

# Practical Security Guide

For Java Developers

## Part I

Luca Capacci  
[luca.capacci@cryptonetslabs.it](mailto:luca.capacci@cryptonetslabs.it)

Vittorio Ballestra  
[vittorio.ballestra@cryptonetslabs.it](mailto:vittorio.ballestra@cryptonetslabs.it)

Bologna, 26/11/2018

## AGENDA

- 1 SAMM & BSIMM
- 2 OWASP Top 10
- 3 Lab & playground



# Software Security Initiative

An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion.

Every organization that develops or integrates software needs a software security initiative

Reference frameworks:

**SAMM**: Software Assurance Maturity Model

**BSIMM**: Building Security In Maturity Model



# BSIMM vs SAMM



## SAMM

methodology to improve the company's security posture



## BSIMM

metric to check the company's security posture against a global benchmark

# SAMM: Software Assurance Maturity Model

The **Software Assurance Maturity Model** (SAMM) is an OWASP project

It's an open framework to help organizations **formulate and implement a strategy for software security** that is tailored to the specific risks facing the organization.

SAMM was defined with **flexibility** in mind such that it can be utilized by small, medium, and large organizations using **any style of development**. Additionally, this model can be applied organization-wide, for a single line-of-business, or even for an individual project.



# SAMM – Business Functions

Start with the core activities tied to any organization performing software development

Named generically, but should resonate with any developer or manager



# SAMM – Business Functions



**Governance** is centered on the processes and activities related to how an organization manages overall software development activities. More specifically, this includes concerns that impact cross-functional groups involved in development, as well as business processes that are established at the organization level.



# SAMM – Business Functions



**Construction** concerns the processes and activities related to how an organization defines goals and creates software within development projects. In general, this will include product management, requirements gathering, high-level architecture specification, detailed design, and implementation.

# SAMM – Business Functions



**Verification** is focused on the processes and activities related to how an organization checks, and tests artifacts produced throughout software development. This typically includes quality assurance work such as testing, but it can also include other review and evaluation activities.

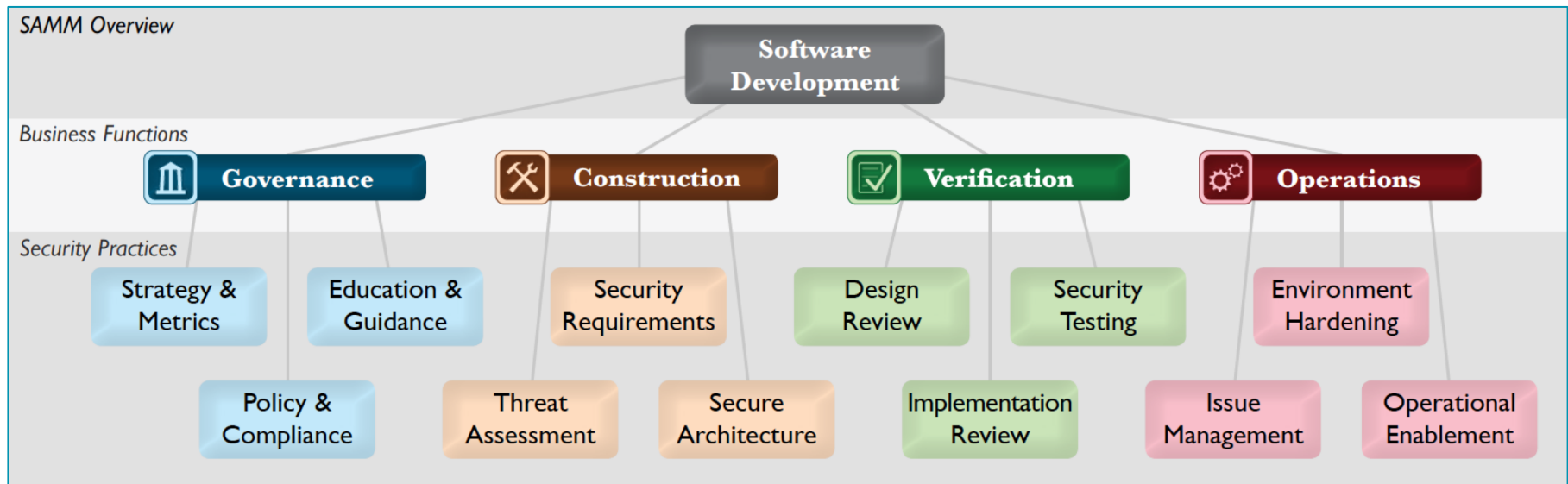
# SAMM – Business Functions



**Operations** entails the processes and activities related to how an organization manages software releases that has been created. This can involve shipping products to end users, deploying products to internal or external hosts, and normal operations of software in the runtime environment.

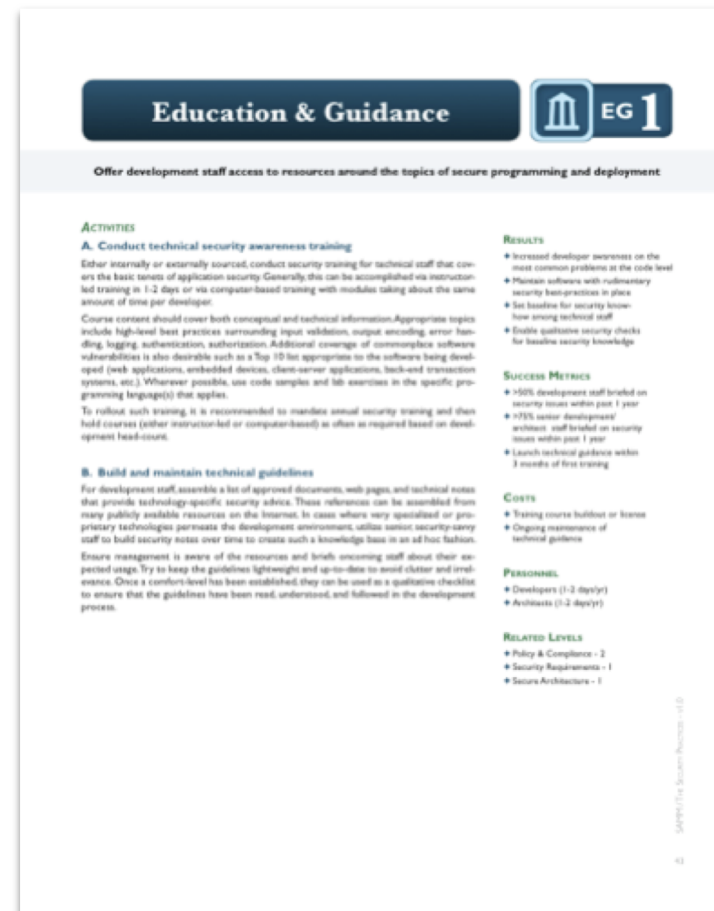
# SAMM – Security Practices

- From each of the Business Functions, 3 Security Practices are defined
- The Security Practices cover all areas relevant to software security assurance
- Each one is a 'silo' for improvement
  - 3 levels for each Security Practice



## Per Level, SAMM defines...

- Objective
- Activities
- Results
- Success Metrics
- Costs
- Personnel
- Related Levels





# Approach to iterative improvement

Simply put, improve an assurance program in phases by:




**Select Security Practices** to improve in next phase of assurance program



**Achieve the next Objective** in each Practice by performing the corresponding Activities at the specified Success Metrics

and the next objective ...

## **Construction**

	Security Requirements		
	 <b>SR 1</b>	 <b>SR 2</b>	 <b>SR 3</b>
<b>OBJECTIVE</b>	Consider security explicitly during the software requirements process	Increase granularity of security requirements derived from business logic and known risks	Mandate security requirements process for all software projects and third-party dependencies
<b>ACTIVITIES</b>	A. Derive security requirements from business functionality B. Evaluate security and compliance guidance for requirements	A. Build an access control matrix for resources and capabilities B. Specify security requirements based on known risks	A. Build security requirements into supplier agreements B. Expand audit program for security requirements

The Security Requirements (SR) Practice is focused on proactively specifying the expected behavior of software with respect to security.



## Governance

Strategy & Metrics → framework, roadmap, KPI  
Policy & Compliance → policies, procedures, contracts  
Education & Guidance → education, guidelines, best practices



## Construction

Threat Assessment → threat model, abuse cases  
Security Requirements → requisiti di sicurezza  
Secure Architecture → principi di progettazione sicura



## Verification

Design Review → review with respect to requirements and best practices  
Implementation Review → manual or automated assessments of the source code  
Security Testing → manual or automated penetration tests and vulnerability assessments



## Operations

Issue Management → remediation process  
Environment Hardening → configuration and patch management  
Operational Enablement → documentation for operators and users

# BSIMM: Building Security In Maturity Model

The **Building Security In Maturity Model** (BSIMM, pronounced “bee simm”) is a study of existing software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variations that make each unique.

BSIMM **is not a how-to guide**, nor is it a one-size-fits-all prescription. Instead, **it is a reflection of software security**.

- Framework derived from SAMM Beta
- Based on collected data from > 100 firms



# What the BSIMM enables you to do



## 1 Start a software security initiative (SSI) using real data

If you don't have a software security initiative yet, you need one. Before you start down that path, the BSIMM will help you identify the core activities that all successful initiatives undertake – no matter what industry you're in.

## 2 Compare your SSI to other firms in your industry

Measure how your SSI stacks up against the rest of your industry peers. With your goals in mind, you can determine where you stand relative to your needs.



# What the BSIMM enables you to do



## 3 Benchmark and track your SSI growth

A repeatable way to measure your SSI's effectiveness. Once your SSI is established, you can use it to measure your continuous improvement year over year. It will also provide concrete details to show your executive team and board how your security efforts are making a difference.

## 4 Evolve your initiative using lessons learned from mature initiatives

The BSIMM is a “what works” report on building and evolving a software security initiative. It comprises proven activities that mature organizations are performing today.

## More than 100 firms in BSIMM



# BSIMM Firms



Dataset:

- 120 organizations

Industries:

- financial services
- independent software vendors
- technology
- healthcare
- the cloud
- the Internet of Things (IoT)
- insurance

# The BSIMM Framework

4 Domains → 12 Practices → 116 Activities



**Governance.** Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.



**Intelligence.** Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.



**SSDL Touchpoints.** Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.



**Deployment.** Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

# The BSIMM Framework





# BSIMM Activities

ATTACK MODELS (AM)		
ACTIVITY DESCRIPTION	ACTIVITY	PARTICIPANT %
LEVEL 1		
Create a data classification scheme and inventory.	AM1.2	62.5
Identify potential attackers.	AM1.3	31.7
Gather and use attack intelligence.	AM1.5	44.2
LEVEL 2		
Build attack patterns and abuse cases tied to potential attackers.	AM2.1	8.3
Create technology-specific attack patterns.	AM2.2	8.3
Build and maintain a top <i>N</i> possible attacks list.	AM2.5	13.3
Collect and publish attack stories.	AM2.6	11.7
Build an internal forum to discuss attacks.	AM2.7	9.2
LEVEL 3		
Have a science team that develops new attack methods.	AM3.1	3.3
Create and use automation to mimic attackers.	AM3.2	1.7

# BSIMM 12 Core Activities

12 activities were observed in at least 62% of the firms

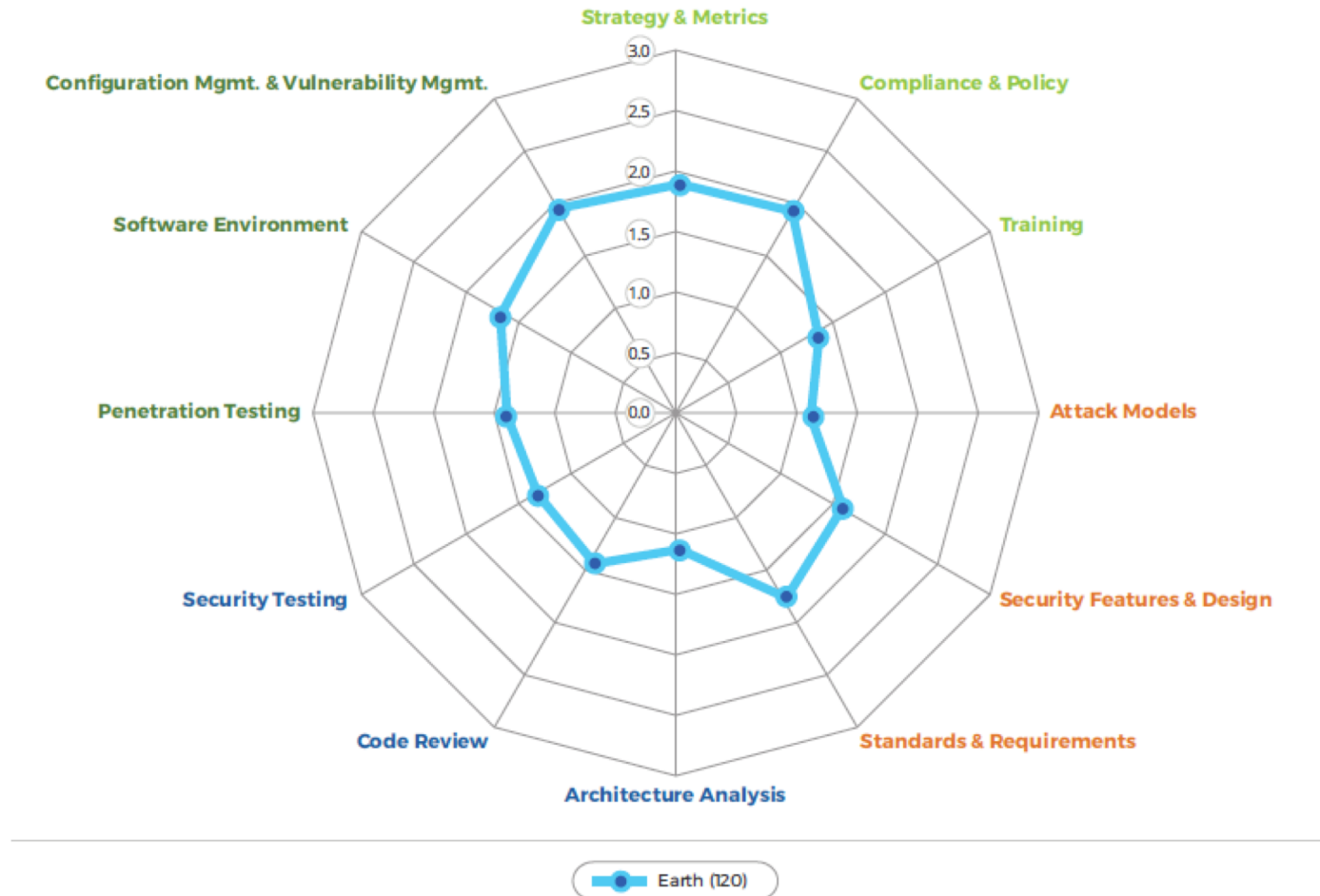


*“Although we can’t directly conclude that these 12 activities are necessary for all SSIs, we can say with confidence that these activities are commonly found in highly successful initiatives. This suggests that if you are working on an initiative of your own, you should consider these 12 activities particularly carefully.”*

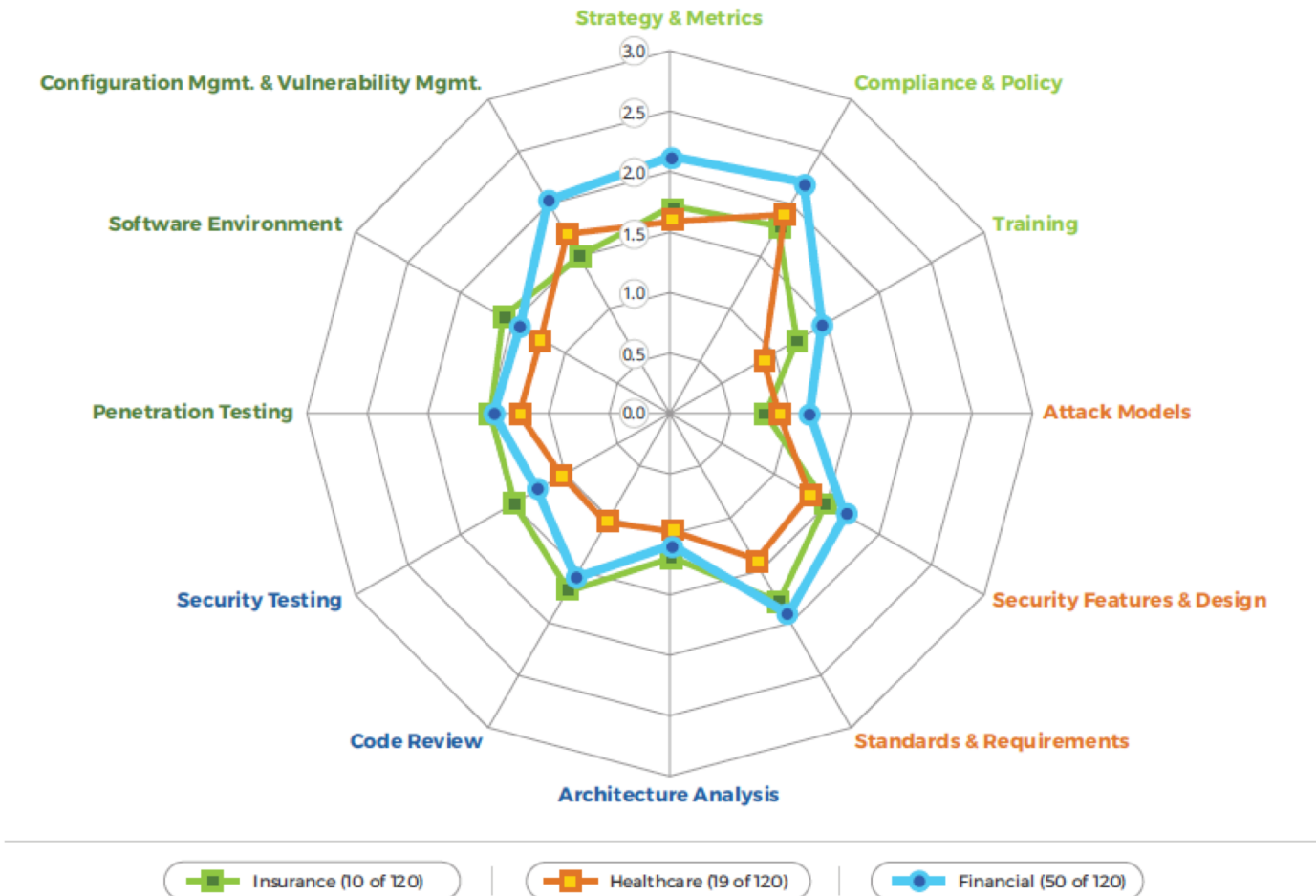
# BSIMM 12 Core Activities

ACTIVITY	DESCRIPTION
[SM1.4]	Identify gate locations and gather necessary artifacts.
[CP1.2]	Identify PII obligations.
[T1.1]	Provide awareness training.
[AM1.2]	Create a data classification scheme and inventory.
[SFD1.1]	Build and publish security features.
[SR1.2]	Create a security portal.
[AA1.1]	Perform security feature review.
[CR1.2]	Have SSG perform ad hoc review.
[ST1.1]	Ensure QA supports edge/boundary value condition testing.
[PT1.1]	Use external penetration testers to find problems.
[SE1.2]	Ensure host and network security basics are in place.
[CMVM1.2]	Identify software bugs found in operations monitoring and feed them back to development.

# Global security posture according to BSIMM



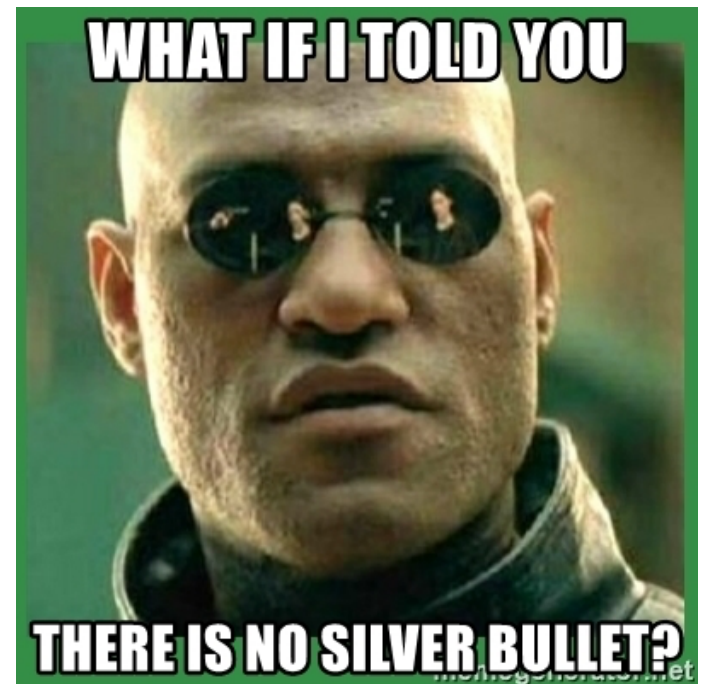
# Global security posture according to BSIMM



# SO ... WHAT'S THIS SECURE PROGRAMMING THING?

# Secure Programming

- First rule of “Secure Programming”
  - There’s no such a thing as “Secure Programming”
  - Remember:
    - Security is an “emergent” feature
    - No “silver bullet” for security



# Security is an “emergent feature”

- What is an “emergent feature” ?
  - A feature that “emerges” only in an aggregation of smaller elements.
- Example:
  - Temperature:
    - There’s no such a thing as the “temperature of a single atom”.
    - “Temperature” is a feature that arises only within an aggregation of atoms like for instance a gas.



# Security is an “emergent feature”

- In the same way
  - A single line of code is not “secure”, it all depends on the whole system
  - Security is a feature that “emerges” from the whole software development process, the deployment environment
- Let’s see it in more details
  - In order to better understand this concept let’s look at some of the most relevant security vulnerabilities
  - In order to fix it learn how to break it !

# OWASP TOP 10 / 2017

1

## A1/2017 - Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

2

## A2/2017 - Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

3

## A3/2017 - Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

4

## A4/2017 - XML External Entities (XXE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

5

## A5/2017 - Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

## 6 A6/2017 - Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/updated in a timely fashion.

## 7 A7/2017 - Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

## 8 A8/2017 - Insecure Deserialization

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

## 9 A9/2017 - Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts

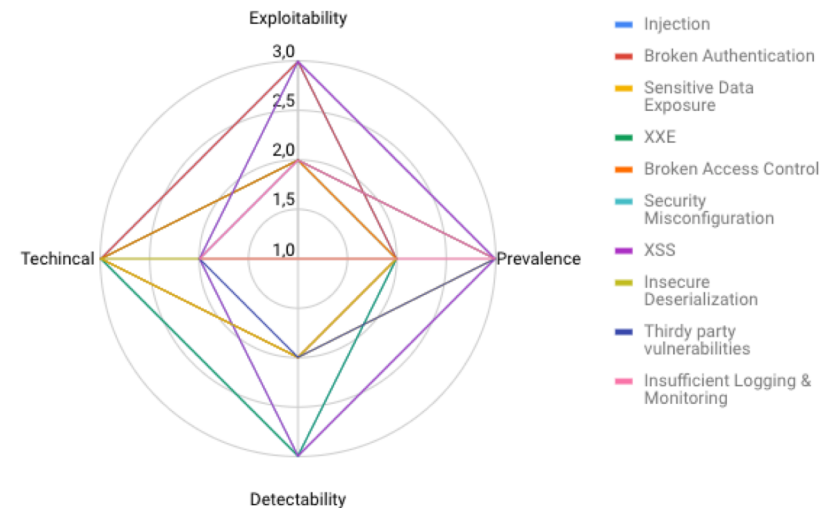
## 10 A10/2017 - Insufficient Logging & Monitoring

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

# OWASP TOP 10 / 2017

Every threat is evaluated according to the following criteria:

- **Exploitability**  
How easy or hard is to exploit that vulnerability (1=hard, 2=medium, 3=easy)
- **Detectability**  
How easy or hard is to find systems affected by the vulnerability (1=hard, 2=medium, 3=easy)
- **Technical**  
The technical impact on the system (1=minor, 2=moderate, 3=severe)
- **Prevalence**  
The spread of the vulnerability (1=uncommon, 2=common, 3=widespread)



# OWASP TOP 10 / HistoryASP TOP

		2017		2013	2010
A1	=	Injection	=	Injection	Injection
A2	=	Broken Authentication	=	Broken Authentication	XSS
A3	+++	Sensitive Data Exposure	=	XSS	Broken Authentication
A4	NEW	XXE	=	Broken Access Control / Insecure Direct Object References	Insecure Direct Object References
A5	-	Broken Access Control	+	Security Misconfiguration	CSRF
A6	-	Security Misconfiguration	++	Sensitive Data Exposure	Security Misconfiguration
A7	----	XSS	+	Broken Access Control / Missing Function Level Access Control	Sensitive Data Exposure / Insecure Cryptographic Storage
A8	NEW	Insecure Deserialization	---	CSRF	Missing Function Level Access Control (Failure to Restrict URL Access)
A9	=	Third party vulnerabilities	NEW	Third party vulnerabilities	Sensitive Data Exposure / Insufficient Transport Layer Protection
A10	NEW	Insufficient Logging & Monitoring	=	Unvalidated Redirects and Forwards	Unvalidated Redirects and Forwards

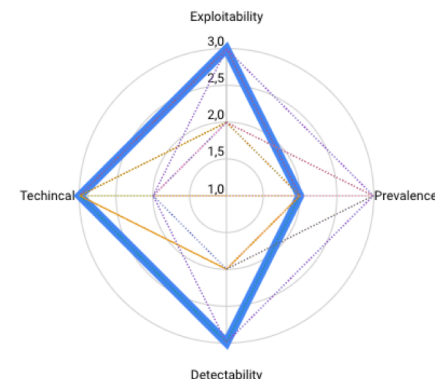
# A1/2017 - Injection



## When it happens ?

«External» input used without validation / sanitization:

- query SQL / JPA-QL / HQL / MongoDB / Elasticsearch / ecc.
- XPath query
- Generating HTML output
- Query LDAP



## How can we prevent it ?

- Always try to use parametrics queries
- Validate through Pattern matching
- Implement sanitization
- [OWASP Injection\\_Prevention\\_Cheat\\_Sheet\\_in\\_Java](#)

# A1/2017 - Injection / Esempio JDBC

```
void updatePassword(String username, String password) {  
    String query = "UPDATE users SET password = '" + password  
        + "' WHERE username='" + username + "'";  
    PreparedStatement ps = con.prepareStatement(query);  
    ps.executeUpdate();  
    ps.close();  
    ...  
}  
...  
updatePassword("mario", "Password_for_everybody' -- ");
```

```
UPDATE users SET password='Password_for_everybody' -- ' WHERE username='mario'
```

# A1/2017 - Injection / JDBC - soluzione

```
void updatePassword(String username, String password) {  
    String query = "UPDATE users SET password = ? WHERE username = ?";  
    PreparedStatement ps = con.prepareStatement(query);  
    ps.setString(1, username);  
    ps.setString(2, password);  
    ps.executeUpdate();  
    ps.close();  
    ...  
}
```



# A1/2017 - Injection / JDBC - lab

- Lab sample
  - Open playground
    - Run demo
    - Inspect code

# A1/2017 – Injection: not only SQL

- JPA / HQL / ecc.
  - Always use parametric queries
    - es. :

```
session.createQuery("FROM MyEntity WHERE id IN :collection")  
    .setParameter("collection",Arrays.asList(1,2,3))  
    .list()
```
- Use QBE
  - es. :

```
session.createCriteria(MyEntity.class)  
    .add(Criteria.in("id",Arrays.asList(1,2,3)))  
    .list();
```

# A1/2017 – Injection: not only SQL


- XPath query
  - Use a variable resolver

```
/**
 * Resolver in order to define parameter for XPATH expression.
 *
 */
public class SimpleVariableResolver implements XPathVariableResolver {

    private final Map<QName, Object> vars = new HashMap<QName, Object>();

    /**
     * External methods to add parameter
     *
     * @param name Parameter name
     * @param value Parameter value
     */
    public void addVariable(QName name, Object value) {
        vars.put(name, value);
    }

    /**
     * {@inheritDoc}
     *
     * @see javax.xml.xpath.XPathVariableResolver#resolveVariable(javax.xml.namespace.QName)
     */
    public Object resolveVariable(QName variableName) {
        return vars.get(variableName);
    }
}
```



```
/*Create and configure XPATH expression*/
XPath xpath = XPathFactory.newInstance().newXPath();
xpath.setXPathVariableResolver(variableResolver);
XPathExpression xpathExpression = xpath.compile("//book[@id=$bookId]");
```

# A1/2017 – Injection: not only SQL

More examples:

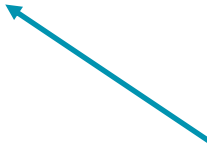
- LDAP (e.g. used as an authentication backend):

String searchlogin= "(&(uid="+user+") (userPassword="+encrypt(password)+"))";

user = "\*)(uid=\*))(|(uid=\*";

Resulting:

(&(uid=\*)(uid=\*))(|(uid=\*)(userPassword={MD5}X03M01qnZdYdgyfeulLPmQ==))



Only the first part of the query gets executed

# A1/2017 – Injection: not only SQL

More examples:

- Command Injection:

```
Process p = Runtime.getRuntime().exec(new String[]{"/bin/bash"});  
PrintWriter w = new PrintWriter(new OutputStreamWriter(p.getOutputStream()));  
w.println("USERNAME=\""+username+"\"");  
w.println("echo \"Check $USERNAME\"");  
w.close();  
p.waitFor();
```

+

```
username = "mario`/bin/echo 'newuser:newpassword' >> /tmp/passwd`";
```

=

```
~ % cat /tmp/passwd  
newuser:newpassword
```

# A1/2017 – Injection: not only SQL

More examples:

- Command Injection:

```
String appHome = System.getProperty("APP_HOME");  
Runtime.getRuntime().exec( "command: appHome+\"/scripts/myscript.sh\"");
```

- APP\_HOME is not sanitized, what will happen if it is set to
  - `"/bin/bash -c 'rm -rf /' ;" ?`

# A1/2017 – Injection: not only SQL

- Lab sample
  - Open playground
  - Run command example
  - Inspect code

# A1/2017 - Injection

What can we do when there are no “safe” way of parametrize your query ?

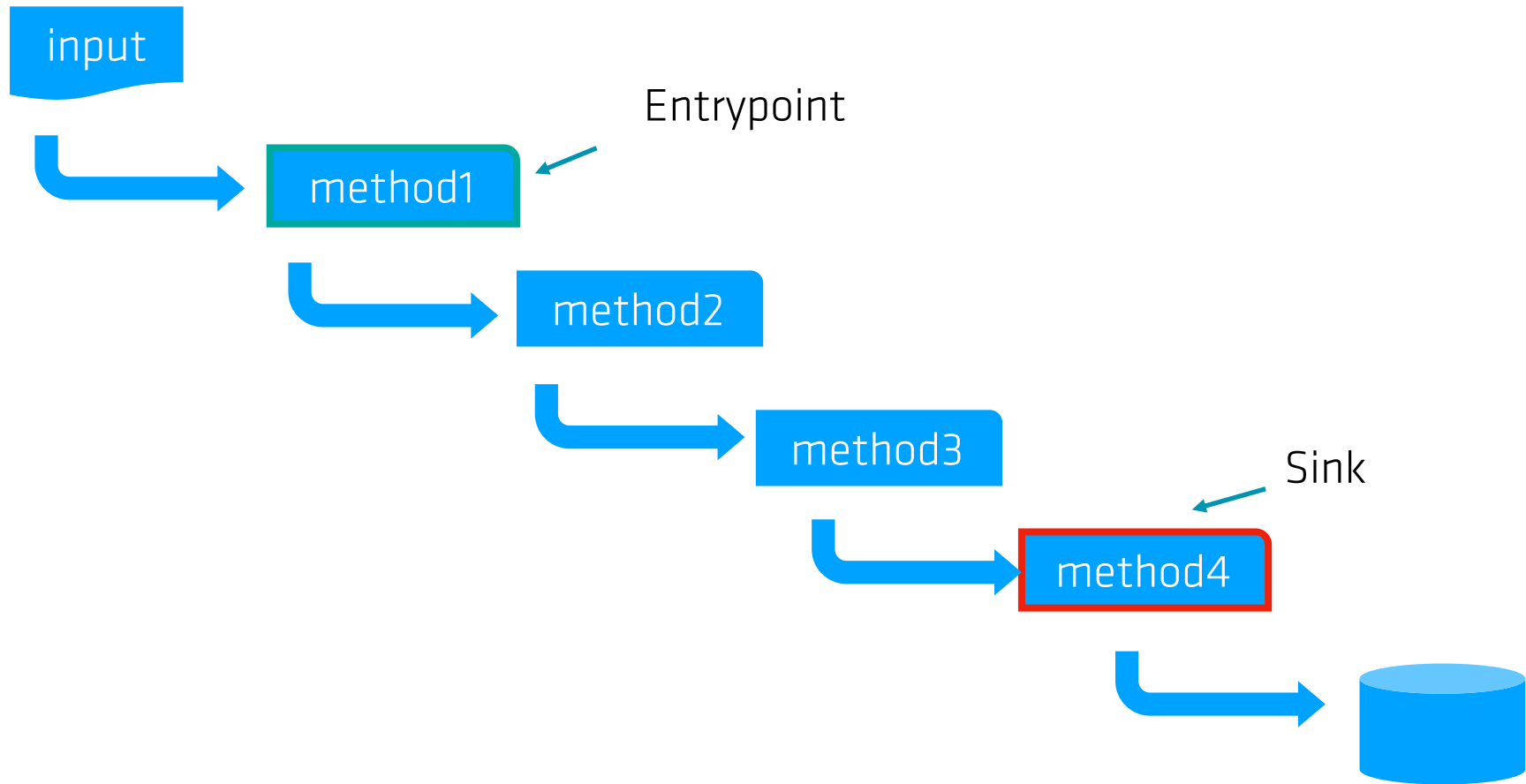
- **Validate the input, e.g.:**
  - use regular expressions to validate the data
  - try to convert to typed data (es.: int / long / Date)
    - A data should be a data, an int should be an int, ...
- **Sanitize input**
  - Escape «dangerous» characters:  
`input.replaceAll("\"", "\\");`
  - Remove ‘dangerous’ characters  
`input.replaceAll("[&]", "_");`



# A1/2017 - Injection

- In general:
  - Use parametrized query when available
  - Validate and sanitize external data before using it
  - Where to do input data validation ?
  - **Validate outputs!**

# Data Validation



# Data Flow

- Entrypoint
  - Where un-trusted data "enters" the system
- Sink
  - Where data get "used" :
    - To query a data source
    - To execute a command
    - To generate a template

# Data validation

- When should I do data validation ?
  - Everywhere :
    - Performance problem
    - Introduces DOS
  - Entry-points :
    - Only validation should be done here (to avoid data mangling, e.g. escaping multiple times, wrong comparisons)
    - Different path of execution can lead to injection
  - At sink :
    - Both sanitization and validation is possible
    - Sinks are generally more difficult to identify than entry points

# A8/2017 - Insecure Deserializatic



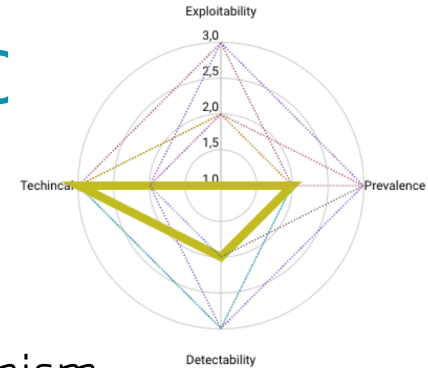
## When it happens ?

- Every time you use an deserialization mechanism
  - Deserialization can be exploited to execute arbitrary code



## How can we prevent it ?

- Use safer serialization formats (e.g. json, google protobuf)
- Strict type checking
- Digital signatures to avoid tampering
- Log and monitoring



# A8/2017 - Insecure Deserialization

- Malicious code execution
  - Gadget classes
    - Classes that can execute code during deserialization
    - Customized serialization can be implemented using the following two methods:
      - **private void writeObject(ObjectOutputStream oos) throws Exception:** This method will be executed automatically by the jvm(also known as Callback Methods) at the time of serialization. Hence to perform any activity during serialization, it must be defined only in this method.
      - **private void readObject(ObjectInputStream ois) throws Exception:** This method will be executed automatically by the jvm(also known as Callback Methods) at the time of deserialization. Hence to perform any activity during deserialization, it must be defined only in this method.

# A8/2017 - Insecure Deserialization

- Gadget classes example

```
package org.acme.security.playground.deser;

import java.io.IOException;
import java.io.Serializable;

public class SampleGadget1 implements Runnable, Serializable {
    private final String command;

    public SampleGadget1(String command) {
        this.command = command;
    }

    @Override
    public void run() {
        try {
            Runtime.getRuntime().exec(command);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

+

```
package org.acme.security.playground.deser;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.Serializable;

public class SampleGadgetFactory1 implements Serializable {

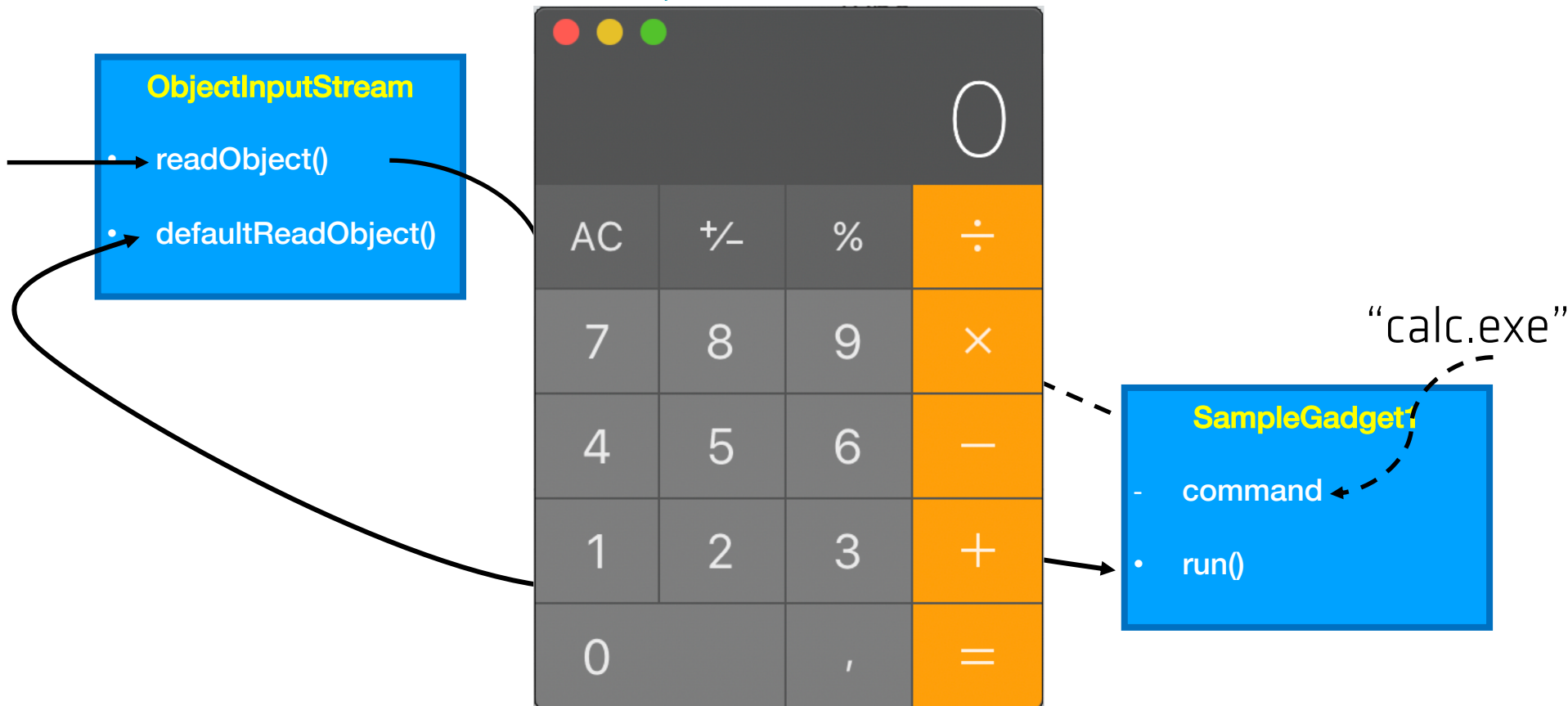
    private final Runnable initHook;

    public SampleGadgetFactory1(Runnable initHook) {
        this.initHook = initHook;
    }

    /**
     * Java serialization allows to customize object deserialization
     */
    public void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
        ois.defaultReadObject();
        initHook.run();
    }
}
```

# A8/2017 - Insecure Deserialization

- Insecure deserialization example



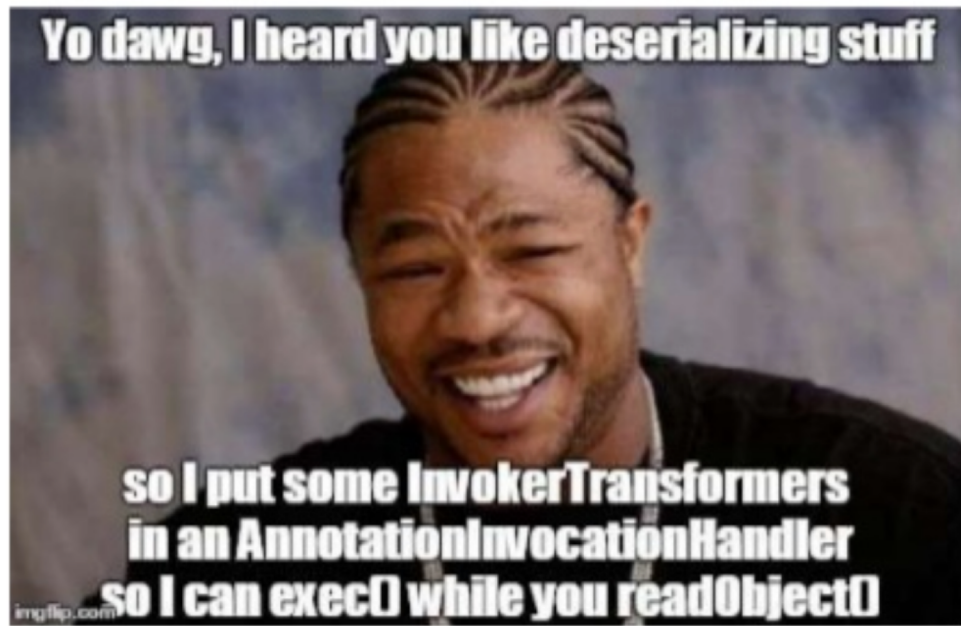


# Lab

- Lab demo
  - Normal method execution
  - Create a malicious payload
  - Use malicious payload

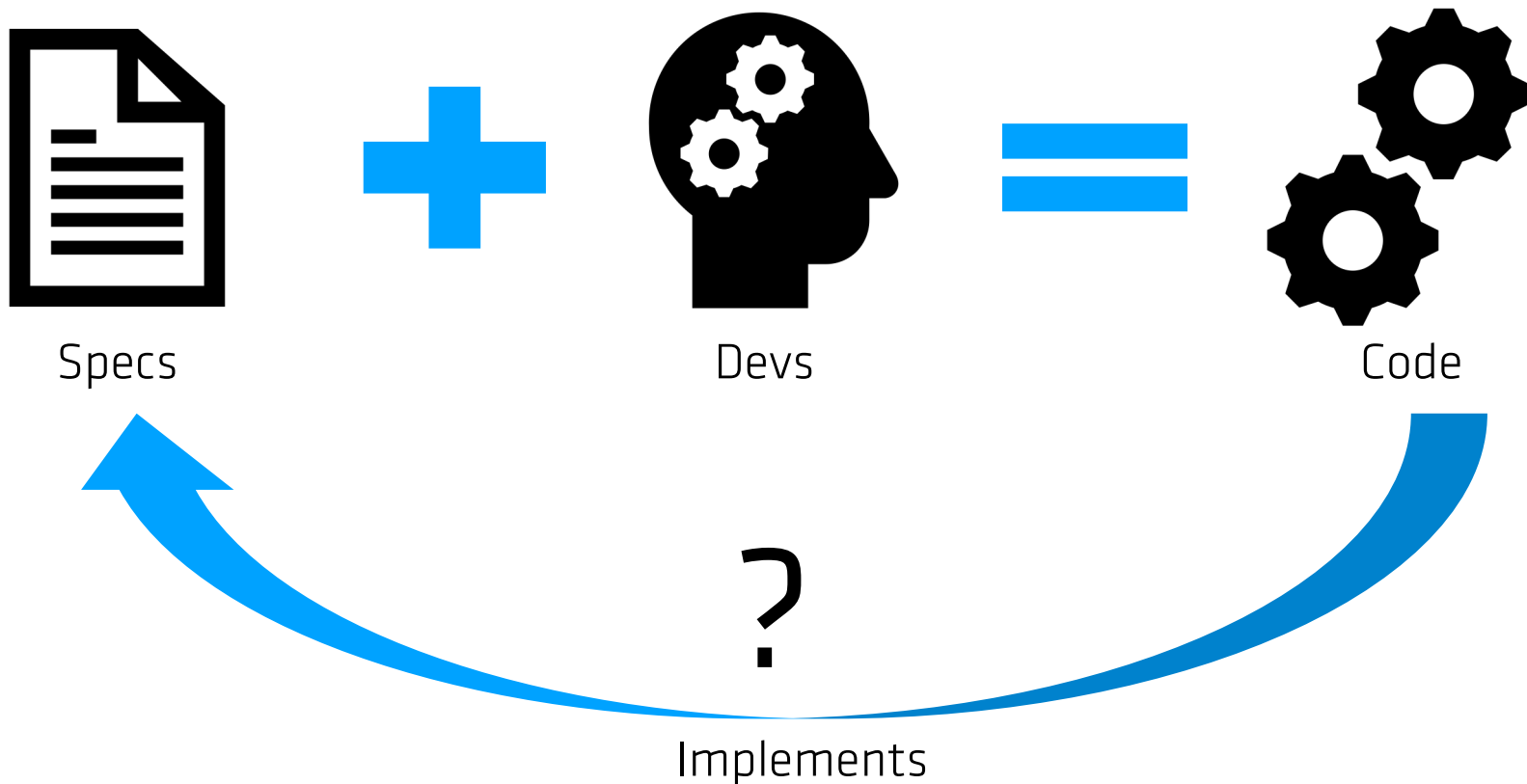
# A8/2017 - Insecure Deserialization

- Libraries containing gadget classes:
  - Apache common collections
  - Apache FileUpload
  - Mozilla Rhino
  - BSH
  - ... and many other



# A secure development process

# Software development process



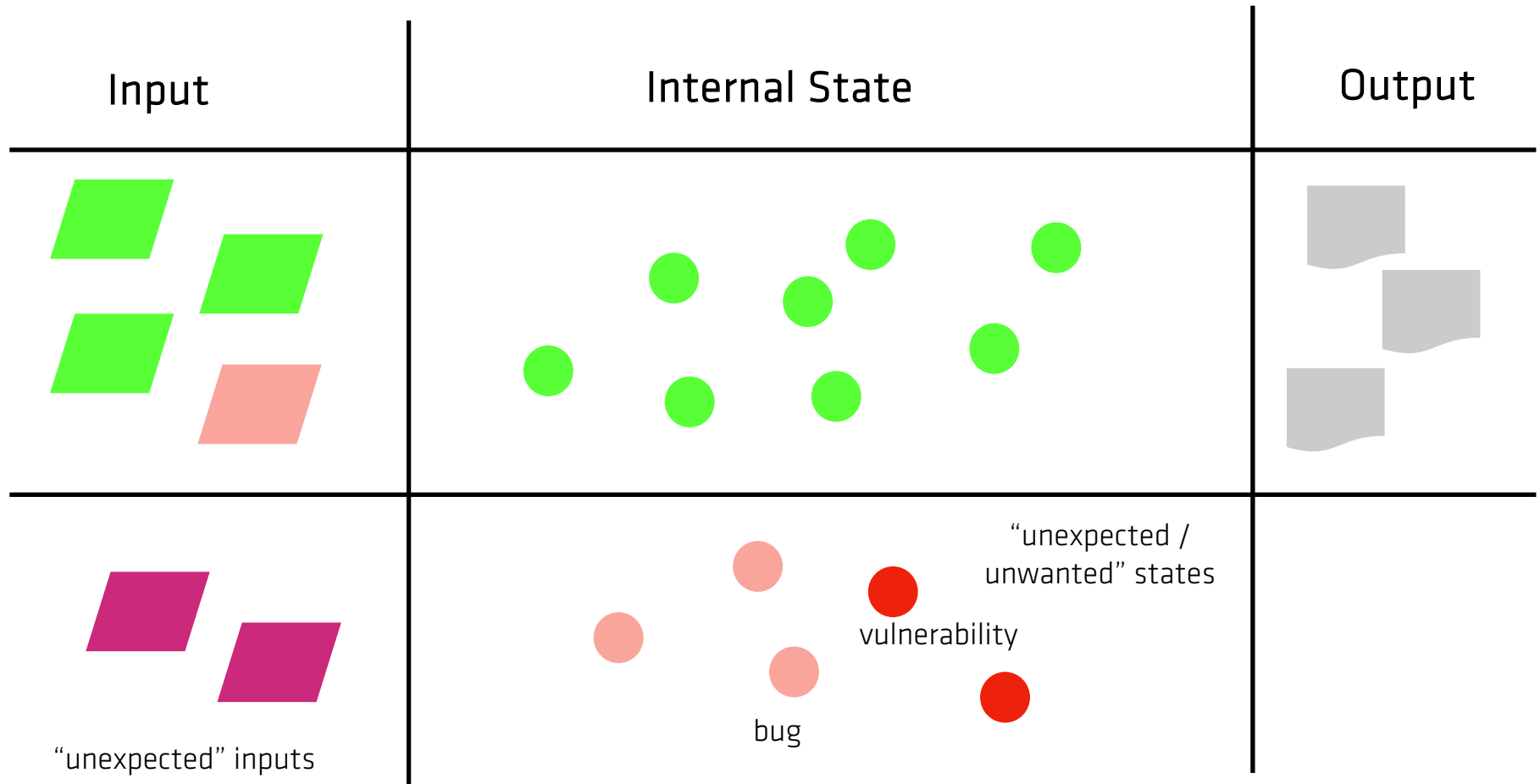
# Software development process

- A possible definition :

*“A process performed by human beings to bring the best algorithmical approximation from a set of ambiguous and incomplete requirements”*

- *The “Halting Problem”, computational theory*
  - Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist. A key part of the proof was a mathematical definition of a computer and program, which became known as a Turing machine; the halting problem is undecidable over Turing machines. It is one of the first examples of a decision problem.

# Software, bugs and vulnerabilities

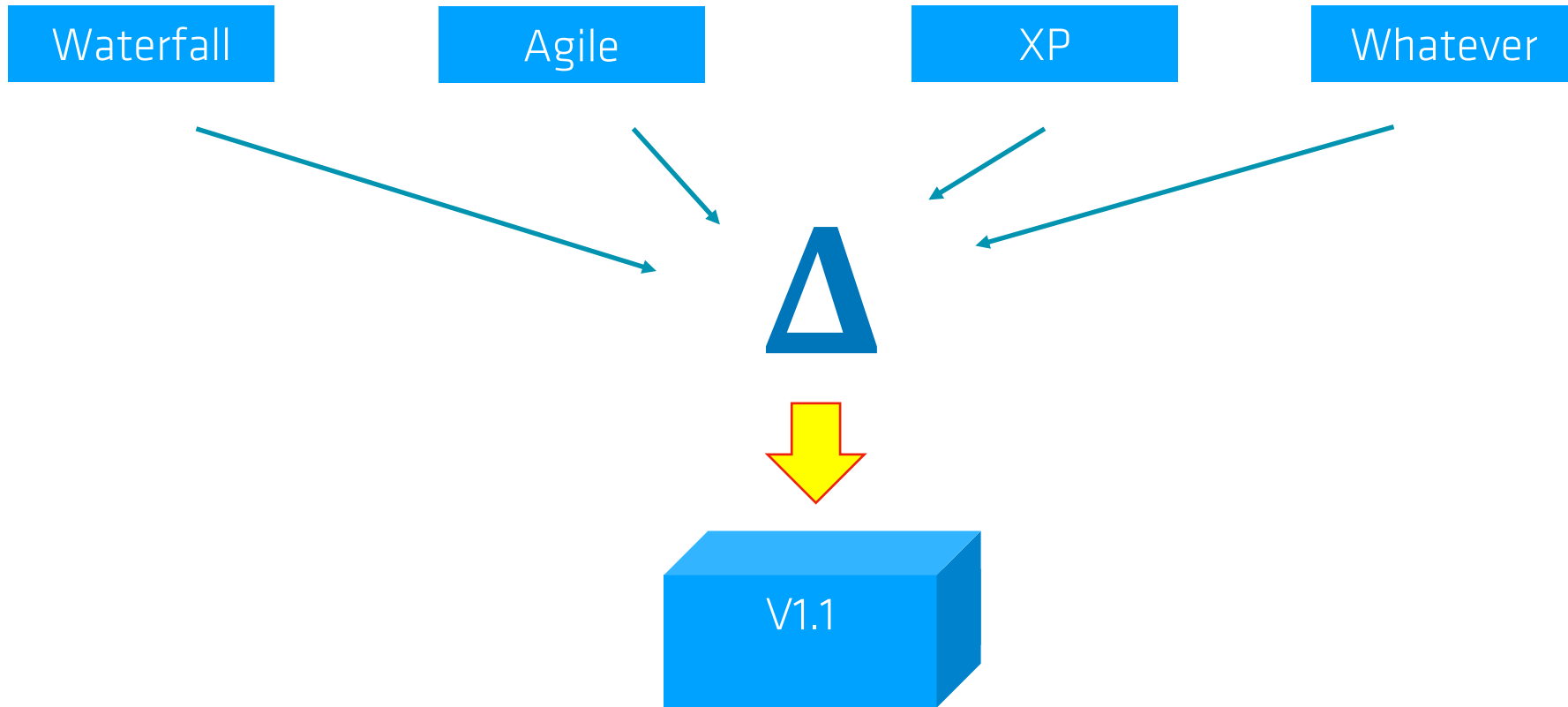


# Secure Development Process

- A secure development process should
  - Limit the occurrence of “unexpected/unwanted states”, in other words limit bugs:
    - a secure development process should be first of all a quality rewarding development software process
  - Limit the range of acceptable inputs
    - Inputs validation
    - Inputs sanitization



# Focus on integration step

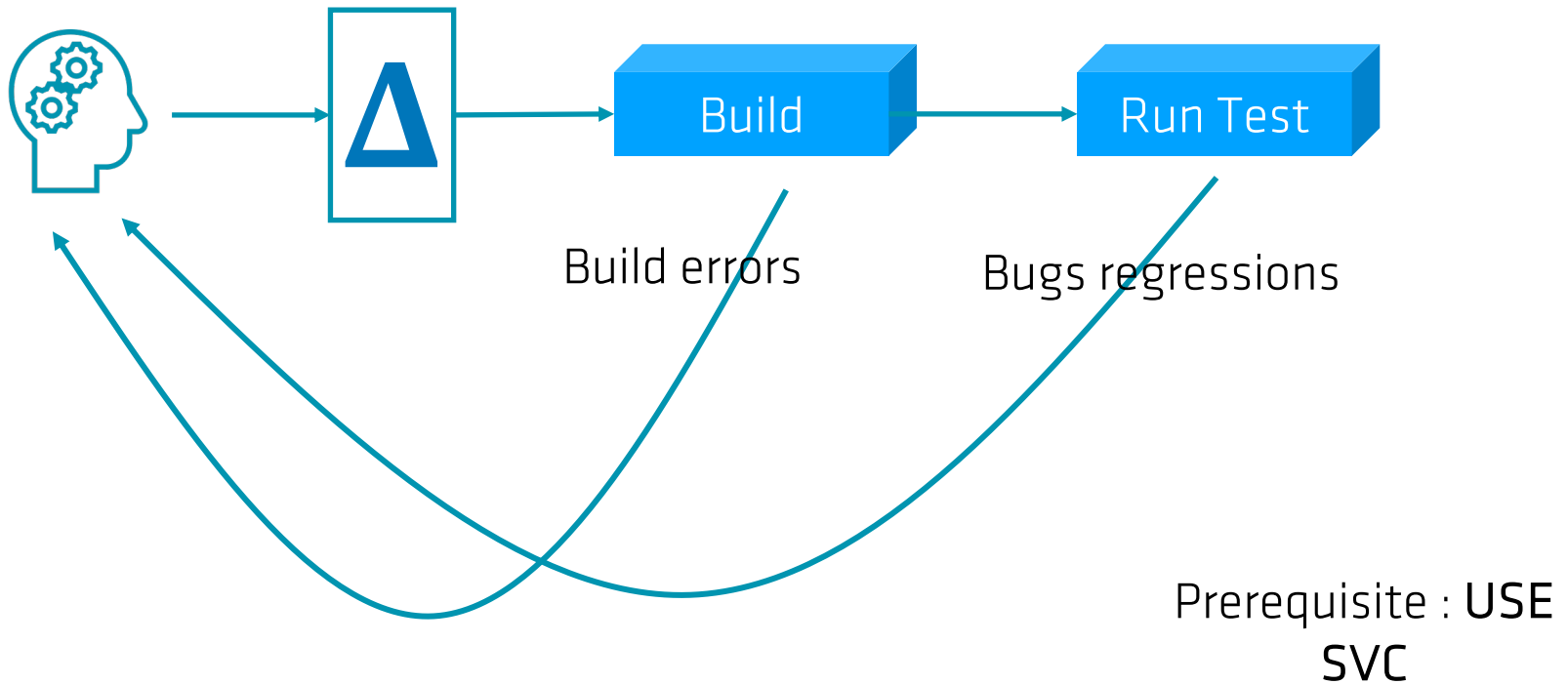




# Integration/merge Step

- A good integration step
  - Should preserve
  - Should not introduce new bugs
  - Should not introduce regressions
- How to achieve this goal ?
  - CI / CD
  - Test automation
  - Metrics
  - Code Review

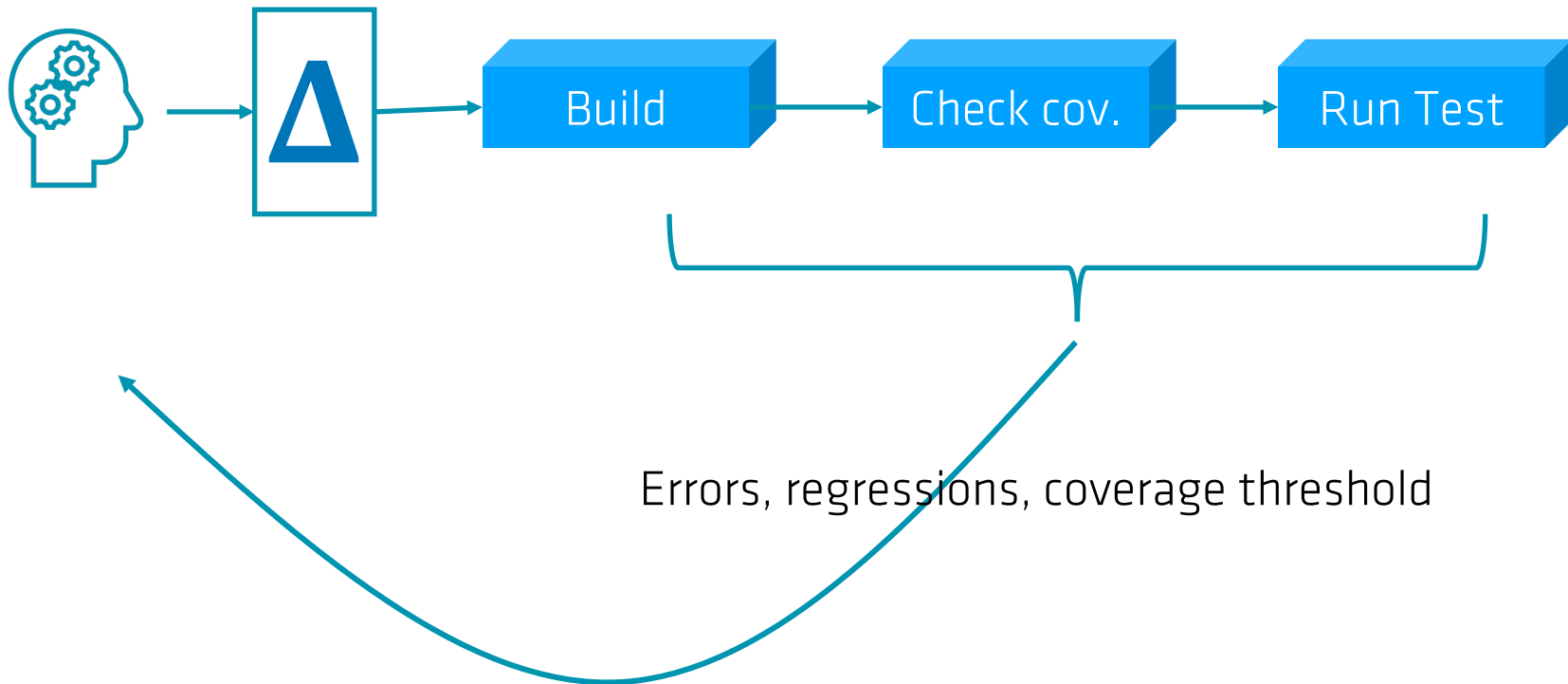
## A sample SDLC (1)



# Is this enough ?

- Test automation
  - In order to be meaningful a sufficient code coverage should be achieved AND preserved
- Enters “Test coverage”
  - Lab :
    - jacoco test coverage
    - Improve test coverage

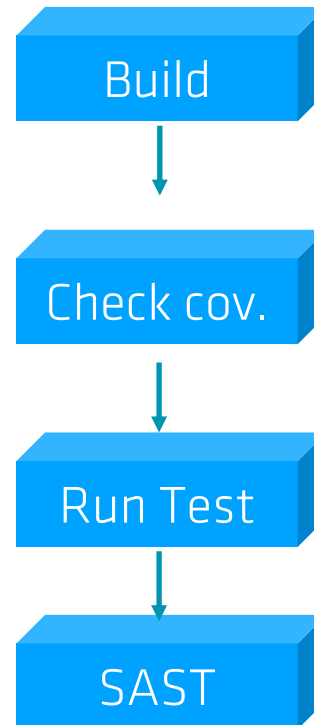
## A sample SDLC (2)



# Is this enough ?

- Software metrics
  - We can generalize the concept by measuring a set of “software metrics” that should be preserved between merges
- Enters “Static Application Security Testing”
  - Lab :
    - Sonarqube analysis
    - Identify vulnerabilities
    - Improve security

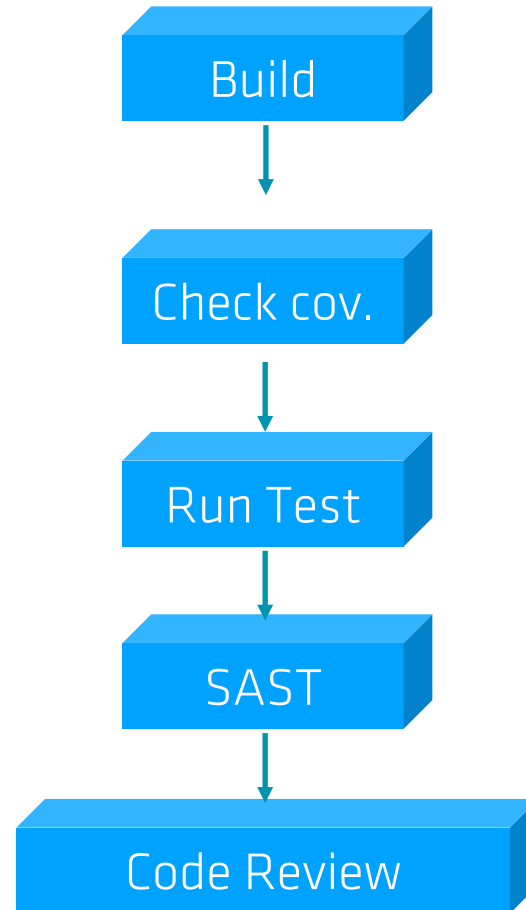
## A sample SDLC (3)



# Is this enough ?

- Are automated tools enough ?
  - Some bugs / vulnerabilities aren't detected (silver bullet ?)
- Enters "Code Review"
  - Lab :
    - Inspect code and find not detected vulnerabilities

## A sample SDLC (4)





# Is this enough ?

- How to further improve security ?
  - Require more than one review
  - Dedicated security reviews
  - Security checks in the integration steps aren't enough to guarantee the final result security
  - DAST : dynamic analysis security testing
  - ...

Q / A

[tesi@cryptonetlabs.it](mailto:tesi@cryptonetlabs.it)