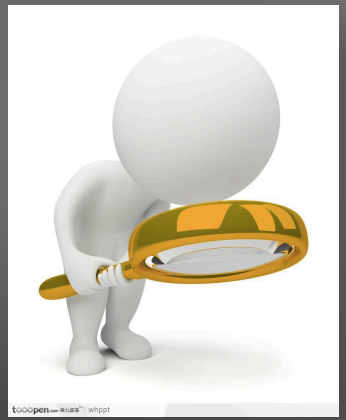
Competition in Kernel

User Mode Rootkits

On Mac OS X 10.10

User Mode 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

launchd(8) BSD System Manager's Manual launchd(8)

NAME

launchd -- System wide and per-user daemon/agent manager

SYNOPSIS

launchd [**-d**] [**-D**] [**-s**] [**-S** SessionType] [**--** command [args ...]]

DESCRIPTION

launchd manages processes, both for the system as a whole and for individual users. The primary and preferred interface to **launchd** is via the launchctl(1) tool which (among other options) allows the user or administrator to load and unload jobs. Where possible, it is preferable for jobs to launch on demand based on criteria specified in their respective configuration files.

During boot **launchd** is invoked by the kernel to run as the first process on the system and to further bootstrap the rest of the system.

You cannot invoke **launchd** directly.

```
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
```

Unlink a job in launchd (OS X 10.9)

Get root permission

Enumerate process launchd and get launchd task

Read launchd memory and find data section

Find root_jobmgr

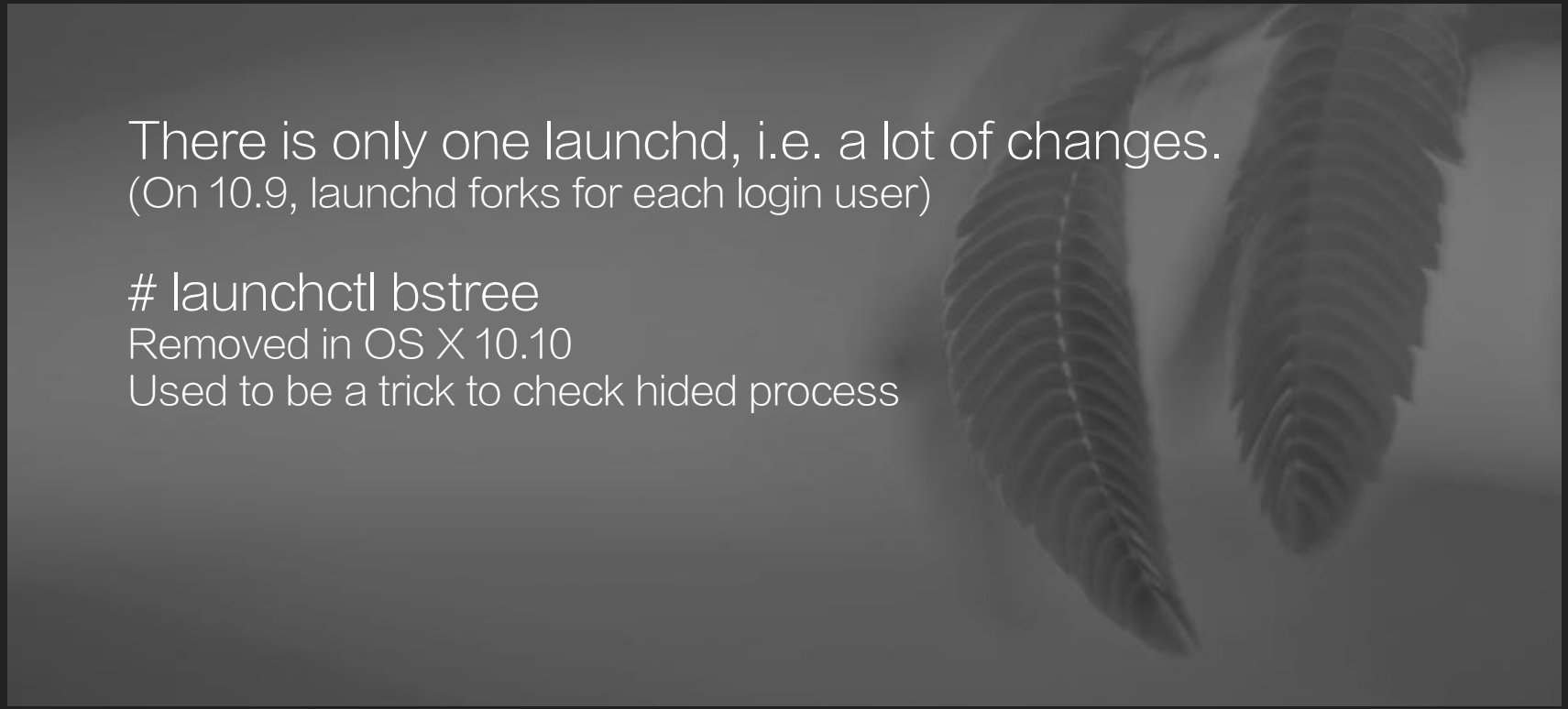
Check root_jobmgr->submgrs and submgrs->parentmgr (verification)

Enumerate jobmgr and get job

Enumerate job and find the target job

Unlink the job

launchd (OS X 10.10)



There is only one launchd, i.e. a lot of changes.
(On 10.9, launchd forks for each login user)

launchctl bstree
Removed in OS X 10.10
Used to be a trick to check hided process

Unlinking a job from new launchd

com.apple.xpc.launchd.user.501.100016.Aqua
com.apple.xpc.launchd.domain.user.501
com.apple.xpc.launchd.domain.system

xpc mgr link
    com.apple.xpc.launchd.domain.pid.kextd.[19]->pid

Unlinking a job from new launchd

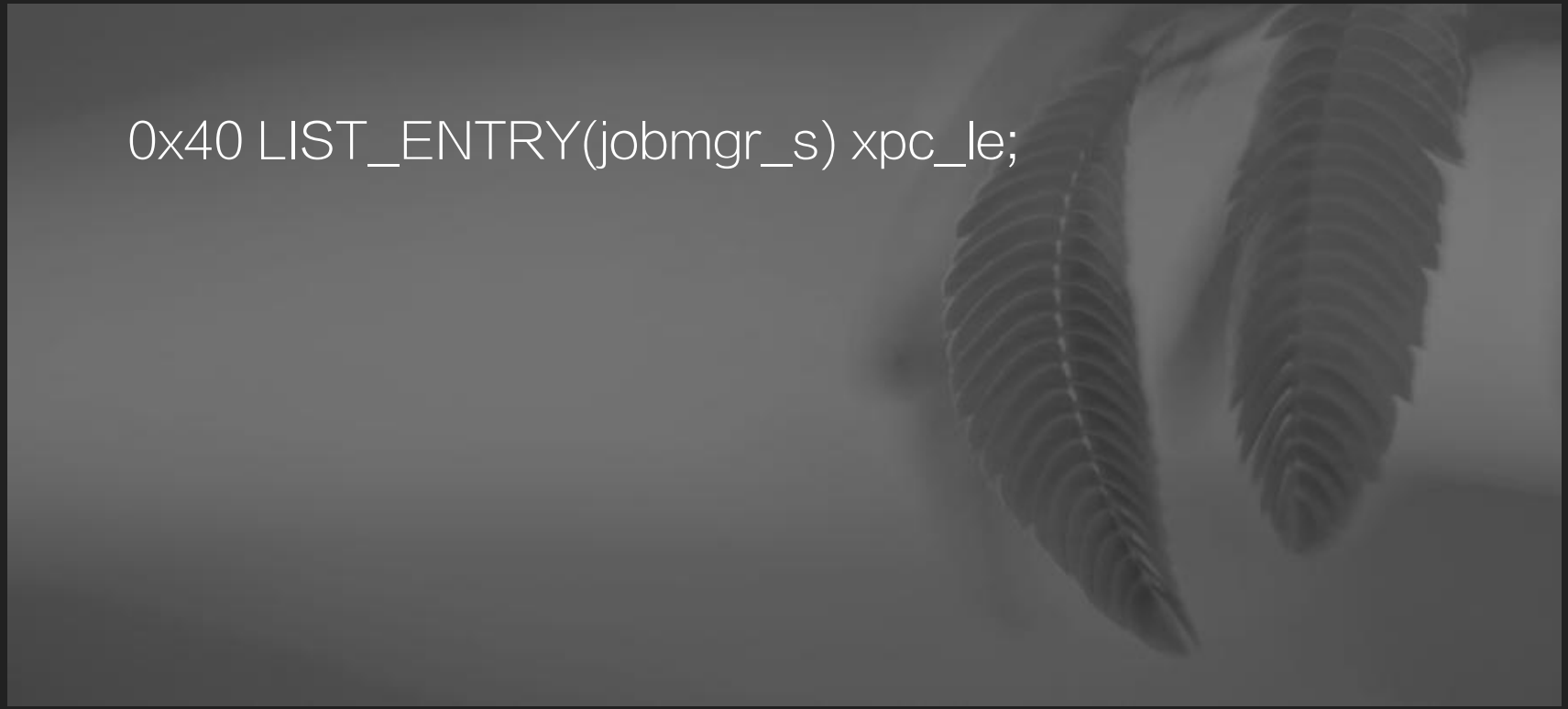
```
struct jobmgr_s10_10 {
    uint64_t kqjobmgr_callback;
    ...
    0x40 LIST_ENTRY(jobmgr_s) xpc_le;
    ...
} size 0x600
```

Unlinking a job from new launchd

```
struct job_s10_10 {
    uint64_t kqjob_callback;
    ...
    0x30 job_link;
    ...
} size=0x600

unlink 0x30 job_link; (DEMO)
```

How to check



launchctl procinfo

```
vmdeMac:Desktop vm$ sudo launchctl procinfo 346
Could not get task ports for pid: 0x5
auditon(): 3: No such process

Could not get responsible PID for PID 346: 3: No such process

proc_get_dirty(): 3: No such process

entitlements = (no entitlements)

com.apple.TextEdit.9688 = {
        active count = 6
        path = (submitted by Dock.214)
        state = running
        bundle id = com.apple.TextEdit

        program = /Applications/TextEdit.app/Contents/MacOS/TextEdit
        arguments = {
                /Applications/TextEdit.app/Contents/MacOS/TextEdit
        }
```

Demo

```
vmdeMac:Desktop vm$ sudo ./check_hiding_proc
************************************************************
EXPLORING YOSEMITE: ABUSING MAC OS X 10.10
System Virginity Verifier for Mac OS X
Check Hiding Process Tool
naninb[@]gmail.com ttsecurity[@]gmail.com
************************************************************
only for osx 10.10
Usage:
check_hiding_proc
sudo ./check_hiding_proc
base:0x10d353000
startaddress:0x10d38e000
findaddrss:0x10d391388 findrootjobmgrs:0x7f9c3301a800
                Alert!---------->uid:501 pid:346 jobmgr:com.apple.xpc.launchd.domain.pid.TextEdit.346
check detail use launchctl procinfo 346
check done!
```

Other Rootkit Tricks

On Mac OS X 10.10

Gain permission

Windows:

Modifying a Process Token

EPROCESS EX_FAST_REF Token;

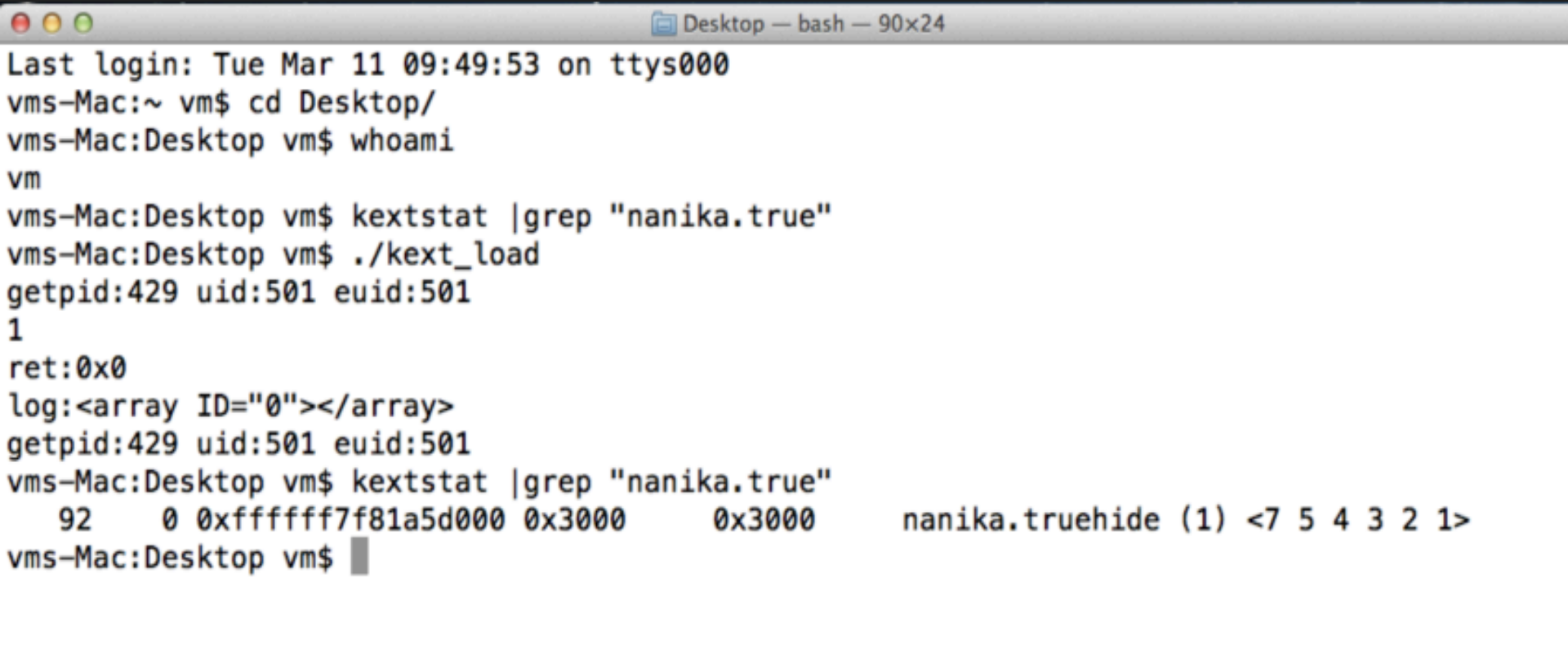
Mac OS X:

proc struct

pcred *p_cred;

How about

A privileged normal user



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.

Reference: http://felinemenace.org/~nemo/mach/manpages/

Rootkit with Host Privilege – Data Patch

```
uint64_t patch_host(){

    realhost=nfind_symbol("_realhost");
    realhost->special[1]=realhost->special[2];

}
```

Direct Task Access

We don’t use task_for_pid()

processor_set_tasks(p_default_set_control, &task_list, &task_count)

task_list[0] is the kernel task

Access all tasks

vm_read / vm_write
Read and write task memory

thread_set_state()
Dynamic library injection

Bypassing Driver Loading Verification

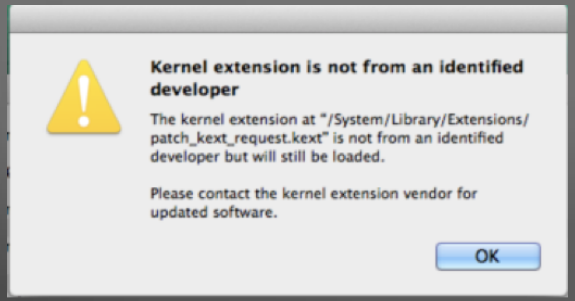
On Mac OS X 10.10

On Mac OS X 10.9, if you want to load a kernel module

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



On Mac OS X 10.10

Instead of a warning message, it blocks the kernel module loading.

mykextload

Load a kernel module from any path.

File is not required.
Load a kernel module on the fly, from memory or even network.

Load a kernel module without verification.
No warning message.

No need to patch kextd.
(ref: http://reverse.put.as/2013/11/23/breaking-os-x-signed-kernel-extensions-with-a-nop/)

kext_request()

```
kern_return_t kext_request(
    host_priv_t                         hostPriv,
    /* in only */  uint32_t             clientLogSpec,
    /* in only */  vm_offset_t          requestIn,
    /* in only */  mach_msg_type_number_t   requestLengthIn,
    /* out only */ vm_offset_t          * responseOut,
    /* out only */ mach_msg_type_number_t * responseLengthOut,
    /* out only */ vm_offset_t          * logDataOut,
    /* out only */ mach_msg_type_number_t * logDataLengthOut,
    /* out only */ kern_return_t        * op_result)
{
```

kext_request()

```
    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, requestLength,
            &logData, &logDataLength);
```

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;
```

```
<dict>
  <key>Kext Request Predicate</key> <string>Load</string>
  <key>Kext Request Arguments</key>...
  <key>_MKEXTInfoDictionaries</key>
  <array>...
```


Introducing SVV-X
System Virginity Verifier for Mac OS X

Hook based rootkits

IDT

SSDT

Win32k SSDT – machtrap

Kernel function inline hook

SVV-X


Releasing Tools

Tools

A PoC tool for hiding process

A tool to load an unsigned kernel module

SVV-X


Contact: tt@teamt5.org