Static Analysis
The Workhorse of a End-to-End Security Testing Strategy

Achim D. Brucker
a.brucker@sheffield.ac.uk http://www.brucker.uk/

Department of Computer Science, The University of Sheffield, Sheffield, UK

Winter School SECENTIS 2016
Security and Trust of Next Generation Enterprise Information Systems
February 8–12, 2016, Trento, Italy
Static Analysis: The Workhorse of a End-to-End Security Testing Strategy

Abstract

Security testing is an important part of any security development lifecycle (SDL) and, thus, should be a part of any software (development) lifecycle. Still, security testing is often understood as an activity done by security testers in the time between “end of development” and “offering the product to customers.” Learning from traditional testing that the fixing of bugs is the more costly the later it is done in development, security testing should be integrated, as early as possible, into the daily development activities. The fact that static analysis can be deployed as soon as the first line of code is written, makes static analysis the right workhorse to start security testing activities.

In this lecture, I will present a risk-based security testing strategy that is used at a large European software vendor. While this security testing strategy combines static and dynamic security testing techniques, I will focus on static analysis. This lecture provides a introduction to the foundations of static analysis as well as insights into the challenges and solutions of rolling out static analysis to more than 20000 developers, distributed across the whole world.
Our Plan

Today:

Tomorrow:

Background and how it works ideally

(Ugly) real world problems and challenges

(or why static analysis is “undecideable” in practice)
Our Plan

- Today:
  - Background and how it works ideally
Our Plan

- **Today:**
  - Background and how it works ideally

- **Tomorrow:**
  - (Ugly) real world problems and challenges
  - (or why static analysis is “undecideable” in practice)
Part I

Background, Motivation, and
An Introduction to Pragmatic Static Analysis
Outline

1. Background
2. Motivation
3. An Introduction to Pragmatic Static Analysis (Code Scanning)
4. Conclusion
Outline

1 Background

2 Motivation

3 An Introduction to Pragmatic Static Analysis (Code Scanning)

4 Conclusion
Personal Background

From Academia to Industry and Back Again . . .

■ **Until 11/2007:**
  ■ PhD student and PostDoc stay at ETH Zurich, Switzerland

■ **Until 11/2015:**
  ■ Member of the central security team, SAP SE (Germany)
    ■ *(Global) Security Testing Strategist*
    ■ Security Research Expert/Architect
  ■ Work areas:
    ■ Defining the risk-based Security Testing Strategy of SAP
    ■ Introducing SAST and DAST tools at SAP
    ■ Identify white spots and evaluate and improve tools/methods
    ■ Secure Software Development Lifecycle integration
    ■ Applied security research
    ■ . . .

■ **Since 12/2015:**
  ■ Senior Lecturer (Security, Testing & Verification, Formal Methods),
    The University of Sheffield, UK

http://www.brucker.ch/
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
- over 68,000 employees worldwide
  - over 25,000 software developers
- Headquarters: Walldorf, Germany (close to Heidelberg)
Why is Software Security Important to Enterprises

Recent Data Breaches

http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

A.D. Brucker The University of Sheffield
## Costs of Data Breaches

- 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)
Has Sony been Hacked this Week?
http://hassonybeenhackedthisweek.com/

Time-line of the Sony Hack(s) (excerpt):

2011-04-20  Sony PSN goes down
2011-05-21  Sony BMG Greece: data 8300 users (SQL Injection)
2011-05-23  Sony Japanese database leaked (SQL Injection)
2011-05-24  Sony Canada: roughly 2,000 leaked (SQL Injection)
2011-06-05  Sony Pictures Russia (SQL Injection)
2011-06-06  Sony Portugal: SQL injection, iFrame injection and XSS
2011-06-20  20th breach within 2 months
            177k email addresses were grabbed via a SQL injection

Consequences:
- account data of close to 100 million individuals exposed
- over 12 million credit and debit cards compromised
- more than 55 class-action lawsuits
- costs of $170 million only in 2011
A Bluffers Guide to SQL Injection (1/2)

Assume an SQL Statement for **selecting all users with “userName” from table “user”**: 

```sql
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 Blufflers Guide to SQL Injection (1/2)

- Assume an SQL Statement for selecting all users with \textit{"userName"} from table \textit{"user"}:

\begin{verbatim}
stmt = "SELECT * FROM 'users' WHERE 'name' = " + userName + "'" + userName + "'";
\end{verbatim}

- What happens if we choose the following \textit{userName}:

\begin{verbatim}
userName = "' or '1'='1"
\end{verbatim}
A Bluffers Guide to SQL Injection (1/2)

- Assume an SQL Statement for selecting all users with "\texttt{userName}" from table "\texttt{user}":

  
  \begin{verbatim}
  stmt = "SELECT * FROM 'users' WHERE 'name' = '" + userName + "';"
  \end{verbatim}

- What happens if we choose the following \texttt{userName}:

  \begin{verbatim}
  userName = "' or '1'=1"
  \end{verbatim}

- Resulting in the following statement:

  \begin{verbatim}
  stmt = "SELECT * FROM 'users' WHERE 'name' = ' or '1'=1';"
  \end{verbatim}
Assume an SQL Statement for selecting all users with "user\Name" from table "user":

```
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'!
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
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
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

```java
void selectUser(HttpServletRequest req, HttpServletResponse resp) 
    throws IOException {
    String userName = req.getParameter("fName"); // source
    userName = Security.whitelistOnlyCharacters(userName); // sanitation
    String stmt = "SELECT * FROM 'users' WHERE 'name' = ";
    stmt += userName + ";";
    SQL.exec(stmt); // sink
}
```
Vulnerability Distribution Since 1999

- Code Execution
- DoS
- Overflow
- Memory Corruption
- SQL Injection
- XSS
- Directory Traversal
- Bypass something
- Gain Privileges
- CSRF
Security-critical (And Safety-critical) Systems Are Small, Right?
Security-critical (And Safety-critical) Systems Are Small, Right?

Pacemaker:
- ca. 100,000 LoC
- supports wireless configuration (up to 50m distance)
Motivation

Security-critical (And Safety-critical) Systems Are Small, Right?

Pacemaker:
- ca. 100,000 LoC
- supports wireless configuration (up to 50m distance)

Typical car:
- ca. 1,000,000 LoC, distributed across ca. 60 ECUs
- ca. 100,000,000 LoC including satnav and entertainment
Security-critical (And Safety-critical) Systems Are Small, Right?

**Pacemaker:**
- ca. 100 000 LoC
- supports wireless configuration (up to 50m distance)

**Typical car:**
- ca. 1 000 000 LoC, distributed across ca. 60 ECUs
- ca. 100 000 000 LoC including satnav and entertainment

**Aircraft:**
- ca. 8 000 000 LoC (on-board), distributed across ca. 200 ECUs
- ca. 16 000 000 LoC (off-airframe)
Security-critical (And Safety-critical) Systems Are Small, Right?

**Pacemaker:**
- ca. 100,000 LoC
- supports wireless configuration (up to 50m distance)

**Typical car:**
- ca. 1,000,000 LoC, distributed across ca. 60 ECUs
- ca. 100,000,000 LoC including satnav and entertainment

**Aircraft:**
- ca. 8,000,000 LoC (on-board), distributed across ca. 200 ECUs
- ca. 16,000,000 LoC (off-airframe)

**Enterprise System (SAP):**
- ca. 500,000,000 LoC (without user interfaces)
- ca. 200,000 screens (user interface definitions)
Evolution of Source Code

- Increase in:
  - code size
  - code complexity
  - number of products
  - product versions
  - used technologies (prog. languages, frameworks)
Languages Used at SAP

- ABAP
- Java
- C
- JavaScript
- Others
Outline

1 Background
2 Motivation
3 An Introduction to Pragmatic Static Analysis (Code Scanning)
4 Conclusion
A Few Questions

1. You are responsible for quality assurance for a large scale IT system (> 10 000 000 LoC)
   - What do you have your team do?
   - Follow coding standards?
   - Test-driven Development?
   - Use Formal Methods?
A Few Questions

1. You are responsible for quality assurance for a large scale IT system (> 10 000 000 LoC)
   - What do you have your team do?
   - Follow coding standards?
   - Test-driven Development?
   - Use Formal Methods?

2. Your system is safety or security critical
   - What changes from #1?
   - Does the distinction between safety versus security matter?
A Few Questions

1. You are responsible for quality assurance for a large scale IT system (> 10,000,000 LoC)
   - What do you have your team do?
   - Follow coding standards?
   - Test-driven Development?
   - Use Formal Methods?

2. Your system is safety or security critical
   - What changes from #1?
   - Does the distinction between safety versus security matter?

3. You are a researcher building code analysis tools.
   - How do you migrate them to large-scale applications?
   - What are the challenges in practise?
   - Would you invest in a high quality (sound???) analysis?
   - Would you invest a good integration into the development environment?
Pragmatic Static Analysis
The Coverity Experience

- Coverity: a tool for finding generic errors in C/C++ code
- Company goal: make money (and build a user community around the tool)
- Guiding principle: if it helps developers to avoid bugs, it’s good
- **Focus on finding bugs/errors**, not proving their absence
- Embrace unsoundness (Focus on low hanging fruit)!

> Circa 2000, unsoundness was controversial in the research community, though it has since become almost a de facto tool bias for commercial products and many research projects.

A few billion lines of code later, CACM, 2010.

- Usability and simplicity are critical!
What We Want to Find
Programming Patterns That May Cause Security Vulnerabilities

Mainly two patterns

Local issues (no data-flow dependency), e.g.,

- Insecure functions

```javascript
var x = Math.random();
```

- Secrets stored in the source code

```javascript
var password = 'secret';
```

Data-flow related issues, e.g.,

- Cross-site Scripting (XSS)

```javascript
var docref = document.location.href;
var input = docref.substring(docref.indexOf("default=")+8);
var fake = function (x) {return x;}
var cleanse = function (x) {
    return 'hello_world';}
document.write(fake(input));
document.write(cleanse(uinput));
```

- Secrets stored in the source code

```javascript
var foo = 'secret';
var x = decrypt(foo,data);
```
What We Want to Find
Programming Patterns That May Cause Security Vulnerabilities

Mainly two patterns

Local issues (no data-flow dependency), e.g.,
- Insecure functions

```javascript
var x = Math.random();
```
- Secrets stored in the source code

```javascript
var password = 'secret';
```

Data-flow related issues, e.g.,
- Cross-site Scripting (XSS)

```javascript
var docref = document.location.href;
var input = docref.substring(docref.indexOf("default=")+8);
var fake = function (x) {return x;}
var cleanse = function (x) {
    return 'hello_world';
}
document.write(fake(input));
document.write(cleanse(uinput));
```

We trust our developers, i.e., we are focusing on finding “obvious” bugs. We do not need to do a sound verification.

- Secrets stored in the source code

```javascript
var foo = 'secret';
var x = decrypt(foo, data);
```
# What We Want to Find

Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors (Tsipenyuk, Chess, and McGraw)

<table>
<thead>
<tr>
<th></th>
<th>Section Title</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input Validation and Representation</td>
<td>Buffer overflows, command injection, . . .</td>
</tr>
<tr>
<td>2</td>
<td>API Abuse</td>
<td>Dangerous functions, unchecked return values, . . .</td>
</tr>
<tr>
<td>3</td>
<td>Security Features</td>
<td>Insecure randomness, password management, . . .</td>
</tr>
<tr>
<td>4</td>
<td>Time and State</td>
<td>Deadlocks, race conditions, . . .</td>
</tr>
<tr>
<td>5</td>
<td>Errors</td>
<td>Catching null pointer ex., empty catch blocks, . . .</td>
</tr>
<tr>
<td>6</td>
<td>Code Quality</td>
<td>Double free, memory leak, . . .</td>
</tr>
<tr>
<td>7</td>
<td>Encapsulation</td>
<td>Comparing classes by name, leftover debug code, . . .</td>
</tr>
<tr>
<td></td>
<td>Environment: J2EE misconfigurations . .</td>
<td></td>
</tr>
</tbody>
</table>

A.D. Brucker  The University of Sheffield
## What We Can Expect to Find

<table>
<thead>
<tr>
<th>visible in the code</th>
<th>visible only in the design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>generic defects</strong></td>
<td>Most likely to be found through architectural analysis.</td>
</tr>
<tr>
<td>Static analysis sweet spot. Built-in rules make it easy</td>
<td>Example: the program executes code downloaded as an email</td>
</tr>
<tr>
<td>for tools to find these without programmer guidance.</td>
<td>attachment.</td>
</tr>
<tr>
<td><em>Example:</em> buffer overflows</td>
<td></td>
</tr>
<tr>
<td><strong>context specific defects</strong></td>
<td>Requires both understanding of general security principles</td>
</tr>
<tr>
<td>Possible to find with static analysis, but customisation</td>
<td>along with domain-specific expertise.</td>
</tr>
<tr>
<td>may be required.</td>
<td>Example: cryptographic keys kept in use for an unsafe duration.</td>
</tr>
<tr>
<td><em>Example:</em> mishandling of credit card information.</td>
<td></td>
</tr>
</tbody>
</table>
Pragmatic static analysis is based on

- successful developments from research community:
  - Type checking
  - Property checking (model-checking, SMT solving, etc.)
  - Abstract interpretation
  - ...

- techniques from the software engineering community
  - Style Checking
  - Program comprehension
  - Security reviews
  - ...

Let's look at examples...
Pragmatic static analysis is based on

- successful developments from research community:
  - Type checking
  - Property checking (model-checking, SMT solving, etc.)
  - Abstract interpretation
  - ...

- techniques from the software engineering community
  - Style Checking
  - Program comprehension
  - Security reviews
  - ...

Let’s look at examples ...
Type Checking

The Java compiler will flag the following as an error. Is it?

```
short s = 0;
int i = s;
short r = i;
```
Type Checking

- The Java compiler will flag the following as an error. Is it?

```java
short s = 0;
int i = s;
short r = i;
```

- How about this:

```java
Object [] objs = new String[1];
objs[0] = new Object();
```

What happens at runtime?
Type Checking

- The Java compiler will flag the following as an error. Is it?

```java
short s = 0;
int i = s;
short r = i;
```

- How about this:

```java
Object[] objs = new String[1];
objs[0] = new Object();
```

What happens at runtime?

- Type checkers are useful
  - But may suffer from false positives/negatives
  - Identifying which computations are harmful is undecidable
Style Checkers

- Enforce more picker and more superficial rules than type checkers
- Some compiler can check these, e.g.,

```c
gcc -Wall enum.c
enum.c:5: warning: enumeration value 'green'
    not handled in switch
enum.c:5: warning: enumeration value 'blue'
    not handled in switch
```

- Or consider `x == 0` vs. `0 == x`
- Style checkers are often extensible, e.g.,
  - PMD (https://pmd.github.io/) for Java
  - JSHint (http://jshint.com/) for JavaScript
- Simple, but very successful in practice
Program Understanding

- Tools can help with
  - Understanding large code bases
  - Reverse engineering abstractions
  - Finding declarations and uses
  - Analysing dependencies
  - ...

- Useful for manual code/architectural reviews
Bug (Pattern) Finders

- Work with a fault model of typical mistakes
Bug (Pattern) Finders

- Work with a fault model of typical mistakes

```java
Person person = aMap.get("bob");
if (person != null) {
    person.updateAccessTime();
}
String name = person.getName();
```

Null-pointer de-reference

String b = "bob";
b.replace('b', 'p');
if(b.equals("pop"))

Ignored return values

Findbugs (http://findbugs.sourceforge.net/) is a good example for Java
Bug (Pattern) Finders

- Work with a fault model of typical mistakes

Person person = aMap.get("bob");
if (person != null) {
    person.updateAccessTime();
}
String name = person.getName();

Null-pointer de-reference
Bug (Pattern) Finders

- Work with a fault model of typical mistakes

```java
Person person = aMap.get("bob");
if (person != null) {
person.updateAccessTime();
}
String name = person.getName();
```

Null-pointer de-reference

```java
String b = "bob";
b.replace('b', 'p');
if(b.equals("pop"))
```
Bug (Pattern) Finders

- Work with a fault model of typical mistakes

```java
Person person = aMap.get("bob");
if (person != null) {
    person.updateDynamicAccessTime();
}
String name = person.getName();
```

- Null-pointer de-reference

```java
String b = "bob";
b.replace('b', 'p');
if (b.equals("pop"))
```

- Ignored return values
Bug (Pattern) Finders

- Work with a fault model of typical mistakes

    ```java
    Person person = aMap.get("bob");
    if (person != null) {
        person.updateAccessTime();
    }
    String name = person.getName();
    ```

  Null-pointer de-reference

    ```java
    String b = "bob";
    b.replace('b', 'p');
    if(b.equals("pop"))
    ```

  Ignored return values

- Findbugs (http://findbugs.sourceforge.net/) is a good example for Java
Sound Methods

- Software model checking
- All the nice methods Anders Møller introduced
Checkmarx: Presentation of Scan Results
Checkmarx: Per Project Reporting
Checkmarx: Dashboard
HP WebInspect

Tool Demo!
Outline

1 Background
2 Motivation
3 An Introduction to Pragmatic Static Analysis (Code Scanning)
4 Conclusion
There are a wide range of tools available that help developers to implement systems securely, safely, and reliably!

Next: How to apply them in a large organisation ...
Part II

Applying Static (And Dynamic) Analysis At SAP
Outline

5  Introducing Static Analysis
6  Application Security at SAP
7  Lesson’s Learned
8  Industry Trends
9  Conclusion
Introducing Static Analysis

Application Security at SAP

Lesson’s Learned

Industry Trends

Conclusion
You are responsible for quality assurance for a large scale IT system (> 10 000 000 LoC)

- What do you have your team do?

### Find Vulnerabilities Using the Running Application
- Manual Application Penetration Testing
- Automated Application Vulnerability Scanning

### Find Vulnerabilities Using the Source Code
- Automated Static Code Analysis
Finding Security Vulnerabilities

You are responsible for quality assurance for a large scale IT system (> 10,000,000 LoC)

- What do you have your team do?

Find Vulnerabilities Using the Running Application
- Manual Application Penetration Testing
- Automated Application Vulnerability Scanning

Find Vulnerabilities Using the Source Code
- Automated Static Code Analysis
**Finding Security Vulnerabilities**

You are responsible for quality assurance for a large scale IT system (> 10 000 000 LoC)

- What do you have your team do?

- Sounds easy, right?

**Find Vulnerabilities Using the Running Application**

- Manual Application Penetration Testing
- Automated Application Vulnerability Scanning

**Find Vulnerabilities Using the Source Code**

- Automated Static Code Analysis
In 2010: Static Analysis Becomes Mandatory

SAST tools used at SAP:

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP</td>
<td>CodeProfiler</td>
<td>Virtual Forge</td>
</tr>
<tr>
<td>Others</td>
<td>Fortify</td>
<td>HP</td>
</tr>
</tbody>
</table>

- Since 2010, mandatory for all SAP products
- Multiple billions lines analysed
- Constant improvement of tool configuration
- Further details:
Introducing Static Analysis

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):
Introducing Static Analysis

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:

```c
brucker@fujikawa:/usr/src/modules/tp-smapi$ cat thinkpad_ec.c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/dmi.h>

static int thinkpad_ec_request_row(const struct thinkpad_ec_row *args) {
    u8 str3;
    int i;

    /* EC protocol requires write to TWRO (function code): */
    if (!!(args->mask & 0x0001)) {
        i8b ue; /* CEC write to TWRO */
        /* ems("TWRO: \%d\n", i8b); */
    }
}
```
Introducing Static Analysis

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**:
Outline

5 Introducing Static Analysis

6 Application Security at SAP
   - How Application Security is Organized at SAP
   - (Risk-based) Security Testing at SAP
   - Measuring Success and Identifying White Spots

7 Lesson’s Learned

8 Industry Trends
   - Agile Development (Towards SecDevOps)
   - From Dynamic to Static and Back Again

9 Conclusion
Moving to a De-Centralized Application Security Approach

How SAP’s Application Development Approach Developed Over Time

- Governance & approvals
- De-centralized approach

One Two SAST tools fit all
- VF CodeProfiler
- Fortify

Blending of Security Testing Tools
- SAST:
  - SAP Netweaver CVA Add-on, Fortify, Synopsis Coverity, Checkmarx, Breakman
- DAST:
  - HP WebInspect, Quotium Seeker
- Others:
  - Burp Suite, OWASP ZAP, Codinomicon Fuzzer, BDD
SAP Uses a De-centralised Secure Development Approach

- **Central security expert team** (S^2DL owner)
  - Organizes security trainings
  - Defines product standard “Security”
  - Defines risk and threat assessment methods
  - Defines security testing strategy
  - Selects and provides security testing tools
  - Validates products
  - Defines and executes response process

- **Local security experts**
  - Embedded into development teams
  - Organize local security activities
  - Support developers and architects
  - Support product owners (responsibles)

- **Development teams**
  - Select technologies
  - Select development model
  - Design and execute security testing plan
  - ...
Focus of the Central Security Team: Security Testing for Developers

Security testing tools for developers, need to

- Be applicable from the start of development
- Automate the security knowledge
- Be deeply integrated into the dev. env., e.g.,
  - IDE (instant feedback)
  - Continuous integration
- Provide easy to understand fix recommendations
- Declare their “sweet spots”
Outline

5 Introducing Static Analysis

6 Application Security at SAP
   ■ How Application Security is Organized at SAP
   ■ (Risk-based) Security Testing at SAP
   ■ Measuring Success and Identifying White Spots

7 Lesson’s Learned

8 Industry Trends
   ■ Agile Development (Towards SecDevOps)
   ■ From Dynamic to Static and Back Again

9 Conclusion
Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software

- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software
- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software

- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
Combining Multiple Security Testing Methods and Tools

- Risks of only using only SAST
  - Wasting effort that could be used more wisely elsewhere
  - Shipping insecure software

- Examples of SAST limitations
  - Not all programming languages supported
  - Covers not all layers of the software stack
SAP’ Secure Software Development Lifecycle (S²DL)

Start of development

Preparation
- Training
  - Security awareness
  - Secure programming
  - Threat modelling
  - Security static analysis
  - Data protection and privacy
  - Security expert curriculum

- Risk Identification
  - SECURIM (Security Risk Identification and Management)
  - Data Privacy Impact Assessment
  - Threat Modeling

Development
- Plan Security Measures
  - Plan product standard compliance
  - Plan security features
  - Plan security tests
  - Plan security response

- Secure development
  - Secure programming
  - Static code scan
  - Code review

Transition
- Security testing
  - Dynamic testing
  - Manual testing
  - External security assessment

Release decision
- Security Validation
  - Independent security assessment

Utilization
- Security Response
  - Execute the security response plan

A.D. Brucker  The University of Sheffield
Security Validation

- Acts as first customer
- Is not a replacement for security testing during development
- Security Validation
  - Check for "flaws" in the implementation of the $S^2DL$
  - Ideally, security validation finds:
  - No issues that can be fixed/detected earlier
  - Only issues that cannot be detected earlier
   (e.g., insecure default configurations, missing security documentation)
Security Validation

- Acts as first customer
- Is not a replacement for security testing during development
- Security Validation
  - Check for “flaws” in the implementation of the S$^2$DL
  - Ideally, security validation finds:
  - No issues that can be fixed/detected earlier
  - Only issues that cannot be detect earlier
    (e.g., insecure default configurations, missing security documentation)

Penetration tests in productive environments are different:
- They test the actual configuration
- They test the productive environment (e.g., cloud/hosting)
SAST and DAST as Part of the S$^2$DL


![Diagram]

**Security Measure Plan**

- **Security Testing Plan**
  - Based on Security Risk Identification and Mitigation Report (Threat Modelling, SECURIM)
  - Describes planned security testing activities
  - Completeness and plausibility check by validation or security enablement team

**Security Measure Report**

- **Security Testing Report**
  - Result of executed security testing activities (e.g., code scan report)
  - Describes deviations from plan
  - Input for validation and operation (cloud)
A Risk-based Test Plan

RISK ASSESSMENT
(e.g., SECURIM, Threat Modelling, OWASP ASVS)

- Combines multiple security testing methods, e.g., code scans, dynamic analysis, manual penetration testing or fuzzing
- Selects the most efficient test tools and test cases based on the risks and the technologies used in the project
- Re-adjusts priorities of test cases based on identified risks for the project
- Monitors false negative findings in the results of risk assessment

Priority of SAP Security Requirements

Implementation details, e.g., programming languages, frameworks

Select from a list of predefined application types

Security Test Plan

A.D. Brucker  The University of Sheffield
Outline

5 Introducing Static Analysis

6 Application Security at SAP
   - How Application Security is Organized at SAP
   - (Risk-based) Security Testing at SAP
   - Measuring Success and Identifying White Spots

7 Lesson’s Learned

8 Industry Trends
   - Agile Development (Towards SecDevOps)
   - From Dynamic to Static and Back Again

9 Conclusion
A Lethal Question

Assume you implemented all this, which

- costs a zillion of dollars license fees each year and
- results in a significant portion of your developers working on improving security instead of new features/products.
A Lethal Question

Assume you implemented all this, which
- costs a zillion of dollars license fees each year and
- results in a significant portion of your developers working on improving security instead of new features/products.

Now your boss enters your office and asks only one question:
- Can you justify these costs/efforts?

Not answering is not an option:
- you might be fired
- the security program will be killed
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools

- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

Success criteria:
- Percentage of vulnerabilities not covered by our security testing tools increases
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools

- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

Success criteria:
Percentage of vulnerabilities not covered by our security testing tools increases
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools

- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

**Success criteria:**
Percentage of vulnerabilities not covered by our security testing tools increases
How to Measure Success (and Identify White Spots)

- Analyze the vulnerabilities reported by
  - Security Validation
  - External security researchers

- Vulnerability not detected by currently used methods
  - Improve tool configuration
  - Introduce new tools

- Vulnerability detected by our security testing tools
  - Vulnerability in older software release
  - Analyze reason for missing vulnerability

**Success criteria:**
Percentage of vulnerabilities not covered by our security testing tools increases
Introducing Static Analysis

Application Security at SAP
- How Application Security is Organized at SAP
- (Risk-based) Security Testing at SAP
- Measuring Success and Identifying White Spots

Lesson's Learned

Industry Trends
- Agile Development (Towards SecDevOps)
- From Dynamic to Static and Back Again

Conclusion
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers
- Yes, security awareness is important
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers

- Yes, security awareness is important **but**
Key Success Factors

- A holistic security awareness program for
  - Developers
  - Managers

- Yes, security awareness is important but
  Developer awareness is even more important!
Lesson’s Learned

Listen to Your Developers And Make Their Life Easy!

We are often talking about a lack of security awareness and, by that, forget the problem of lacking development awareness.

- Building a secure system more difficult than finding a successful attack.
- Do not expect your developers to become penetration testers (or security experts)!

Often, organisations make it hard for developers to apply their security testing skills!

- Don’t ask developers to do security testing, if their work contract doesn’t allow for it
- Budget application security activities centrally
  (in particular, in a decentralised model)
Select tools that are

- easy to integrate into your development process and tools
  - central scan infrastructure
  - source code upload, CLI, Jenkins, github, ...
- easy to use by developers
  - easy to understand descriptions of findings
  - actionable fix recommendations
- easy to adapt to your security policies and prioritisation
  - report issues that are relevant for you
  - focus developers effort on the issues that are critical for you
- allow for tracking your success
  - tool internal reporting
  - interfaces to your own reporting infrastructure
Outline

5 Introducing Static Analysis

6 Application Security at SAP
  - How Application Security is Organized at SAP
  - (Risk-based) Security Testing at SAP
  - Measuring Success and Identifying White Spots

7 Lesson’s Learned

8 Industry Trends
  - Agile Development (Towards SecDevOps)
  - From Dynamic to Static and Back Again

9 Conclusion
Agile Development

- What is agile for you?
  SCRUM, Continuous Delivery, DevOps, SCRUM, Cloud development, ...
- Cloud/agile development lifecycle
Secure Agile Development

- **PSC Security**
- **Security Measures**
- **Security Testing**
- **Threat Modelling**
- **Risk Identification**
- **Level of Trust**

- **Secure Programming**
  - Static Testing
  - Dynamic Testing
  - Manual Testing
  - Security Validation

- **Risk Mitigation & Testing**

- **Define**
  - Strategic Planning
  - Design Thinking
  - Design-led development

- **Build**
  - Agile Software Engineering
  - Product READINESS

- **Operate**
  - Early Customer Feedback
  - Org Customer End-User READINESS
  - Spot Monitoring Usage analysis

- **Release**
  - Early Access
  - Customer Assisted Testing

- **Security Response**

- **Build Decision**

**Flow Diagram:***

- **Build**
  - Agile Software Engineering
  - Product READINESS

- **Operate**
  - Early Customer Feedback
  - Org Customer End-User READINESS
  - Spot Monitoring Usage analysis

- **Release**
  - Early Access
  - Customer Assisted Testing

**Notes:**

- **Build Decision** connects to Define, Operate, and Release

- **RD = Release Decision**
Secure Agile Development and SecDevOps

Open (Research) Questions

- **Social aspects**
  - Does the SecDevOps model increase security awareness? 
    (Developers and their managers are also responsible for operational risks)
  - Does this impact the willingness to take (security) risks and/or the risk assessment?

- **Process and organisational aspects**
  - What services should be offered centrally?
  - How to ensure a certain level of security across all products?
  - How to ensure a certain level of security across the end-to-end supply chain?

- **Technical and fundamental aspects**
  - How do we need to adapt development support
  - How do we need to adapt threat modelling or risk assessment methods
  - How do we need to adapt security testing techniques

- **The big challenge in practice:**
  Products are often offered in the cloud (SaaS) and on premise
From Dynamic to Static and Back Again

Observations

Let’s have a look on memory corruption analysis

- until 1995: random testing, simple fuzzing
- 1995-2005: the decade of runtime analysis (dynamic testing)
- 2005-2015: the decade of static analysis
- 2015-????: dynamic approaches and combined techniques are getting popular

(dates are rough estimates)

There are (at least) two reasons why people are looking again at dynamic approaches:

- People are not happy with false-positive e rates of static approaches
  (Warning: dynamic approaches are not false-positive free either)
- DevOps pushes dynamic approaches to development, as operations uses pre-dominantly dynamic testing
From Dynamic to Static and Back Again

A few thoughts (not final conclusions yet)

- On the long-run, people will not be happy with (simple) DAST solutions
  - IAST (concolic testing) is a logical next step
  - Improved coverage and increased test case complexity (lowering false-negative rate)
  - Grey-box attack validation (lowering false-positive rate)

- My feeling:
  Runtime protection is hyped, but hard to sell to traditional software companies (e.g., SAP)
  - requires a close collaboration of development and operations (close to DevOps)
  - Why not use runtime-technology (e.g., end-to-end tainting) for security testing during development
    - improves results of manual or automated dynamic tests
    - compute advanced dynamic test cases or rule out false positives

Test systems
- are not as performance critical as production systems
- are less risky to change (runtime environments, instrumentation, etc.)
5 Introducing Static Analysis

6 Application Security at SAP
   - How Application Security is Organized at SAP
   - (Risk-based) Security Testing at SAP
   - Measuring Success and Identifying White Spots

7 Lesson’s Learned

8 Industry Trends
   - Agile Development (Towards SecDevOps)
   - From Dynamic to Static and Back Again

9 Conclusion
Secure software development is a prerequisite for the secure and compliant operation and minimises the risk of operating and maintaining IT systems.

Developers are your most important ally.
- Make life easy for them.

SAST, DAST (or IAST), and runtime technologies are friends: they complement each other.
Part III

Problems in Practise
(And Pragmatic Mitigation Strategies)
Why is Static Analysis Hard (Vendor Perspective)?

Why is Static Analysis Hard (User Perspective)?
Why is Static Analysis Hard (Vendor Perspective)?

Why is Static Analysis Hard (User Perspective)?
Why is Static Analysis Hard (Vendor Perspective)?

Theory

- **Problem:**
  Many properties are undecidable
  (recall the nice explanation during the lecture of Anders Møller)

- **Consequence:**
  - Tools **over-approximate**: might result in false positives
  - Tools **under-approximate**: might result in false negatives
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Getting And Understanding The Source Files

Where is the code?

- **Problem:** Home-grown build environments (build tools, code repositories, etc.)
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Getting And Understanding The Source Files

Where is the code?

- **Problem:** Home-grown build environments (build tools, code repositories, etc.)
- **Solution:**
  - Wrap build and intercept all system calls
    - System still needs to build
    - Mapping issues in translated/pre-processed files back to manually written files
  - Virtual compiler
    - What about pre-processed files (software product lines)
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Getting And Understanding The Source Files

Where is the code?

- **Problem:** Home-grown build environments (build tools, code repositories, etc.)
- **Solution:**
  - Wrap build and intercept all system calls
  - System still needs to build
  - Mapping issues in translated/pre-processed files back to manually written files
  - Virtual compiler
    - What about pre-processed files (software product lines)

You need to parse the input language

- **Problem:** Experience: the C language doesn’t exist
  (neither does JavaScript, Perl, ABAP, etc.)
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Getting And Understanding The Source Files

Where is the code?

- **Problem:** Home-grown build environments (build tools, code repositories, etc.)

- **Solution:**
  - Wrap build and intercept all system calls
    - System still needs to build
    - Mapping issues in translated/pre-processed files back to manually written files
  - Virtual compiler
    - What about pre-processed files (software product lines)

You need to parse the input language

- **Problem:** Experience: the C language doesn’t exist
  (neither does JavaScript, Perl, ABAP, etc.)

- **Solution:** Relaxed parsing, ignoring unknown constructs
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

- **Problem:** Programming languages change over time (Objective C vs. Java)
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

- **Problem:** Programming languages change over time (Objective C vs. Java)
- **Solution:** More developers???
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

- **Problem:** Programming languages change over time (Objective C vs. Java)
- **Solution:** More developers???

Customers expect stable results across upgrades

- **Problem:** Findings should not change across system upgrades (but you need to improve the tool . . . )
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

■ **Problem:** Programming languages change over time (Objective C vs. Java)
■ **Solution:** More developers???

Customers expect stable results across upgrades

■ **Problem:** Findings should not change across system upgrades
  (but you need to improve the tool . . . )
■ **Solution:**
  ■ Separate engines and rules
  ■ Fingerprint (or assign and log unique versions of) rule sets/configuration
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

- **Problem:** Programming languages change over time (Objective C vs. Java)
- **Solution:** More developers???

Customers expect stable results across upgrades

- **Problem:** Findings should not change across system upgrades
  (but you need to improve the tool . . .)
- **Solution:**
  - Separate engines and rules
  - Fingerprint (or assign and log unique versions of) rule sets/configuration

Audit/Review results need to be preserved across code changes

- **Problem:** Audits are expensive, thus re-scans should not require a full audit
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Change Management

Ever changing source languages (and compilers/development environments)

- **Problem:** Programming languages change over time (Objective C vs. Java)
- **Solution:** More developers???

Customers expect stable results across upgrades

- **Problem:** Findings should not change across system upgrades (but you need to improve the tool . . .)
- **Solution:**
  - Separate engines and rules
  - Fingerprint (or assign and log unique versions of) rule sets/configuration

Audit/Review results need to be preserved across code changes

- **Problem:** Audits are expensive, thus re-scans should not require a full audit
- **Solution:**
  - Compute “invariant” hash values of findings
Customers must understand the bugs and care about them

■ **Problem:** How to prioritise findings to point users to the important ones?
Customers must understand the bugs and care about them

- **Problem**: How to prioritise findings to point users to the important ones?
- **Solution**:
  - Quality/precision of checks
  - Length of data-flows
  - Ranking of sources/sinks
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Prioritising Findings And Explaining Them

Customers must understand the bugs and care about them

■ **Problem:** How to prioritise findings to point users to the important ones?

■ **Solution:**
  - Quality/precision of checks
  - Length of data-flows
  - Ranking of sources/sinks

How to explain findings to customers

■ **Problem:** Customers need to understand findings to prioritise them as well as develop fixes
Why is Static Analysis Hard (Vendor Perspective)?

Practice: Prioritising Findings And Explaining Them

Customers must understand the bugs and care about them

- **Problem:** How to prioritise findings to point users to the important ones?
- **Solution:**
  - Quality/precision of checks
  - Length of data-flows
  - Ranking of sources/sinks

How to explain findings to customers

- **Problem:** Customers need to understand findings to prioritise them as well as develop fixes
- **Solution:**
  - (Static) fix recommendations pointing to standard recommendations
  - Computing bet fix locations
Outline

Why is Static Analysis Hard (Vendor Perspective)?

Why is Static Analysis Hard (User Perspective)?
- A Selection of Open Issues
- False Positives and False Negatives
- Evaluating Static Analysis Tools
- Changing/Improving Tool Configurations
- Conclusion and Recommendations
A Selection of Challenges for Users

- Estimating the risk of not fixing security issues is hard
  - How to prioritize security vs. functionality
  - In case of doubt, functionality wins
A Selection of Challenges for Users

- Estimating the risk of not fixing security issues is hard
  - How to prioritize security vs. functionality
  - In case of doubt, functionality wins

- Pushing SAST across the software supply chain
  - Consumed software (OSS, third-party products)
  - SAP Customers, partners, and OEM products
A Selection of Challenges for Users

- Estimating the risk of not fixing security issues is hard
  - How to prioritize security vs. functionality
  - In case of doubt, functionality wins
- Pushing SAST across the software supply chain
  - Consumed software (OSS, third-party products)
  - SAP Customers, partners, and OEM products
- Huge and hybrid multi-language applications
  - Client-server applications
  - Web-frameworks
A Selection of Challenges for Users

- Estimating the risk of not fixing security issues is hard
  - How to prioritize security vs. functionality
  - In case of doubt, functionality wins

- Pushing SAST across the software supply chain
  - Consumed software (OSS, third-party products)
  - SAP Customers, partners, and OEM products

- Huge and hybrid multi-language applications
  - Client-server applications
  - Web-frameworks

- Dynamic programming paradigms and languages
  - JavaScript, Ruby, etc.
A Selection of Challenges for Users

- Estimating the risk of not fixing security issues is hard
  - How to prioritize security vs. functionality
  - In case of doubt, functionality wins

- Pushing SAST across the software supply chain
  - Consumed software (OSS, third-party products)
  - SAP Customers, partners, and OEM products

- Huge and hybrid multi-language applications
  - Client-server applications
  - Web-frameworks

- Dynamic programming paradigms and languages
  - JavaScript, Ruby, etc.

- Lack of standardized regression test suites
  - Different tools
  - Different versions of the same tool
Outline

10 Why is Static Analysis Hard (Vendor Perspective)?

11 Why is Static Analysis Hard (User Perspective)?
   - A Selection of Open Issues
   - False Positives and False Negatives
   - Evaluating Static Analysis Tools
   - Changing/Improving Tool Configurations
   - Conclusion and Recommendations
Pragmatics: False Positives (Unwanted Findings)

An informal definition:

- If a static analysis tool reports a finding, this finding
  - can be exploitable (true positive)
  - cannot be exploitable (false positive)

- If a static analysis tool does not report a finding,
  - the code is secure (true negative)
  - the code contains a vulnerability (false negative)

Let us take the view point of a

- **Developer**: “I want a tool with zero false positives!”
  *False positives* create unnecessary effort

- **Security expert**: “I want a tool with zero false negatives!”
  *False negatives* increase the overall security risk
False Negatives
Reasons and Recommendations (Examples)

- **Fundamental**: under-approximation of the tool (method), e.g.,
  - missing language features (might intercept data flow analysis)
  - missing support for complete syntax (parsing errors)

  Report to tool vendor

- **Configuration**: lacking knowledge of insecure frameworks, e.g.,
  - insecure sinks (output) and sources (input)

  Improve configuration

- **Unknown security threats**: For us, e.g.,
  - XML verb tampering

  Develop new analysis for tool (might require support from tool vendor)
False Positives
Reasons and Recommendations (Examples)

- **Fundamental:** over-approximation of the tool (method), e.g.,
  - pointer analysis
  - call stack
  - control-flow analysis
  Report to tool vendor

- **Configuration:** lacking knowledge of security framework, e.g.,
  - sanitation functions
  - secure APIs
  Improve configuration

- **Mitigated by attack surface:** strictly speaking a true finding, e.g,
  - No external communication due to firewall
  - SQL injections in a database admin tool

Should be fixed.
In practice often mitigated during audit, or local analysis configuration
Prioritisation of Findings
A Pragmatic Solution for Too Many Findings

- What needs to be audited
- What needs to be fixed
  - as security issue (response effort)
  - quality issue
- Different rules for
  - old code
  - new code
Prioritisation of Findings
A Pragmatic Solution for Too Many Findings

- What needs to be audited
- What needs to be fixed
  - as security issue (response effort)
  - quality issue
- Different rules for
  - old code
  - new code
Prioritisation of Findings

A Pragmatic Solution for Too Many Findings

- What needs to be audited
- What needs to be fixed
  - as security issue (response effort)
  - quality issue
- Different rules for
  - old code
  - new code
Prioritisation of Findings
A Pragmatic Solution for Too Many Findings

- What needs to be audited
- What needs to be fixed
  - as security issue (response effort)
  - quality issue
- Different rules for
  - old code
  - new code
Outline

10 Why is Static Analysis Hard (Vendor Perspective)?

11 Why is Static Analysis Hard (User Perspective)?
  - A Selection of Open Issues
  - False Positives and False Negatives
  - Evaluating Static Analysis Tools
  - Changing/Improving Tool Configurations
  - Conclusion and Recommendations
SAP Development Experienced a Change in 2012/2013
In 2012: Rumours Began to Spread

The existing solution does not work for JavaScript!

It looks like the existing solution
- reports less issues per line of code (compared to Java, C/C++, ...)
- has some noise checks

We need to make ourselves aware that
- scanning JavaScript is easy (compared to Java, C/C++)
- fewer reported issues allow for a more diligent audit

Food for thought:
- many issues being reported (without careful review) might result in a false sense of security
- due to low effort, it might still be valuable (good cost-benefit ratio)
In 2012: We Asked Ourselves

- Is SCA is useful for JavaScript?

- Are there better SCA Tools available?

- Can we use the tools more effectively and efficiently?
In 2012: We Asked Ourselves

- Is SCA is useful for JavaScript?
  **Yes:** Serious flaws are found and fixed!

- Are there better SCA Tools available?

- Can we use the tools more effectively and efficiently?
In 2012: We Asked Ourselves

- Is SCA is useful for JavaScript?
  **Yes:** Serious flaws are found and fixed!

- Are there better SCA Tools available?
  Checkmarx, Fortify, IBM AppScan Source Edition

- Can we use the tools more effectively and efficiently?
In 2012: We Asked Ourselves

- Is SCA is useful for JavaScript?
  **Yes:** Serious flaws are found and fixed!

- Are there better SCA Tools available?
  Checkmarx, Fortify, IBM AppScan Source Edition

- Can we use the tools more effectively and efficiently?
  Most likely, **yes**
In 2013: What Tools Were Available

Our market study revealed three classes of tools:

- Scale-able analyzers with a broad security scope
  - Fortify
  - Checkmarx
  - IBM AppScan Source Edition

- Light-weight analyzers
  - JSPPrime (focused on DOM-based XSS)
  - JSLint (very useful, focused on coding styles)
  - HSInt (early stage, extensible JSLint)
  - YASCA (simple grep)
  - ... 

- Research prototypes
  - TAJS (scalability – jQuery?)
  - ...
In 2013: Evaluation

We evaluated in detail:

- Fortify
- Checkmarx
- IBM AppScan Source Edition

For all tools, we used

- most sensitive “default” configuration (no SAP template/filters)
- the same evaluation targets
  - library of JavaScript “challenges” (self-made)
  - three (fourth ongoing) SAP applications of different size (including one with server-side JavaScript using the XS Engine)
  - detailed comparison for
    - XSS-variants
    - All findings of the two topmost priorities (high)
Assume we want to compare tool (configuration) A and B:

1. Analyse same test target with both tools (configurations)
2. For all findings (or well-defined subset, e.g., one vulnerability type):
   1. Ignore all findings reported by both tools (configurations)
      (Regardless if you use A or B, you need to cope with these findings)
   2. Analyse all findings only reported by A
      - True positives of A are false negatives for B
   3. Analyse all findings only reported by B
      - True positives of B are false negatives for A
3. Compare the number of false/true positives for both tools
   (how to weight — depends on your actual efforts . . .)
In 2013: A First Evaluation Result

No tool is perfect (for us) in its default configuration

- For real SAP Applications,
  - there is no clear winner in the category “JavaScript semantics”
  - interesting difference in terms of available checks (categories)

- Important follow-up:
  How can we adapt the tools to our needs?
  - We tried to write custom checks/rules for two test cases
    - an eval-example
    - an SQL-injection example

Only one tool (partially) successful
Generalised Evaluation and Roll-out Approach

1. Identify need
2. Market research
3. For a larger set of candidates:
   Limited evaluation by central security team based on
   - artificial test cases
   - one or two selected SAP applications
4. For a smaller set of candidates:
   Proof-of-Concept (pilot) with SAP development team and vendor
5. In case of success (roll-out decision):
   - Ramp-up projects on a per project/team basis
   - Adaption to SAP technologies
   - Integration into SAP build systems, IDEs, reporting, ...

Note, step 1) and 2) are sometimes replaced by vendor marketing ...
Outline

10 Why is Static Analysis Hard (Vendor Perspective)?

11 Why is Static Analysis Hard (User Perspective)?
- A Selection of Open Issues
- False Positives and False Negatives
- Evaluating Static Analysis Tools
- Changing/Improving Tool Configurations
- Conclusion and Recommendations
Report all occurrences where the first argument to system can be influenced by an attacker (input not sanitised)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DataflowSinkRule formatVersion="3.2" language="cpp">
  <MetaInfo>
    <Group name="package">C Core</Group>
  </MetaInfo>
  <RuleID>AA212456-92CD-48E0-A5D5-E74CC26A276F</RuleID>
  <VulnKingdom>Input Validation and Representation</VulnKingdom>
  <VulnCategory>Command Injection</VulnCategory>
  <DefaultSeverity>4.0</DefaultSeverity>
  <Description ref="desc.dataflow.cpp.command_injection"/>
...
```
Insight: Changing/Improving Tool Configurations

Fortify

- Report all occurrences where the first argument to system can be influenced by an attacker (input not sanitised)

```xml
...<Sink>
  <InArguments>0</InArguments>
  <Conditional>
    <Not>
      <TaintFlagSet taintFlag="VALIDATED_COMMAND_INJECTION"/>
    </Not>
  </Conditional>
</Sink>
<FunctionIdentifier>
  <FunctionName>
    <Value>system</Value>
  </FunctionName>
</FunctionIdentifier>
</DataflowSinkRule>
```
Insight: Changing/Improving Tool Configurations

Checkmarx

- Report all occurrences where the first argument to `system` can be influenced by an attacker (input not sanitised)

```java
CxList inputs = Find_Inputs();
CxList validSanitation = getSanitizers()
CxList SystemCalls = All.FindByShortName("system")
CxList output = All.GetParameters(SystemCalls, 0));

result = outputs.InfluencedByAndNotSanitized(inputs, validSanitation);
```
Outline

Why is Static Analysis Hard (Vendor Perspective)?

Why is Static Analysis Hard (User Perspective)?
- A Selection of Open Issues
- False Positives and False Negatives
- Evaluating Static Analysis Tools
- Changing/Improving Tool Configurations
- Conclusion and Recommendations
Conclusion

- Static analysis is a challenging problem both theoretically and pragmatically

Recommendations for users

- Adapt the tools to your needs!
- Provide clear guidelines which findings are important to your organisation!
- Choose your tools carefully
- Scan daily (or at least weakly)
Part IV

Outlook
Outline

12 A Few Notes on Practical JavaScript Challenges
13 Analysing Hybrid Mobile Apps
var docref = document.location.href
var input = docref.substring(docref.indexOf("default=")+8);
var fake = function (x) {return x;}
var cleanse = function (x) {return 'hello_world';}

var uinput = unknown(input);  // unknown is nowhere defined
document.write(uinput);  // secure!?

var finput = fake(input);
document.write(finput);  // not secure

var cinput = cleanse(input);
document.write(cinput);  // secure

var extfinput = extfake(input);  // defined externally (part of scan)
document.write(extfinput);  // not secure

var extcinput = extcleanse(input); defined externally (part of scan)
document.write(extcinput);  // secure

var nobodyKnows = toCleanOrNotToCleanse(input); multiply defined (underspecified)
document.write(nobodyKnows);  // not secure!?
Functions as First-Class Objects

```javascript
var href = document.location.href;
var unsafeInput = href.substring(href.indexOf("default=")+8); // unsafe input
var safeInput = "1+2"; // safe input

// aliasing eval
var exec = eval;
var doit = exec;

var func_eval1 = function (x) {eval(x);};
var func_eval2 = function (x,y) {eval(y);};

var func_eval_eval = function (x) {func_eval1(x);};
var func_doit = function (x) {doit(x);};
var func_exec = function (x) {exec(y);};
var run = func_eval1;
var inject_code = func_exec;
```
CSRF Prevention

```javascript
var request = {
    headers: {
        "X-Requested-With": "XMLHttpRequest",
        "Content-Type": "application/atom+xml",
        "X-CSRF-Token": "Fetch"
    }
},

if (Appcc.CSRFToken)
    var request = {
        headers: {
            "X-Requested-With": "XMLHttpRequest",
            "Content-Type": "application/atom+xml",
            "X-CSRF-Token": Appcc.CSRFToken
        }
    },
else var request = {
    headers: {
        "X-Requested-With": "XMLHttpRequest",
        "Content-Type": "application/atom+xml",
        "X-CSRF-Token": "etch"
    }
},

var response = this.oServiceManager.read(request, this, this.batch, this.busy);`
Prototype-based Inheritance

```javascript
var vl = new sap.ui.commons.layout.VerticalLayout();
sap.ui.core.Control.extend("foobar.Label", {
    metadata : {
        properties : {
            "text" : "string"
        }
    },
    renderer : function(oRm, oControl) {
        oRm.write("<span>XSSLabel: ");
        oRm.write(oControl.getText());
        oRm.write("</span>");
    }
});
var p = jQuery.sap.getUriParameters().get("xss");
vl.addContent(new foobar.Label({text:p}));
return vl;
```
A Few Notes on Practical JavaScript Challenges

Analysing Hybrid Mobile Apps
- Motivation: Hybrid Mobile Apps and Their Security Challenges
- Static Analysis for Hybrid Apps: Building a unified call graph
- An assessment of hybrid Apps (in Google Play)
- Recommendations & Conclusions
What is a Hybrid App?
Native, HTML5, or hybrid

Native apps
- Java \ Swift \ C#
- Developed for a specific platform
- All features available

Web apps
- HTML5 and JS
- Hosted on server, all platforms
- No access to device features

Platform-specific

Platform-independent
What is a Hybrid App?
Native, HTML5, or hybrid

- Native apps: Java \ Swift \ C#
  - Developed for a specific platform
  - All features available

- Hybrid apps: HTML5, JS, and native
  - Build once, run everywhere
  - Access to device features through plugins

- Web apps: HTML5 and JS
  - Hosted on server, all platforms
  - No access to device features

Platform-specific

Platform-independent
The architecture of Apache Cordova

- HTML Android App
  - UI Layer (HTML, CSS, JS)
  - Application Logic in JS

- Android WebView

- Cordova Plugins
  - Camera
  - In-App Browser
  - Media
  - Geolocation
  - Network
  - File
  - Vibration
  - Custom Plugins

- Android APIs

- Cordova JS API

- Cordova Native API

- Android APIs

- Android Platform
**Example: Get Contacts**

```javascript
function showPhoneNumber(name) {
    var successCallback = function(contact) {
        alert("Phone number: " + contacts.phone);
    }
    exec(successCallback, null, "ContactsPlugin", "find",
        [{"name" : name}]);
}
```

```java
class ContactsPlugin extends CordovaPlugin {
    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        if ("find".equals(action)) {
            String name = args.get(0).name;
            find(name, callbackContext);
        } else if ("create".equals(action)) ... 
    }
    void find(String name, CallbackContext callbackContext) {
        Contact contact = query("SELECT ... where_name=" + name);
        callbackContext.success(contact);
    }
}
```
From Apache Cordova to SAP Kapsel (Fiori/Kapsel Browser)

- Based on Apache Cordova (FOSS Framework)
- Apache Cordova plus plugins for
  - Encrypted Storage
  - Authentication
  - Logging
  - ...
- Enterprise features
  - Single sign-on
  - Application management (SMP)
  - Mobile Device Management (MDM)
- SAP UI5
  (JavaScript framework for UIs)
Why is it hard to ensure the security of hybrid apps

Web technologies (i.e., JavaScript)
- lack of typing, higher order functions, asynchronous programming models
- highly dynamic (e.g., eval(...), dynamic loading)
- ...

Large Libraries and Modules
- large (≈ 100kLOC) third party (FOSS, proprietary) libraries
- both native (Java) and JavaScript
- complex core framework
- ...

Cross-Language-Analysis
- many data-flows across language boundaries
- datatype conversion
- not only for accessing sensors (e.g., session plugin requires > 10 language switches)
- ...

A.D. Brucker  The University of Sheffield
Exploiting the JavaScript to Java Bridge (CVE-2013-4710)

- We can expose Java methods in JavaScript

```
foo.addJavascriptInterface(new FileUtils(), "FUtil");
```

- And use them in JavaScript easily

```
<script type="text/javascript">
// <![CDATA[
filename = '/data/data/com.livingsocial.www/' + id + '_cache.txt';
FUtil.write(filename, data, false);
// ]]></script>
```

- Which might expose much more than expected

```
function execute(cmd){
    return
    window._cordovaNative.getClass().forName('java.lang.Runtime').
        getMethod('getRuntime',null).invoke(null,null).exec(cmd);
}
```
Never, really never, use http without SSL

Thanks to Jens Heider from Fraunhofer SIT.
Never, really never, use http without SSL

Thanks to Jens Heider from Fraunhofer SIT.
What did we learn from this

There are many subtle things to consider:

- always use http\textcolor{red}{s} and \textcolor{red}{validate} certificates
- dynamically loaded code from third parties can be dangerous (even if “iframed”)
- in Cordova apps, XSS attackers can be very powerful
- ship only the plugins that you need (unused plugins can still be exploited)
- if you need only limited functionality, secure the plugin in the native/Java code
- Did you know that

\begin{verbatim}
<application android:debuggable="true" />
\end{verbatim}

\textcolor{red}{disables} certificates checks in WebViews!
A Few Notes on Practical JavaScript Challenges

Analysing Hybrid Mobile Apps

- Motivation: Hybrid Mobile Apps and Their Security Challenges
- Static Analysis for Hybrid Apps: Building a unified call graph
- An assessment of hybrid Apps (in Google Play)
- Recommendations & Conclusions
How to help the developer?

We want to find bugs in Cordova apps

- Idea: Static program analysis, build a call graph of the Cordova app
- But how to find cross-language calls?

Four heuristics that model the Cordova framework:

- ConvertModules
- ReplaceCordovaExec
- FilterJavaCallSites
- FilterJSFrameworks

Based on examination of real Cordova apps
Exploit frequent coding patterns to improve precision
**Problem:**

- Not all callback functions are defined within the plugin
- Difficult to track callback functions from app code

**Solution:**

- Substitute dynamic mechanism with unique, global variable
**Problem:**
- Not all callback functions are defined within the plugin
- Difficult to track callback functions from app code

**Solution:**
- Substitute dynamic mechanism with unique, global variable
**ConvertModules: results**

- Most useful for
  - small plugins
  - more precise analysis
- Allowed finding of callback functions in app code
- Less errors due to less ambiguity of dynamic mechanism
function showPhoneNumber(name) {
    var successCallback = function(contact) {
        alert("Phone number:"+contacts.phone);
    }

    exec(successCallback, null, "ContactsPlugin", "find", {}
        [{"name" : name}]);
}

Problem:
- Callback call sites are hard to find
- No context-sensitivity

Solution:
- Stub the exec method
ReplaceCordovaExec

```javascript
function showPhoneNumber(name) {
    var successCallback = function(contact) {
        alert("Phone number: "+contacts.phone);
    }
    function stub1(succ, fail) {
        succ(null);
        fail(null);
    }
    stub1(successCallback, null, "ContactsPlugin", "find",
         [{'name' : name}]);
}
```

**Problem:**
- Callback call sites are hard to find
- No context-sensitivity

**Solution:**
- Stub the exec method
ReplaceCordovaExec: Results

- Necessary to find any Java to JavaScript calls
- Most apps use exec to communicate, only some bypass it
- Inexpensive way to get context-sensitivity where it is needed the most
class ContactsPlugin extends CordovaPlugin {
    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        if ("find".equals(action)) {
            String name = args.get(0).name;
            find(name, callbackContext);
        } else if ("create".equals(action)) {
            }
    }

    void find(String name, CallbackContext callbackContext) {
        Contact contact = query("SELECT name + name);
        callbackContext.success(contact);
    }
}

Problem:
- How to determine the targets of the callbackContext calls?
- Can we use the pattern of the action usage?

Solution:
- Determine which callbackContext calls are reachable
FilterJavaCallSites: details

```
showPhoneNumber
  Invoke exec

... Invoke alert

execute
  Entry
  if action == "find"
  if action == "create"
    Invoke CordovaArgs.get
    ... Invoke find
    Exit

find
  Entry
  Invoke query
  Invoke callbackContext.success
  Exit
```
FilterJavaCallSites: details

showPhoneNumber
  - Invoke exec

execute
  - Entry
    - if action == "find"
    - if action == "create"
      - Invoke CordovaArgs.get
      - ...
FilterJavaCallSites: details

- showPhoneNumber
  - Invoke exec

- execute
  - Entry
  - if action == "find"
    - if action == "create"
      - Invoke CordovaArgs.get
      - ... Invoke find
  - Exit

- find
  - Entry
  - Invoke query
  - Invoke callbackContext.success
  - Exit

- ... Invoke alert
  - ... Invoke callbackContext.success
FilterJavaCallSites: results

- Developers all use action variable similarly
- Therefore: Many incorrect edges avoided
- But: A few calls from Java to JavaScript are missed now
- Some store the callbackContext and call asynchronously
A Few Notes on Practical JavaScript Challenges

Analysing Hybrid Mobile Apps
- Motivation: Hybrid Mobile Apps and Their Security Challenges
- Static Analysis for Hybrid Apps: Building a unified call graph
- An assessment of hybrid Apps (in Google Play)
- Recommendations & Conclusions
What we were interested in

Main goals:
- Understand the use of Cordova
- Learn requirements for Cordova security testing tools

Looking for answers for questions like
- How many apps are using Cordova?
- How is Cordova used by app developers?
- Are cross-language calls common or not?
What we did

Selection of apps
- all apps that ship Cordova from Google’s Top 1000:
  - 100 apps ship Cordova plugins
  - only 50 actually use Cordova (5%)
- three selected apps from SAP (using SAP Kapsel)
- one artificial test app (to test our tool)

Development of a static analysis tool
- analysing Android apps (*.apk files)
- specialised in data-flows from Java to JavaScript and vice versa
- based on WALA
- in addition: list used plugins

Manual analysis of 8 apps (including one from SAP)
- to understand the use of Cordova
- to assess the quality of our automated analysis
What we have learned: plugin use

Plugins are used for

- accessing device information
- showing native dialog boxes and splash screens
- accessing network information
- accessing the file storage
- accessing the camera
- ...

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Use (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>52%</td>
</tr>
<tr>
<td>inappbrowser</td>
<td>50%</td>
</tr>
<tr>
<td>dialogs</td>
<td>40%</td>
</tr>
<tr>
<td>splashscreen</td>
<td>36%</td>
</tr>
<tr>
<td>network-information</td>
<td>28%</td>
</tr>
<tr>
<td>file</td>
<td>28%</td>
</tr>
<tr>
<td>console</td>
<td>24%</td>
</tr>
<tr>
<td>camera</td>
<td>22%</td>
</tr>
<tr>
<td>statusbar</td>
<td>22%</td>
</tr>
<tr>
<td>PushPlugin</td>
<td>22%</td>
</tr>
</tbody>
</table>
What we have learned: app size and cross-language calls

### App size:
- Mobile apps are not always small
- SAP apps seem to be larger than the average

### Cross-language calls:
- Calls from Java to JS: very common
- Calls from JS to Java: surprisingly uncommon

<table>
<thead>
<tr>
<th>App</th>
<th>Category</th>
<th>Java2JS</th>
<th>JS2Java</th>
<th>JS [kLoC]</th>
<th>Java [kLoC]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sap01</td>
<td>Finance</td>
<td>2</td>
<td>12</td>
<td>35.5</td>
<td>17.0</td>
</tr>
<tr>
<td>sap02</td>
<td>Business</td>
<td>20814</td>
<td>39</td>
<td>345.3</td>
<td>53.5</td>
</tr>
<tr>
<td>sap03</td>
<td>Business</td>
<td>9531</td>
<td>75</td>
<td>572.3</td>
<td>135.8</td>
</tr>
<tr>
<td>app01</td>
<td>Finance</td>
<td>9</td>
<td>13</td>
<td>26.3</td>
<td>17.8</td>
</tr>
<tr>
<td>app02</td>
<td>Finance</td>
<td>2</td>
<td>10</td>
<td>11.2</td>
<td>16.8</td>
</tr>
<tr>
<td>app03</td>
<td>Social</td>
<td>2349</td>
<td>31</td>
<td>4.6</td>
<td>103.7</td>
</tr>
<tr>
<td>app04</td>
<td>Business</td>
<td>1</td>
<td>6</td>
<td>37.5</td>
<td>16.8</td>
</tr>
<tr>
<td>app05</td>
<td>Finance</td>
<td>6</td>
<td>26</td>
<td>20.0</td>
<td>44.8</td>
</tr>
<tr>
<td>app06</td>
<td>Finance</td>
<td>693</td>
<td>70</td>
<td>30.4</td>
<td>24.3</td>
</tr>
<tr>
<td>app07</td>
<td>Travel &amp; Local</td>
<td>3430</td>
<td>43</td>
<td>129.0</td>
<td>304.0</td>
</tr>
<tr>
<td>app08</td>
<td>Entertainment</td>
<td>14220</td>
<td>67</td>
<td>36.7</td>
<td>23.0</td>
</tr>
<tr>
<td>app09</td>
<td>Lifestyle</td>
<td>51553</td>
<td>89</td>
<td>36.3</td>
<td>44.7</td>
</tr>
<tr>
<td>app10</td>
<td>Finance</td>
<td>8</td>
<td>36</td>
<td>43.7</td>
<td>18.4</td>
</tr>
<tr>
<td>app11</td>
<td>Business</td>
<td>0</td>
<td>0</td>
<td>14.0</td>
<td>438.9</td>
</tr>
</tbody>
</table>

...
### Recall and Precision

**Recall:**

- **Correctly reported calls**
  - **All reported calls**

**Precision:**

- **Correctly reported calls**
  - **Calls actually present**

<table>
<thead>
<tr>
<th>App</th>
<th>kLoC</th>
<th>kNodes</th>
<th>Plugins</th>
<th>Recall</th>
<th>Precision</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>app01</td>
<td>43</td>
<td>9</td>
<td>5</td>
<td>33%</td>
<td>75%</td>
<td>17</td>
</tr>
<tr>
<td>app02</td>
<td>27</td>
<td>8</td>
<td>4</td>
<td>100%</td>
<td>66%</td>
<td>13</td>
</tr>
<tr>
<td>app03</td>
<td>106</td>
<td>18</td>
<td>8</td>
<td>1%</td>
<td>93%</td>
<td>61</td>
</tr>
<tr>
<td>app04</td>
<td>53</td>
<td>14</td>
<td>3</td>
<td>100%</td>
<td>100%</td>
<td>7</td>
</tr>
<tr>
<td>app05</td>
<td>64</td>
<td>10</td>
<td>7</td>
<td>33%</td>
<td>66%</td>
<td>29</td>
</tr>
<tr>
<td>app06</td>
<td>53</td>
<td>8</td>
<td>12</td>
<td>35%</td>
<td>97%</td>
<td>316</td>
</tr>
<tr>
<td>app08</td>
<td>58</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app20</td>
<td>68</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app22</td>
<td>20</td>
<td>9</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app25</td>
<td>161</td>
<td>59</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app37</td>
<td>280</td>
<td>65</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app38</td>
<td>77</td>
<td>56</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app45</td>
<td>18</td>
<td>7</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| sap01 | 52   | 19     | 6       | 100%   | 66%       | 15    |
| sap02 | 398  | 15     | 17      |        |           |       |
| sap03 | 708  | 118    | 15      |        |           |       |

| dvhma | 17   | 7      | 4       | 100%   | 100%      | 15    |
What we have learned: exceptional behaviours

Cordova use:
- no HTML/JS in the app
- no use of Cordova

Plugin use:
- often callbacks are not used (missing error handling)
- plugins are modified
- plugins might use JNI
A Few Notes on Practical JavaScript Challenges

Analysing Hybrid Mobile Apps

- Motivation: Hybrid Mobile Apps and Their Security Challenges
- Static Analysis for Hybrid Apps: Building a unified call graph
- An assessment of hybrid Apps (in Google Play)
- Recommendations & Conclusions
Recommendations: The (hopefully) obvious parts (1/2)

Cordova apps are **Web applications**:
- do secure JavaScript programming
- content security policy, same origin policy
- ...

**Warning**: the WebView sandbox is not as strong as on desktop Web browsers

Cordova apps are **native/Java apps**:
- do secure Java/Objective-C/... programming
- do not trust validations done in the JavaScript part of the plugin
- ...

A.D. Brucker  The University of Sheffield  Static Analysis  February 8-12., 2016  129
Cordova apps are **mobile applications**:  
- permissions  
- ...  

Cordova apps are **cordova applications**:  
- plugin whitelisting  
- read the Cordova security guide:  
  https://cordova.apache.org/docs/en/5.4.0/guide/appdev/security/index.html
Recommendation: Use the latest framework version

Frameworks (and the underlying OS) can have vulnerabilities:
- use the latest version of Cordova (SAP Kapsel)
- monitor for public known vulnerabilities (e.g., CVEs)

Framework vulnerabilities can be severe:
- Java code execution via JavaScript: CVE-2013-4710
  Avoid Cordova on Android below 4.1 & use AddJavaScriptInterface annotation
- (incomplete) overview: https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-27153/Apache-Cordova.html
Summary

- Hybrid mobile apps are getting more popular
  - they are recommended at SAP
  - everything running in the Kapsel/Fiori Browser is a hybrid app

- Securing hybrid apps is a challenge and requires expertise in
  - Web application security
  - native/Java security
  - mobile security
  - Cordova/SAP Kapsel security

- Check the Cordova security guide:
  https://cordova.apache.org/docs/en/5.4.0/guide/appdev/security/index.html
Part V

Conclusions
Conclusion

Static and dynamic security testing approaches are an important means for improving software security.

From an industrial perspective
- They can be rolled out to 25,000 developers, but it is not easy
- Still problems that need to be solved
  - On the management/organizational level
  - On the technical level

From an academic (researcher) perspective
- While here is a wealth of literature, there are still many open questions
- Interesting area
  - crossing the boundary between verification and falsification
  - combining dynamic and static approaches
  - security and software/language engineering (“secure by construction”)
Thank you for your attention!

Any questions or remarks?

Contact:
Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP
UK

Phone: +44 114 22 21806
https://de.linkedin.com/in/adbrucker
https://www.brucker.uk
https://www.logicalhacking.com
a.brucker@sheffield.ac.uk
Bibliography I

Ruediger Bachmann and Achim D. Brucker.
Developing secure software: A holistic approach to security testing.

Achim D. Brucker and Michael Herzberg.
On the static analysis of hybrid mobile apps: A report on the state of apache cordova nation.

Lotfi ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker.
Time for addressing software security issues: Prediction models and impacting factors.

Factors impacting the effort required to fix security vulnerabilities: An industrial case study.
Bibliography II

Achim D. Brucker and Uwe Sodan.
Deploying static application security testing on a large scale.

Stanislav Dashevskyi, Achim D. Brucker, and Fabio Massacci.
On the security cost of using a free and open source component in a proprietary product.