Adaptive and Self-Configurable Honeypots

Radu State
University of Luxembourg
9 September 2011
Outline

Introduction

Modeling a High Interaction Honeypot

Honeypot Games

Learning Honeypots
  Reinforcement Learning Honeypots
  Fast Concurrent Learning Honeypots

Results

Conclusions and Future Work
Introduction

Context

- Honeypots are resources dedicated to be attacked [4]
- The level of interaction with a honeypot depends on its capabilities
- The more features are exposed to attackers, the more interaction may be observed
- Honeypots are classified into low-mid- or high-interaction honeypots
Introduction

Limitations

- Existing honeypots have a static behavior
- Usage of static configuration
- Restricted view of attacker capabilities

Need for triggering more interaction with attackers

- Keep attackers for a longer time → learn more from them
- Collect more tools from them
- Study their behavior when they face resistance
- Reveal their social background
# Introduction

## 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>Returns shell</td>
<td></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>Execute id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ps aux</td>
<td></td>
<td>Already compromised?</td>
</tr>
<tr>
<td>5</td>
<td>Execute ps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>wget URL_0</code></td>
<td></td>
<td>Acquire tool</td>
</tr>
<tr>
<td>7</td>
<td>Execute wget</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><code>./ssh – brute</code></td>
<td></td>
<td>Misuse the system</td>
</tr>
<tr>
<td>9</td>
<td>Return error</td>
<td></td>
<td>Artificial block</td>
</tr>
<tr>
<td>10</td>
<td><code>wget URL_1</code></td>
<td></td>
<td>Additional tool</td>
</tr>
<tr>
<td>11</td>
<td>Execute wget</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td><code>./local-exploit</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Execute local-exploit</td>
<td></td>
<td>Make attacker happy</td>
</tr>
</tbody>
</table>
Introduction
Real attacker 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
Du-te dracului
Modeling a High Interaction Honeypot

- Linux operating system
- Exposing SSH server → popular attack vector [3]
- Presented by a hierarchical probabilistic automaton

Automaton description

- Macro states: programs
- Micro states: command line arguments
- Edges: transition probability
Honeypot Games

• Honeypot actions
  ◦ Block a `sys_execve` system call with the probability $Pr(block)$
  ◦ Allow a `sys_execve` system call with the probability $1 - Pr(block)$

• Attacker actions if they were blocked
  ◦ Retry the command with a probability $Pr(Retry)$
  ◦ Select an alternative command with the probability $Pr(Alternative)$
  ◦ Leave the honeypot with a probability $Pr(Quit)$

• Each action involves a payoff / reward

• Purpose: Compute Nash Equilibria
Honeypot Games

- **Attacker**
- **Honeypot**

Diagram:

- **wget**
- **uname**
- **nmap**
- **sshd**
- **nmap**

Diagram nodes with probabilities:

- **wget**: 0.95
- **Allow**: 1
- **Pr(block)**: 0.05
- **Pr(quit)**: 0.1
- **Pr(retry)**: 0.2
- **Pr(alternative)**: 0.6
- **uname**: 0.3
- **sshd**: 0.1
- **wget**: 0.1
- **nmap**: 0.05
- **wget**: 0.95
- **nmap**: 0.95

Graph showing interactions and probabilities.
Honeypot Games

- Attacker
- Honeypot

Diagram:

- nmap: 0.6
- wget: 0.95
- uname: 0.6
- sshd: 0.3
- nmap: 0.8
- wget: 0.2
- uname: 0.1
- sshd: 0.05
- nmap: 0.6

Pr(Alternative)
Pr(retry)
Pr(Block)
Pr(Quiet)
Pr(block)
Allow
1 - Pr(block)
Honeypot Games

- Attacker
- Honeypot

```
wget
[36muname\nsshdd
nmap
```

```
Pr(alternative) = Pr(uname) = 0.3
Pr(retry) = Pr(sshd) = 0.1
Pr(Quit) = Pr(wget) = 0.95
```

```
Pr(Block) = 1 - Pr(block)
```

```
Allow
```

```
Block
```

```
uname 0.6
```

```
wget 0.95
```

```
nmap 0.6
```

```
sshd
```

```
wget
```

```
Pr(alternative)
```

```
uname 0.6
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Honeypot Games

Attacker

Honeypot

wget

uname

sshd

nmap

Pr(alternative)

Pr(retry)

Pr(Quit)

Block

Pr(Block)

1 - Pr(block)

Allow

1 - Pr(block)

Pr(Block)

Pr(Quit)

Pr(retry)

uname 0.6

nmap 0.6

wget 0.95

ssh 0.6

nmap

wget

uname 0.3

ssh 0.1

wget 0.8

nmap

wget 0.95

nmap
Honeypot Games

- Attacker
- Honeypot

1. wget
2. uname
3. sshd
4. nmap

Pr(alternative)

Pr(retry)

Pr(Quit)

Pr(Block)

Pr(1-Block)

Allow

Block

Pr(alternative)

uname 0.6

nmap 0.6

wget 0.95

sshd

nmap

uname
Honeypot Games

Attacker

Honeypot

wget 0.95
uname 0.6
sshd
nmap

Pr(alternative)

Pr(retry)

Pr(Quit)

Pr(Block)

Pr(Block)

Pr(alternative)

uname

wget

nmap

sshd

uname

wget

nmap
Honeypot Games
Computing Payoffs

Transition payoff
Honeypot payoff = The number of transitions an attacker did
Attacker payoff = - Honeypot payoff

Path probability payoff
Given an initial state $s_0$ (sshd) and a given final state $s_1$:
Let $Pr^*(path)$ be the most probable path from $s_0$ to $s_1$
Let $Pr(path)$ be the probability of the path the attacker has chosen

\[
\text{Attacker payoff} = R_a^p = \frac{Pr(Path)}{Pr^*(Path)}
\]
\[
\text{Honeypot payoff} = R_h^p = 1 - \frac{Pr(Path)}{Pr^*(Path)}
\]

Take into account whether attacker reach their goal or not
Honeypot Games

Limitations

• Coarse-grained adaptation mechanism
• Independent of the contextual state
• Transition probabilities must be estimated
• Payoffs must be simulated and may change over time
• Players are assumed to make errors
Reinforcement Learning Honeypots

The environment is the high-interaction honeypot model.
Attackers do transitions in the hierarchical probabilistic automaton.
The honeypot can: allow, block, substitute the command or insult the attacker.
Each action results into a reward that is optimized for each state.
The honeypot learns an optimal behavior for each action-state pair.
Limitations

- Attackers are considered as the environment
- Ignores the competitive nature among attackers and the honeypot
- May impact the convergence to optimal values for each state
Fast Concurrent Learning Honeypots

• Combine game theory and reinforcement learning

• Approaches
  ○ Nash-Q by Hu et al. [2]
    • High algorithmic complexity
    • Slow decision making (processing speed)
  ○ Fast concurrent learning by Banerjee et al. [1]
    • Same convergence aspects than Nash-Q
    • Fast decision making
Fast Concurrent Learning

Stochastic game

- Attackers penetrating the Honeypot are modeled as stochastic game $< S, A^1, A^2, r^1, r^2, p >$
- $S$ is the discrete state space
- $A^k$ is the discrete action space of player $k$, $k = 1, 2$
- $r^k : S \times A^1 \times A^2 \rightarrow R$ is the payoff function for player $k$
- $p$ is the transition probability
- Constants: $\alpha$ represents the learning rate, $\gamma$ discounting factor

Learning rule

$$Q_{k}^{t+1}(s_t, a^k_t, \bar{a}^k_t) = (1 - \alpha_t)Q_k^t(s_t, a^k_t, \bar{a}^k_t) + \alpha_t[r^k_t + \gamma Q_k^t(s_t, a^k_{t+1}, \bar{a}^k_{t+1})]$$
Fast Concurrent Learning Honeypot

Honeypots actions applicable for each state

• Allow the execution
  ◦ \sim high-interaction honeypot

• Block the execution
  ◦ Strategic blocking
  ◦ Attacker challenge

• Substitute the execution
  ◦ Make attacker believe to have downloaded the wrong program
  ◦ Repository is not available

• Insult the attacker
  ◦ Irritate attacker
  ◦ Human vs. automated attacks (reverse turning test)
Attacker actions

• Do transitions in the hierarchical probabilistic automaton
• Actions when attacker has been detoured
  ◦ Retry the command → same transition
  ◦ Select an alternative command → select an alternative transition
  ◦ Leave the honeypot → make a transition to an absorbing state
Fast Concurrent Learning Honeypot

Learning rule

$$Q^{t+1}_k(s_t, a^k_t, \bar{a}^k_t) = (1 - \alpha_t)Q^t_k(s_t, a^k_t, \bar{a}^k_t) + \alpha_t[r^k_t + \gamma Q^t_k(s_t, a^k_{t+1}, \bar{a}^k_{t+1})]$$

$$Q^{t+1}_{\text{attacker}}(ps, block, retry) = ?$$
Fast concurrent learning honeypot

Rewards

- Honeypot
  - \( r^2 = \delta \times (l + 1) \)
  - \( \forall s \in S \ l = \min \text{Levenshtein}(c, s) \)
  - \( c \) is the input given by an attacker
    - Attacker enter valid commands
    - Attacker execute custom tools
    - Attacker type insults and make typographic errors
  - \( s \) is the used state in the hierarchical probabilistic automaton
  - If attackers enter typographic errors or execute their own programs, the better it is for the honeypot
Implementation

- **Linux system call hooks**
  - `sys_execve`
  - `sys_clone`
  - `sys_exit`
- **Decision making**
- **Exchange Messages**
  - Export message
  - Reply message

→ git.quuxlabs.com
Results

• Honeypot operation: 48 days
• Number of states: 47
• Successful logins: 637
• Number of transitions: 4360
• Number of insults 1183
## Insult Analysis

<table>
<thead>
<tr>
<th>Language</th>
<th>Proportion</th>
<th>Country Code</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romanian</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>Russian</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>N/A</td>
<td>others</td>
<td>15%</td>
</tr>
</tbody>
</table>
Results

Q value evolution

Figure: honeypot’s perspective

Figure: attacker’s perspective
Results

Q value evolution

Figure: Blocking

Figure: Substituting
Conclusions & Future Work

Conclusions

• Self-management paradigm for high-interaction honeypots
• Learning strategic decisions facing attackers
• Framed in a stochastic game
• Usage of fast competitive learning
• Analyzed the insults collected from attackers
• Revealed the impact of strategic honeypot actions

Limitations and future work

• Attackers could replace states → state representation
• Ignorance of the command semantics
• Explore other learning techniques i.e. attackers who collude
Questions and Answers

Thank you for your attention

Questions?

Radu State  radu.state@uni.lu
Gerard Wagener  gerard.wagener@circl.lu
Alexandre Dulaunoy  alexandre.dulaunoy@circl.lu
Thomas Engel  thomas.engel@uni.lu
Bikramjit Banerjee, Sandip Sen, and Jing Peng.
Fast Concurrent Reinforcement Learners.

Junling Hu and Michael P. Wellman.
Nash Q-Learning for General-Sum Stochastic Games.

Daniel Ramsbrock, Robin Berthier, and Michel Cukier.
Profiling attacker behavior following SSH compromises.

L. Spitzner.
Honeypots: Tracking Hackers.