Improving the security of Software Defined Infrastructures

Theodoor Scholte and Thomas Kraus
{t.scholte, t.kraus}@sig.eu

Who am I? A few words about myself…

Current:

- Software Security Consultant at the Software Improvement Group
- Secure design reviews, secure source code reviews
- Like to hack and code

Past:

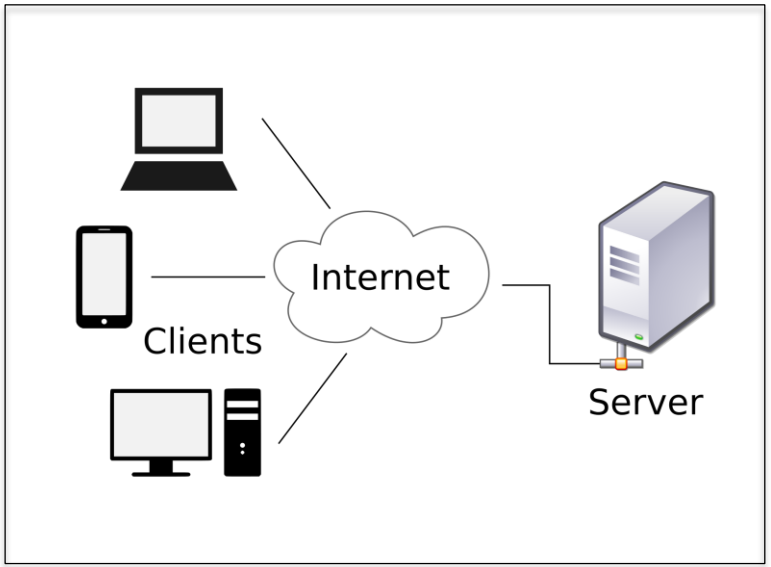
- Product Security Response at SAP SE
- Security Researcher at SAP SE
- PhD in Web Application Security from Institut Eurécom / Télécom ParisTech

Nearly 10 years of experience in software security

“Classic” Infrastructure

- Can be defined as “a composition of software, hardware and network resources to run software applications”

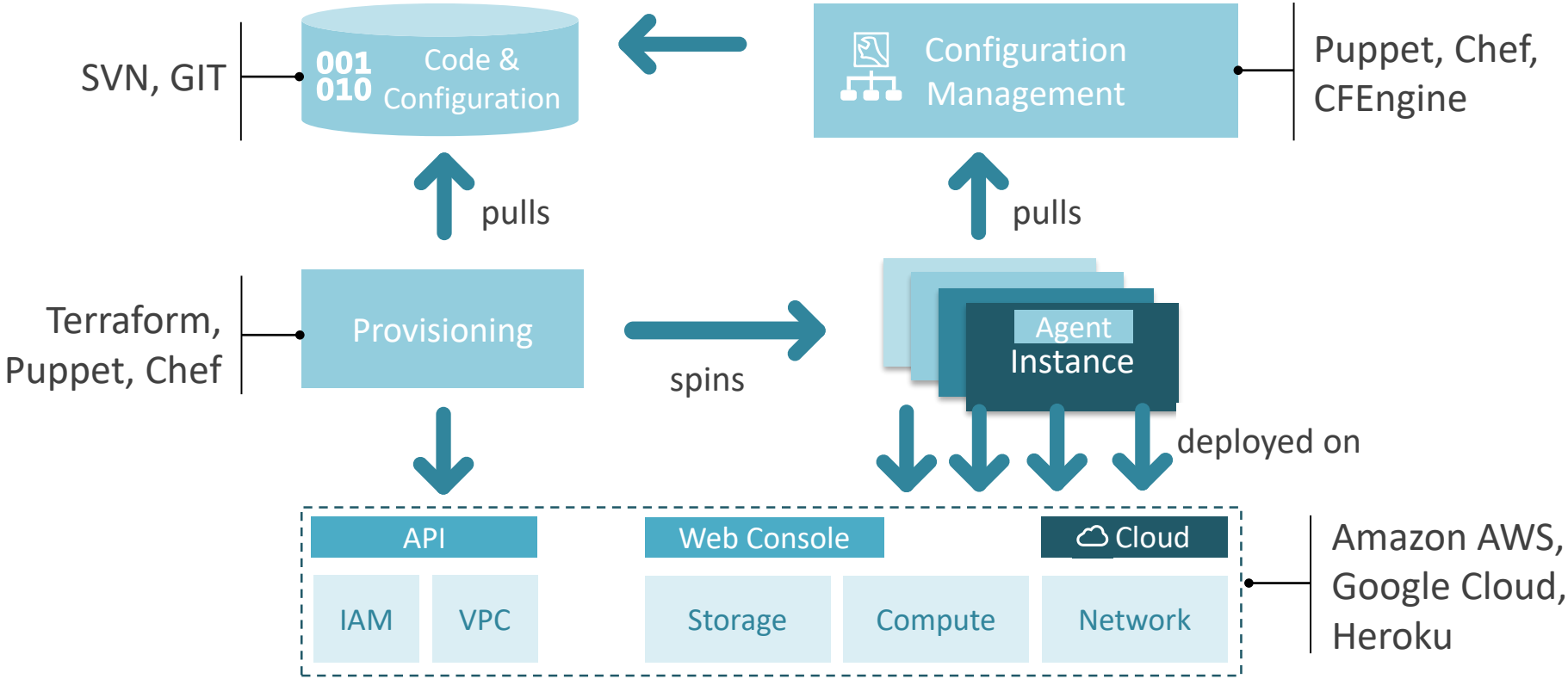
- Disadvantages
  - Lack of automation impedes scaling
  - Poor testing abilities
  - Poor change control



Software Defined Infrastructure

- Infrastructure = Code + Configuration + (Virtualized) Hardware
  - “Infrastructure as code”
- The whole infrastructure is defined in code, configuration templates and configuration files
  - It is stored in a repository, ideally version control. Examples include SVN or GIT
- Tools exist to run this code; to provision system environments
  - Examples include: Puppet, Chef or Ansible
- By executing this code N number of times, you can get N number of environments
- Development and maintenance of an SDI is like normal software development

What does an SDI look like? - An example



Why is Software Defined Infrastructure good for security?

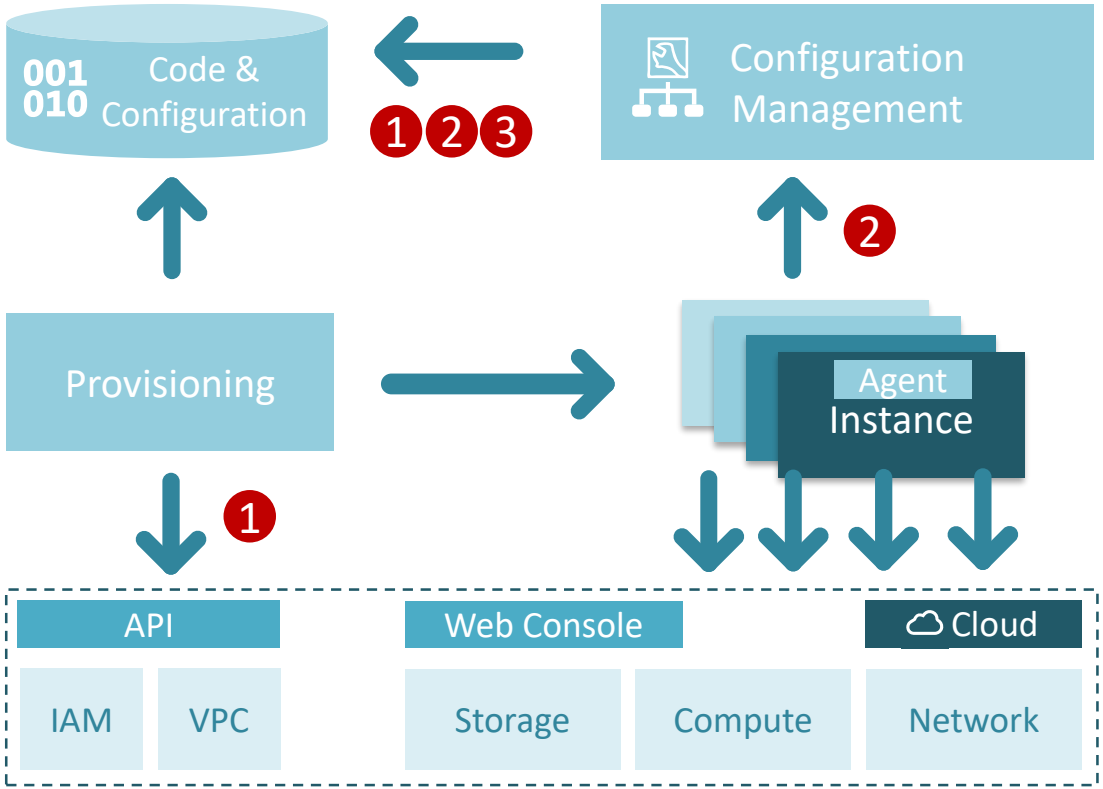
SDI as security tool:

- Define a security baseline once in code, deploy everywhere
  - Enforce common security settings everywhere
  - Examples: login rules, disable SSH1 protocol, local firewall settings, centralized logging / monitoring
- Configure security rules in code based on role of the machine
- Automated security & compliance testing of all systems with tools such as RSpec, ServerSpec and Inspec
- Deploy security patches faster
- Faster recovery after compromise
- Demonstrate compliance, auditing with version control

But…

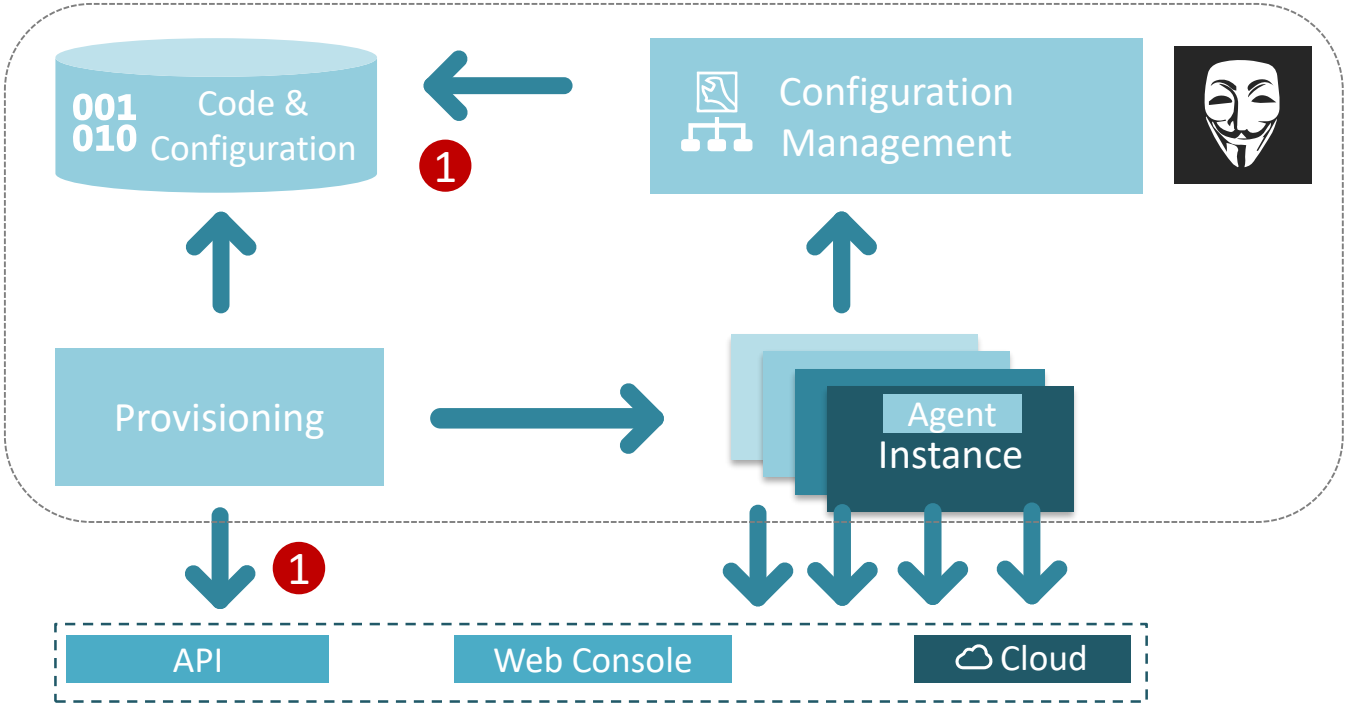
- SDI is also a very powerful instrument
- How to avoid that the SDI itself becomes an attack vector?

Security issues that are often overlooked (non-exhaustive)



1. Insufficiently protected interfaces
2. Insecure handling of secrets
3. Remote code inclusion

Insufficiently protected interfaces

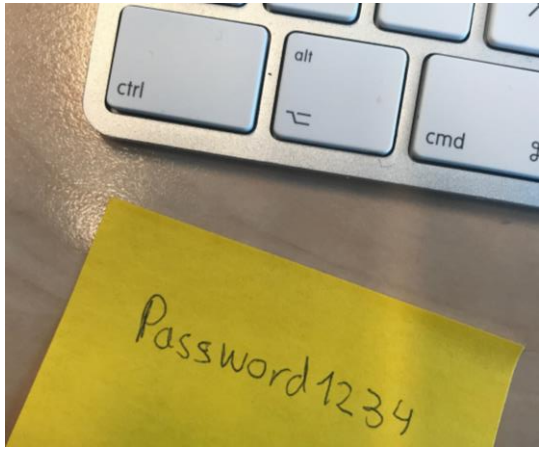


- Root accounts
- No authentication
- Weak passwords
- No LAC/RBAC
- Too many privileges

Insecure handling of secrets

Configuration management systems separate code from configuration to enable code-reuse

- Configuration files specify site-specific configurations
  - Chef: attributes and data bags vs recipes/cookbooks
  - Puppet. Hiera data stored in YAML files vs manifests



A configuration file:

hiera.eyaml hiera.eyaml hiera.eyaml

```
1 sig_mywebapp::properties:
2   mywebapp.mysql_password:
3     value: 'Password1234'
```

Bad Practice: ends up in version control!

- Many users do typically have access to version control
- Version control maintains history, credentials are stored for a very long time!

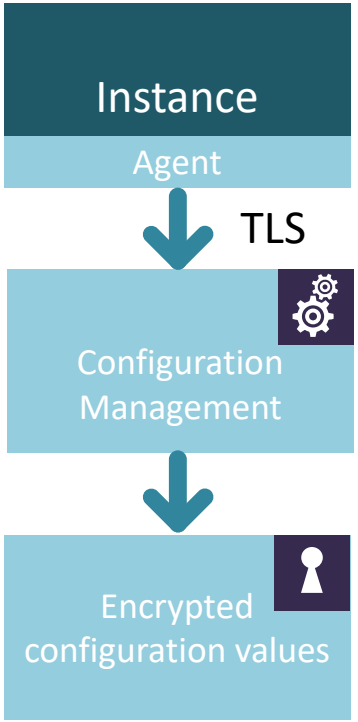
Insecure handling of secrets. So what to do?

Encrypt Configurations!

- Encrypt individual attribute values in the configuration file encryption
- Only the configuration server can decrypt the values

hiera.eyaml | hiera.eyaml | hiera.eyaml | hiera.eyaml | hiera.eyaml | *hiera.eyaml | »87

```
1  mywebapp.mysql_password:
2    value: >
3      ENC[PKCS7,MIIBeQYJKoZIhvcNAQcDoIIBajCCAWYCAQAxggEhMIIBHQIBADAFMAACAQEw
4      DQYJKoZIhvcNAQEBBQAEggEAQKS61/NjZa8V2ozOyxZnNk7SimVL50DQaOla
5      BfRXReOJNGNIMDYKTSM0EJW3ZMRzizo6h0u5mtTJHzmXIXcUkWgFmN4UUJBK
6      jGM56QMPIMgKe1p1fcXtoHsC2RfpXqxVc94BoHT/sPoTU5WX1Dke9+p6/Ko4
7      9RVFysMuD+dbxVoJfHboaZHWapt8wuA4psom0+b6RCWon9DFMdzNlOjHEaU2
8      +zdIYuYxzTSGRL07BOe7RKxj5PJJc6wbzazxYugFH+qWtEmr1+sR6+a352E/
9      MifvLcFC9YEkawZMp990iY3M0b6UgSebFURtG1TsmSPn636dr6N6aLoOzBKk
10     KiOVRDA8BgkqhkiG1e0BBwEwHQYJYIZIAWUDBAEqBBBzipA8VAV4uOQuALD6
11     C9BUgBATyM8s/FchJnZqRzajZ0GPJ
```



Insecure handling of secrets. Other ways of leaking…

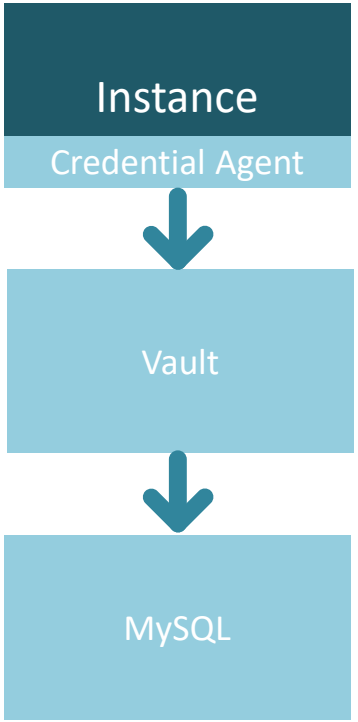
- Configuration management tools update configuration files on instances
- Differences in configuration files on an instance can be logged by the configuration management system
- If a configuration file contains a secret, a change of the secret may get logged
- This logged information can end up in many places:
  - Syslog
  - SIEM
  - ?

- Be careful with logging changes of configuration files!

Insecure handling of secrets. So what is next?

Avoid long living credentials altogether with dynamic credentials:

- A service such as HashiCorp's Vault creates a subaccount with credentials for a service (e.g. MySQL, PostgreSQL)
- Credentials expire after lease period and needs to be renewed on time
  - Configuration of application accessing the service needs to be updated
  - Failure leads to revocation and authentication/access failure!
- A separate process on an instance is needed to update configuration files with passwords (e.g. Consul Template)



Inclusion of untrusted code

- Docker can build images automatically by reading and executing instructions from a Dockerfile

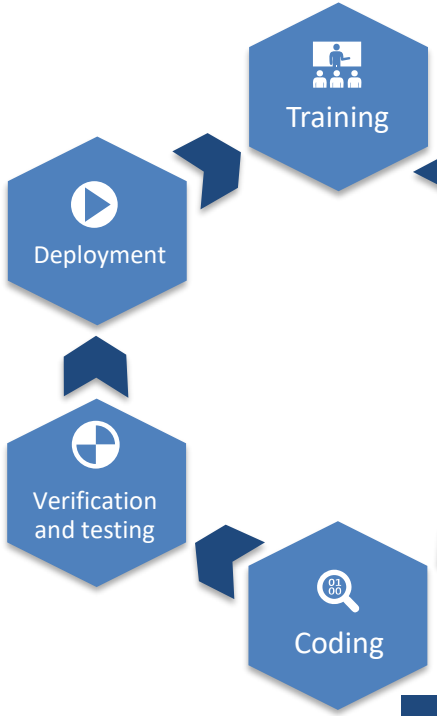
- Example of a Dockerfile from Docker Hub:

```
# Install Supervisor
RUN curl -sS -o - https://bootstrap.pypa.io/ez_setup.py | python && \
    easy_install -q supervisor

# Install Chrome WebDriver
RUN CHROMEDRIVER_VERSION=`curl -sS chromedriver.storage.googleapis.com/LATEST_RELEASE` && \
```

- Can code from **bootstrap.pypa.io** be trusted at any time?
  - What if **bootstrap.pypa.io** has been compromised?

Fixing SDI security structurally



- Establish security and privacy requirements

- Threat modeling
- Attack surface analysis

- Code review
- Use secure third-party code

- Vulnerability scanning
- Penetration testing

- Final security review
- Incident response plan

Secure SDLC needs to cover infrastructure

Code review

- What to look at?
  - Your own code
  - Code provided by external parties, e.g. Puppet Forge, Chef's Supermarket, Docker Hub

- Examples:
  - Remote code inclusion:
    - # find . -type f | xargs grep -i -e "(curl\\|wget)"
  - Too permissive file permissions (e.g. mode 777):
    - # find . -type f | xargs grep '[0-9]\{2\}7'
  - Passwords / secrets
    - # find . -type f | xargs grep -i -e "(pw\\|password\\|pass\\|api\\|key\\|username\\|user\\|email\\|mail\\|token\\|uname\\|credential\\|secret\\|login)"
    - Tool support: gittyleaks for GIT repo's

Summary

- SDI can be used as a tool to improve security, but it can also become an attack vector if not done properly

- Common security issues that are often overlooked:
  - Lack of or coarse-grained access control on code & configuration repositories and cloud API's
  - Insecure handling of secrets
  - Remote code inclusion

- Secure Development Lifecycle also applies to SDIs / "infrastructure as code"
  - Threat modeling & secure code review & automated security testing help to address security issues early on