whoami

- Security researcher at Cisco in the Talos group
- Ph.D. Telecom ParisTech/Eurecom
- Hackademic
- Malware analysis / memory forensics
“Software to maintain a **persistent** and **stealthy** access on a compromised machine”
HOW?

RING 3
RING 0
RING -1
RING -2
RING -3

PRIVILEGES
HOW?

DETECTION

RING 3
RING 0
RING -1
RING -2
RING -3

PRIVILEGES
HOW?

COMMON ROOTKITS

RING 0
RING -1
RING -2
RING -3
RING 3

DETECTION

PRIVILEGES
HOW?

RING -1

- “Subvirt: Implementing malware with virtual machines“ - S&P 06
- Blue Pill - Joanna Rutkowska - Syscan 06
- Vitriol - Dino Dai Zovi - BHUS 06

RING 0

RING -2

RING -3

DETECTION

PRIVILEGES
Duflot SMM research
“SMM rootkits: A new breed of OS independent malware” - SP 08
“System Management Mode Hacks” - Phrack #65 - ’08
“Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers” - Phrack #66 - ’09
“Implementing SMM PS/2 Keyboard sniffer” - Beist - 2009
NSA
http://blog.cr4.sh/2016/02/exploiting-smm-callout-vulnerabilities.html
**HOW?**

**SOUFFLETHROUGH**

ANT Product Data

(TS/SI/REL) SOUFFLETHROUGH is a BIOS persistence implant for Juniper SSG 500 and SSG 300 series firewalls. It persists DNT’s BANANAGLEE software implant. SOUFFLETHROUGH also has an advanced persistent back-door capability.

Command, Control, and Data Exfiltration using DNT Implant Communications Protocol (typical)

(Target Network)

(TS/SI/REL) SOUFFLETHROUGH Persistence Implant Concept of Operations

(TS/SI/REL) SOUFFLETHROUGH is a BIOS persistence implant for Juniper SSG 500 and SSG 300 series firewalls (320M, 350M, 520, 550, 520M, 550M). It persists DNT’s BANANAGLEE software implant and modifies the Juniper firewall’s operating system (ScreenOS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE’s communications structure, so that full access can be reacquired at a later time. It takes advantage of Intel’s System Management Mode for enhanced reliability and covertness. The PBD is also able to beacon home, and is fully configurable.

(TS/SI/REL) A typical SOUFFLETHROUGH deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. SOUFFLETHROUGH is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

Status: (C/REL) Released. Has been deployed. There are no availability restrictions preventing ongoing deployments.

POC: [redacted] @nsa.ic.gov

Unit Cost: $0

Derived From: NSAICS5

Dated: 20

Declassify On: 20

TOP SECRET//COMINT//REL TO USA, FVEY
HOW?

RING -3

- “Introducing Ring -3 Rootkits” - Tereshkin & Wojtczuk - BHUS’09
- “Understanding DMA Malware” - Stewin et al. - DIMVA ‘12
- http://me.bios.io/Resources
HOW?

- Bootkits
- DKOM
- ROP Rootkits
- Bluepills
- Firmware
HOW?

- BOOTKITS
- DKOM
- ROP ROOTKITS
- BLUEPILLS
- FIRMWARE
ROP ROOTKIT?

- Motivation

- “Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms” - USENIX Security 09

- “Persistent Data-only Malware: Function Hooks without Code” - NDSS ‘14
ROP ROOTKIT?

- Persistence technique:
  - CVE-2013-2094
  - sysenter
    - IA32_SYSENTER_ESP (0x175)
    - IA32_SYSENTER_EIP (0x176)
Chuck ROP chains:

<table>
<thead>
<tr>
<th>CHAIN</th>
<th>INSTRUCTIONS</th>
<th>GADGETS</th>
<th>BLOCKS</th>
<th>BRANCHES</th>
<th>FUNCTIONS</th>
<th>CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY</td>
<td>414,275</td>
<td>184,126</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DISPATCHER</td>
<td>63,515</td>
<td>28,874</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>PAYLOAD</td>
<td>6320</td>
<td>2913</td>
<td>34</td>
<td>26</td>
<td>9</td>
<td>17</td>
</tr>
</tbody>
</table>
DKOM

“Direct Kernel Object Manipulation”
TRADITIONAL DKOM
TRADITIONAL DKOM
DKOM vs PROCESSES

- DKOM is a generic technique
- Processes:
  - Windows: KPROCESS/EPROCESS/PEB
  - Linux: task_struct
  - OSX: proc/task
(E)PROCESS?

```
lkd> dt nt!_EPROCESS
+0x0000 Pcb                  : _KPROCESS
+0x160 ProcessLock          : _EX_PUSH_LOCK
+0x168 CreateTime           : _LARGE_INTEGER
+0x170 ExitTime             : _LARGE_INTEGER
+0x178 RundownProtect       : _EX_RUNDOWN_REF
+0x180 UniqueProcessId      : Pt64 _uid
+0x188 ActiveProcessLinks   : _LIST_ENTRY
+0x198 ProcessQuotaUsage    : [2] Uint8B
+0x1a8 ProcessQuotaPeak     : [2] Uint8B
+0x1b8 CommitCharge         : Uint8B
+0x1c0 QuotaBlock           : Ptr64 _EPROCESS_QUOTA_BLOCK
+0x1c8 CpuQuotaBlock        : Ptr64 _PS_CPU_QUOTA_BLOCK
+0x1d0 PeakVirtualSize      : Uint8B
+0x1d8 VirtualSize          : Uint8B
+0x1e0 SessionProcessLinks  : _LIST_ENTRY
+0x1f0 DebugPort            : Ptr64 Void
+0x1f8 ExceptionPortData    : Ptr64 Void
+0x1f8 ExceptionPortValue   : Uint8B
+0x1f8 ExceptionPortState   : Pos 0, 3 Bits
+0x200 ObjectTable          : Ptr64 _HANDLE_TABLE
+0x208 Token                : _EX_FAST_REF
+0x210 WorkingSetPage       : Uint8B
+0x218 AddressCreationLock  : _EX_PUSH_LOCK
+0x220 RotateInProgress     : Ptr64 _ETHREAD
+0x228 ForkInProgress       : Ptr64 _ETHREAD
+0x230 HardwareTrigger      : Uint8B
+0x238 PhysicalVadRoot      : Ptr64 _MM_AVL_TABLE
+0x240 CloneRoot            : Ptr64 Void
```

Winter 2018 Edition
(E)PROCESS?
Process

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>Header</td>
<td>_DISPATCHER_HEADER</td>
</tr>
<tr>
<td>0x018</td>
<td>ProcessLock</td>
<td>Uint8B</td>
</tr>
<tr>
<td>0x028</td>
<td>DirectoryTableBase</td>
<td>LIST_ENTRY</td>
</tr>
<tr>
<td>0x040</td>
<td>ProcessLock</td>
<td>Uint8B</td>
</tr>
<tr>
<td>0x048</td>
<td>Affinity</td>
<td>_KAFFINITY_EX</td>
</tr>
<tr>
<td>0x070</td>
<td>ReadyListHead</td>
<td>_LIST_ENTRY</td>
</tr>
<tr>
<td>0x088</td>
<td>ActiveProcessors</td>
<td>_SINGLE_LIST_ENTRY</td>
</tr>
<tr>
<td>0x0b0</td>
<td>AutoAlignment</td>
<td>Pos 0, 1 Bit</td>
</tr>
<tr>
<td>0x0b0</td>
<td>DisableBoost</td>
<td>Pos 1, 1 Bit</td>
</tr>
<tr>
<td>0x0b0</td>
<td>DisableQuantum</td>
<td>Pos 2, 1 Bit</td>
</tr>
<tr>
<td>0x0b0</td>
<td>ActiveGroupsMask</td>
<td>Pos 3, 4 Bits</td>
</tr>
<tr>
<td>0x0b0</td>
<td>ReservedFlags</td>
<td>Pos 7, 25 Bits</td>
</tr>
<tr>
<td>0x0b0</td>
<td>ProcessFlags</td>
<td>Int4B</td>
</tr>
<tr>
<td>0x0b4</td>
<td>BasePriority</td>
<td>Char</td>
</tr>
<tr>
<td>0x0b5</td>
<td>QuantumReset</td>
<td>Char</td>
</tr>
<tr>
<td>0x0b6</td>
<td>Visited</td>
<td>UChar</td>
</tr>
<tr>
<td>0x0b7</td>
<td>Unused3</td>
<td>UChar</td>
</tr>
<tr>
<td>0x0b8</td>
<td>ThreadSeed</td>
<td>[4] Uint4B</td>
</tr>
<tr>
<td>0x0c8</td>
<td>IdealNode</td>
<td>[4] Uint2B</td>
</tr>
<tr>
<td>0x0d0</td>
<td>IdealGlobalNode</td>
<td>Uint2B</td>
</tr>
<tr>
<td>0x0d2</td>
<td>Flags</td>
<td>_KEKECUTE_OPTIONS</td>
</tr>
<tr>
<td>0x0d3</td>
<td>Unused1</td>
<td>UChar</td>
</tr>
<tr>
<td>0x0d4</td>
<td>Unused2</td>
<td>Uint4B</td>
</tr>
<tr>
<td>0x0d8</td>
<td>Unused4</td>
<td>Uint4B</td>
</tr>
<tr>
<td>0x0dc</td>
<td>StackCount</td>
<td>_KSTACK_COUNT</td>
</tr>
<tr>
<td>0xe0</td>
<td>ProcessListEntry</td>
<td>_LIST_ENTRY</td>
</tr>
<tr>
<td>0xf0</td>
<td>CycleTime</td>
<td>Uint8B</td>
</tr>
<tr>
<td>0xf8</td>
<td>KernelTime</td>
<td>Uint4B</td>
</tr>
<tr>
<td>0xfc</td>
<td>UserTime</td>
<td>Uint4B</td>
</tr>
<tr>
<td>100</td>
<td>InstrumentationCallback</td>
<td>Ptr64 Void</td>
</tr>
<tr>
<td>108</td>
<td>LdtSystemDescriptor</td>
<td>_KGDENTRY64</td>
</tr>
<tr>
<td>110</td>
<td>LdtBaseAddress</td>
<td>Ptr64 Void</td>
</tr>
<tr>
<td>120</td>
<td>LdtProcessLock</td>
<td>_KGUARDED_MUTEX</td>
</tr>
<tr>
<td>158</td>
<td>LdtFreeSelectorHint</td>
<td>Uint2B</td>
</tr>
<tr>
<td>15a</td>
<td>LdtTableLength</td>
<td>Uint2B</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>InheritedAddressSpace</td>
<td>UChar</td>
<td></td>
</tr>
<tr>
<td>ReadImageFileExecutionOptions</td>
<td>UChar</td>
<td></td>
</tr>
<tr>
<td>BeingDebugged</td>
<td>UChar</td>
<td></td>
</tr>
<tr>
<td>BitField</td>
<td>UChar</td>
<td></td>
</tr>
<tr>
<td>ImageUsesLargePages</td>
<td>Pos 0, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>IsProtectedProcess</td>
<td>Pos 1, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>IsLegacyProcess</td>
<td>Pos 2, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ImageDynamicallyRelocated</td>
<td>Pos 3, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>SkipPatchingUser32Forwards</td>
<td>Pos 4, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>SpareBits</td>
<td>Pos 5, 3 Bits</td>
<td></td>
</tr>
<tr>
<td>Mutant</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>ImageBaseAddress</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>Ldr</td>
<td>Ptr64 __PEB_LDR_DATA</td>
<td></td>
</tr>
<tr>
<td>ProcessParameters</td>
<td>Ptr64 _RTL_USER_PROCESS_PARAMETERS</td>
<td></td>
</tr>
<tr>
<td>SubSystemData</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>ProcessHeap</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>FastPebLock</td>
<td>Ptr64 _RTL_CRITICAL_SECTION</td>
<td></td>
</tr>
<tr>
<td>AllThreadStubs</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>IFEOKey</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>CrossProcessFlags</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>ProcessInJob</td>
<td>Pos 0, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ProcessInitializing</td>
<td>Pos 1, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ProcessUsingVEH</td>
<td>Pos 2, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ProcessUsingVCH</td>
<td>Pos 3, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ProcessUsingFTH</td>
<td>Pos 4, 1 Bit</td>
<td></td>
</tr>
<tr>
<td>ReservedBits0</td>
<td>Pos 5, 27 Bits</td>
<td></td>
</tr>
<tr>
<td>KernelCallbackTable</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>UserSharedInfoPtr</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>SystemReserved</td>
<td>[1] Uint4B</td>
<td></td>
</tr>
<tr>
<td>AllThunkSlListPtr</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>ApiSetMap</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>TlsExpansionCounter</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>TlsBitmap</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>TlsBitmapBits</td>
<td>[2] Uint4B</td>
<td></td>
</tr>
<tr>
<td>ReadOnlySharedMemoryBase</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>HotpatchInformation</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>ReadOnlyStaticServerData</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>AnsiCodePageData</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>OemCodePageData</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>UnicodeCaseTableData</td>
<td>Ptr64 Void</td>
<td></td>
</tr>
<tr>
<td>NumberOfProcessors</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>MtlGlobalFlag</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>CriticalSectionTimeout</td>
<td>_LARGE_INTEGER</td>
<td></td>
</tr>
<tr>
<td>HeapSegmentReserves</td>
<td>Uint8</td>
<td></td>
</tr>
<tr>
<td>HeapSegmentComit</td>
<td>Uint8B</td>
<td></td>
</tr>
<tr>
<td>HeapDeCommitTotalFreeThreshold</td>
<td>Uint8B</td>
<td></td>
</tr>
<tr>
<td>HeapDeComitFreeBlockThreshold</td>
<td>Uint8B</td>
<td></td>
</tr>
<tr>
<td>NumberOfHeaps</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>MaximumNumberOfHeaps</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>ProcessHeaps</td>
<td>Pptr64 Pptr64 Void</td>
<td></td>
</tr>
<tr>
<td>GdiSharedHandleTable</td>
<td>Pptr64 Void</td>
<td></td>
</tr>
<tr>
<td>ProcessStarterHelper</td>
<td>Pptr64 Void</td>
<td></td>
</tr>
<tr>
<td>GdiDCAtributeList</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>LoaderLock</td>
<td>Pptr64 _RTL_CRITICAL_SECTION</td>
<td></td>
</tr>
<tr>
<td>OSMajorVersion</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>OSMinorVersion</td>
<td>Uint4B</td>
<td></td>
</tr>
<tr>
<td>OSBuildNumber</td>
<td>Uint2B</td>
<td></td>
</tr>
<tr>
<td>OSWindowsVersion</td>
<td>Uint2B</td>
<td></td>
</tr>
<tr>
<td>OSPagefiled</td>
<td>Uint2B</td>
<td></td>
</tr>
<tr>
<td>OSKernel32</td>
<td>Uint2B</td>
<td></td>
</tr>
<tr>
<td>OSImage</td>
<td>Uint2B</td>
<td></td>
</tr>
</tbody>
</table>
PROCESS?

- EPROCESS info:
  - Creation and exit time
  - PID and PPID
  - Pointer to the handle table
  - VAD, etc

- PEB info:
  - Pointer to the Image Base Address
  - Pointer to the DLLs loaded
  - Heap size, etc
DKOM DEFENSES

- Kernel data integrity solutions:
  - invariants
    - external systems
    - memory analysis
  - data partitioning
def plist(addr_space):
    """ A Generator for _EPROCESS objects ""
    for p in get_kdbg(addr_space).processes():
        yield p

def processes(self):
    """Enumerate processes""
    # This is defined as a pointer to _LIST_ENTRY in the overlay
    list_head = self.PsActiveProcessHead.dereference()
    if not list_head:
        raise AttributeError("Could not list tasks, please verify your --profile with kdbgscan")
    for l in list_head.list_of_type("_EPROCESS", "ActiveProcessLinks"):
        yield l
DEMO

“DKOM DEMO”
E-DKOM

“Evolutionary Direct Kernel Object Manipulation”

“Subverting Operating System Properties through Evolutionary DKOM Attacks”
Mariano Graziano, Lorenzo Flore, Andrea Lanzi, Davide Balzarotti
DIMVA 2016, San Sebastian, Spain
Violation of a \textit{temporal} property
The attack cannot be detected looking at a single snapshot
STATE vs PROPERTY

- Traditional DKOM affects the state and are discrete
- Evolutionary DKOM (E-DKOM) affects the evolution in time of a given property and are continuous
LINUX CFS SCHEDULER

```c
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}

struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```c
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}

struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```
LINUX CFS SCHEDULER

```c
struct task_struct {
    volatile long state;
    void *stack;
    unsigned int flags;
    int prio, static_prio, normal_prio;
    const struct sched_class *sched_class;
    struct sched_entity se;
    ...
}

struct sched_entity {
    struct load_weight load;
    struct rb_node run_node;
    struct list_head group_node;
    ...
}

struct cfs_rq {
    ...
    struct rb_root tasks_timeline;
    ...
}

struct rb_node{
    unsigned long rb_parent_color;
    struct rb_node *rb_right;
    struct rb_node *rb_left;
};
```

`target`
LINUX CFS SCHEDULER
LINUX CFS SCHEDULER

struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}

struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};

struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}

struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  ...
}

Set target_{vruntime} > rightmost_{vruntime}
We affect the evolution of the data structure over time. We altered the scheduler property (fair execution).
DEMO

“E-DKOM DEMO”
DEFENSES?

- Reference monitor that mimics the OS property:
  - OS specific
  - Difficult to generalize
DEFENSE FRAMEWORK

CFS rb-tree

27

19

7

25

31

34

99

Operating system

Hypervisor

Periodic VMexit

VMexit on scheduler events

Update tasks info

Periodic Monitor

Inferred Task List

ALERT!

CFS ATTACK

Raise target virtual runtime
FUTURE

- Minimalism

- Possible trends:
  - Infections for the masses
  - Stealthy and multi stage attacks

- Cat and mouse game

- Microsoft approach:
  - Credential Guard
  - Application Guard
CONCLUSION

- Rootkit technology evolution
- New attack based on data structure evolution
- Experiment on the Linux CFS scheduler
- Defense based on hypervisor
- General mitigation/solution very hard
THE END

THANK YOU

email: magrazia@cisco.com
twitter: @emd3l