Industrial Challenges of Secure Software Development

Achim D. Brucker
achim.brucker@sap.com

SAP SE
Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany

CSP Forum Cluster Workshop, Florence, Italy
October 8, 2014

Agenda

1 Introduction & Motivation

2 Secure Software Development at SAP

3 Challenges in Industrial Software Development

4 Discussion About Future Research Directions

Fact Sheet: SAP SE

- Leader in Business Software
  - Cloud
  - Mobile
  - On premise
- Many different technologies and platforms, e.g.,
  - In-memory database and application server (HANA)
  - Netweaver for ABAP and Java
- More than 25 industries
- 63% of the world's transaction revenue touches an SAP system
- More than 67 000 employees worldwide
- Headquartered in Walldorf, Germany (close to Heidelberg)

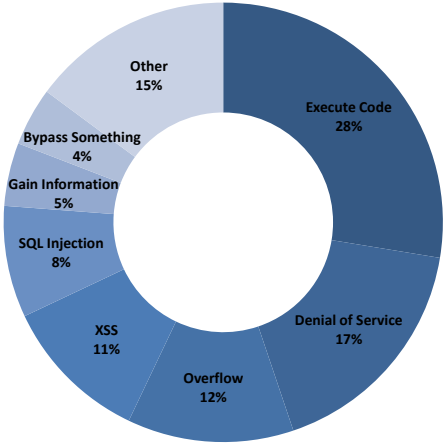
Costs of Vulnerabilities (Attacks on IT Systems)

- TJX Company, Inc. (2007) $250 million
- Sony (2011) $170 million
- Heartland Payment Systems (2009) $41 million

> “A hack not only costs a company money, but also its **reputation** and the **trust** of its customers. It can take years and millions of dollars to repair the damage that a single computer hack inflicts.

(http://financialedge.investopedia.com/financial-edge/0711/Most-Costly-Computer-Hacks-Of-All-Time.aspx)

Vulnerability types of CVE reports since 1999



- Causes for most vulnerabilities are
  - programming errors
  - configuration errors
- Patching
  - is expensive
  - may introduce new bugs

How can we ensure that no vulnerable code is shipped?

Agenda

Secure Software Development Lifecycle

Secure Software Development Lifecycle

Training

- Goals:
  - Create security awareness across all areas and roles
  - Role-specific education, e.g., in development (examples):
    - Threat modeling
    - Secure Programming: ABAP, Java, C/C++
    - Security testing (including static analysis)
- Security expert curriculum
  - one year (part-time)
  - official job specialization

Secure Software Development Lifecycle

Requirements: SAP Product Standard Security

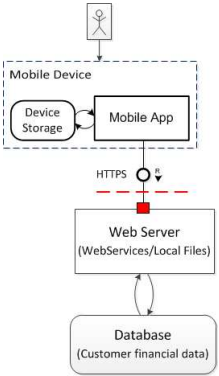
Security and data protection requirements for all products

1. Regulatory Compliance
   - SAP software shall be able to log security relevant events
2. Data Protection and Privacy
   - SAP software shall support erasure of personal data
3. Vulnerability Prevention
   - SAP software shall be free of SQL Injection vulnerabilities
4. Strategy and Attack Surface Reduction
   - SAP products shall support to be deployed and run securely in segmented networks

Secure Software Development Lifecycle

Requirements and Design: Threat Modeling

Training → Requirements → Design → Implementation → Validation → Shipment → Response



Workshops with

- Threat modeling experts
- Security experts
- Product owner
- Architect
- . . .

Systematic analysis, e.g.,

- What data is stored on device?
- How is the user authenticated?

Secure Software Development Lifecycle

Implementation: Security Testing

- Goal: ensure that security controls are effective
- Static analysis since 2010 mandatory for all products

Language	Tool	Vendor
ABAP	Netweaver CVA	SAP
C/C++	Coverity	Coverity
JavaScript, Ruby	Checkmarx	Checkmarx
Others	Fortify	HP

- Additional security testing activities based on requirements and design (IBM AppScan, HP WebInspect, Burp Suite, Fuzzers, own tooling, . . .)

Secure Software Development Lifecycle

Validation

Before shipment:

- Check that products comply to SAP standards
- Important part: penetration testing
- Executed by independent group within SAP
- Validates quality of previous steps in SSDL

Secure Software Development Lifecycle

Response

After shipment:

- Handle security related bug reports
- Monitor external security community
- Organize roll-out of security patches

Secure Software Development Lifecycle

So Everything is Secure Now, Right?

> Our tool reports all vulnerabilities in your software – you only need to fix them and you are secure.
>
> Undisclosed sales engineer from a SAST tool vendor.

So Everything is Secure Now, Right?

> Our tool reports all vulnerabilities in your software – you only need to fix them and you are secure.
>
> Undisclosed sales engineer from a SAST tool vendor.

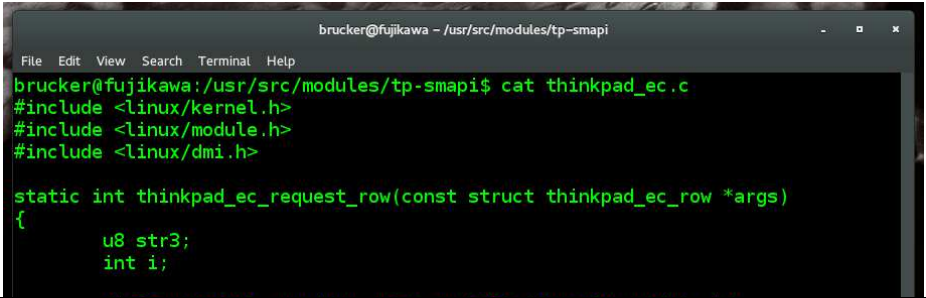
Yes, this tools exists! It is called Code Assurance Tool (cat):

So Everything is Secure Now, Right?

> Our tool reports all vulnerabilities in your software – you only need to fix them and you are secure.
>
> Undisclosed sales engineer from a SAST tool vendor.

Yes, this tools exists! It is called Code Assurance Tool (cat):

- The cat tool reports each line, that might contain a vulnerability:

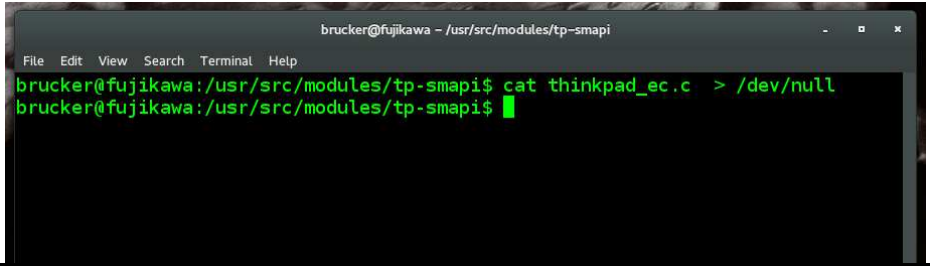


So Everything is Secure Now, Right?

> Our tool reports all vulnerabilities in your software – you only need to fix them and you are secure.
>
> Undisclosed sales engineer from a SAST tool vendor.

Yes, this tools exists! It is called Code Assurance Tool (cat):

- The cat tool reports each line, that might contain a vulnerability:
- It supports also a mode that reports no false positives:



So Everything is Secure Now, Right?

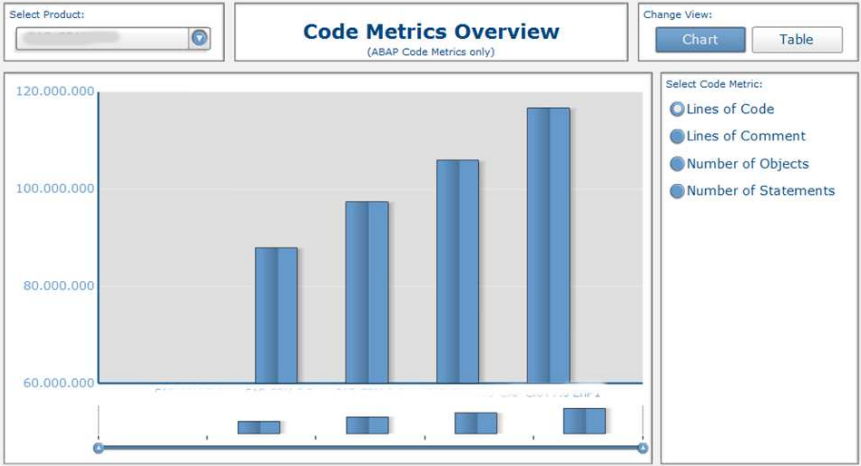
> Our tool reports all vulnerabilities in your software – you only need to fix them and you are secure.
>
> Undisclosed sales engineer from a SAST tool vendor.

Yes, this tools exists! It is called Code Assurance Tool (cat):

- The cat tool reports each line, that might contain a vulnerability:
- It supports also a mode that reports no false positives:
- Note:
  - There are sound or complete tools, but only for specific domains
  - In practice,
    - requirements are not formal enough to be sound and complete
    - scalability is very important
    - modularity is very important

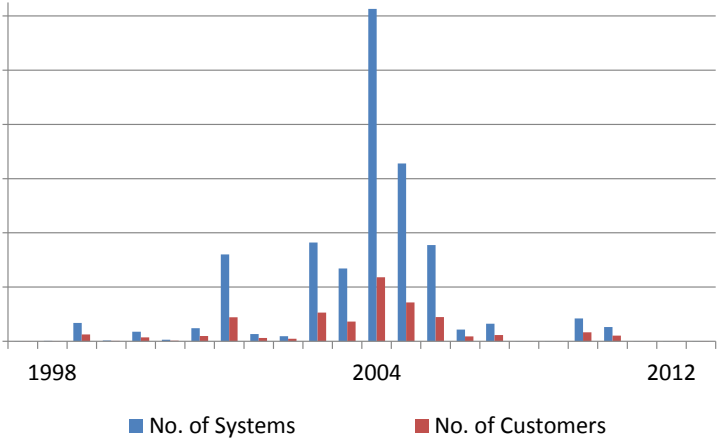
Agenda

The Scalability Challenge

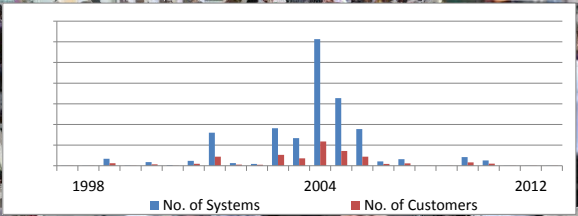


The Software Maintenance Challenge (Modularity)

The Software Maintenance Challenge (Modularity)



The Software Maintenance Challenge (Modularity)



Example

Product	Release	EOL	ext. EOL
Windows XP	2001	2009	2014
Windows 8	2012	2018	2023
Red Hat Ent. Linux	2012	2020	2023

Agenda

1 Introduction & Motivation

2 Secure Software Development at SAP

3 Challenges in Industrial Software Development

4 Discussion About Future Research Directions

Security is not a Binary Property

> Systems are either secure or insecure.

- Security is only one property out of many:
  - Usability
  - New features
  - Time-to-market
- We will never achieve 100% security
- **Question:** Where should I spent my (limited) budget?
  Or: What is the risk of not fixing an issue and how to balance it with other requirements?
- **Claim:** We need more research in
  - risk-based security
  - security economics (cost of fixing vs. costs of not fixing, etc.)

Soundness is not Binary Either

> Security testing methods should be sound.

- Observations:
  - Proving soundness seems to be a prerequisite for getting an academic paper accepted.
  - (Nearly) no “real-world” tool is sound (the underlying method/theory might be sound)
  - Even worse: your sound tool will not report anything, on our frameworks
- What I need (from vendors/researchers) to provide the best “blend” to my developers:
  - A Clear specification what it “in-scope”
  - A Clear specification what it “out-scope”
  - Test cases that validate the expected behavior (e.g., similar to qualification kits for DO178C)
- **Claim:** We need more research in
  - “well-defined” unsound security testing methods
  - clear specifications of unsoundness
  - test sets for comparing security testing tools
  - extension/adaption points for security testing tools
- If you want to read more: `http://www.soundiness.org`

Automation is Too important to Lie

> My tool is fully automated

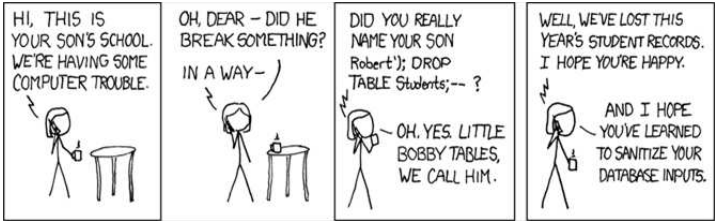
- No, it is (usually) not. And, btw, calling it interactive does not help either
- Again, clearly specify
  - what is automated
  - what needs to be configured “one-time”
  - what needs to be done manually/interactively “on each use”
- **Claim:** We need more research in
  - “automating” the knowledge of security experts
  - automation of “learning new frameworks and policies”
  - closing the gap between security (non-functional) and functional testing
  - need to be integrated into development and built environments
    - instant feedback (could be imprecise)
    - on each check-in
    - nightly/weekly (high quality, should generate compliance reports)

Software is Not Developed on The “Greenfield”

> Security testing is done by the developer of a software component

- Observations:
  - Software evolves over time (both, on-premise and Cloud): small changes are the norm
  - Software is build using
    - Free and Open Source Software
    - third party libraries (closed source)
    - assets of acquired companies As vendor, you are responsible for all code you ship to customers
- **Claim:** We need more research in
  - composable security testing techniques, e.g.,
    - impact/change analysis for selecting (security) test cases
    - automated inference of security specifications of software components
  - in pushing security testing across the whole software supply chain
    - techniques that generate “security certificates”
    - formats and guidelines for exchanging “security test tool configurations”

Thank you!



http://xkcd.com/327/

Bibliography I

Ruediger Bachmann and Achim D. Brucker.
Developing secure software: A holistic approach to security testing.
Datenschutz und Datensicherheit (DuD), 38(4):257–261, April 2014.

Achim D. Brucker and Uwe Sodan.
Deploying static application security testing on a large scale.
In Stefan Katzenbeisser, Volkmar Lotz, and Edgar Weippl, editors, *GI Sicherheit 2014*, volume 228 of *Lecture Notes in Informatics (LNI)*, pages 91–101. GI, March 2014.

Part II

Appendix

A Bluffers Guide to SQL Injection (1/2)

- **Assume an SQL Statement for**

```
selecting all users with “userName” from table “user”
```

A Bluffers Guide to SQL Injection (1/2)

- **Assume an SQL Statement for**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’" + userName + "’;"
```

A Bluffers Guide to SQL Injection (1/2)

- **Assume an SQL Statement for**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’" + userName + "’;"
```

- **What happens if we choose the following *userName*:**

```
userName = "’ or ’1’=’1"
```

A Bluffers Guide to SQL Injection (1/2)

- **Assume an SQL Statement for**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’" + userName + "’;"
```

- **What happens if we choose the following *userName*:**

```
userName = "’ or ’1’=’1"
```

- **Resulting in the following statement:**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’’ or ’1’=’1’;"
```

A Bluffers Guide to SQL Injection (1/2)

- **Assume an SQL Statement for**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’" + userName + "’;"
```

- **What happens if we choose the following *userName*:**

```
userName = "’ or ’1’=’1"
```

- **Resulting in the following statement:**

```
stmt = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’’ or ’1’=’1’;"
```

- **Which is equivalent to**

```
stmt = "SELECT * FROM ‘users‘;"
```

selecting the information of **all users** stored in the table ‘users’!

A Bluffers Guide to SQL Injection (2/2)

```
void selectUser(HttpServletRequest req, HttpServletResponse resp)
     throws IOException {
  String userName = req.getParameter("fName"); // source

  String stmt     = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’"
                    + userName +"’;"
  SQL.exec(stmt); //sink
}
```

- Many vulnerabilities have similar causes:
  - cross-site-scripting (XSS), code-injection, buffer-overflows, . . .
- Root cause of a wide range of vulnerabilities
  - “bad” programming
  - mis-configuration
- *Warning:*
  - for preventing SQL injections, consider the use of prepared statements
  - do whitelisting (specify what is allowed) and *do not* blacklisting

A Bluffers Guide to SQL Injection (2/2)

```
void selectUser(HttpServletRequest req, HttpServletResponse resp)
     throws IOException {
  String userName = req.getParameter("fName"); // source

  String stmt     = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’"
                  + userName +"’;"
  SQL.exec(stmt); //sink
}
```

- Many vulnerabilities have similar causes:
  - cross-site-scripting (XSS), code-injection, buffer-overflows, . . .
- Root cause of a wide range of vulnerabilities
  - “bad” programming
  - mis-configuration
- *Warning:*
  - for preventing SQL injections, consider the use of prepared statements
  - do whitelisting (specify what is allowed) and *do not* blacklisting

A Bluffers Guide to SQL Injection (2/2)

```
void selectUser(HttpServletRequest req, HttpServletResponse resp)
     throws IOException {
  String userName = req.getParameter("fName"); // source
         userName = Security.whitelistOnlyLetter(userName); // sanitation
  String stmt     = "SELECT * FROM ‘users‘ WHERE ‘name‘ = ’"
                  + userName +"’;"
  SQL.exec(stmt); //sink
}
```

- Many vulnerabilities have similar causes:
  - cross-site-scripting (XSS), code-injection, buffer-overflows, . . .
- Root cause of a wide range of vulnerabilities
  - “bad” programming
  - mis-configuration
- *Warning:*
  - for preventing SQL injections, consider the use of prepared statements
  - do whitelisting (specify what is allowed) and *do not* blacklisting

© 2014 SAP SE. All rights reserved