Continuous Assessment of a Unix Configuration: Integrating Intrusion Detection & Configuration Analysis

A. Mounji    B. Le Charlier,

ASAX Project

Institut d’Informatique,
FUNDP,
Namur - Belgium,

E-mail: {amo, ble}@info.fundp.ac.be

URL:
http://www.info.fundp.ac.be/~cri/DOCS/asax.html
Contents

1. Configuration Analysis Systems

2. Intrusion Detection Systems

3. Architecture of the Integrated System

4. Configuration Analysis Language

5. Specification of the Inference Engine

6. Implementation

7. Performance Evaluation

8. Conclusions and Future Works
Configuration Analysis

Systems

- Check the presence of vulnerabilities in the configuration

- What can potentially be done to the system?

- Can be based on Predicate Logic
  - Existing Systems are snapshot oriented (Eg.:\(Kuang\)\(NetKuang\))
  - ASAX: Declarative\(\) Real-time.
    Eg.:

```
become(User\(\)root) :-
    replace(User\(\)/etc/passwd).
```
Intrusion Detection Systems

- Observe user actions

- What has *actually* been done to the system?

- A Rule-based Language approach is powerful:

  \[
  \text{if } \text{cond} \text{ then } \text{action}
  \]

- Example System: ASAX

- Can be made more powerful by taking advantage of the knowledge about the state of the configuration
Architecture of the Integrated System

Audit System

Dynamic Fact Base
- Updates basic facts
- Updates derived facts

Fact Base Initializer

Audit Trail
- Audit data generation
- Alarms, Enabling Security Procedures

Audit Trail Analysis
- Ei
- Trigger new Detection rules

RUSSEL
- Audit Trail

Configuration Analysis
- Datalog
RUSSEL Language (Example)

rule detect_root_access(Username: string);
begin
if (event = 7 or event = 23)
/* exec(2) or execve(2) */
and file_owner_id = 0 /* root */
and uid = uid(Username)
and illegalSetUID(file_name) = 1
--> println('Suspicious Execution of the setUID program', file_name,
' By User ', Username,
' At Time ', gettime(time))
fi;
trigger off for_next
detect_root_access(Username)
end.
Configuration Analysis

Language

- **Goal:** represent the security state of a Unix Configuration using Predicate Logic

- **Specialized version of Datalog:**
  - constants: users, groups, file names
  - built-ins (ex: homeDir(amo/users/amo))
  - Deductions: (incremental evaluation)
    \[ \langle r\Gamma \sigma \rangle = r\sigma \] (instance of a rule)
  - a fact \( f \) contributes to \( \langle r\Gamma \sigma \rangle \) iff
    \[ h\sigma := a_1\sigma, ..., a_i-1\sigma, f\sigma, a_i+1\sigma, ..., a_n\sigma \]
write(U, F) :- worldWrite(F).
write(U, F) :- groupWrite(F, Group),
    inGroup(U, Group).
write(U, F) :- parentDir(D, F), write(U, D).
inGroup(U, G) :- groupMember(U, G).
inGroup(U, G) :- groupMember(V, G), become(U, V).
become(U, V) :- parentDir(D, F),
    write(U, F),
    homeDir(V, D).
become(U, root) :- write(U, /etc/passwd).
become(U, root) :- write(U, /etc/group).
become(U, root) :- write(U, /etc/rc).
become(U, root) :- write(U, /etc/aliases).
Interfacing ASAX with Datalog

Trigger (resp. cancel) a detection rule as the configuration changes:

- **on_new** `fact_name(X_1, ..., X_n)`
  trigger off for next `rule_name(X_1, ..., X_n)`

- **on_dispose** `fact_name(X_1, ..., X_n)`
  cancel `rule_name(X_1, ..., X_n)`

Update the fact base by monitoring critical events:

- **is_fact**(`fact_name(X_1, ..., X_n)`)
- **assert**(`fact_name(X_1, ..., X_n)`)
- **retract**(`fact_name(X_1, ..., X_n)`)
- **commit**
Interfacing ASAX with Datalog: Example

1. Datalog

   on_new become(UΓroot)
   trigger off for_next detect_root_access(U).

   on_dispose become(UΓroot)
   cancel detect_root_access(U).

2. RUSSEL

   if
     grp_read(path, gid, mode) = 1
     -->
     begin
       assert(groupRead(gid, path));
       commit
     end
   fi
Specification of the Inference Engine

- Given
  - a set of basic facts BF
  - a set of rules SR

- we define the set of derived facts DF and the set of deductions SD corresponding to BF and SR as the smallest sets such that:

\[
\begin{align*}
r &\in SR, \\
r &\equiv h : -a_1, \ldots, a_n, \\
f_1, \ldots, f_n &\in BF \cup DF, \\
\sigma &\equiv mgu \{a_1 = f_1, \ldots, a_n = f_n\} \quad \Rightarrow \quad h\sigma \in DF \text{ and } r\sigma \in SD
\end{align*}
\]

- and we note \(\langle SD \Gamma DF \rangle = \text{Ded} \ (BF \Gamma SR)\).
Incremental Update of the Fact Base

Upon occurrence of an event we compute:

- $\Delta^-$: basic facts to be retracted.

- $\Delta^+$: basic facts to be added.

Example:
rename `~amo/.cshrc` to `~amo/.login`

- $\Delta^- =$
  
  {parentDir(`~amo, `~amo/.cshrc)`}
  
  worldWrite(`~amo/.cshrc`)

- $\Delta^+ =$
  
  {parentDir(`~amo, `~amo/.login)`}
  
  worldWrite(`~amo/.login`)
Incremental Update of the Fact Base

Given

- a set of rules $SR$ a set of basic facts $BF$ and 
  \( \langle SD\Gamma DF \rangle = \text{Ded}(BF\Gamma SR) \)

- $\Delta^-$ and $\Delta^+$

Compute $\text{Ded}((BF \setminus \Delta^-) \cup \Delta^+\Gamma SR)$

This is done \textit{incrementally} in 2 steps:

1. compute $(SD^-\Gamma DF^-) = \text{Ded}(BF \setminus \Delta^-\Gamma SR)$
   from $\Delta^-\Gamma SD\Gamma DF$ and $BF$

2. compute $\text{Ded}((BF\setminus \Delta^-) \cup \Delta^+\Gamma SR)$
   from $\Delta^+\Gamma SD^-\Gamma DF^-$ and $BF\setminus \Delta^-$
Retracting a list of facts

For each removed fact:

- for each deduction to which it contributes
  - remove deduction
  - decrement ref. count of implied fact
  - if ref. count = 0 recursively remove the fact
Retracting a list of facts

Retract_ded(Δ⁻)
begin
Δ := Δ⁻;
while (Δ ≠ ∅) do
begin
    Remove(Δ, f);
    Sup_ded := list_ded(f);
    while (Sup_ded ≠ ∅) do
    begin
        Remove(Sup_ded, d);
        f' := Fact(d);
        SD := SD \ {d};
        if (Nb_ded(f') = 0)
            then Δ := Δ \ {f'}
end;
    DF := DF \ {f}
end
end
Adding a list of facts

Generate_ded(\(\Delta^+\))
begin
\(\Delta := \Delta^+;\)
while (\(\Delta \neq \emptyset\)) do
begin
Remove(\(\Delta, f;\))
FB := FB \cup \{f\};
\(\Gamma := \text{rule_match}(f);\)
while (\(\Gamma \neq \emptyset\)) do
begin
Remove(\(\Gamma, (r, i));\)
Gen_ded_fact(r, i, f, \(\Delta\))
end
end
end.

Gen_ded_fact(r, i, f, \(\Delta\))
begin
Let \(r \equiv h -: a_1, \ldots, a_n;\)
Let \(\sigma = \text{mgu}(a_i, f);\)
Gen_case(r, \(\sigma, i, 0, \Delta\))
end.
Adding a list of facts (continued)

Gen_case(r, α, i, j, Δ)
begin
Let r ≡ h :- a_1, ..., a_n;
if (j = n+1) and (rα ∉ SD) then
  begin
    SD := SD ∪ {rα};
    if (hα ∈ BF ∪ DF) then
      begin
        DF := DF ∪ {hα};
        Δ := Δ ∪ {hα}
      end
    else increment_ref(hα)
  end
else if (j = i) then Gen_case(r, α, i, j+1, Δ)
else begin
  list_facts := find_all_facts(a_jα);
  while (list_facts ≠ ∅) do
    begin
      Remove(list_facts, f');
      σ := mgu(f', a_jα);
      if (σ ≠ fail)
        then Gen_case(r, ασ, i, j+1, Δ)
    end
  end
end
end.
Implementation

- Predicate: list of rules where it appears in the body

- Fact: (its predicate, array of args, ref count, list of deductions to which it contributes)

- Deduction: (array of facts contributing to it, the implied fact)

- Hash code to ensure unicity of representation

![Diagram of relationships between facts and deductions]
Performance Evaluation

- **Detection Rules:**

<table>
<thead>
<tr>
<th>Rules</th>
<th>Exploitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setuid program writes another setuid</td>
</tr>
<tr>
<td>2</td>
<td>Programs writing to executable files</td>
</tr>
<tr>
<td>3</td>
<td><em>lpr</em> overwrites a file outside of <em>/var/spool</em></td>
</tr>
<tr>
<td>4</td>
<td>Execution of known attack programs (crack, cops, etc)</td>
</tr>
<tr>
<td>5</td>
<td>Creation of setuid programs</td>
</tr>
<tr>
<td>6</td>
<td>Creation of a device file using <em>mknod()</em></td>
</tr>
<tr>
<td>7</td>
<td>Writing non owned files</td>
</tr>
<tr>
<td>8</td>
<td>Execution of a suspicious setuid program</td>
</tr>
<tr>
<td>9</td>
<td>Illegal read access to <em>/dev/kmem</em> or <em>/dev/mem</em></td>
</tr>
<tr>
<td>10</td>
<td>Linking an <em>at</em> job to root mail box</td>
</tr>
<tr>
<td>11</td>
<td>Copying a shell in root mail box when empty</td>
</tr>
</tbody>
</table>

- **Audit Trail Description:**

<table>
<thead>
<tr>
<th>#Users</th>
<th>#Grps</th>
<th>#Rec</th>
<th>#SRec</th>
<th>Size</th>
<th>Time</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>23</td>
<td>173,828</td>
<td>5,641</td>
<td>14.5 MB</td>
<td>25:35:42</td>
<td>1.89</td>
</tr>
</tbody>
</table>
Performance Evaluation (continued)

- Fact Base Initialization:

<table>
<thead>
<tr>
<th>#Facts</th>
<th>#Deds</th>
<th>Size Kb</th>
<th>IFB sec</th>
<th>UFB sec</th>
<th>UPR msec</th>
</tr>
</thead>
<tbody>
<tr>
<td>5084</td>
<td>5721</td>
<td>568</td>
<td>26.52</td>
<td>62</td>
<td>0.36</td>
</tr>
</tbody>
</table>

- Audit Trail Analysis:

<table>
<thead>
<tr>
<th>type</th>
<th>usr sec</th>
<th>sys sec</th>
<th>total sec</th>
<th>#RPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated</td>
<td>458.47</td>
<td>95.63</td>
<td>554.10</td>
<td>313.71</td>
</tr>
<tr>
<td>Not Integrated</td>
<td>4062.70</td>
<td>106.70</td>
<td>4169.10</td>
<td>41.00</td>
</tr>
</tbody>
</table>
Conclusions and Future Works

Conclusions:

- Integrating Intrusion Detection with Configuration Analysis achieves:
  - A continuous assessment of the configuration.
  - A dynamically adaptive IDS \textit{wrt} the configuration

- Computationally feasible

Future Works:

- Further extend current deductive rules

- Further tuning of the system
ASAX package and papers available at:

http://www.info.fundp.ac.be/~cri/DOCS/asax.html