An Autonomic Cloud Management System for Enforcing Security and Assurance Properties

CLHS'15

Laurent Bobelin, Aline Bousquet, Jérémy Briffaut

Laboratoire d'Informatique, Tours, France
INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022

June 15, 2015

Plan

Problems with Cloud security

Objectives:

- Enforce security properties
  - Confidentiality, Integrity, Availability
- Check security properties enforcement
  - Assurance, Assurance Scripts

- Many available system and network security mechanisms
  - iptables
  - SELinux
  - Secure Elements (SE)
  - OpenVPN
  - ...
- Complexity of security configuration
  - System, VM, Host, Hypervisor, Network, ...

No security mechanism can protect a whole system/Cloud on its own
$\Rightarrow$ Propose a model to easily guarantee security properties.

Plan

Global Objective

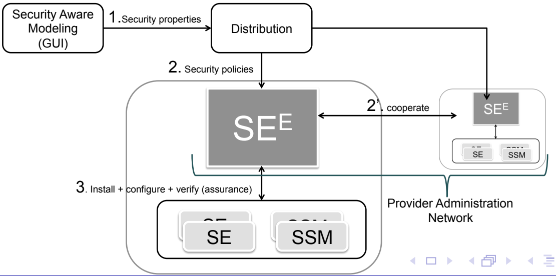
Automatic deployment of security and assurance in a Cloud environment

- Define the global Cloud software architecture
- Define the security requirements using properties
- Enforce the security properties using existing mechanisms
- Check that the security properties are enforced as expected

Global Architecture

Seed4C's solution: a three-parts model

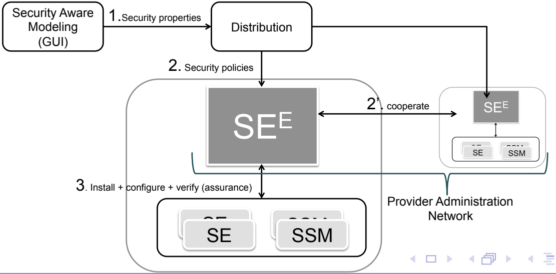
1. A modeling tool (GUI)
   - The user describes his software architecture
   - He graphically defines abstract security properties (Confidentiality, ...)
2. A distribution engine
   - Splits the properties into sub-properties to be applied on the nodes
3. An enforcement & assurance engine: the SE^E (Secure Element Extended)
   - Selects and configures the Software Security Mechanisms (SSM)



Global Architecture

Seed4C's solution: a three-parts model

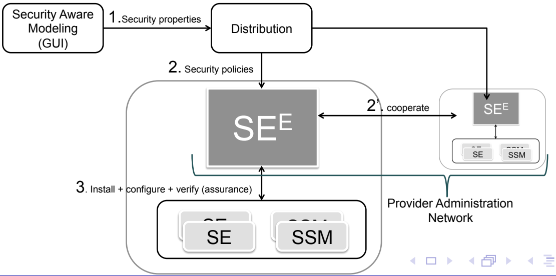
1. A modeling tool (GUI)
   - The user describes his software architecture
   - He graphically defines abstract security properties (Confidentiality, ...)
2. A distribution engine
   - Splits the properties into sub-properties to be applied on the nodes
3. An enforcement & assurance engine: the SE^E (Secure Element Extended)
   - Selects and configures the Software Security Mechanisms (SSM)



Global Architecture

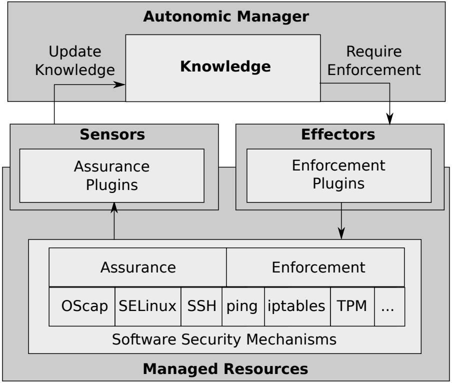
Seed4C's solution: a three-parts model

1. A modeling tool (GUI)
   - The user describes his software architecture
   - He graphically defines abstract security properties (Confidentiality, ...)
2. A distribution engine
   - Splits the properties into sub-properties to be applied on the nodes
3. An enforcement & assurance engine: the SE^E (Secure Element Extended)
   - Selects and configures the Software Security Mechanisms (SSM)



Autonomic architecture: Application to SE^E

1. Autonomic Manager: Component that manages the resources
2. Managed Resources: Elements of the system
3. Effectors: Elements that configure the resources
4. Sensors: Elements that collect data about the resources



Plan

Security Policy Language

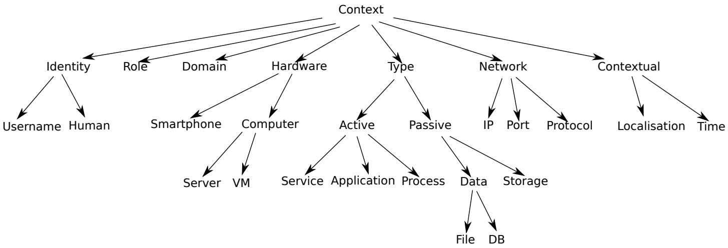
To easily express the security requirements, we propose a dedicated language with:

- **Contexts:**
  - Identify the resources (VM, applications, processes, users, files...)
- **Properties:**
  - Define the security requirements between contexts

Security Contexts

- A context is a label identifying a real resource
- It is composed of a set of attributes
- Each attribute characterizes a part of the identified resource
    - IP address, localization, encryption key, owner identity...
- Reports owned by Bob:

  `Type.Passive.Data.File="report":Id.Username="bob"`



Security properties

Property Templates:

- Two blocks: **enforcement** & **assurance**
- Defined using *capabilies*
  - Capability = abstract functionality offered by security mechanisms
  - Enforcement
    - **generate_key**: generate an encryption key
    - **deny_all_write_accesses**: deny all write accesses to a resource
  - Assurance
    - **check_encrypt_flow**: check that a network flow is encrypted
    - **check_write**: check that resource cannot be read

Property instances:

- Defined during modelization
- Only Bob can read his report files:
  `Confidentiality (Type.Passive.Data.File="report":Id.Username ="bob", Id.Username="bob")`

Security properties

Property Templates:

- Two blocks: **enforcement** & **assurance**
- Defined using *capabilies*
  - Capability = abstract functionality offered by security mechanisms
  - Enforcement
    - **generate_key**: generate an encryption key
    - **deny_all_write_accesses**: deny all write accesses to a resource
  - Assurance
    - **check_encrypt_flow**: check that a network flow is encrypted
    - **check_write**: check that resource cannot be read

Property instances:

- Defined during modelization
- Only Bob can read his report files:
  `Confidentiality (Type.Passive.Data.File="report":Id.Username ="bob", Id.Username="bob")`

Security properties

Property Templates:

- Two blocks: **enforcement** & **assurance**
- Defined using *capabilies*
  - Capability = abstract functionality offered by security mechanisms
  - Enforcement
    - **generate_key**: generate an encryption key
    - **deny_all_write_accesses**: deny all write accesses to a resource
  - Assurance
    - **check_encrypt_flow**: check that a network flow is encrypted
    - **check_write**: check that resource cannot be read

Property instances:

- Defined during modelization
- Only Bob can read his report files:

  `Confidentiality (Type.Passive.Data.File="report":Id.Username ="bob", Id.Username="bob")`

Property Templates: Example

- File confidentiality through access control:

```
boolean Confidentiality_Access_Control (Type.Passive.Data.File SCFile, Id.User SCUser) {
  enforcement {
    deny_all_read_accesses (SCFile);
    return allow_read_access (SCFile, SCUser);
  }
  assurance {
    boolean c = true;
    for (SCUserTmp IN get_all_users()) {
      if (SCUserTmp.Id.User == SCUser.Id.User) {
        c &= check_read (SCFile, SCUser);
      } else {
        c &= (NOT check_read (SCFile, SCUser));
      }
    }
    return c;
  }
}
```

Plan

Assurance property

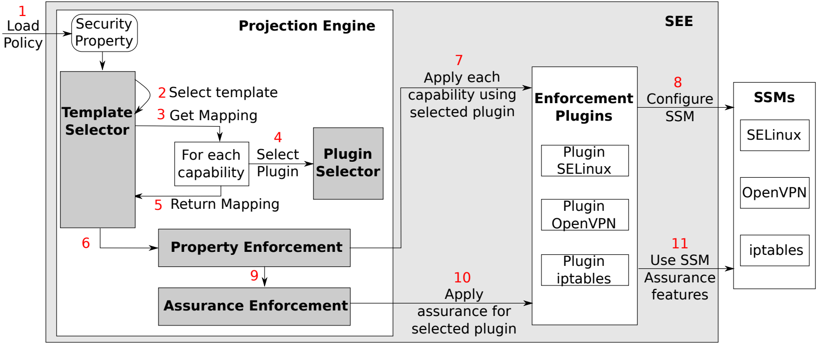
Assurance generation

- Two types:
  - Assurance for mechanisms: generated by each plugin
  - Assurance for properties: defined with the properties, using the language
- Generate scripts
- Scripts' execution defined in an Assurance property:

```
T3:= boolean Assurance (Tests.Frequency SCFrequency) {
 enforcement {
  return run_xccdf_tests (SCFrequency);
 }
}
```

Assurance engine

Enforcement and assurance projection for mechanisms:



Policy → Contexts, Properties → Plugins → Mechanisms Configuration

Assurance

What is generated:

- Scripts to check mechanisms' status
- Scripts to check properties' enforcement

What is done:

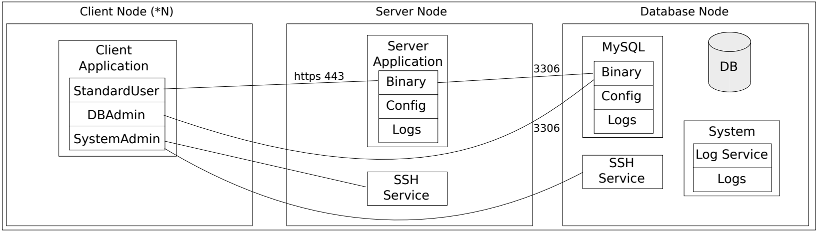
- Scripts are executed by a plugin (e.g. Oscap) according to Assurance properties
- Results stored in XCCDF file

```
$ cat xccdf-test.xml
 [...]
<rule-result idref="ssm-SELinux" time="..." severity="medium" weight="1">
 <result>pass</result>
 <check system="http://open-scap.org/page/SCE">
  <check-import import-name="stdout"></check-import>
  <check-content-ref href="selinux-assurance.sh"/>
 </check>
</rule-result>
 [...]
<score system="urn:xccdf:scoring:default" maximum="100">100</score>
 [...]
```

Plan

Usecase's description

- Cloud database storage architecture



- Objective: isolate the database application and protect its data

Usecase's policy

Contexts:

```
hostServerDB= (Hardware.Computer = "vm_db");
domainDB = (Domain="App_db");
configDB = (Type.Passive.Data.File.Category="Configuration"):domainDB;
logDB = (Type.Passive.Data.File.Category="Log"):domainDB;\\
 [...]
adminRoot = (Id.User="idDBAdmin"):(Id.Role="StandardUser|DBAdmin");
adminOperator = (Id.User="idDBOperator"):(Id.Role="StandardUser|DBOperator");
```

Properties:

Isolation_System(domainDB);
Integrity(configDB,adminRoot);
Confidentiality_access_control(logDB, adminOperator);
[...]
Assurance (frequency, ssmXccdf);

Usecase's policy

- XCCDF file generate by the SE^E and used by Oscap
- Test the enforcement of the properties
- Can also be used to test the status of the mechanisms

```
$ cat prop-xccdf.xml
 [...]
<Rule id="prop-fileConf" severity="medium" selected="true">
 <title>Confidentiality Status</title>
 <description>Check that property is properly enforced</description>
 <check system="http://open-scap.org/page/SCE">
   <check-import import-name="stdout" />
   <check-content-ref href="fileConf.sh"/>
 </check>
</Rule>
 [...]
```

Usecase's policy

- Assurance script generated by the SE^E

```
$ cat fileConf.sh
#!/bin/bash
RET=$XCCDF_RESULT_PASS
check_read(){su -c "test -r "$1"" $2; return $?;}
FILES=[...]      # list of confidential files
USERS=[...]      # list of all users
OK_USERS=[...] # list of authorized users

for file in "${FILES[@]}" ; do
 for user in "${USERS[@]}" ; do
  check_read $file $user
  READ_OK=$?

  if [[ " ${OK_USERS[@]} " =~ " $user " ]] ; then
   if [[ $READ_OK -ne "0" ]] ; then
    RET=$XCCDF_RESULT_FAIL
    echo "Unexpected access denial: $user->$file"
   fi
  else
   if  [[ $READ_OK -eq "0" ]] ; then
    RET=$XCCDF_RESULT_FAIL
    echo "Unauthorized access: $user->$file"
   fi
  fi
 done
done
exit $RET
```

Usecase's policy

- Assurance stats

Number of	
Security properties	8
Assurance aggregation properties	1
SSMs collaborating to enforce the security properties (SELinux, iptables, PAM, SSH)	4
SSMs collaborating to enforce the assurance properties (Oscap)	1
Assurance scripts for the properties	8
Assurance scripts for the SSMs	4

Plan

Conclusion and future works

Conclusion:

- A new language to express security properties in a distributed and heterogeneous environment
- An architecture to enforce the security policy and to check the enforcement
- A solution independent from the security mechanisms
- Experiments on industrial usecases defined by partners of the European project Seed4C (`http://www.celticplus-seed4c.org/`)
- Now: automatic reconfiguration of mechanisms when the assurance process detects an error

Future works:

- Check the coherence of the properties before enforcement

Thank you for your attention!

Questions?