Towards an Invisible
Honeypot Monitoring Tool

HITB06

Nguyen Anh Quynh
<aquynh –at- gmail com>
Keio university, Japan
Who am I?

- Nguyen Anh Quynh, a PhD student of Takefuji-lab, Keio university, Japan
- Interests: Network/Computer Security, Operating system, Robust system, Virtualization
- Non-geek hobby: traveling, reading and playing soccer
**Motivation**

- **Sebek** is a de-facto data capture tool of honeynet architecture
- But there are various ways to defeat **Sebek** because **Sebek** is not "invisible" enough
- **Xebek** is our solution on Xen Virtual Machines to address **Sebek**'s problems
  - More "invisible"
  - More flexible
  - Better performance
Overview

- Honeynet architecture and Sebek
- Sebek's problems
- Xebek comes to rescue
  - Introduction to Xen Virtual Machine
  - Xebek architecture & implementation
  - Demonstration
- Q & A
Part I

- Honeynet architecture and Sebek
  - Honeypot introduction
  - Honeynet architecture
  - Sebek technology
Honeypot technology

What is a honeypot?

- The information system resource whose value lies in unauthorized or illicit use of that resource
- Has no production value, anything going in/out the honeypot is likely a probe/attack/compromise
- Primary value to most organizations is information
Honeypot impact

**Advantage**
- High valuable data
- Reduce false positives
- Catch new attacks (0-day bug?) & false negatives

**Disadvantage**
- Limited view
- Risk of take over
Honeypot types

Categorized based on level of interaction

**Low-interaction**
- Emulate services, applications, OSes
  - Low risk and easy to deploy/maintain
  - **But** capture limited information

  Honeyd

**High-interaction**
- Real services, application, OSes
  - Capture extensive information
  - **But** high risk and hard to maintain

  Honeynet
How honeynet works

- A highly controlled network where every packet entering or leaving is monitored, captured and analyzed.
Honeynet components

2 key components

- Data capture
- Data logging & analysis
Data capture

- Capture activities at various levels
  - Application
  - Network
  - OS level
Data analysis

- Manage and analysis captured data from honeypots
  - Investigate malware
  - Forensic purpose
Honeynet generations

- Gen I
- Gen II, Gen III (currently)
  - radical change in architecture focuses on the data capture tool

**Sebek** as a data capture tool
Sebek: a data capture tool

- Born in Honeynet Gen II
- Play a key role in Honeynet architecture
- Gen III (currently)
Sebek architecture

- **Sebek components**
  - Data capture tool
  - Central logging server

Capture host based data: Syslogs and Sebek

Capture network based data: packet captures, IDS, Firewall, OS Fingerprints, and netflow.
Sebek client technique

- Data capture tool: patches system-calls (open/fork/read/write/socket)
- Send out gathered data via network-stack (UDP protocol)
Sebek features

- **hidden** kernel module
- **dumps activity to the network** via UDP protocol to a central logging machine
- fool attacker by modifying network stack, so **Sebek traffic is invisible** (well, almost!)
Part 2

Current problems of Sebek

- Easy to identify
- How easy it is?
  - Possible even with unprivileged user
- How?
- 7 methods to defeat Sebek
Sebek client requirement

- Most vital requirement for a data capture tool: Function as covert as possible  \( \Rightarrow \) Invisible problem
  - Otherwise, game over
    - No more chance to watch out the attacker
    - No more chance to catch 0-day bug (daydream?)
    - Attacker can destroy the honeypot
    - Who fools who then?
But can **Sebek** deliver?

- Hmm, not really. Various ways to defeat **Sebek**
  - 1. Can be discover by even unprivileged user
  - 2. Network statistics disclose **Sebek**
  - 3. Brute-force scanning method
  - 4. System-call address checking
  - 5. Remove **Sebek** is feasible
  - 6. Sniff at the right place
  - 7. Bring down the central logging server
Method (1)

- **Sebek** can be discovered by even unprivileged users.
  - dd-attack
    - Generate lots of data
      ```bash
      dd if=/dev/zero of=/dev/null bs=1
      ```
    - Check to see if network congestion?

 Why?

Network stack is employed to send data out.
Method (2)

Network statistics disclose Sebek

- Compare `dev->getstats()->tx_bytes` with value found in `/proc/net/dev`

Why?

Network stack is employed to send data out
Method (3)

- Brute-force scanning method to detect hidden kernel module
  - Look for hidden kernel module
  - **linux-airt** tool by madsys

Why?

Sebek is implemented as a (hidden)kernel module
Method (4)

- System-call addresses checking
  - System-call addresses at abnormal places?

Why?

Sebek replaces original system-calls
Method (5)

- Remove Sebek is feasible
  - (unsebek.c by J.Corey)
  - Look for the `sys_call_table` by scanning in memory
  - Recover original system-call with exported symbols

Why?

Sebek replaces original system-calls
Method (6)

- Detect honeypot with Sebek
  - Sniff at the right place from outside

Why?
Sebek sends data out to other machine

Attacker with sniffer
sebekd

Sebek client
UDP packets
Method (7)

- Bring down the central logging server
  - Data logging server (sebekd) has vulnerable libpcap?

Why?

sebekd exposed to network

Sebekd with buggy libpcap
Reasons make **Sebek** sux 😊

1. Uses network stack to send data out
2. Logging data sent out can be sniffed online
3. Function as kernel module + replace original system-calls
4. Central logging server (**sebekd**) exposed to the network
5. Data transfer might not be reliable (UDP)
Do you still think that current honeynet can fool skillful hackers?

- I seriously doubt that!
- Should we give up?
- No, let's keep fighting and raise the bar a little bit ;-)
Part 3

Xebek comes to rescue

- Virtual honeypot on virtual machine
- Xen Virtual Machine technology
- Xebek solution
Fix *Sebek*'s problems

- Bring up virtual machine technology: Xen
- Exploit the advantage introduced by Xen to address discussed problems
Xen 3.0 Architecture
Xen's main components

- Xen hypervisor runs on top of hardware
- Domains with modified kernel for Xen architecture, run on top of Xen
- Special device drivers in Dom0 & DomU (backend-frontend architecture)
- Xen control tools in Dom0 (*xend, xm*)
- Others: *xenbus, xenstore, event-channel, balloon driver,* ...
x86 CPU virtualization

- Xen runs in ring 0 (most privileged)
- Ring 1/2 for guest OS, 3 for user-space
  - GPF if guest attempts to use privileged instr
- Xen lives in top 64MB of linear addr space
  - Segmentation used to protect Xen as switching page tables too slow on standard x86
- Hypercalls jump to Xen in ring 0
MMU Micro-Benchmarks

![Graph showing page fault and process fork times for Linux (L), Xen (X), VMWare Workstation (V), and UML (U).]

Imbench results on Linux (L), Xen (X), VMWare Workstation (V), and UML (U)
Xen's future: Bright

- Xen 3.0 was released at the end of 2005
- Object: to be gradually merged into Linux kernel in 2006
- Already adopted by ISPs, datacenters, banks,...
- Will be widely used in the near future
Xen-based honeynet

Domain-0

Device Manager & Control s/w

GuestOS (XenLinux)

Back-End

Native Device Driver

Unmodified User Software

GuestOS (XenLinux)

SMP

Front-End Device Drivers

Unmodified User Software

GuestOS (XenLinux)

Front-End Device Drivers

Unmodified User Software

GuestOS (WinXP))

Front-End Device Drivers

Control IF

Safe HW IF

Event Channel

Virtual CPU

Virtual MMU

Xen Virtual Machine Monitor

Hardware (SMP, MMU, physical memory, Ethernet, SCSI/IDE)
Xebek solution for Xen-based honeynet

- **Xebek**: Goals and approaches
- **Xebek** Architecture
- **Xebek** Implementation’s issues
- **Xebek** Evaluation
- Hardening Xebek
- Detecting Xebek
Xebek goals and approaches

(1) Capture data as Sebek does, but with some improvements
(2) Eliminate problems of leaving too many traces when forwarding data out
(3) Harden the central logging server
Goal (1)

- Capture data as Sebek does, but with some improvements
  - Sebek3 captures data by intercepting system-calls (read/write/open/fork/socket)
  - So Xebek does.
  - But Xebek patches the system-calls, so Xebek does not run as a kernel module

(1) Uses network stack to send data out
(2) Data can be sniffed

(3) Function as KLM & replace original system-calls

(4) Central logging server exposed to the network
(5) Data transfer might not be reliable (UDP)
Goal (2)

- Eliminate problems of leaving too many traces when forwarding data out
  - **Xebek** does not use network stack to deliver data as **Sebek** does
  - Using **shared memory** between DomU and Dom0 instead to exchange data

(1) Uses network stack to send data out
(2) Logging data can be sniffed online
(3) Function as KLM & replace original system-calls
(4) Central logging server exposed to the network
(5) Data transfer might not be reliable (UDP)
Goal (3)

- Harden the central logging server
  - Put the central logging server in Dom0 to pick up data forwarded from DomU
  - No more exposed to the network

(1) Uses network stack to send data out
(2) Data can be sniffed
(3) Function as KLM & replace original system-calls

**4) Central logging server exposed to the network**

(5) Data transfer might not be reliable (UDP)
Xebek architecture
Xebek component in DomU's kernel

- patch the system-calls (open/read/write/fork/socket)
- establish shared memory with Dom0
- put the gathered data from system-calls to shared-memory, then notify xebekd
xebekd

- logging recorder in Dom0
  - waits for notification from xebekU
  - pick up data in shared-memory, then save to corresponding logging file
  - notify xebekU on completion
Xebek utilities

Diagram showing the flow of data from xebekd to a logging files system, then to a keystroke extractor, data uploader, SQL database server, and finally to data analysis (Walleye).
Implementation issues

Shared memory structure
- Need to be accessed at the same time by 2 parties
  - xebekU writes to shared memory
  - xebekd reads from shared memory

ring buffer format
Ring buffer format

Buffer with 2 heads

- **Write head**: fill up buffer
- **Read head**: reallocate buffer space

```c
struct ringbuf {
    u32 write; /* write head */
    u32 read; /* read head */
    u32 size; /* buffer size */
    char buf[0];
} __attribute__((packed));
```
**xebekd**: multiple threading

- **main thread**
- **worker thread**
- **host_ring**: ring buffer structure
Coding

- Version 0.2 – Linux based DomU only ATM
  - Kernel patch
  - Kernel module is also available (NOT encourage!)
- xebekd + xebeklive + xkeys: 1676 lines
- xebekU: 1848 lines (linux-2.6.16-rc2)
  - Small increase in kernel binary size
    - 946550 bytes -> 948494 bytes
  - Small patch to kernel

<table>
<thead>
<tr>
<th>File name</th>
<th>Modified lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel/fork.c</td>
<td>54</td>
</tr>
<tr>
<td>fs/open.c</td>
<td>21</td>
</tr>
<tr>
<td>fs/read_write.c</td>
<td>148</td>
</tr>
<tr>
<td>net/socket.c</td>
<td>44</td>
</tr>
</tbody>
</table>
#ifdef CONFIG_XEN_XEBEK
    struct xebek_packet p;
    if (my_private.active) {
        p.event = EVT_FORK;
        fill_time(&p.time);
        p.size = sizeof(current->comm);
        p.version = XEBEK_VERSION;
        p.magic = XEBEK_MAGIC;
        p.uid = current->uid;
        p.ppid = current->parent->pid;
        p.pid = current->pid;
        copy_to_buffer(&p, current->comm, p.size, 0);
    }
#endif
Compile Configuration
### Xebek evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Native</th>
<th>Sebek</th>
<th>Xebek</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>8.194</td>
<td>1509.073 (≈184 times)</td>
<td>9.720 (18.62%)</td>
</tr>
<tr>
<td>READ</td>
<td>1.221</td>
<td>972.649 (≈976 times)</td>
<td>1.968 (61.13%)</td>
</tr>
<tr>
<td>WRITE</td>
<td>1.106</td>
<td>1.113 (-)</td>
<td>1.822 (64.69%)</td>
</tr>
<tr>
<td>FORK</td>
<td>900.380</td>
<td>900.433 (≈0%)</td>
<td>900.421 (≈0%)</td>
</tr>
<tr>
<td>TCP</td>
<td>842.256</td>
<td>1276.562 (51.56%)</td>
<td>1004.912 (19.31%)</td>
</tr>
<tr>
<td>UDP</td>
<td>1050.991</td>
<td>1100.262 (4.68%)</td>
<td>1085.241 (3.25%)</td>
</tr>
</tbody>
</table>

LMBench benchmark results
Hardening Xebek

- **Harden DomU:**
  - Protect kernel binary? No need 😊
  - Protect kernel symbol? No need 😊
  - Shutdown all the paths to the kernel
    - No kernel module loading
    - `/dev/{kmem, mem, port}` removed

- **Harden Dom0**
  - Harden system (SELinux, LIDS, AppArmor)
  - Run Dom0 with no network access
Detecting **Xebek**

- **Intruder gains kernel access?**
  - We are hopeless against brute-force scanning kernel memory
  - Block all path to kernel.

- **Intruder has no kernel access?**
  - Timing attack based on syscall latency?
  - Impossible to solve completely !!! 😞

- Removing kernel access might be suspicious !!!
Demonstration
Future work

- Analysis tool: Adapt Walleye for Xebek
- Maintenance Xebek patch for different kernel versions (costly?)
- Make Xebek more flexible
  - Adapt Xebek to the Sebek scheme
  - Optimize to reduce latency
  - Port Xebek to other platforms like *BSD/Solaris/...
Conclusions

Xebek is a robust data capture tool for Xen-based virtual honeypot

- More “invisible”
- More reliable/flexible
- Open source: To be released under GPL licence soon (when I have more free time 😞)
Towards an Invisible Honeypot Monitoring Tool

HITB06

Nguyen Anh Quynh
<aquynh –at- gmail com>
Keio university, Japan

Thank you!

Questions/Comments?