AHA - Adaptive Honeypot Alternative

Gérard Wagener

October 29, 2010
Introduction

Related work

- Honeypots are resources designed to be under attack [5]
- End eighties / early nineties first experiments by Clifford Stoll [6], Steven Bellovin [1] and Bill Cheswick [2]
- They mainly reported how they trapped attackers and the related activities
- In 1998 Fred Cohen discussed the deception techniques that can be used while dealing with attackers [3]
- Lance Spitzner writes that honeypots are particularly useful to learn from attackers
- Jose Antonio Coret re-implemented an SSH server in python as honeypot [4]
## Attack Scenario

<table>
<thead>
<tr>
<th>Step</th>
<th>Attacker</th>
<th>Honypot</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SSH connect</td>
<td></td>
<td>Attacker penetration</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Returns shell</td>
<td>Full access</td>
</tr>
<tr>
<td>2</td>
<td>id</td>
<td></td>
<td>System identification</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Execute id</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uname</td>
<td></td>
<td>System identification</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Execute uname</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ps aux</td>
<td></td>
<td>Already compromised?</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Execute ps</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>wget $URL_0$</td>
<td></td>
<td>Acquire tool</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Execute wget</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>./ssh − brute</td>
<td></td>
<td>Misuse the system</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Return error</td>
<td>Strategical block</td>
</tr>
<tr>
<td>12</td>
<td>wget $URL_1$</td>
<td></td>
<td>Additional tool</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Execute wget</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>./configure</td>
<td>Build attacker tool</td>
<td>Source code</td>
</tr>
<tr>
<td>15</td>
<td>allow</td>
<td></td>
<td>Make attacker happy</td>
</tr>
</tbody>
</table>
Introduction

Contribution

- Create framework serving as building blocks for adaptive honeypots
- Optimize information retrieval from attackers (skills, tools, used time, social background, used language)
- Based on a Linux operating system exposing a vulnerable SSH server
Adaptation mechanisms

- Allow the execution of a program
  - Behave like a regular high-interaction honeypot
  - Do not interfere with the execution flow
- Block the execution of a program
  - Strategically block the execution of a program
  - Challenge the attacker
- Substitute the executed program
  - Make attacker believe that they downloaded the wrong program
  - Make attacker believe that their repository is not available
- Insult the attacker
  - Irritate attacker
  - Reveal his ethic background
  - Differentiate between automated attacks and human attackers
  - See if attackers bounce through compromised hosts
AHA framework - Overview

- **Attacker**
- **Internet**
- **Host Operating System (User space)**
- **Kernel**
- **User Mode Linux**
- **SSHD**
- **AHAD**
- **Output**
- **Input**
- **AHA Eye**
- **AHA Worker**
- **Report**
- **Log file**
Components interaction

- Linux system call hooks
  - `sys_execve`
  - `sys_clone`
  - `sys_exit`

- Send messages to AHA daemon

- A decision must be taken (not included in the framework)

- Exchange Messages
  - Export message → export kernel information to the daemon
  - Reply message → decision taken by the daemon
  - Export and reply messages are tightly linked → unique message identifier
Components interaction

type=1
file=/usr/bin/vi
argument=vi
env=TERM=screen
env=SSH_CLIENT=192.168.1.2 41836 22
env=SSH_TTY=/dev/pts/0
env=USER=gabriela
env=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
env=LANG=en_US.UTF-8
env=HISTCONTROL=ignoreboth
env=SHLVL=1
env=HOME=/home/gabriela
env=LOGNAME=gabriela
env=SSH_CONNECTION=192.168.1.2 41836 192.168.1.1 22
env=_=/usr/bin/vi
pid=1100
ppid=1075
rppid=1075
DONE=1
struct ReplyMessage{
  int block;
  int substitute;
  int insult;
};
Building an UML from a vanilla kernel

1. make defconfig ARCH=um
2. make ARCH=um

Modified kernel files

<table>
<thead>
<tr>
<th>File</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch/um/kernel/exec.c</td>
<td>sys_execve</td>
</tr>
<tr>
<td>arch/um/kernel/process.c</td>
<td>exit_thread</td>
</tr>
<tr>
<td>arch/um/sys-i386/syscalls.c</td>
<td>sys_clone</td>
</tr>
<tr>
<td>os-Linux/main.c</td>
<td>__init</td>
</tr>
</tbody>
</table>

Purpose: Export program execution data and let the daemon take the decisions
Components description
User Mode Linux surgeries

Sys_execve hook

```c
long sys_execve(char __user *file, char __user **argv,
                char __user **env)
{
    long error;
    char *filename;
    struct ReplyMessage msg;
    filename = aha_dump_execve(file,argv,env);
    if (filename){
        aha_get_reply_message(filename,&msg);
        kfree(filename);
        /* Implement decisions taken by AHA */
        if (msg.block) {
            error = msg.block;
            goto out;
        }
        if (msg.insult) {
            aha_handle_insult_messages(&msg,file,argv);
        }else {
            if (msg.substitute) {
                aha_handle_substitutes(&msg,file,argv);
            }
        }
    }
}
```
Components description

AHA daemon

Operation

▶ Read messages initiated by the User Mode Linux
▶ Is the program execution related to an attacker?
▶ Take a decision and put it in the input queue

Code Organization

▶ AHAActions → core functions to interact with the User Mode Linux
▶ KERNEL_ERRORS → Strategical blocking (taken from the Kernel Source)
▶ ReplyMessage → Create a binary reply message for the User Mode Linux
▶ ProcessTree → Maintain in the daemon a clone of the process tree of the UML
Components description

AHA Worker

- Execution performance is critical
- AHA daemon only takes decisions
- AHA Worker periodically polls the queues
- Merges messages in a log file
- Avoid overfilled queues
Components description

AHA Eye

- Monitoring is essential for honeypot operation
- Human readable form is desired
- AHA Eye uses the log file from AHA Worker
- Creates a report → attacker’s bash session
Is the program related to an attacker or the system?

Legend
- Program name
- PID
- Command line argument

Classified programs

<table>
<thead>
<tr>
<th>∑ ts</th>
<th>Program name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sshd</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Is the program related to an attacker or the system?

Legend
- Blue: Program name
- Orange: PID
- Green: Command line argument

Classified programs

<table>
<thead>
<tr>
<th>∑ ts</th>
<th>Program name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sshd</td>
</tr>
<tr>
<td>2</td>
<td>bash</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Attacker 1 \((A_1)\)
Is the program related to an attacker or the system?

Legend
- Blue: Program name
- Orange: PID
- Green: Command line argument

Classified programs

<table>
<thead>
<tr>
<th>$\sum ts$</th>
<th>Program name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sshd</td>
</tr>
<tr>
<td>2</td>
<td>bash</td>
</tr>
<tr>
<td>5</td>
<td>uname</td>
</tr>
</tbody>
</table>

Attacker 1 ($A_1$)
Is the program related to an attacker or the system?

Attacker 1 ($A_1$)

Legend
- Program name
- PID
- Command line argument

Classified programs

<table>
<thead>
<tr>
<th>$\sum ts$</th>
<th>Program name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sshd</td>
</tr>
<tr>
<td>2</td>
<td>bash</td>
</tr>
<tr>
<td>5</td>
<td>uname</td>
</tr>
</tbody>
</table>

$\vec{A}_1 = \langle sshd, bash, uname \rangle$
Insulting the attacker

```c
void aha_handle_insult_messages(struct ReplyMessage *msg,
                                  char __user* file,
                                  char __user* __user* argv)
{
  char buf[16];
  char* addr;
  int cnt;

  if(!copy_to_user(file,"/sbin/insult",13)){
    cnt = snprintf((char*)&buf,16,"%d",msg->insult);
    if ((cnt > 0) && (cnt<15))
      buf[cnt+1]=0;
    if (!get_user(addr,argv))
      copy_to_user(addr,buf,cnt+1);
  }
}

Substituting programs works similarly
```
Gathering insults from an attacker

Problem

- Insults = invalid programs
- Handled by Bash

Solution (ugly)

- Hook bash using the `NOTFOUND_HOOK`
- Use helper application that just accepts the arguments
- When the helper application is started a `sys_execve` is induced
- This is then visible for the AHA daemon
Case Studies
Example Session (94.52.64.x username: test)

```
w
  .. scbrute.tar .wp
w
18:28:21 up 6:46, 1 user, load average: 0.15, 0.03, 0.01
bash
I dont wanna do that
sh
wget http://www.dragutrau.xxx.su/xxx/yyy
I love you
kill -9 1
Core dumped
l
  .. scbrute.tar .wp
fetch
fuck you
exit
```
Case Studies

Experiment #1

Adaptive Honeypot vs High-Interaction Honeypot

![Graph showing process vector length distribution for high-interaction honeypot and adaptive honeypot.](image)
# Case Studies

## Experiment #2

### Insult Analysis

<table>
<thead>
<tr>
<th>Language</th>
<th>Proportion</th>
<th>Country Code</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined</td>
<td>51.8%</td>
<td>RO</td>
<td>47%</td>
</tr>
<tr>
<td>Typographic errors</td>
<td>17.1%</td>
<td>DE</td>
<td>16%</td>
</tr>
<tr>
<td>Romanian</td>
<td>11.8%</td>
<td>ES</td>
<td>4%</td>
</tr>
<tr>
<td>English</td>
<td>9.2%</td>
<td>LU</td>
<td>4%</td>
</tr>
<tr>
<td>Smiley</td>
<td>5.3%</td>
<td>IT</td>
<td>4%</td>
</tr>
<tr>
<td>Slovak</td>
<td>5.3%</td>
<td>MK</td>
<td>4%</td>
</tr>
<tr>
<td>Croatian</td>
<td>1.0</td>
<td>LB</td>
<td>3%</td>
</tr>
<tr>
<td>Polish</td>
<td>1.0%</td>
<td>NL</td>
<td>2%</td>
</tr>
<tr>
<td>German</td>
<td>0.2%</td>
<td>GB</td>
<td>1%</td>
</tr>
<tr>
<td>others</td>
<td>33.06%</td>
<td>others</td>
<td>15%</td>
</tr>
</tbody>
</table>

### Examples

muie, sex, fuck me, gogo, beto,hahahah, :)), pla, sugeo, please, sucker, bine, ?, noaon, qwerty ...
Future work and conclusions

- **Future work**
  - Execution slow-down → AHA is slower than an high-interaction honeypot
  - Evaluate timing attacks
  - Explore faster interprocess communication techniques
  - Insult program needs to be protected with rootkit techniques
  - Substituting a program can crash the program when the stack frame is too small
  - Vulnerable against indirect attacks → let the system continue the attack
  - Tests with the SKAS patch could be done
  - tty_read and tty_write could be monitored → insights about keystrokes
  - Instrument a virtual machine instead of User Mode Linux
Future work and conclusions

Conclusions

- Honeypots should become more intelligent and adaptive
- Optimize information retrieval from attackers
- Created an adaptive honeypot framework to investigate learning techniques
- Extended User Mode Linux
- Each system call related to program execution needs to be acknowledged by the AHA daemon
- Freely available at git.quuxlabs.com
Demo in progress ...

Thank you for your patience ...
Thank you for your attention
Questions?
Comments for improvement?
Bibliography

Steven M. Bellovin.
There be dragons.

Bill Cheswick.
An evening with Berferd in which a cracker is lured, endured, and studied.

Fred Cohen.
A note on the role of deception in information protection.

Jose Antonio Coret.
Kojoney - a honeypot for the SSH service.

L. Spitzner.
Honeypots: Tracking Hackers.

Clifford Stoll.
Stalking the wily hacker.