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

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?

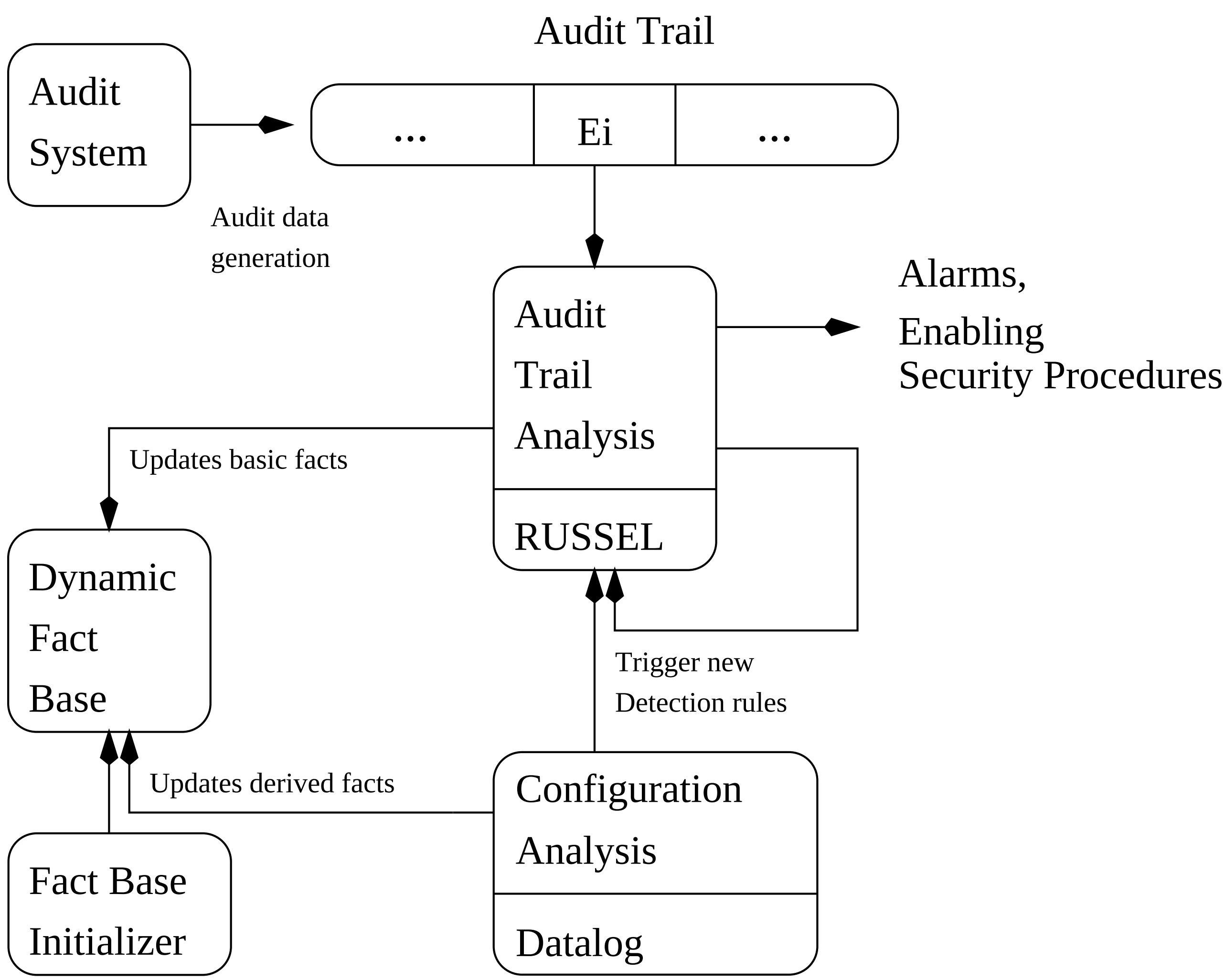
- A Rule-based Language approach is powerful:

  if *cond* **then** *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



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)
    $\langle r, \sigma \rangle = r\sigma$ (instance of a rule)
  - a fact f contributes to $\langle r, \sigma \rangle$ *iff*
    $h\sigma$:- $a_1\sigma$, ..., $a_{i-1}\sigma$, $f\sigma$, $a_{i+1}\sigma$, ..., $a_n\sigma$

Configuration Analysis Language (Example)

```
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(U, root)
   trigger off for_next detect_root_access(U).

   on_dispose become(U, root)
   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:

$$\left.\begin{array}{l} r \in SR, \\ r \equiv h : -a_1, ..., a_n, \\ f_1, ..., f_n \in BF \cup DF, \\ \sigma = mgu\{a_1 = f_1, ..., a_n = f_n\} \end{array}\right\} \Rightarrow \begin{array}{l} h\sigma \in DF \ \ and \\ r\sigma \in SD \end{array}$$

- and we note $\langle$SD, DF$\rangle$ = Ded (BF, SR).

Incremental Update of the Fact Base

Upon occurrence of an event, we compute:

- Δ^-: basic facts to be retracted.
- Δ^+: 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, DF$\rangle$ = Ded(BF, SR)
- Δ^- and Δ^+

Compute Ded((BF $\setminus \Delta^-) \cup \Delta^+$, SR)

This is done *incrementally*, in 2 steps:

1. compute (SD$^-$, DF$^-$) = Ded(*BF* $\setminus \Delta^-$, SR) from Δ^-, *SD*, *DF* and *BF*
2. compute Ded((BF$\setminus \Delta^-) \cup \Delta^+$, SR) from Δ^+, SD$^-$, 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(Δ+)
begin
    Δ := Δ+;
    while (Δ ≠ ∅) do
    begin
        Remove(Δ, f);
        FB := FB ∪ {f};
        ,  := rule_match(f);
        while (,  ≠ ∅) do
        begin
            Remove(, , (r, i));
            Gen_ded_fact(r, i, f, Δ)
        end
    end
end.
```

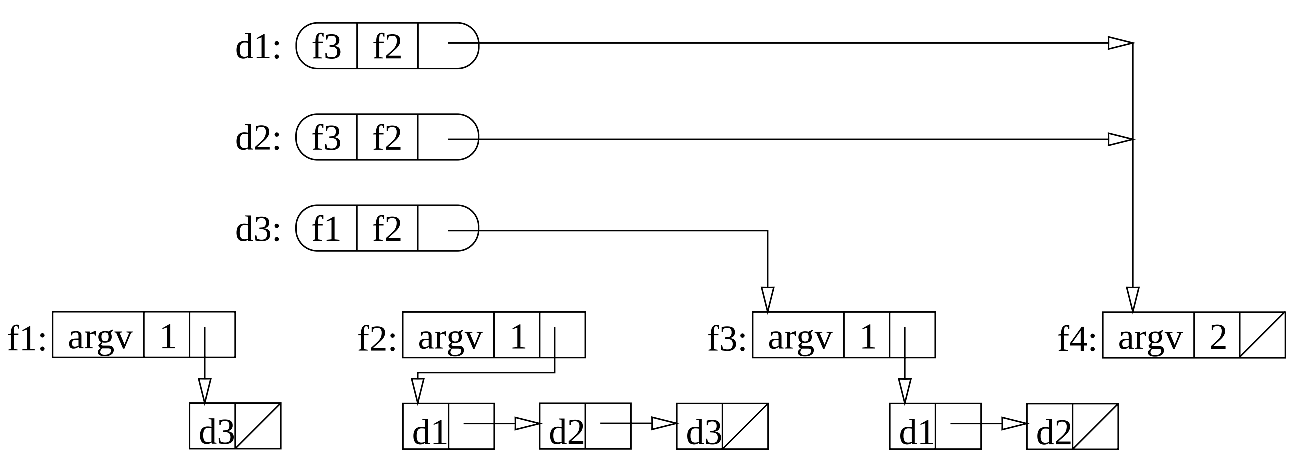
```
Gen_ded_fact(r, i, f, Δ)
begin
    Let r ≡ h -:  a_1, ..., a_n;
    Let σ = mgu(a_i, f);
    Gen_case(r, σ, i, 0, Δ)
end.
```

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_j α);
      while (list_facts ≠ ∅) do
         begin
            Remove(list_facts, f′);
            σ := mgu(f′, a_j α);
            if (σ ≠ fail)
            then Gen_case(r, ασ, i, j+1, Δ)
         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



Performance Evaluation

- **Detection Rules**:

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

- **Audit Trail Description**:

#Users	*#Grps*	*#Rec*	*#SRec*	*Size* Mb	*Time* hh:mm:ss	*Rate*
128	23	173,828	5,641	14.5 MB	25 : 35 : 42	1.89

Performance Evaluation (*continued*)

- **Fact Base Initialization**:

#Facts	*#Deds*	*Size* *Kb*	*IFB* *sec*	*UFB* *sec*	*UPR* *msec*
5084	5721	568	26.52	62	0.36

- **Audit Trail Analysis**:

type	*usr* *sec*	*sys* *sec*	*total* *sec*	*#RPS*
Integrated	458.47	95.63	554.10	313.71
Not Integrated	4062.70	106.70	4169.10	41.00

Conclusions and Future Works

Conclusions:

- Integrating Intrusion Detection with Configuration Analysis achieves:
  - A continuous assessment of the configuration.
  - A dynamically adaptive IDS *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`