RSA CONFERENCE 2014
FEBRUARY 24 – 28 | MOSCONE CENTER | SAN FRANCISCO

Share.
Learn.
Secure.
Capitalizing on
Collective Intelligence

Hunting for OS X Rootkits in Memory

SESSION ID: ANF-R03

Cem Gurkok

Threat Intelligence R & D Manager
Verizon Terremark
@CGurkok

It all started with…

[1], [2]

What's this all about?

> There are known knowns; there are things we know that we know.
> There are known unknowns; that is to say, there are things that we now know we don't know.
> But there are also unknown unknowns – there are things we do not know we don't know.
>
> —United States Secretary of Defense, Donald Rumsfeld

Rootkit Method	Type	Detecting Plugin
DTrace Hooks	Known Unknown	check_dtrace
Syscall Table Hooks	Known Unknown	check_hooks
Shadow Syscall Table	Known Unknown	check_hooks
IDT Hooks	Unknown Unknown	check_idt
Call Reference Modification	Known Unknown	check_hooks
Shadow TrustedBSD/mac_policy_list	Known Unknown	check_hooks

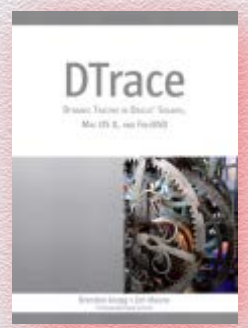
Why it matters?

- OS X Kernel has been increasingly targeted
- More users = more attackers
- Need better tools for detection
- Best place to detect: Memory
- Nowhere to hide

Some Definitions…

- **Syscall Table**: Functions that permit a userland process to interact with the kernel (BSD level)
- **Mach Trap Table**: Prototypes of traps as seen from userland (Mach level syscalls)
- Function Hooking
  - **Direct**: Replace the function entry with the modified version's address
  - **Inline**: Keep original function entry in place, modify the function itself (e.g. prologue)

What's DTrace?



- Dynamic Tracing Framework [3]
- Built for Solaris, now on OS X and TrustedBSD
- Used for troubleshooting system issues in real time via providers, for example:
  - **syscall**: Monitor the entry point into the kernel from applications in userland
  - **fbt** (function boundary tracing): probes for almost all kernel functions
  - **mach_trap**: fires on entry or return of the specified Mach library function
- Used for rootkit detection in the past by Beaucham and Weston [4]

DTrace Artifacts in Memory

- How to detect DTrace activity?
- After some research…
- Artifacts depend on the provider (syscall, fbt, mach_trap etc.)
  - **syscall**: Direct modification/hooking of the Syscall Table
  - **fbt**: Inline modification of the probed function
  - **mach_trap**: Direct modification/hooking of the Mach Trap Table

DTrace Hooks Detection

$ python vol.py **mac_check_dtrace** -f ~/Downloads/MacMemoryReader/ram_dump-after.mach-o --profile=MacLion_10_7_5_AMDx64

```
Volatile Systems Volatility Framework 2.3_beta
Table Name      Index  Address            Symbol                          D-Trace Probe
--------------- ------ ------------------ ------------------------------- ---------------
Syscall_Table   344    0xffffff80005c89e0 _dtrace_systrace_syscall        syscall_probe
```

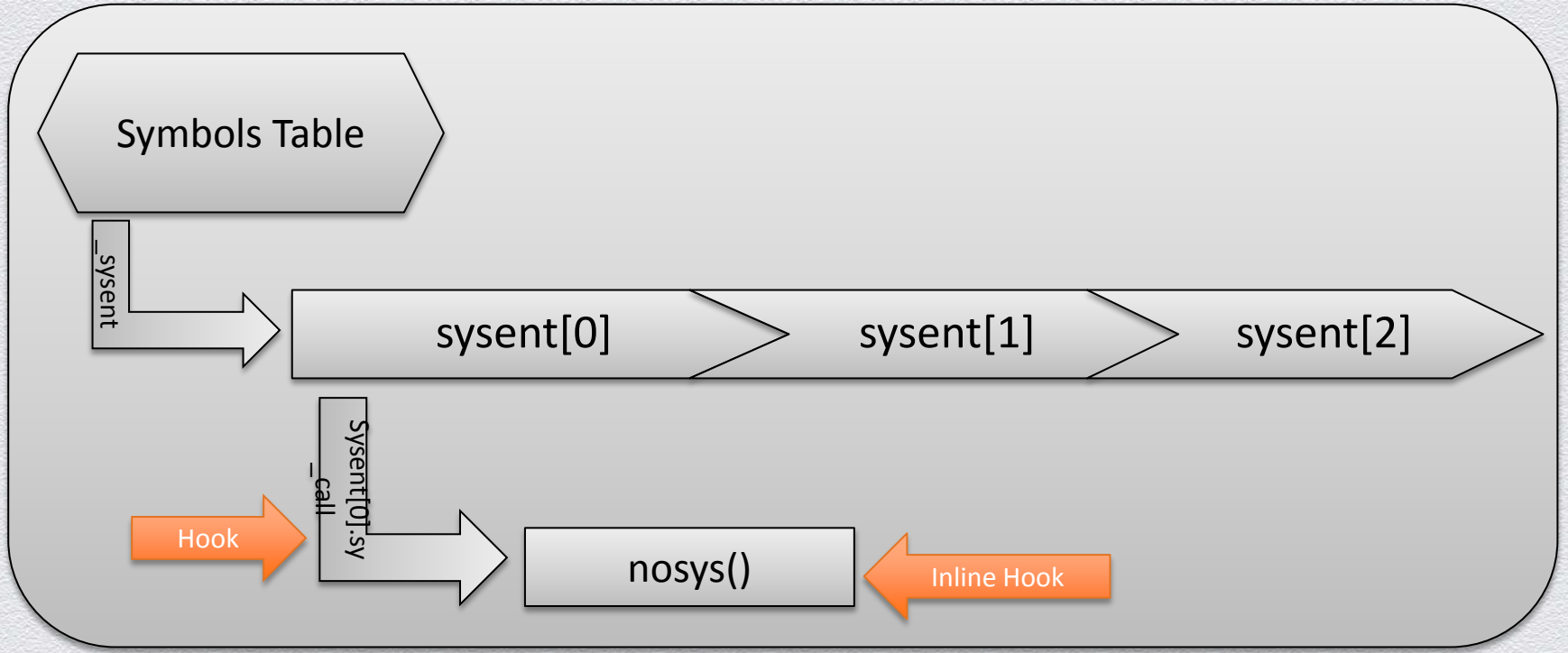
$ python vol.py **mac_check_dtrace** -f ~/Downloads/MacMemoryReader/ram_dump-trap.mach-o --profile=MacLion_10_7_5_AMDx64

```
Volatile Systems Volatility Framework 2.3_beta
Table Name      Index  Address            Symbol                          D-Trace Probe
--------------- ------ ------------------ ------------------------------- ---------------
Trap_Table      46     0xffffff80285dbc30 _dtrace_machtrace_syscall       mach_trap_probe
```

$ python vol.py **mac_check_dtrace** -f ~/Downloads/MacMemoryReader/ram_dump-fbt.mach-o --profile=MacLion_10_7_5_AMDx64

```
Volatile Systems Volatility Framework 2.3_beta
Table Name      Index  Address            Symbol                          D-Trace Probe
--------------- ------ ------------------ ------------------------------- ---------------
Syscall_Table   344    0xffffff8000306fb0 _getdirentries64                fbt_probe
```

Syscall Table Hooks



Detecting Syscall Table Hooks

$ python vol.py mac_check_hooks -f ~/Downloads/MacMemoryReader/ram_dump-dtrace.mach-o --profile=MacLion_10_7_4_AMDx64

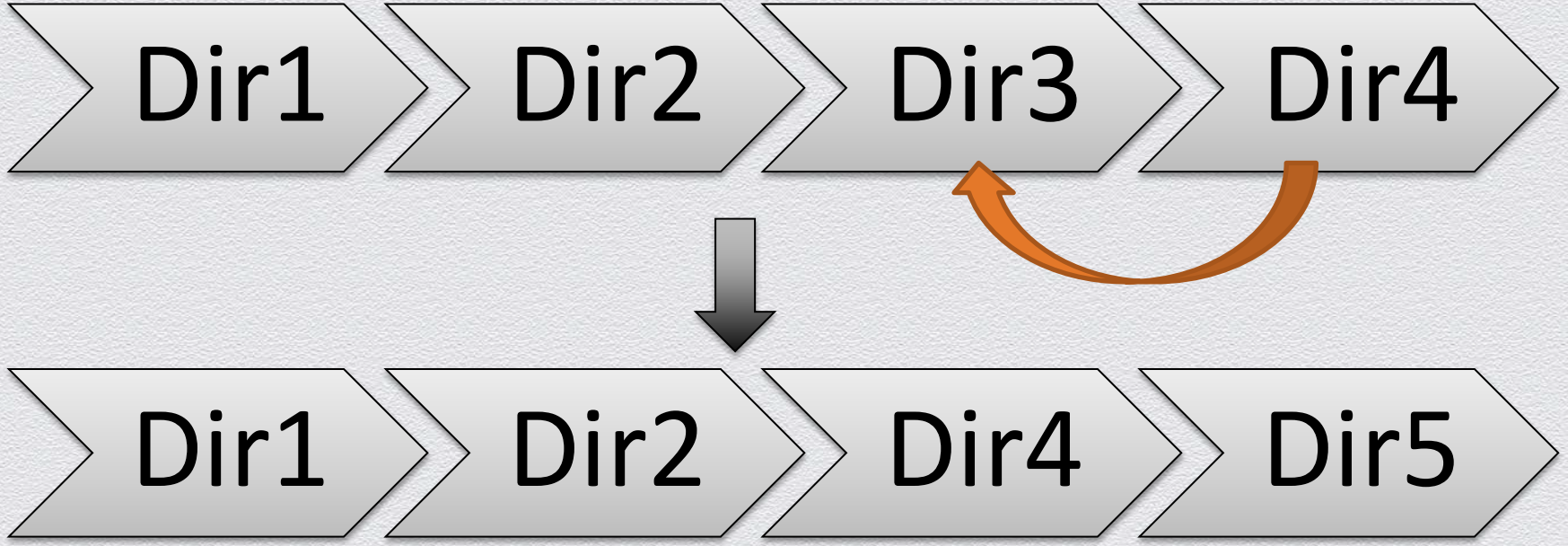
```
Volatile Systems Volatility Framework 2.3_beta
Table Name   Index  Address            Symbol                          Inlined Shadowed Perms  Hook In
------------ ------ ------------------ ------------------------------- ------- -------- ------ ----------
SyscallTable 344    0xffffff80005c89e0 [HOOKED] _dtrace_systrace_syscall No      No       -      __kernel__
```

$ python vol.py mac_check_hooks -f ~/Downloads/MacMemoryReader/ram_dump-fbt.mach-o --profile=MacLion_10_7_5_AMDx64

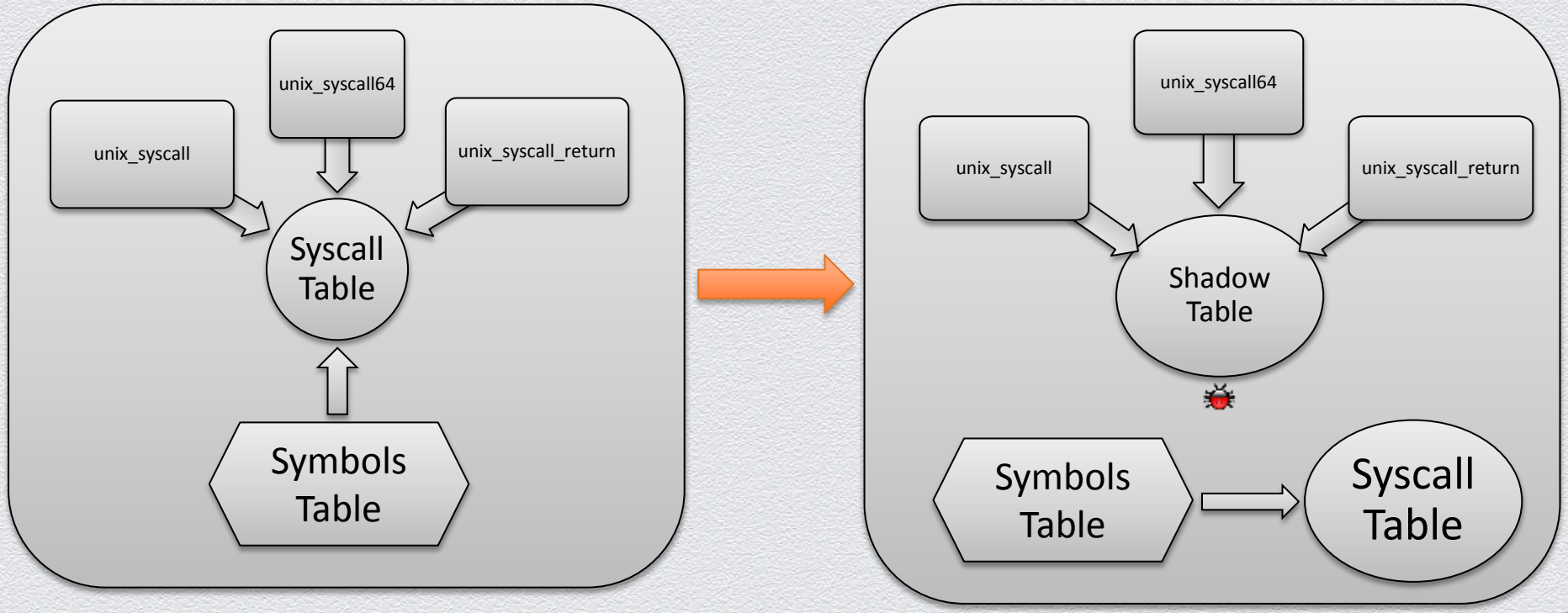
```
Volatile Systems Volatility Framework 2.3_beta
Table Name   Index  Address            Symbol                          Inlined Shadowed Perms  Hook In
------------ ------ ------------------ ------------------------------- ------- -------- ------ ----------
SyscallTable 344    0xffffff8000306fb0 _getdirentries64                Yes     No       -      __kernel__
```

Dtrace/Syscall Hook Detection Demo

Demo: Hiding a File/Folder and Detection



Shadow Syscall Table



Detecting the Shadow Syscall Table

- To detect the Shadow Syscall Table
  1. Check functions known to have references to the syscall table: unix_syscall_return, unix_syscall64, unix_syscall
  2. Disassemble them to find the syscall table references
  3. Obtain the references in the function and compare to the address in the symbols table
- All incorporated into the check_hooks plugin!

Detecting the Shadow Syscall Table

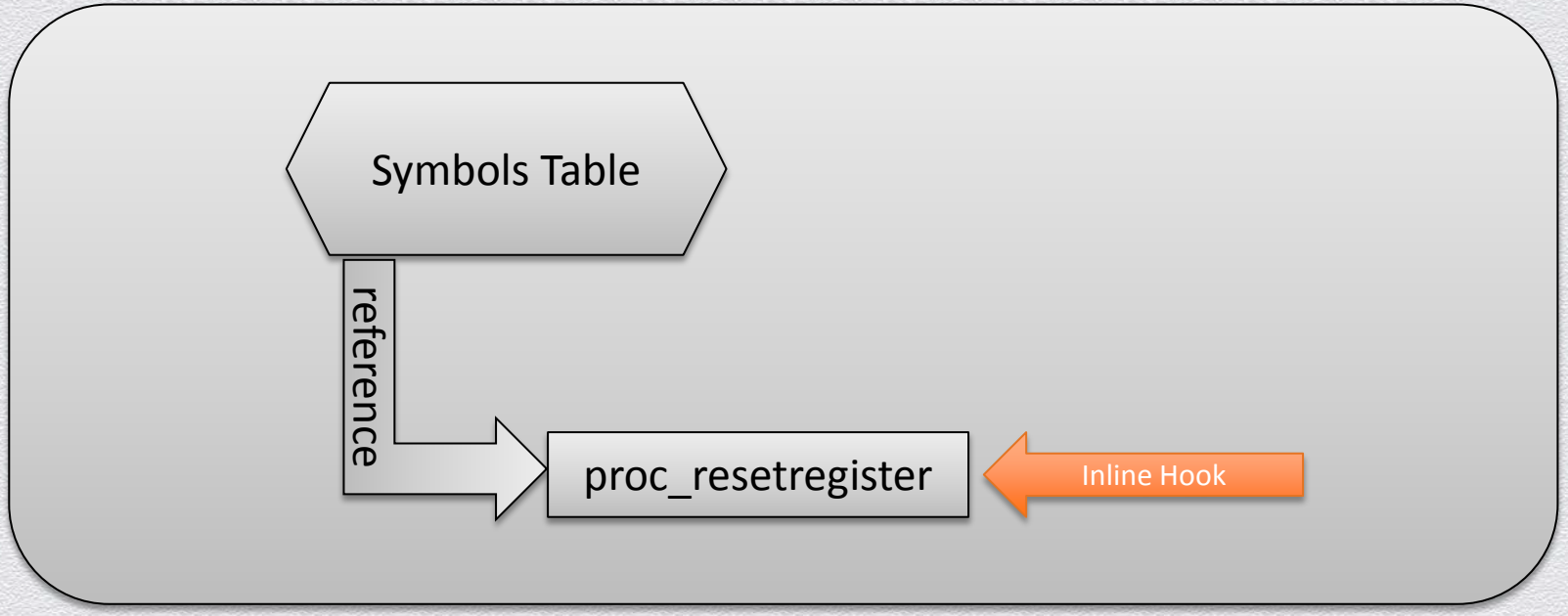
python vol.py mac_check_hooks -f /Volumes/Storage/HITB/ShadowSyscall-MountainLion_10_8_3_AMDx64.vmem --profile=MacMountainLion_10_8_3_AMDx64

sysent table is shadowed at _unix_syscall_return: 0xffffff800f3e084b ADD R15, [RIP+0x21f87e]
shadow sysent table is at 0xffffff7f907b2350
...
sysent table is shadowed at _unix_syscall64: 0xffffff800f3e04ac ADD R13, [RIP+0x21fc1d]
shadow sysent table is at 0xffffff7f907b2350
...
sysent table is shadowed at _unix_syscall: 0xffffff800f3e0246 ADD RBX, [RIP+0x21fe83]
shadow sysent table is at 0xffffff7f907b2350

Symbols Table Hooks

- Functions are exposed by the kernel and kexts in their symbols tables
- Can also be direct or inline hooked
- To check the functions, need to obtain the list of symbols
- Then check for modifications that cause the execution to continue in an external kext/module

Symbols Table Hooks



Hooking Symbols Table Functions

- Hydra [7], a kext that intercepts a process's creation
- Inline hooks **proc_resetregister**, a function in the kernel symbols
- The destination of the hook is in the 'put.as.hydra' kext
- Used the **check_hooks** plugin to find the hook

Detecting Symbols Table Hooks

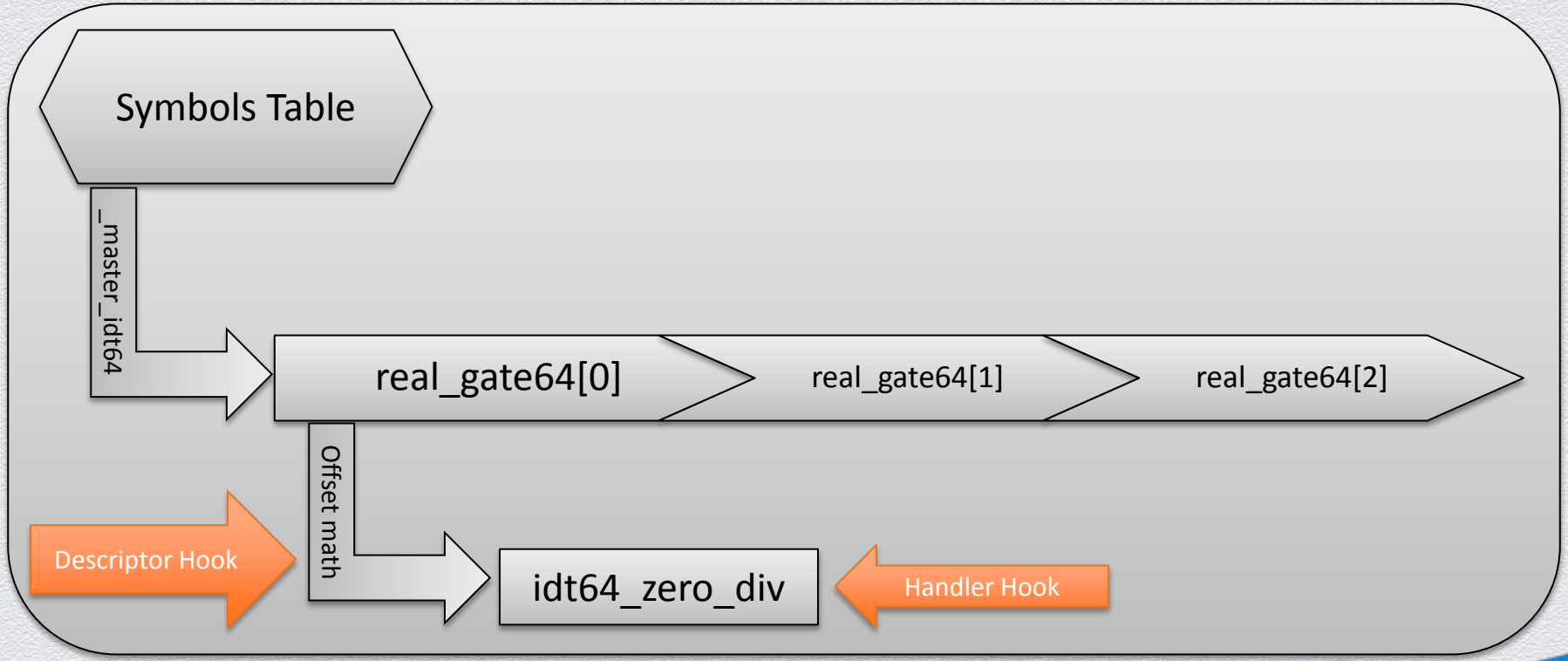
$ python vol.py mac_check_hooks -f ~/Documents/Virtual\ Machines/Mac\ OS\ X\ 10.8\ 64-bit-DEMO.vmwarevm/564d438d-cc29-2121-3dd6-ac473e701f8d.vmem --profile=MacMountainLion_10_8_5_AMDx64 **-K**

```
Volatility Foundation Volatility Framework 2.3
Table Name     Index  Address            Symbol                          Inlined Shadowed Perms  Hook In
-------------- ------ ------------------ ------------------------------- ------- -------- ------ ------------
SymbolsTable -       0xffffff8000b68fe0 _proc_resetregister             Yes     No       put.as.hydra
```

Hooking the IDT

- Interrupt descriptor table (IDT)
- Associates each interrupt or exception identifier (handler) with a descriptor (vector).
- Descriptors have the instructions for the associated event.
- An interrupt is usually defined as an event that alters the sequence of instructions executed by a processor.
- IDT entries: Interrupt Gates, Task Gates and Trap Gates…
- Why hook the IDT?
- Because it gives us ring 0 or root access!

Hooking the IDT



Hooking the IDT Descriptor

- Hooked **idt64_zero_div** and redirected to **idt64_stack_fault**
- Used both hooking methods

```
testers-Mac:~ tester$ ./div0
Floating point exception: 8
```

```
testers-Mac:~ tester$ ./div0
Illegal instruction: 4
```

Detecting the Descriptor Hook

$ python vol.py mac_check_idt -f /Volumes/Storage/HITB/IDTDescriptorHook-MountainLion_10_8_3_AMDx64.vmem --profile=MacMountainLion_10_8_3_AMDx64

```
Volatile Systems Volatility Framework 2.3_beta
CPU# Index Address            Symbol                 Ring Selector     Module                    Hooked Inlined
---- ----- ------------------ ---------------------- ---- ------------ ------------------------- ------ -------
0    0     0xffffff7f8dfba6e5 UNKNOWN                0    KERNEL64_CS  com.vmware.kext.vmhgfs    Yes    Yes
0    1     0xffffff800c8cd860 _idt64_debug           0    KERNEL64_CS  __kernel__                No     No
0    2     0xffffff800c8cac40 __intr_0x02            0    KERNEL64_CS  __kernel__                No     No
0    3     0xffffff800c8cac60 _idt64_int3            3    KERNEL64_CS  __kernel__                No     No
0    4     0xffffff800c8cac80 _idt64_into            3    KERNEL64_CS  __kernel__                No     No
0    5     0xffffff800c8caca0 _idt64_bounds          3    KERNEL64_CS  __kernel__                No     No
0    6     0xffffff800c8cacc0 _idt64_invop           0    KERNEL64_CS  __kernel__                No     No
0    7     0xffffff800c8cace0 _idt64_nofpu           0    KERNEL64_CS  __kernel__                No     No
0    8     0xffffff800c8cd0e0 _idt64_double_fault    0    KERNEL64_CS  __kernel__                No     No
0    9     0xffffff800c8cad00 _idt64_fpu_over        0    KERNEL64_CS  __kernel__                No     No
0    10    0xffffff800c8cad20 _idt64_inv_tss         0    KERNEL64_CS  __kernel__                No     No
0    11    0xffffff800c8cd160 _idt64_segnp           0    KERNEL64_CS  __kernel__                No     No
0    12    0xffffff800c8cd140 _idt64_stack_fault     0    KERNEL64_CS  __kernel__                No     No
```

Detecting the Handler Hook

$ python vol.py mac_check_idt -f /Volumes/Storage/HITB/IDTHandlerHook-MountainLion_10_8_3_AMDx64.vmem --profile=MacMountainLion_10_8_3_AMDx64

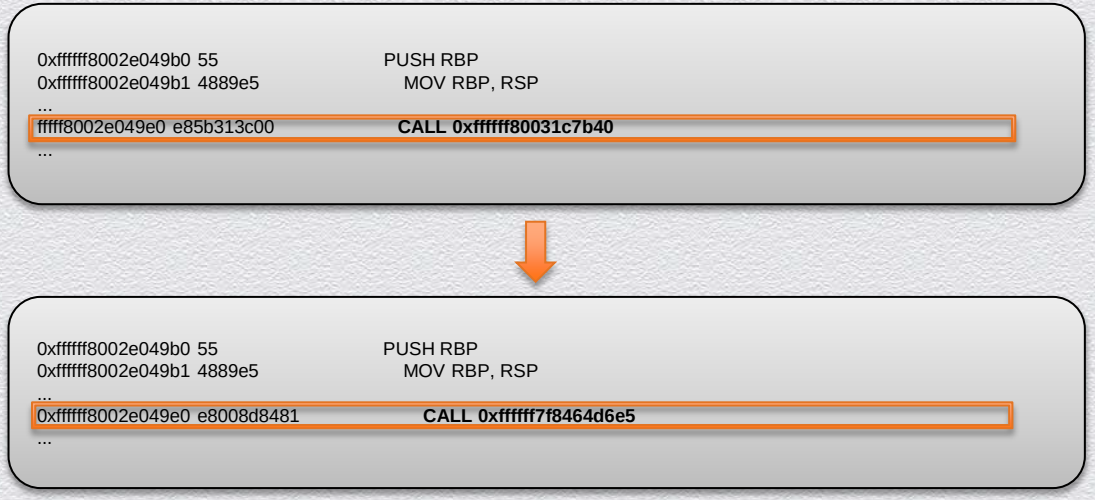
```
Volatile Systems Volatility Framework 2.3_beta
CPU# Index Address            Symbol                Ring Selector     Module                                  Hooked Inlined
---- ----- ------------------ --------------------- ---- ------------ --------------------------------------- ------ ------
0    0     0xffffff802a4cac20 _idt64_zero_div       0    KERNEL64_CS  __kernel__                              No     Yes
0    1     0xffffff802a4cad80 _idt64_debug          0    KERNEL64_CS  __kernel__                              No     No
0    2     0xffffff802a4cac40 __intr_0x02           0    KERNEL64_CS  __kernel__                              No     No
0    3     0xffffff802a4cac60 _idt64_int3           3    KERNEL64_CS  __kernel__                              No     No
0    4     0xffffff802a4cac80 _idt64_into           3    KERNEL64_CS  __kernel__                              No     No
0    5     0xffffff802a4caca0 _idt64_bounds         3    KERNEL64_CS  __kernel__                              No     No
0    6     0xffffff802a4cacc0 _idt64_invop          0    KERNEL64_CS  __kernel__                              No     No
0    7     0xffffff802a4cace0 _idt64_nofpu          0    KERNEL64_CS  __kernel__                              No     No
0    8     0xffffff802a4cd0e0 _idt64_double_fault   0    KERNEL64_CS  __kernel__                              No     No
0    9     0xffffff802a4cad00 _idt64_fpu_over       0    KERNEL64_CS  __kernel__                              No     No
0    10    0xffffff802a4cad20 _idt64_inv_tss        0    KERNEL64_CS  __kernel__                              No     No
0    11    0xffffff802a4cd160 _idt64_segnp          0    KERNEL64_CS  __kernel__                              No     No
0    12    0xffffff802a4cd140 _idt64_stack_fault    0    KERNEL64_CS  __kernel__                              No     No
```

Breaking Volatility?

- fG! once more claims to break Volatility by:
  - Modifying Call References
  - Shadow TrustedBSD/mac_policy_list
  - Hiding from Memory Acquisition… Irrelevant!

Call Reference Modification

- Modified **ps_read_file** function
- Calls **vnode_pagein**
- Redirected call to an address in the kext **com.vmware.kext.vmhgfs**
- Tool? Volatility!



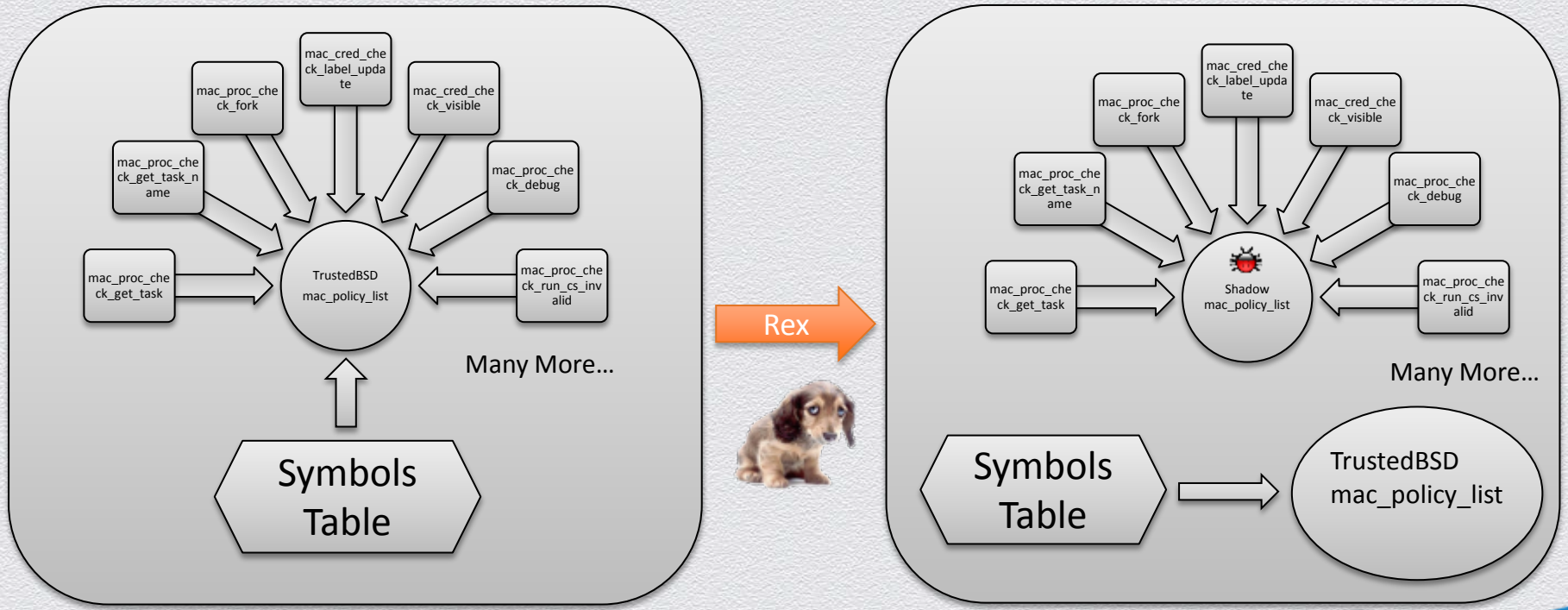
Detecting Call Reference Modification

$ python vol.py mac_check_hooks -f ~/Documents/Virtual\ Machines/Mac\ OS\ X\ 10.8\ 64-bit-DEMO.vmwarevm/564d438d-cc29-2121-3dd6-ac473e701f8d.vmem --profile=MacMountainLion_10_8_5_AMDx64 -K

Volatility Foundation Volatility Framework 2.3.1

Table Name	Index	Address	Symbol	Inlined	Shadowed	Perms	Hook In
SymbolsTable	-	0xffffff8002e049b0	_ps_read_file	Yes	No	-	com.vmware.kext.vmhgfs

Shadow TrustedBSD or mac_policy_list



Detecting Shadow TrustedBSD

- All functions for TrustedBSD include the macro MAC_CHECK
- Not as easy as Shadow Symbols table
- Need to scan all TrustedBSD related functions for referencing
- For Rex scan only **mac_proc_check_get_task**
- Could have used the **mac_policy_list.entries** instead

```
$ python vol.py mac_check_hooks -f ~/Desktop/OMFW-2013/564d438d-cc29-2121-3dd6-ac473e701f8d.vmem --
profile=MacMountainLion_10_8_5_AMDx64

Volatility Foundation Volatility Framework 2.3.1
Table Name    Index  Address          Symbol                    Inlined Shadowed Perms  Hook In
------------ ------ ---------------- ------------------------- ------- -------- ------ ----------------------
mac_policy_address is shadowed! Original Address: 0xffffff8024af4d28, Shadow Address: 0xffffff7fa5c4d6e5, Modification at: 0xffffff802488ee34
```

Conclusion

- DTrace is part of OS X and readily available
- Can be used to detect and create rootkits
- Syscalls and other system functions/structures are easy targets for rootkits
- Memory analysis with Volatility reveals rootkit artifacts
- Detection methods trivially wrapped into a plugin for automation
- If there is no detection mechanism, write a Volatility plugin!

References

- [1] http://felinemenace.org/~nemo/dtrace-infiltrate.pdf
- [2] http://reverse.put.as/wp-content/uploads/2013/05/SysScan-13-Presentation.pdf
- [3] http://www.dtracebook.com
- [4] http://blackhat.com/presentations/bh-usa-08/Beauchamp_Weston/BH_US_08_Beauchamp-Weston_DTrace.pdf
- [5] http://nostarch.com/rootkits.htm
- [6] http://www.opensource.apple.com
- [7] https://github.com/gdbinit/

Questions?

Thank you!

- Blog: siliconblade.blogspot.com
- Code: github.com/siliconblade/
- Twitter: @CGurkok
- E-mail: cemgurkok <at/> gmail.com