



# TECHNIQUES AND TOOLS TO DEFEND AGAINST WEB APPLICATION'S SOFTWARE VULNERABILITIES

DSN 2015  
Rio de Janeiro – Brazil  
22 June 2015

**DEVISSES**

Nuno Antunes, Marco Vieira  
[nmsa@dei.uc.pt](mailto:nmsa@dei.uc.pt), [mvieira@dei.uc.pt](mailto:mvieira@dei.uc.pt)  
Department of Informatics Engineering  
University of Coimbra - Portugal




## NUNO ANTUNES

From the **University of Coimbra**  
– Portugal ☺


- **BSc** and **MSc** in Informatics Engineering
- **PhD** in Information Science and Technology  
– Since 2014
- Researcher at **CISUC** since 2008  
– Software and Systems Engineering group (**SSE**)
- Contact: [nmsa@dei.uc.pt](mailto:nmsa@dei.uc.pt)

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 2




## NUNO'S BACKGROUND

- Dependable and Secure Software
  - Software Vulnerabilities
  - Vulnerability Detection Tools
  - Experimental Dependability Evaluation
  - Benchmarking
  - Service Oriented Architectures
  - Verification and Validation
  - Intrusion Detection Systems
  - Virtualized Infrastructures



N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      3




## MARCO VIEIRA

Professor at the University of Coimbra

- Teaching experience of 17 years
  - Hum... getting old ☹
- BSc and MSc in Informatics Engineering
- PhD in Computer Science
- Large experience in projects with industry
  - Portugal Telecom, Critical Software, ESA, ...

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      4



## MARCO'S BACKGROUND

**Research:**

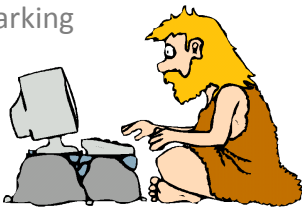
- Dependable and secure computing
- Experimental evaluation and benchmarking
- Software engineering

■ **Teaching:**

- Software engineering
- Databases
- And many other things...
  - Discrete math, telecommunications, data analysis, strategic information systems, programming languages, etc., etc.

■ **With industry:**


- Software engineering
- Databases and data warehousing (*decision support*)



N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

5



## COIMBRA


City in the center of Portugal

- 200 Km to the north of Lisbon

■ ~ 150 000 people

■ Most activity around the University

■ Many centuries of history



N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

6




## UNIVERSITY OF COIMBRA

### University of Coimbra

- One of the oldest in the world
  - Created in **1290**
- 9 schools (faculties)
  - **Sciences and Technology**
  - Law
  - Pharmacy
  - Economics
  - Psychology and Education Sciences
  - Sport Sciences and Physical Education
  - Medicine
  - Arts and Humanities
- About 23000 students
  - 18% of which are foreigners, including > 2000 Brazilians
- [www.uc.pt](http://www.uc.pt)

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 7




## SOFTWARE AND SYSTEMS ENGINEERING

### SSE

- Part of the Centre for Informatics and Systems of the University of Coimbra
- **Key people:**
  - Lead by Marco Vieira
  - 15 PhDs (Full Members) + 8 PhDs (Associate Members)
  - 30 PhD students
- **Areas of interest:**
  - Trustworthy and Resilient Software and Systems
  - Critical Services on the Cloud
  - Efficiency in Software Development
  - Reconfigurable Hardware for Resilient Systems

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 8






# INTRODUCTION

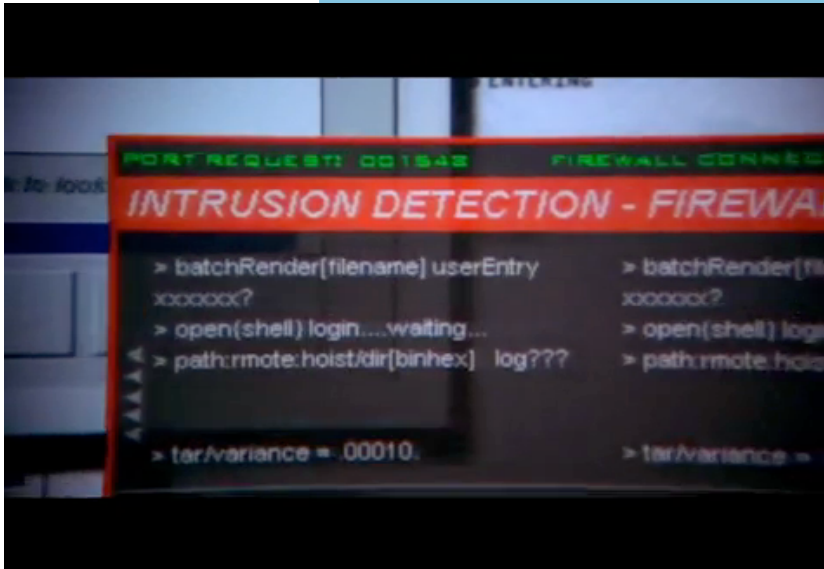
---

## *BACKGROUND*


N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      9



# COMPUTER SYSTEMS SECURITY




N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      10



## WHAT IS A VULNERABILITY?


A **weakness** that may allow **attackers** to gain access to the system or info

- [Stock07]



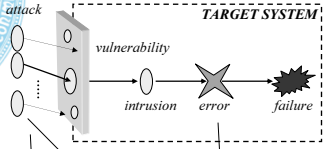
- There are many causes:
  - Complexity
  - Password and privileges management flaws
  - Operating system design flaws
  - Software bugs
  - Unchecked user input
  - ...

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 11



## VULNERABILITY VS FAULT


**Fault → Error → Failure**  
**(attack + vulnerability) → intrusion**



- Attack
  - Malicious activities that intentionally attempt to violate security properties of the system
- Vulnerability is a **fault** that leave space for malicious exploitation of a system
- Intrusion
  - An attack that successfully activates a vulnerability


Neves et al. "Using Attack Injection to Discover New Vulnerabilities", DSN'06

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 12




## AN IMPORTANT PROBLEM ...

Create and feed an underground economy



- Companies are aware of that:
  - OWASP Security Spending Benchmarks 2009 shows that investment in security is **increasing**
- However...

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 13




## ... THAT IS NOT GETTING BETTER!

NTA Web Application Security Reports show that Web Security is **decreasing**

- According to the WhiteHat Security Website Security Statistics Report, **63%** of assessed websites are vulnerable
- Something is **wrong** in the development of web applications!

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 14




## SOFTWARE VULNERABILITIES

Most security problems in computer systems are related to **unknown** attacks and vulnerabilities

- Very hard to prevent!
- Some anomaly detection systems can catch one or another attack by detecting abnormal activities
  - Not much more to do other than apply best practices and "pray" ☺

- But, there are vulnerabilities / attacks that **we know!**
  - Some of these vulnerabilities are considered as "solved"
- **Knowing them does not make them less devastating**
  - **SQL Injection** is a good example
    - it surely has the potential for catastrophic consequences...
    - ... and it is somewhat easy to avoid!

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 15




## EXAMPLE SOFTWARE VULNERABILITIES

Denial of Service

- Oversize Payload, Coercive Parsing, Oversize Cryptography, Attack Obfuscation, Unvalidated Redirects and Forwards

- Brute Force
  - Insecure Cryptographic Storage, Broken Authentication and Session Management
- Spoofing
  - Insufficient Transport Layer Protection, Metadata Spoofing, Security misconfiguration
- Injection
  - SQL Injection, Cross site Scripting (XSS), Cross Site Request Forgery, XPath Injection
- Flooding
  - Instantiation Flood, Indirect Flooding, BPEL State Deviation

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 16




## WEB APPLICATION VULNERABILITIES

Web applications are widely exposed

- Publicly visible face of an organization
- Became the preferred targets for hackers

- Hackers moved their focus from the network to application's code
- Traditional security mechanisms are not effective
  - Firewall, IDS, encryption can't mitigate these vulnerabilities
- Application level attacks
  - Use network ports that are used for regular web traffic
  - Use specially tampered values
  - Exploit inputs of improperly coded applications

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 17




## VULNERABILITIES IN WEB APPS (1)

### SQL Injection

- It is possible to alter the construction of backend SQL statements
- An attacker can read or modify database data and
- In some cases, execute database administration operations or commands in the system

- XPath Injection
  - It is possible to modify an XPath query to be parsed in a way differing from the programmer's intention
  - Attackers may gain access to information in XML documents

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 18



## VULNERABILITIES IN WEB APPS (2)

### Code Execution


- It is possible to manipulate the application inputs to trigger server-side code execution
- An attacker can exploit this vulnerability to execute malicious code in the server machine

### ■ Username/Password Disclosure

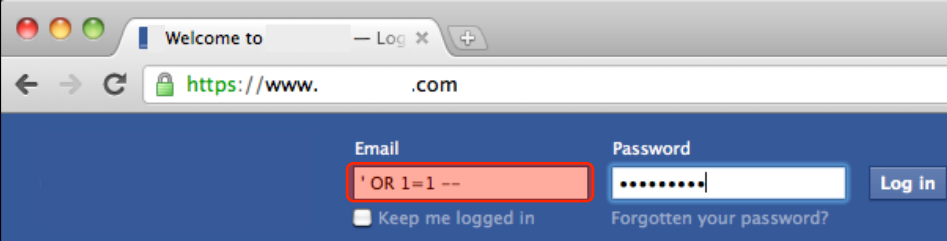
- A response contains information related to usernames and/or passwords
- An attacker can use this information to get access to private data

■ ...

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 19



## EXAMPLE OF SQL INJECTION

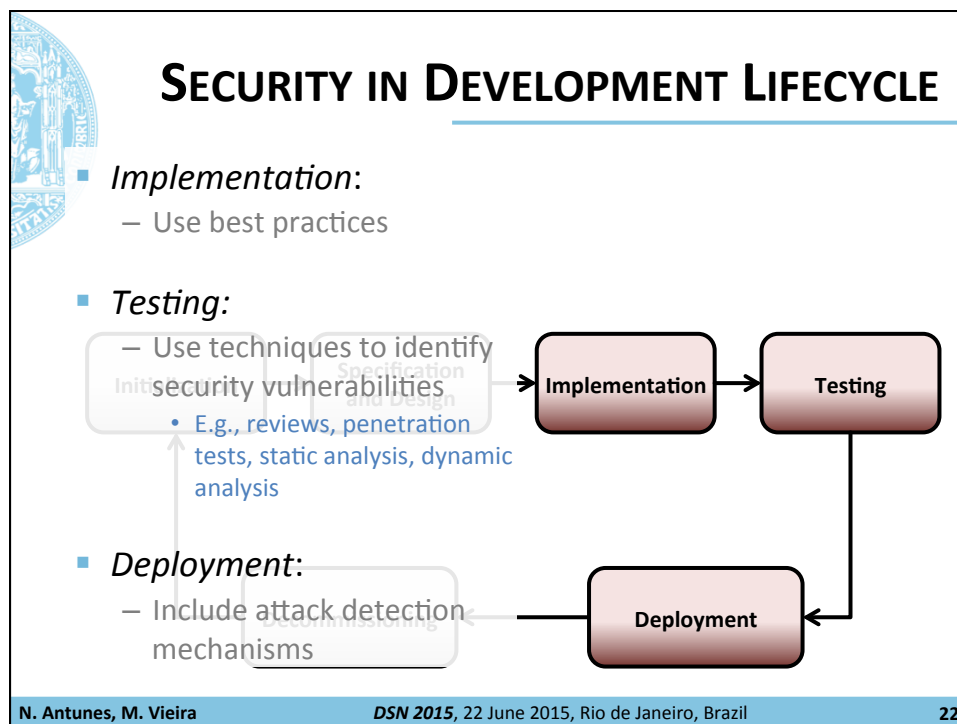
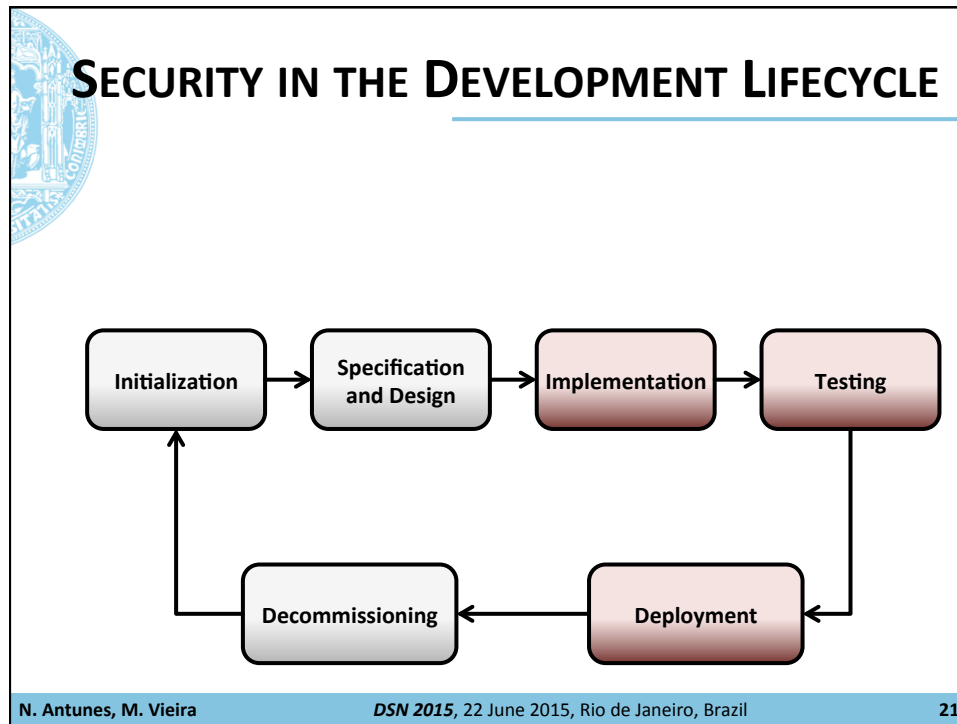


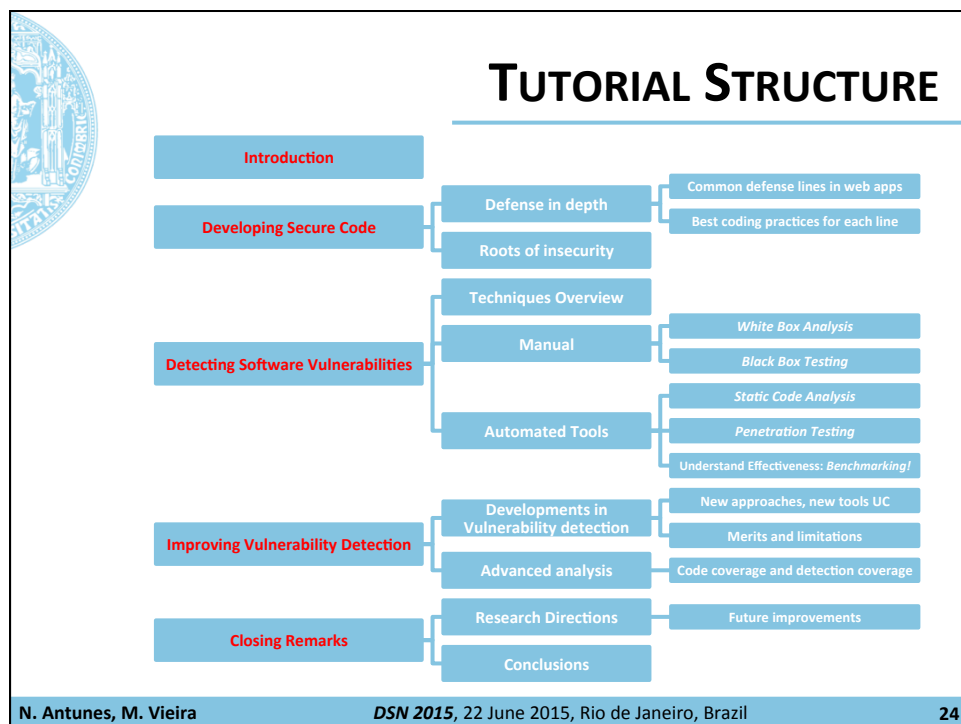
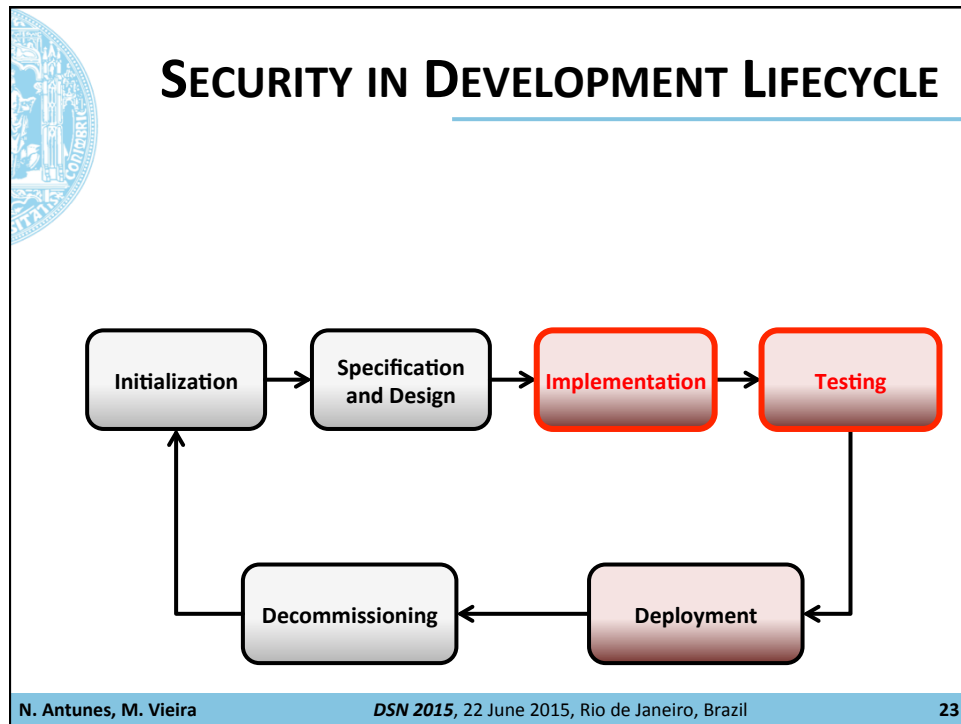
The screenshot shows a web browser window with a login form. The email field contains the payload `' OR 1=1 --`. The password field is masked with dots. Below the form, the original SQL query is shown, and a highlighted box shows the modified query after injection.

```
(...)  
String sql = "SELECT * FROM users WHERE "+  
    "username='" +      + "'" AND "+  
    "password='" + Password + "'";  
(...)
```


```
"SELECT * FROM users WHERE  
    username=' ' OR 1=1 -- ' AND  
    password=' '";
```

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 20











# QUESTIONS?

## *INTRODUCTION*




N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      25



# TUTORIAL STRUCTURE

- Introduction
- Developing Secure Code
  - Defense in depth
    - Common defense lines in web apps
  - Roots of insecurity
    - Best coding practices for each line
- Detecting Software Vulnerabilities
  - Techniques Overview
    - Manual
      - White Box Analysis
      - Black Box Testing
    - Automated Tools
      - Static Code Analysis
      - Penetration Testing
      - Understand Effectiveness: Benchmarking!
- Improving Vulnerability Detection
  - Developments in Vulnerability detection
    - New approaches, new tools UC
    - Merits and limitations
  - Advanced analysis
    - Code coverage and detection coverage
- Closing Remarks
  - Research Directions
    - Future improvements
  - Conclusions

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      26




## DEVELOPING SECURE CODE

---

### *DEFENSE IN DEPTH*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      27




## MOTIVATION

---


Most security problems in computer systems are related to **unknown** attacks and vulnerabilities

- Even in this case, it is better to apply best practices
- There are vulnerabilities / attacks that **we do know!**
  - Some of these vulnerabilities are considered as “solved”
    - But they keep showing up in the applications
  - **Knowing them does not make them less devastating**
    - Unless the developers avoid them
- Owners need to give the due importance to the matter
- Developers need to apply the best practices

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      28



## THE SOLUTION




*«...a **defense-in-depth** approach,  
with overlapping protections,  
can help secure Web applications»*

[Howard02]

N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

29



## USE DEFENSE-IN-DEPTH


Assume that every security precaution can fail

- Security depends on several layers of mechanisms
  - Cover the failures of each other
- Every defensive protecting can be destroyed
  - Imagine your application is the last component standing!
- Be prepared to defend your system
  - The "security features" defending it might be annihilated
  - E.g. although you are protected by a firewall, build the system as if the firewall is compromised
    - A great deal of software is designed and written in a way that leads to total compromise when a firewall is breached
    - **Not good enough today**

N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

30




## THE BANK EXAMPLE

To get to the big money you need to get to the vault

- Requires that you go through multiple layers of defense:
- Examples of the defensive layers:
  - There is often a guard at the bank's entrance
  - Some banks have time-release doors
    - You cannot rush in and rush out
    - A teller can lock the doors remotely, trapping a exiting thief
  - There are guards inside the bank
  - Closed-circuit cameras monitor the movements of everyone
  - Tellers do not have access to the vault
    - An example of least privilege principle
  - The vault itself has multiple layers of defense, such as:
    - It opens only at certain controlled times
    - It's made of very thick metal
    - Multiple compartments in the vault require other access means

[Howard02]

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 31




## APPLY DEFENSE IN DEPTH

Harden all layers of the technological stack

- If a server's OS is insecure, an attacker exploiting a command injection flaw may be able to escalate privileges
  - May propagate through network if other hosts were not hardened
  - But, if the underlying servers were secured, an attack may be fully contained within one or more tiers of the application
- Sensitive data persisted in any tier of the application should be encrypted
  - Prevent trivial disclosure in the event that that tier is compromised
  - User credentials and other sensitive information, such as credit card numbers, should be stored in encrypted form within the database
  - Built-in mechanisms should be used to protect database credentials
    - In ASP.NET 2.0, an encrypted database connection string can be stored in the web.config file.

[Howard02]

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 32




## DEFENSE-IN-DEPTH ADVANTAGES

Even in a complete compromise, may give you the ability to minimize the impacts of the incident

- Impose restrictions on the web server's capabilities from other, autonomous components of the application
  - E. g., if the DB account used by the application has only INSERT access to the **audit log tables**, an attacker cannot delete any log entries that have already been created
- Impose strict network level filters on traffic
  - To and from the web server
- Use an IDS to identify any anomalous network activity
  - May indicate that a breach has occurred
  - After compromising a web server, attackers will attempt to create a reverse connection out to the Internet, or scan for other hosts on the DMZ network

[Howard02]

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 33

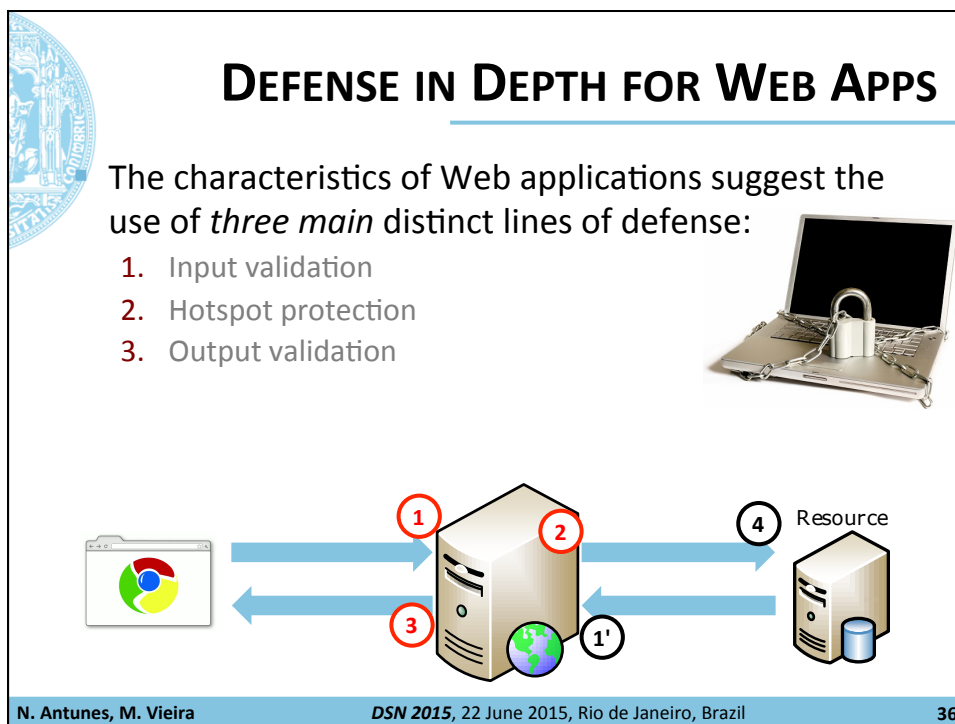
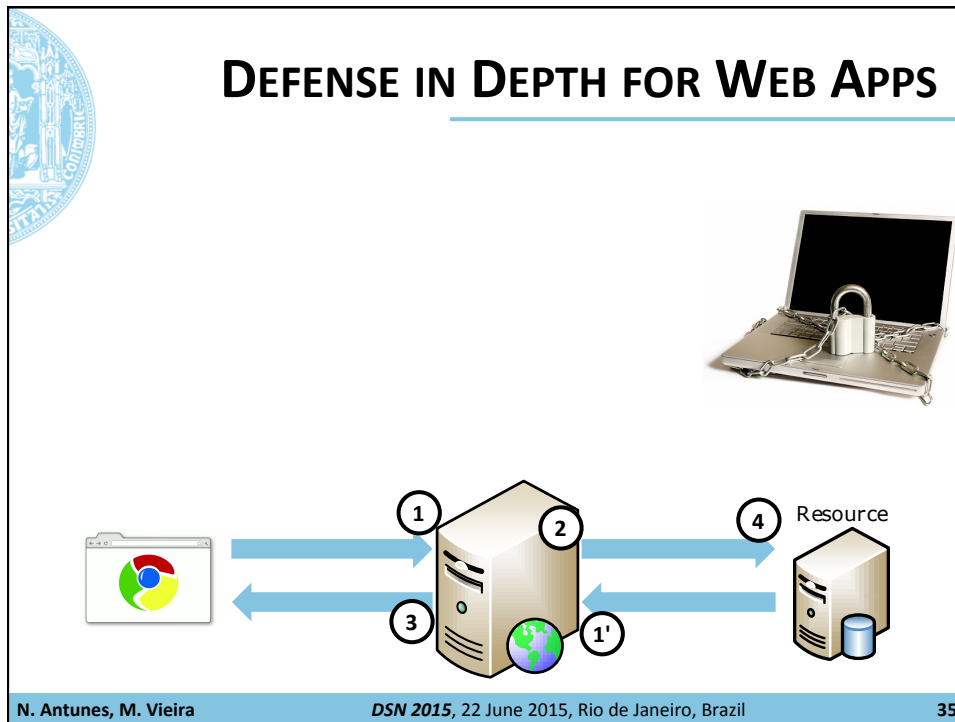


## REACTING TO ATTACKS

*Security-critical* applications may contain built-in mechanisms to react defensively

- Against users identified as potentially malicious
- Besides alerting addition to alerting administrators
- Of course, it is not a substitute for fixing vulnerabilities
  - However, in real world, even the most verified applications may have some exploitable defects
- Creating other obstacles to an attacker reduces the chances that remaining vulnerabilities will be exploited
  - And may reduce the impact

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 34

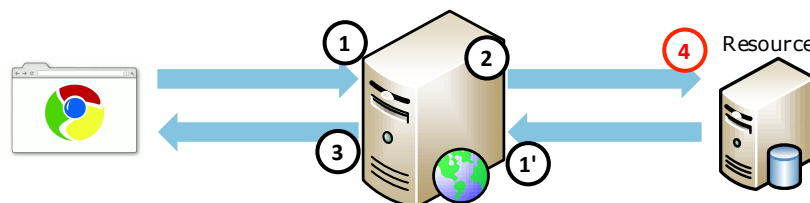


## DEFENSE IN DEPTH FOR WEB APPS

The characteristics of Web applications suggest the use of *three main* distinct lines of defense:

1. Input validation
2. Hotspot protection
3. Output validation

- Another line may be considered
- 4. Protect back-end resources

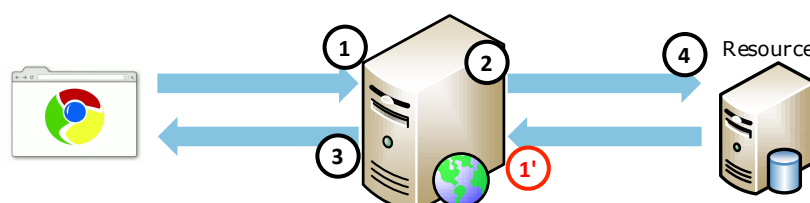


N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 37


## DEFENSE IN DEPTH FOR WEB APPS

Every input must be considered malicious, until proven otherwise

- Applies for any data coming from untrusted environments
- You may even have to consider malicious everything that crosses boundaries in system
  - Second order attacks take advantage of who does not do that



N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 38



## LINE 1: INPUT VALIDATION

Reduce an application's input domain


- All inputs are malicious until proven otherwise!

- Start with normalization of the inputs
  - To a baseline character set and encoding
- Use filtering strategies to reject values outside domain
- Or, use positive pattern matching
  - positive validation
  - "Whitelisting"

✗ Information domain can allow malicious data:

- e.g. < or > in the case of XSS

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 39



## LINE 2: HOTSPOT PROTECTION

**Hotspot** is a set of statements that is *prone* to a specific type of vulnerabilities

- An attack always targets a hotspot

- Hotspot is not a synonym of vulnerability
  - It is prone to a vulnerability, but it can be perfectly secure!!!
- This line of defense focuses on protecting the hotspots
  - The goal is to ensure that these specific set of lines is correctly implemented
    - Or the best possible ☺

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 40





## LINE 3: OUTPUT VALIDATION

Validate the output of a process before it is sent to the recipient

- "Sanitize the output"
- Prevents users from receiving restricted information:
  - Internal Exceptions that can lead to other attacks
  - Credit card numbers
- **Output encoding** is a example of output validation
  - Avoids XSS vulnerabilities
  - A common mistake is to encode only the characters that appear to allow attacks directly
    - E.g. Encode only the " char when a item is inserted in a string
    - An attacker can exploit browsers' tolerance of invalid HTML and JavaScript to change context or inject code in unexpected ways

N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

41



## LINE 4: PROTECT BACKEND RESOURCES


Use the lowest possible level of privileges

- A *principle transversal* to every security good practice
- Unnecessary functionalities should be removed or disabled
  - In some cases the attacker may be able to restore them
    - But it creates difficulties to his job 😊
- Follow the best practices for the specific resource
  - All vendor-issued security patches should be evaluated tested, and applied in a timely way
- Give special attention when dealing with information
  - IDS, Anomaly detection techniques, etc.

N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

42



## DEFENDING AGAINST SQL INJECTION


SQL injection is in general one of the easier vulnerabilities to prevent

- Despite the frequency of its occurrences...
- ... its different manifestations
- ... the consequences of its exploitation

- Discussion about countermeasures is often misleading
  - Many people rely upon defensive measures that are only **partially** effective

**So, how should we proceed to avoid these vulnerabilities?**

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      43



## INPUT VALIDATION FOR SQL INJECTION

An important first step

- Helps in some cases, but does not solve the problem
  - Additional layer to make the life of the attacker harder


✗ Domain can allow malicious data:

- e.g. ' in the case of SQL injection
- This means that you cannot filter out all the potentially malicious values

✗ Vulnerable to second order attacks:

- If the focus is on the inputs only

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      44




## USE STORED PROCEDURES

Take 1 on Hotspot Protection

- Procedures can provide security and performance benefits
- ✗ A poorly written procedure can contain SQL injection vulnerabilities within its own code
- Even if correctly implement
- ✗ SQL injection vulnerabilities can arise if it is invoked in an unsafe way using user-supplied input
  - `exec sp_RegisterUser 'joe', 'secret'`
  - If the user input is used incorrectly, this statement may be just as vulnerable as an INSERT

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      45




## INPUT ESCAPING

Take 2 on Hotspot Protection

- Escape the characters that affect the specific hotspots
- ✗ Often escaping is incorrectly implemented
  - e.g. it is not enough to escape `'`
- Even if correctly implemented...
- ✗ Escaping characters increases the length of the string
  - May cause data truncation

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      46




## PARAMETERIZED QUERIES

Take 3 on Hotspot Protection

- Also called prepared statements

- Define query structure using placeholders for variables
- Before each execution, values are attached
  - The interpreter can use them correctly
    - No matter the data's content, it will always be used as a value
      - Never as code!
- ✗ SQL injection vulnerabilities can arise if the query is prepared using user-supplied input
  - Must be used correctly

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 47




## WHEN YOU CANNOT PARAMETERIZE?

Parameter placeholders cannot be used to specify the table and column names used in the query

- In very rare cases, applications need to specify these items within an SQL query on the basis of user-supplied data
- Best approach: use a whitelist of known good values
  - Reject any input that does not match an item on this list!
- If not possible, apply strict validation
  - E.g. allow only alphanumeric characters, exclude white-space, and enforce a suitable length limit

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 48




## LINE 4: PROTECTING THE DATABASE

Give any database user the minimum privileges


- If possible, employ accounts for performing different actions
  - E.g. if 90% of database queries only require read access, then these **should** be performed using an account **without write privileges**
  - If a query only reads a subset of data, then use an account with the corresponding level of access
- Unnecessary functionalities should be disabled or not even installed
  - In some cases the attacker may be able to restore them
- Security patches must be evaluated and applied timely
  - In security-critical systems, subscribe services to obtain advance notification of some found vulnerabilities

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      49




## QUESTIONS?

*DEFENSE IN DEPTH*



N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      50




## SECURE CODING

---

### *ROOTS OF INSECURITY*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      51



## KEY FACTORS (1)

---


### Immature Security Awareness

- The awareness of web application security issues is less mature than there is in other areas as networks or Oss

- In-House Development
  - Many web applications are developed in-house by an organization's own staff or contractors
    - Every application is different and may contain its own defects
- Deceptive Simplicity
  - It is possible for a novice programmer to create a powerful application from scratch in a short period of time
  - Code that usually is far from secure

[Scutard07]

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      52




## KEY FACTORS (2)

- Rapidly evolving threat profile
  - Research in web attacks and defenses is a thriving
    - Concepts and threats are conceived at a fast rate
    - Developing team's knowledge becomes outdated during the projects
- Resource and time constraints
  - Development projects are time and resources limitations
  - Producing functional applications is more important than security
- Overextended technologies
  - Many technologies were developed for a different WWW
  - Since have been pushed far beyond their original purposes
    - e.g. , the use of JavaScript for data transmission in many AJAX-apps
  - Unforeseen side effects lead to vulnerabilities

[Sutard07]

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 53



## WHY DEVELOPERS DO NOT APPLY THIS?

- Training and education
  - Developers cannot prevent flaws if they do not know how
  - Many programs lack courses about secure design/coding/testing
- Security is boring and uninteresting
  - Does not directly contribute to developing new and exciting functionalities
  - More attention to functional requirements
- Someone else should “take care” of security
  - like network management staff.
  - Developers ignore that their code is the main target of attackers


N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 54



## WHY DEVELOPERS DO NOT APPLY THIS?


- Security “limits” application functionality
  - More features increase the attack surface
    - Adds potential for vulnerabilities
  - Simplifying authentication/authorization procedures might enhance usability but it also makes it easier for attackers
- Security is not seen as a core feature of most products
  - Many times companies hire testers or SQ experts
  - Companies hire security experts...
    - ... but much less than programmers, designers or ux experts
- Developers are not the "owners" of the product
  - They do not set the priorities
  - There are different priorities between the client, the supplier and the developers

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 55



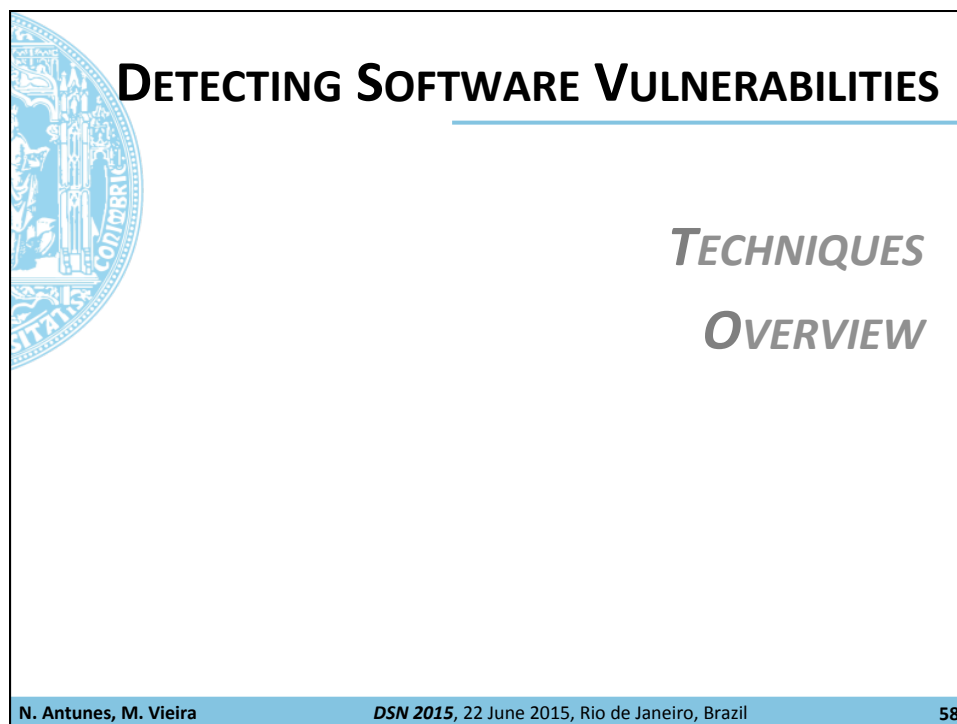
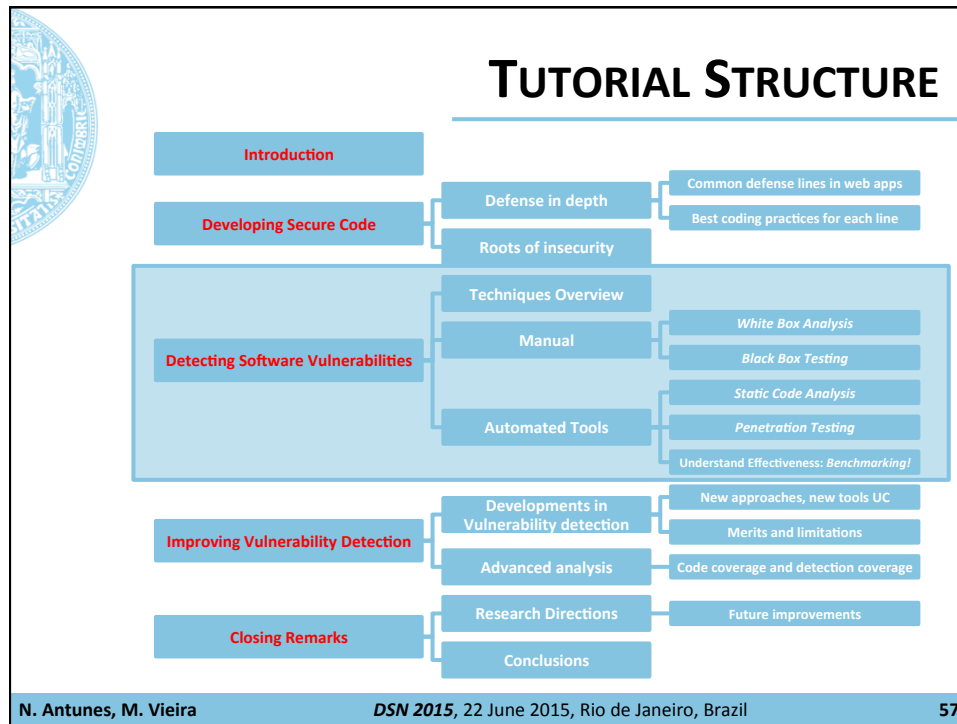
## QUESTIONS?


### *ROOTS OF INSECURITY*



N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 56








## VULNERABILITY DETECTION TECHNIQUES

**Goal:** find vulnerabilities, so they can be corrected!

- Several techniques available
  - **White box analysis**, **black box testing**, dynamic analysis, white box testing, anomaly detection, ...
  - Can usually be performed both **manually** or using automated **tools**
- **Detection coverage** is important
  - We do not want to leave vulnerabilities undetected
- **False positives** are also important
  - We do not want to waste resources fixing problems that do not exist!

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      59




## WHITE-BOX ANALYSIS

Techniques that analyze the code without actually executing it

- Look for potential vulnerabilities
  - Among other types of software defects
- Automated tools provide an automatic way for highlighting possible coding errors
- ✗ Require access to the source code or bytecode
- ✗ Ignore the runtime perspective

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      60




## BLACK-BOX TESTING

Techniques that test the program from an external perspective

- The point of view of the **attacker**...
- Automated tools provide an automatic way to search for vulnerabilities
  - Avoid a large number of manual tests
- ✓ Does NOT require access to the source code or bytecode
- ✗ Ignores the internals of the application ☹

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 61




## WHITE-BOX (GLASS-BOX) TESTING (1)

Examines the program structure and derives test data from the program logic/code

- Also known as clear box testing, open box testing, logic driven testing, path driven testing, structural testing, ...
- Approaches for structural testing:
  - Statement Coverage
    - Aimed at exercising all programming statements with minimal tests
  - Branch Coverage
    - Running a series of tests to ensure that all branches are tested at least once
  - Path Coverage
    - Testing all possible paths which means that each statement and branch are covered

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 62



## WHITE-BOX (GLASS-BOX) TESTING (2)


**Advantages:**

- Forces the tester to analyze carefully the implementation
- Addresses *Dead Code* and other issues related to best coding practices

■ **Disadvantages:**

- Expensive as it requires time and money to perform white box analysis and testing
- Requires in-depth knowledge about the programming language

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      63



## GRAY BOX TESTING

The tester has some (limited) knowledge of the internal details of the program

- A gray box is a device, program or system whose workings are partially understood


■ **Dynamic Analysis**

- Testing and evaluation of an application during runtime
- Code typically instrumented to observe measures of interest

■ **Runtime Anomaly Detection**

- Consists on the detection of behavior deviations at runtime
- Frequently requires a learning phase to serve as reference

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      64




## NOTES ON THE PRESENTED TECHNIQUES

All these techniques can be used for more than just detecting vulnerabilities

- But here, we are **mainly interested** in this perspective
- Although not always the most effective, white box analysis and black box testing are still...
  - ... the most widely known
  - ... the most used
  - ... the ones with more tools implementing it
    - And thus have the better cost/benefit relation in most cases


N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 65



## DETECTING SOFTWARE VULNERABILITIES

### *WHITE BOX ANALYSIS*

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 66




Adapted from Prof. Madeira slides, University of Coimbra

## SOFTWARE REVIEWS

Software reviews are a *“quality improvement processes for written material”*


- Follow the general idea that written material (e.g., papers, books, leaflets, ... and software) must go to successive review interactions to improve quality
- Help supporting:
  - Project management
  - Systems engineering
  - Verification and validation
  - Configuration management
  - Quality assurance



N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

67



Adapted from Prof. Madeira slides, University of Coimbra

## FAGAN'S INSPECTION

Software inspections were proposed by Michel Fagan

- in 1976

*«A highly structured, clearly defined process by which software documents are reviewed in detail by a team including the author and, ideally, the customer»*

- **Goal**
  - Identify defects as closely as possible to the point of occurrence in order to facilitate corrective actions
- Many variants have been proposed since then...

N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

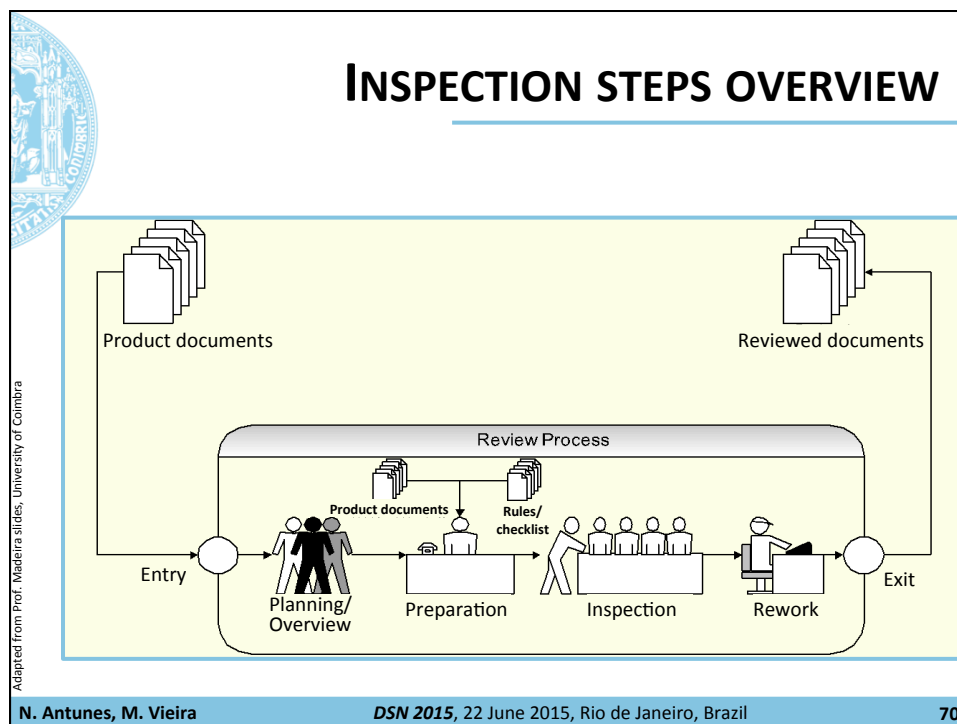
68


**TYPES OF SOFTWARE REVIEWS**

- **Buddy checking**
  - Informal review by a person other than the author
    - Usually, does not involve the use of checklists to guide the process
    - Not repeatable ☹️
- **Walkthrough**
  - Presentation to a peer audience that provides comments and feedbacks
    - Usually, limited documentation of the process and the issues uncovered
    - Difficult defect tracking
- **Review by circulation (asynchronous review)**
  - Consist of circulating an artifact among peers for comments
    - Like a walkthrough but without holding a meeting
    - Avoids potential arguments over issues...
    - ... but also avoids the benefits of discussion
- **Inspection**
  - Formal and managed peer review process

Adapted from Dr. Lisa Traore slides, University of Victoria, Canada

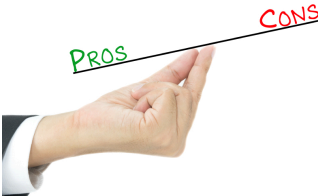
**N. Antunes, M. Vieira** *DSN 2015*, 22 June 2015, Rio de Janeiro, Brazil **69**






## PROS AND CONS OF INSPECTIONS

- Pros
  - Regarded as the most effective technique to find bugs and vulnerabilities
  - Allow an easier correction of the bugs
  - Allow learning to improve the processes
- Cons
  - Need for expertise
  - Cost!



N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      71




## STATIC CODE ANALYSIS

“white-box” approach

- Consists in analyzing the source code of the application without execution it
- Looks for potential vulnerabilities
  - Among other types of software defects
- Can be performed manually or automatically
- Does require access to the source code (or bytecode)

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      72






## STATIC CODE ANALYSIS TOOLS

Provide an automatic way for highlighting possible coding errors

- The analysis varies depending on the tool sophistication
  - Ranging from tools that consider only individual statements and declarations
  - To others that consider the complete code
- Have other usages
  - e.g., model checking and data flow analysis

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 73



## EXAMPLES OF LIMITATIONS


```
public void operation(String str) {
    int i = Integer.parseInt(str);
    try {
        String sql = "DELETE FROM table" +
            "WHERE id='" + str + "'";
        statement.executeUpdate(sql);
    } catch (SQLException se) {}
}
```

**Analyzers identify the vulnerability because the SQL query is a non-constant string**

```
public String dumpDepositInfo(String str) {
    try {
        String path = "//DepositInfo/Deposit"+
            "[@accNum='" + str + "']";
        return csvFromPath(path);
    } catch (XPathException e) {}
    return null;
}
```

**Depending on the complexity of csvFromPath method, a static analysis tool may not be able to find the vulnerability**

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 74




## SOME STATIC ANALYSIS TOOLS

- FindBugs
  - Yasca (Yet Another Source Code Analyzer)
  - IntelliJ IDEA
  - ...

Case Examples

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 75




## FINDBUGS

*“A program which uses static analysis to look for bugs in Java code”*

- It is able to scan the bytecode of Java applications
- Detects, among other problems, security issues
- It is one of the most used tools for static code analysis

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 76




## YASCA

*Yet Another Source Code Analyzer*

- “A framework for conducting source code analyses”
- Wide range of programming languages, including java
- Yasca includes two components:
  - A framework for conducting source code analyses
  - An implementation of that framework that allows integration with other static code analyzers
    - e.g., FindBugs, PMD, Jlint


N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      77




## INTELIJ IDEA

A commercial tool that provides a powerful IDE for Java development

- Includes “inspection gadgets” plug-ins with automated code inspection functionalities
- IntelliJ IDEA is able to detect security issues in java source code




N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      78



# DETECTING SOFTWARE VULNERABILITIES

## *BLACK BOX TESTING*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      79




## BLACK BOX TESTING

Software is treated as a **black-box**

- ... and tested against a specification of its external behavior
- ... without knowledge of internal implementation details
  - Behavior defined in functional or requirements specs, API docs, etc.
    - Also called **Functional Testing**

- Attention is focused on the information domain
  - Program structure is disregarded
  - Implementation details do not matter
- Requires an end-user perspective
- Criteria are not always precise

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      80



## BLACK BOX TESTING (2)


Tends to find different kinds of errors

- Missing functionalities
- Usability problems
- Performance problems
- Concurrency and timing errors
- Initialization and termination errors
- ...

- **Challenge:** create effective test cases that represent the potentially **infinite input space**
  - Equivalence class partitioning, and Boundary values analysis are used

Adapted from Prof. Ken Rodham slides, Brigham Young University

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      81




## ROBUSTNESS TESTING

A specific form of black box testing

- *Characterize the behavior of a system in presence of erroneous, unexpected or exceptional input conditions*

- Stimulate the system in a way that triggers internal errors
  - To expose programming and design defects
  - Systems can be improved and/or differentiated based on the errors uncovered
- Used very often to test software interfaces such as system calls, APIs, web services, etc.

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      82




## PENETRATION TESTING

Specialization of robustness testing

- Consists in stressing the application from the point of view of an attacker
  - “black-box” approach
  - Fuzzing techniques to test the inputs
  - Uses specific malicious inputs
    - e.g., for SQL Injection: ' or 1=1
- The tester needs no knowledge about the implementation details

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 83



## PENETRATION TESTING TOOLS

Provide an automatic way to search for vulnerabilities

- Avoid the repetitive and tedious task of doing hundreds or even thousands of tests by hand
- Many tools available
  - Including commercial and open-source
- Different tools target different types of vulnerabilities
- The effectiveness of penetration testing tools is doubtful

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 84

## EXAMPLES OF LIMITATIONS

```
public void operation(String str) {  
    try {  
        String sql = "DELETE FROM table" +  
            "WHERE id='" + str + "'";  
        statement.executeUpdate(sql);  
    } catch (SQLException se) {}  
}
```

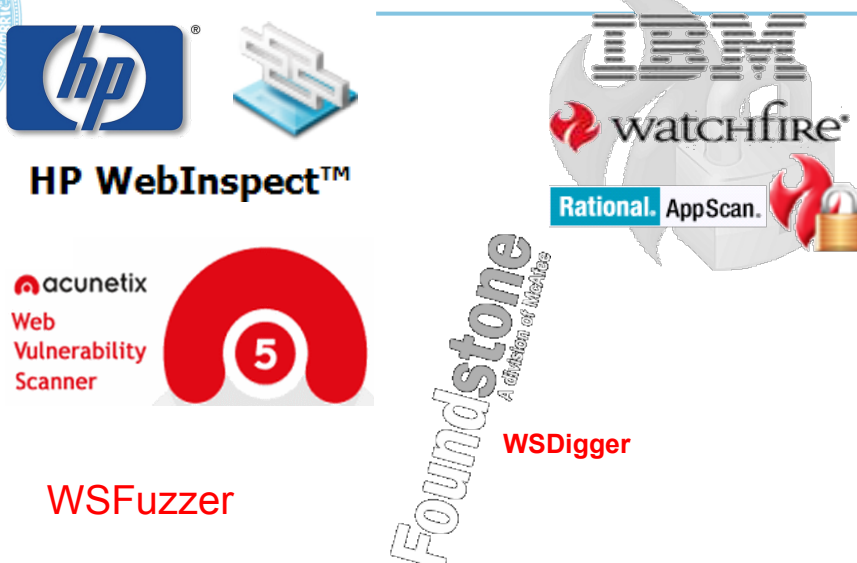
**No return value and exceptions related with SQL malformation do not leak out to the invocator**

```
public String dumpDepositInfo(String str) {  
    try {  
        String path = "//DepositInfo/Deposit"+  
            "[@accNum='" + str + "']";  
        return csvFromPath(path);  
    } catch (XPathException e) {}  
    return null;  
}
```

**Lack of output information**

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 85

## SOME PENETRATION TESTING TOOLS



HP WebInspect™

acunetix Web Vulnerability Scanner


WSFuzzer

Foundstone WSDigger

IBM Watchfire

Rational AppScan

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 86




## HP WEBINSPECT

*“Web application security testing and assessment for complex web applications*

- *Built on emerging Web 2.0 technologies*
- *Fast scanning capabilities, broad security assessment coverage*
- *Accurate web application security scanning results”*
- Includes pioneering assessment technology
  - Including simultaneous crawl and audit (SCA) and concurrent application scanning
- Targets many different types of vulnerabilities

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      87




## IBM RATIONAL APPSCAN

*“a leading suite of automated Web application security and compliance assessment tools Scan for common application vulnerabilities”*

- Suitable for users ranging from non-security experts to advanced users
- Supports extensions for customized scanning environments

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      88






## ACUNETIX WEB VULNERABILITY SCANNER

*“Is an automated web application security testing tool*

- *Audits your web applications by checking for exploitable hacking vulnerabilities”*
- Broad scanning capabilities
  - Targets many different types of vulnerabilities
- Can be applied for security testing in web applications in general


N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 89



## DETECTING SOFTWARE VULNERABILITIES

***CASE STUDY:  
BENCHMARKING VULNERABILITY  
DETECTION TOOLS***

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 90




## HOW TO SELECT THE TOOLS TO USE?

Existing evaluations have limited value

- By the limited number of tools used
- By the representativeness of the experiments

- Developers urge a practical way to compare alternative tools concerning their ability to detect vulnerabilities
- The solution: **Benchmarking!**

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      91




## BENCHMARKING VD TOOLS

Benchmarks are standard approaches to evaluate and compare different systems

- According to specific characteristics

- Evaluate and compare the existing tools
- Select the most effective tools
- Guide the improvement of methodologies
  - As performance benchmarks have contributed to improve performance of systems


N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      92



## BENCHMARKING COMPONENTS

- Workload:**
  - Work that a tool must perform during the benchmark execution
  - i.e. services to exercise the Vulnerability Detection Tools
- **Measures:**
  - Characterize the effectiveness of the tools
  - Must be easy to understand
  - Must allow the comparison among different tools
- **Procedure:**
  - The procedures and rules that must be followed during the benchmark execution

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      93



## BENCH. FOR SQL INJECTION VD TOOLS

Targets the following domain:

- Class of web services: SOAP web services
- Type of vulnerabilities: SQL Injection
- Vulnerability detection approaches: penetration-testing, static code analysis, and runtime anomaly detection

- **Workload** composed by code from standard benchmarks:
- TPC-App
- TPC-W\*
- TPC-C\*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      94

## WORKLOAD

Benchmark	Service Name	Vuln. Inputs	Vuln. Queries	LOC	Avg. C. Comp.
TPC-App	ProductDetail	0	0	121	5
	NewProducts	15	1	103	4.5
	NewCustomer	1	4	205	5.6
	ChangePaymentMethod	2	1	99	5
TPC-C	Delivery	2	7	227	21
	NewOrder	3	5	331	33
	OrderStatus	4	5	209	13
	Payment	6	11	327	25
	StockLevel	2	2	80	4
TPC-W	AdminUpdate	2	1	81	5
	CreateNewCustomer	11	4	163	3
	CreateShoppingCart	0	0	207	2.67
	DoAuthorSearch	1	1	44	3
	DoSubjectSearch	1	1	45	3
	DoTitleSearch	1	1	45	3
	GetBestSellers	1	1	62	3
	GetCustomer	1	1	46	4
	GetMostRecentOrder	1	1	129	6
	GetNewProducts	1	1	50	3
	GetPassword	1	1	40	2
	GetUsername	0	0	40	2
Total		56	49	2654	-

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 95


## ENHANCING THE WORKLOAD

To create a more realistic workload we created new versions of the services

- For each web service we have:
  - one version without known vulnerabilities
  - one version with N vulnerabilities
  - N versions with one vulnerable SQL query each
- This accounts for:

Services + Versions	Vuln. Inputs	Vuln. lines
80	158	87

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 96




## MEASURES

Computed from the information collected during the execution of the vulnerability detection tools

- Relative measures
  - Can be used for comparison or for improvement and tuning
- Different tools report vulnerabilities in different ways
  - Precision
  - Recall
  - F-Measure

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TV}$$
$$F\text{-Measure} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 97



## PROCEDURE

- Step 1: Preparation
  - Select the tools to be benchmarked
- Step 2: Execution
  - Use the tools under benchmarking to detect vulnerabilities in the workload
- Step 3: Measures calculation
  - Analyze the vulnerabilities reported by the tools and calculate the measures.
- Step 4: Ranking and selection
  - Rank the tools using the measures
  - Select the most effective tool

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 98

## STEP 1: PREPARATION

The tools under benchmarking

Provider	Tool	Technique
HP	WebInspect	Penetration testing
IBM	Rational AppScan	
Acunetix	Web Vulnerability Scanner	
Univ. Coimbra	VS-WS	
Univ. Maryland	FindBugs	Static code analysis
SourceForge	Yasca	
JetBrains	IntelliJ IDEA	
Univ. Coimbra	RAD-WS	Anomaly detection

- Vulnerability Scanners: VS1, VS2, VS3, VS4
- Static code analyzers: SA1, SA2, SA3

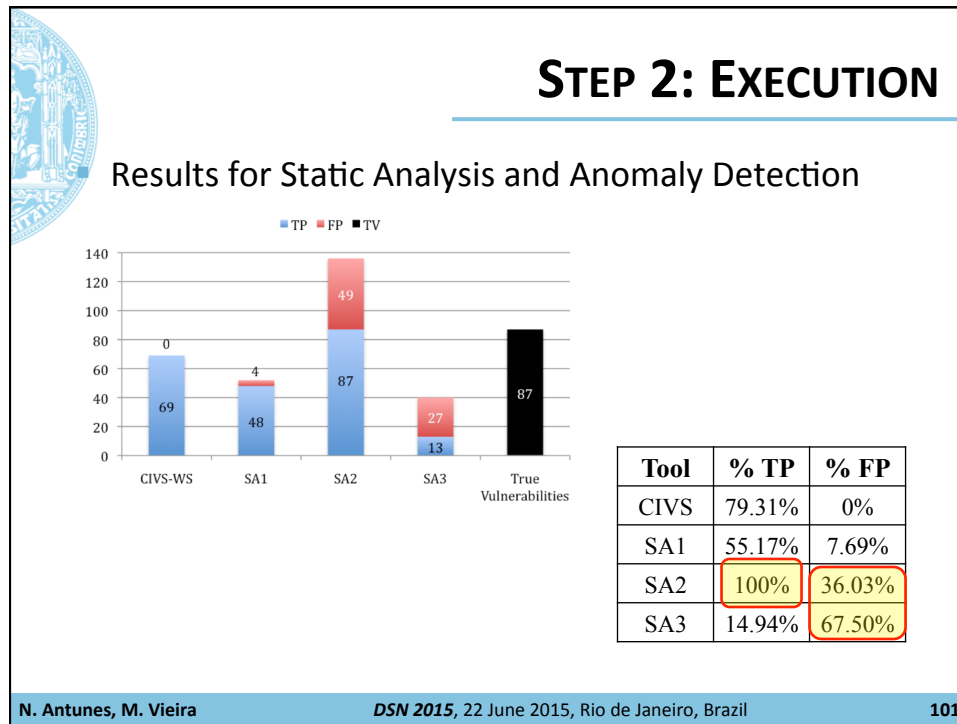
N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      99

## STEP 2: EXECUTION

Results for Penetration Testing

Tool	% TP	% FP
VS1	32.28%	54.46%
VS2	24.05%	61.22%
VS3	1.9%	0%
VS4	24.05%	43.28%

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      100




## STEP 3: MEASURES CALCULATION

### Benchmarking results

Tool	F-Measure	Precision	Recall
CIVS-WS	0.885	1	0.793
SA1	0.691	0.923	0.552
SA2	0.780	0.640	1
SA3	0.204	0.325	0.149
VS1	0.378	0.455	0.323
VS2	0.297	0.388	0.241
VS3	0.037	1	0.019
VS4	0.338	0.567	0.241

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      102




## STEP 4: RANKING AND SELECTION

Rank the tools using the measures

- Select the most effective tool

	Criteria	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Inputs	F-Measure	VS1	VS4	VS2	VS3
	Precision	VS3	VS4	VS1	VS2
	Recall	VS1	VS2/VS4		VS3
Queries	F-Measure	CIVS	SA2	SA1	SA3
	Precision	CIVS	SA1	SA2	SA3
	Recall	SA2	CIVS	SA1	SA3

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      103




## BENCHMARK PROPERTIES

Portability

- Non-intrusiveness
- Simple to use
- Repeatability
- Representativeness

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      104





## ABOUT THE TOOLS...


Not so effective even for a very well known type of vulnerability...

- More visibility improves the effectiveness
- Static analysis outperforms penetration testing in terms of coverage
  - But the reverse seems to happen regarding false positives
- How to improve the current situation?
  - Further research is need...

N. Antunes, M. Vieira


DSN 2015, 22 June 2015, Rio de Janeiro, Brazil

105



## QUESTIONS?

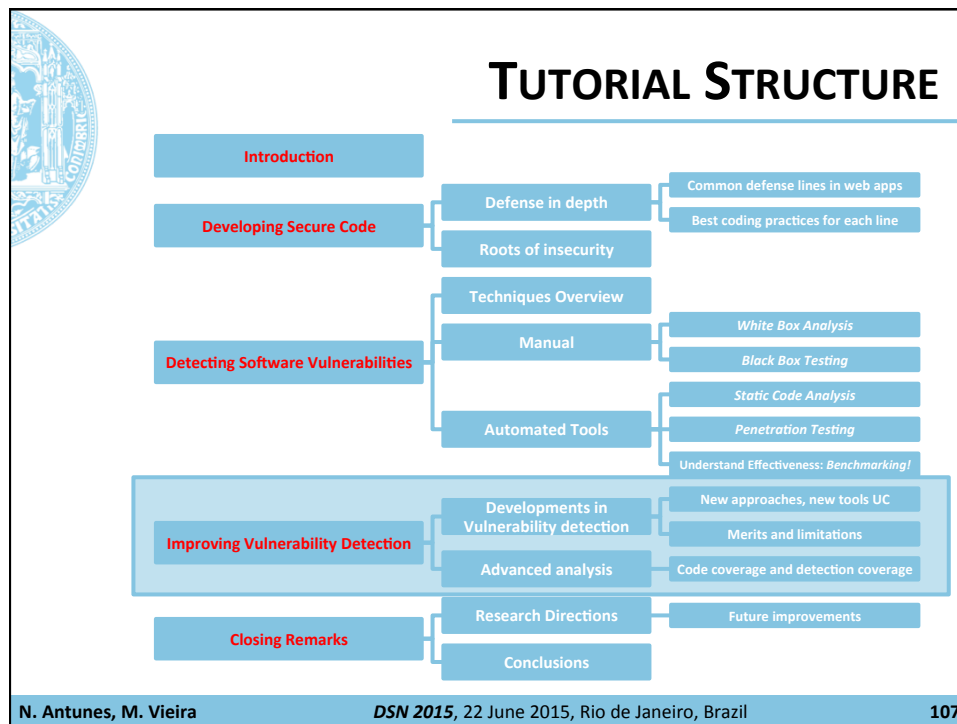
*DETECTING  
SOFTWARE  
VULNERABILITIES*



N. Antunes, M. Vieira

DSN 2015, 22 June 2015, Rio de Janeiro, Brazil


106



**REMEMBER: PENETRATION TESTING...**

- Many automated tools available
  - Including commercial and open-source
  - An automatic way to search for vulnerabilities
- ✓ Does not require access to the code
- ✗ **The Problem:** vulnerability detection can only rely on the analysis of the output
  - Effectiveness is limited by the lack of visibility on the internal behavior of the service

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 108




## HOW CAN WE OVERCOME THIS?

Add visibility to the process

- Create new vulnerability detection mechanisms
- Generate better tests
- Assess the quality of the results and of the tests
  - «*You cannot improve what you cannot measure*»


N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 109




## OUTLINE

Sign-WS

- RAD-WS
- Merits and Limitations
- Code Coverage Analysis



N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 110



## APPROACH #1: SIGN-WS

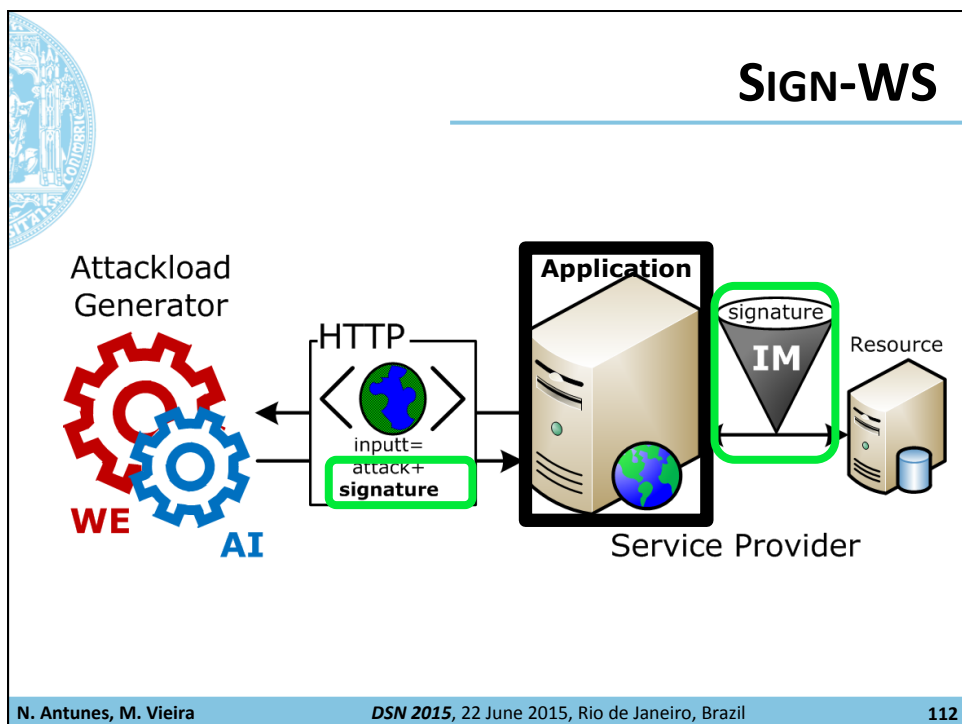
Adds visibility to the process


- Yet, keeping it as black-box as possible

- How? It uses Interface Monitoring together with Attack Signatures
  - It is possible to obtain the information necessary to improve the Penetration Testing process...
  - ... without accessing or modifying the internals of the application!!!
- **Key assumption:** injection attacks manifest in the interfaces of the attacked web service

[Antunes11]

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      111





## ATTACK SIGNATURE

Token introduced inside a injection attack


- In an successful attack the token is:
  - Observable in the interfaces of the service
  - Active: outside literal strings
  - Changing the structure of the backed command

**Active:** `Select n from t where dsc LIKE '%input' TOKEN%`

**Inactive:** `Select n from t where dsc LIKE '%input TOKEN%`

- A successful token must be:
  - Unambiguous
  - Inoffensive
  - Informative
  - Short

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 113



## PERFORM SIGNED INJECTION ATTACKS

Reversed token used to confirm vulnerabilities


- Reinforce unambiguity and avoid false positives and

- Proposed Model:
  - Delimiters “\_”
  - Identifiers
  - Qualifier “o|p”

Regular	Reversed
<code>_12_p</code>	<code>_21_o</code>

- Token must be carefully placed inside the malicious string
  - The location depends on the vulnerabilities tested
  - Must be easily configurable

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 114



## INTERFACE MONITORING


Multiple options available:

- Network packet sniffing
- Proxy
- Driver instrumentation

- Process commands to find “active” signatures
  1. Remove escaped slashes, apostrophes and quotes
  2. Remove literal strings
  3. Remaining signatures are active

1: Select n from t where dsc LIKE '%input' _28_p%';
2: Select n from t where dsc LIKE '%input' _28_p%';
3: Select n from t where dsc LIKE _28_p%';
1: Select n from t where dsc LIKE '%input\' _28_p%';
2: Select n from t where dsc LIKE '%input _28_p%';
3: Select n from t where dsc LIKE ;

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 115




## SIGN-WS: PROTOTYPE IMPLEMENTATION

Targets SQL Injection vulnerabilities

- The most common

- Uses JDBC driver instrumentation
  - Using Aspect-Oriented Programming (AOP)
  - Developed with attention to avoid introducing bugs
  - Very practical to test different systems
- For each web service operation:
  - Workload is generated combining valid values for parameters
  - Attackload is generated by mutating the workload calls
    - One parameter at a time!

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 116




## SIGN-WS: CONFIGURATION

A config. file allows defining attacks with signatures

- A set of attacks is defined in the prototype
- Examples of attacks:

' %SIGNATURE%
\ ' %SIGNATURE%
' -- %SIGNATURE%
' = ' ' or %SIGNATURE% = ' '
%WORKLOAD%' %SIGNATURE%
%WORKL_%' or %SIGNATURE%= %SIGNATURE% --
- Special placeholders
  - %SIGNATURE% - replaced by the actual signature token
  - %WORKLOAD% - replaced at runtime by the value of that input in the original work-load request
  - %WORKL\_% - only the initial WL characters are used in order to maintain the total length

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 117




## LIMITATIONS

It does not provide information on the internal behavior of the application

- It is dependent on the signatures
  - It may affect the patterns of the inputs and be intercepted by validators
- It is dependent on the quality of the tests
  - As are the other techniques

*We'll present evaluation results later...*

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 118




## APPROACH #2: RAD-WS

### Detect Command Injection in Web Services

- Structure of commands is learned in a profiling phase  
`select n from t where dsc=?INT`
- In attack phase deviations from these profiles are reported as vulnerabilities  
`select n from t where dsc=?INT or ?INT=?INT`

[Antunes09]

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      119

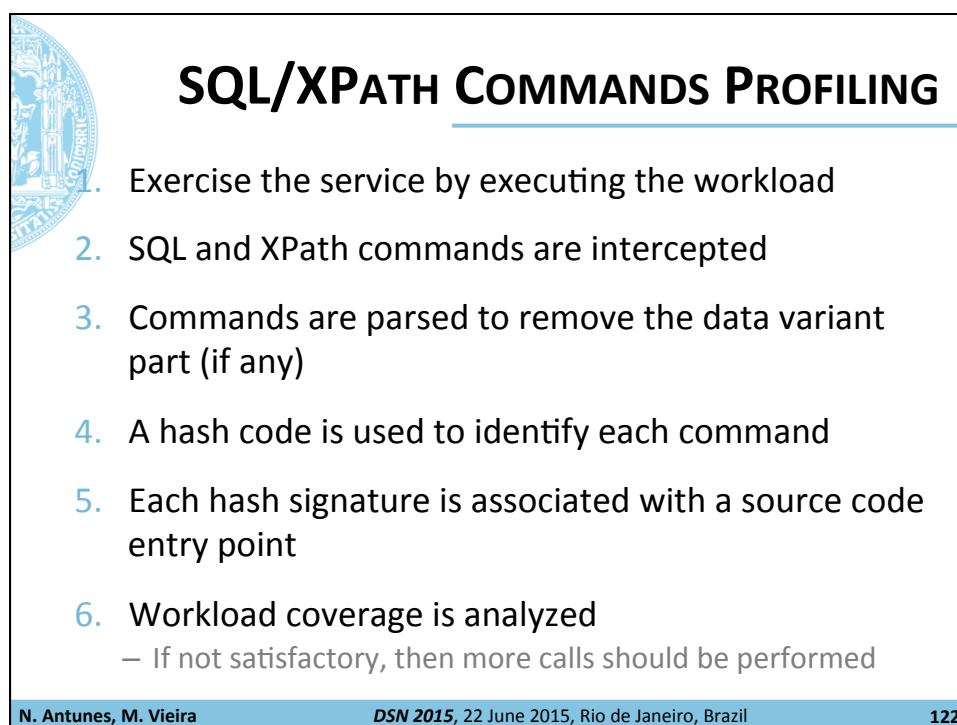
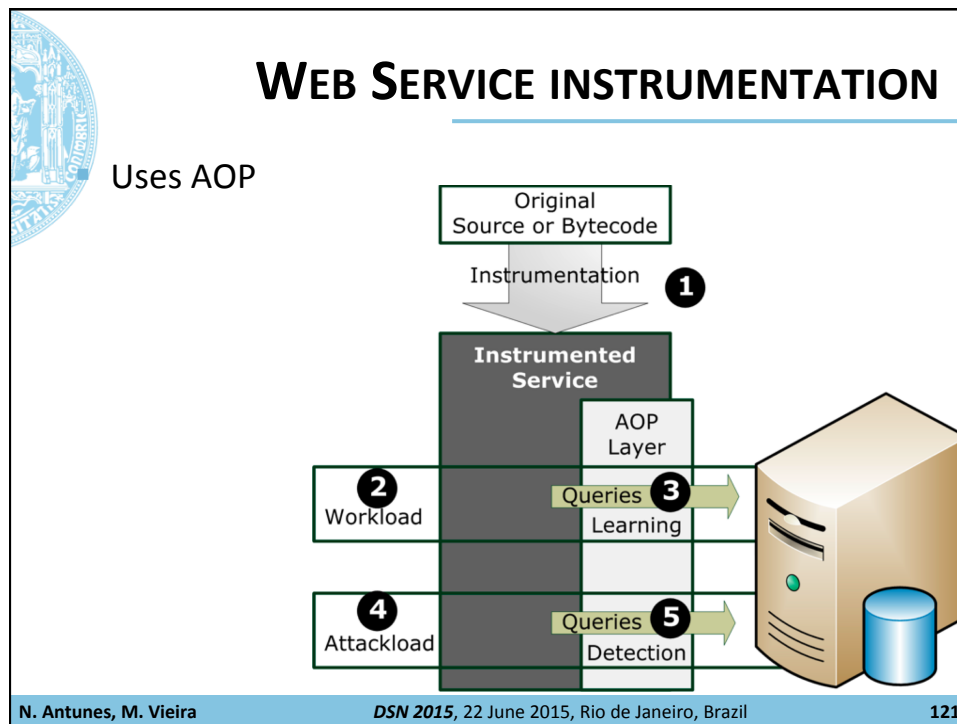



## STEPS OF THE APPROACH

- Instrument the web service to intercept all SQL/XPath commands executed
- Generate a workload
- Execute the workload to learn SQL commands and XPath queries issued
- Generate an attackload based on a large set of SQL Injection and XPath Injection attacks
- Execute the attackload to detect vulnerabilities

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      120








## VULNERABILITIES DETECTION

1. Execute the attackload and perform security checks per each data access executed
2. SQL and XPath commands are intercepted and hashed
3. The calculated hash codes are compared to the values of the learned valid commands
  - For the code point at which the command was submitted
4. If hash code is NOT found then:
  - There is a vulnerability
  - The source code location was not learned correctly

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      123




## TO SUMMARIZE

Two different approaches, two prototypes

- Sign-WS
  - Based on attack signatures and interface monitoring
  - Does not need to access the source code
- RAD-WS
  - Based on the behavior of the application
  - Finds deviations during the attack phase

*Now let's see how they perform*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      124




# IMPROVING VULNERABILITY DETECTION

---

## *MERITS AND LIMITATIONS*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      125




# EXPERIMENTAL EVALUATION

---

Prototype tool to demonstrate the approach

- Available at: <http://eden.dei.uc.pt/~mvieira>
- Experiments to assess its effectiveness
  - Detecting vulnerabilities in a set of Java-based Web Services coded by independent developers
  - Comparison with existing scanners and code analyzers

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      126




## EXPERIMENTAL EVALUATION

Used a “*Benchmark for SQL Injection Vulnerability Detection Tools*” [Antunes10]

- 21 services, 80 operations, 158 known vulnerabilities
- Compared with three commercial scanners:
  - Acunetix WVS, HP WebInspect, IBM Rational AppScan
  - Similarly to the last results presented (named VS1, VS2, VS3)
- Evaluate vulnerability detection tools based on
  - Detection coverage
    - Percentage of existing vulnerabilities detected by the tool
  - False detection rate
    - Percentage of vulnerabilities detected by the tool that do not exist
- Assess the effectiveness the proposed tools
  - Understand their merits and limitations

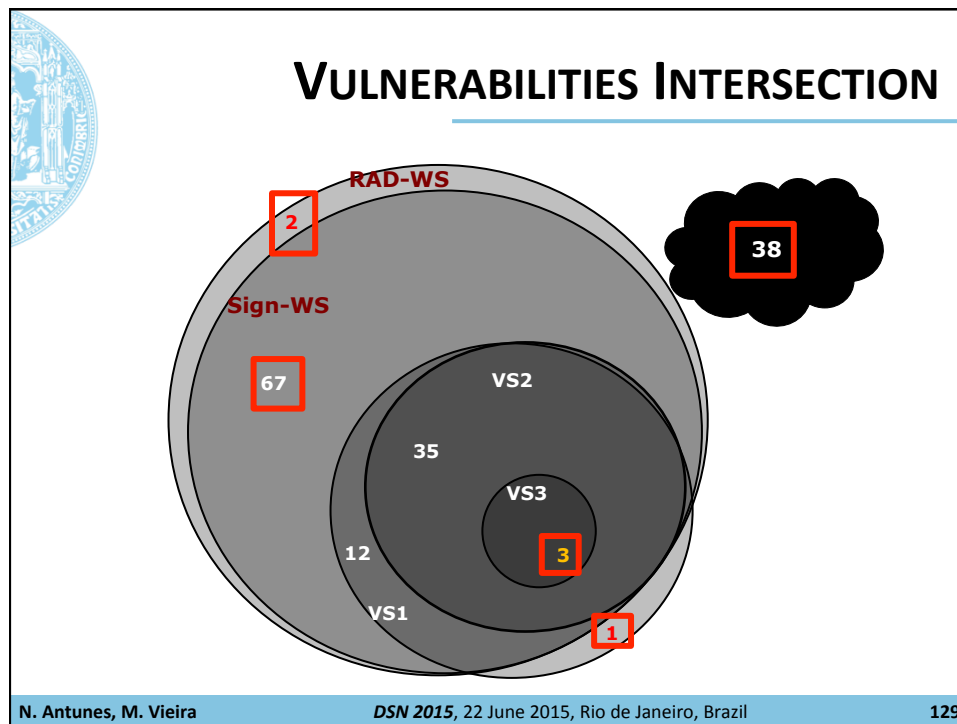
N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 127



## OVERALL RESULTS

Tool	True Vulnerabilities	True Positives	False Positives	Coverage	F. Detection Rate
RAD-WS	158	119	0	75,32%	0,00%
Sign-WS	158	117	0	74,05%	0,00%
VS1	158	51	61	32,28%	54,46%
VS2	158	38	60	24,05%	61,22%
VS3	158	3	0	1,90%	0,00%

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 128



### MERITS AND LIMITATIONS

✓ **Merit:** more effective tools


- Detect much more vulnerabilities
- Avoid reporting false positives

✗ **Limitation:** vulnerabilities left undetected

- Code coverage far from satisfactory
  - A problem for many types of testing
- Interdependent vulnerabilities
  - Iterative (detection/correction) processes are necessary


*How can we take this analysis one step further??*

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 130



## CODE COVERAGE ANALYSIS

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      131



## COVERAGE ANALYSIS


Which parts of the program were or not tested?

- Also called Testing Coverage
- Can be based on the functional specification or...
- ... based on the program internal structure:  
Code coverage

- Relation with SW Reliability is demonstrated
  - At least since the 90s [Lyu94], [Maldonado97]
  - Some use this relations to improve testing [McCabe]

**But the relation with vulnerabilities detected is yet to be confirmed or validated!**

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      132




## CODE COVERAGE

Can be measured at various granularities:

- statements, blocks, conditions, and methods, etc.

- Several tools available
  - Work with specific unit-testing libraries
  - Very limited in terms of the analyzed Criteria
- Stronger criteria: control flow, data flow
  - Extracted from CFG (control flow graph) and DUG (def-use graph)

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 133



## RESEARCH STUDY

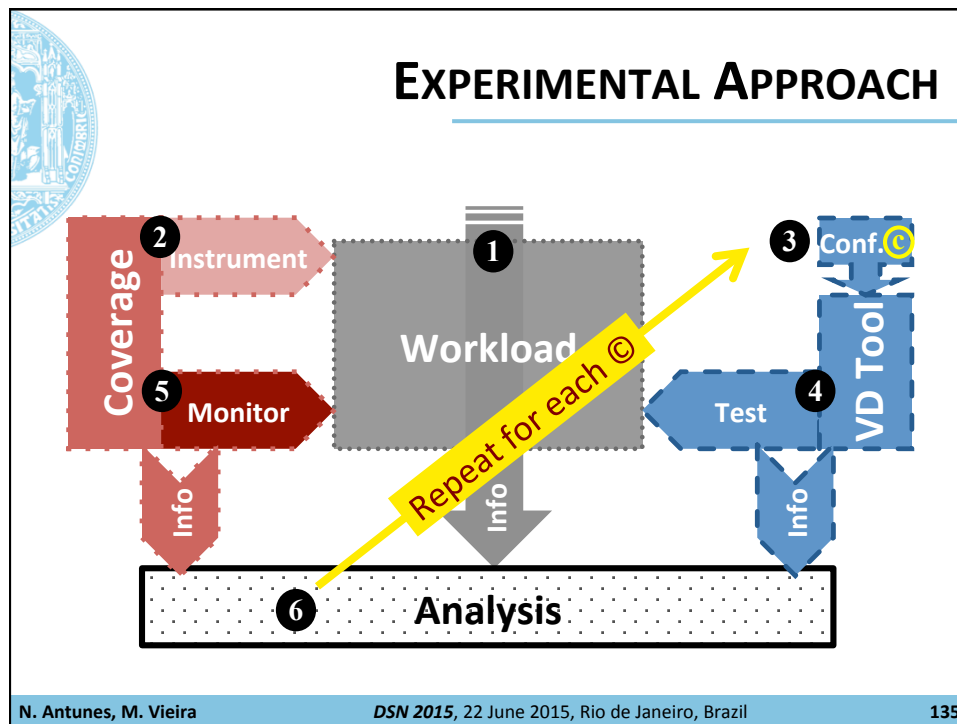
A first study on relation between code coverage and quality of vulnerability testing results

- Focused on true positives rate (TPR)
  - FPs more affected by vulnerability identification
  - SotA tools avoid FPs, but none found all vulnerabilities

**Q1.** *Is there a relation between code coverage and the number of reported vulnerabilities?*

**Q2.** *Are code coverage metrics effective to compare different vulnerability detection tests?*

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 134



## VULNERABILITY DETECTION TOOL

One tool: Sign-WS [Antunes11]

- Avoids false positives
- High TPRs, although with margin for improvement
- Effectiveness depends only on test quality


■ Four configurations:

- DL    **Many tests** of valid (...) tests based on **Better tests**
- RL    **Many tests** generated L (...) base of **Worse tests**
- DS    **Fewer tests** valid req (...) based on **Better tests**
- RS    **Fewer tests** generated S (...) base of **Worse tests**

**One variable: the quality of the testing campaign!**

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      136






## CODE COVERAGE ANALYSIS

Existing tools do not work with external testing


- Seamlessly applicable to other testing tools
  - We want web services to run independently
  - We want testing tool to test independently
    - Through the standard (and tech-independent) interface
- Apply state of the art criteria

■ Jabuti\*

- JaBUTi\* Code Instrumenter
  - Instruments the web services classes with probes
- JaBUTi\* Trace Analyzer
  - Compute metrics for control flow and data flow criteria



N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 137

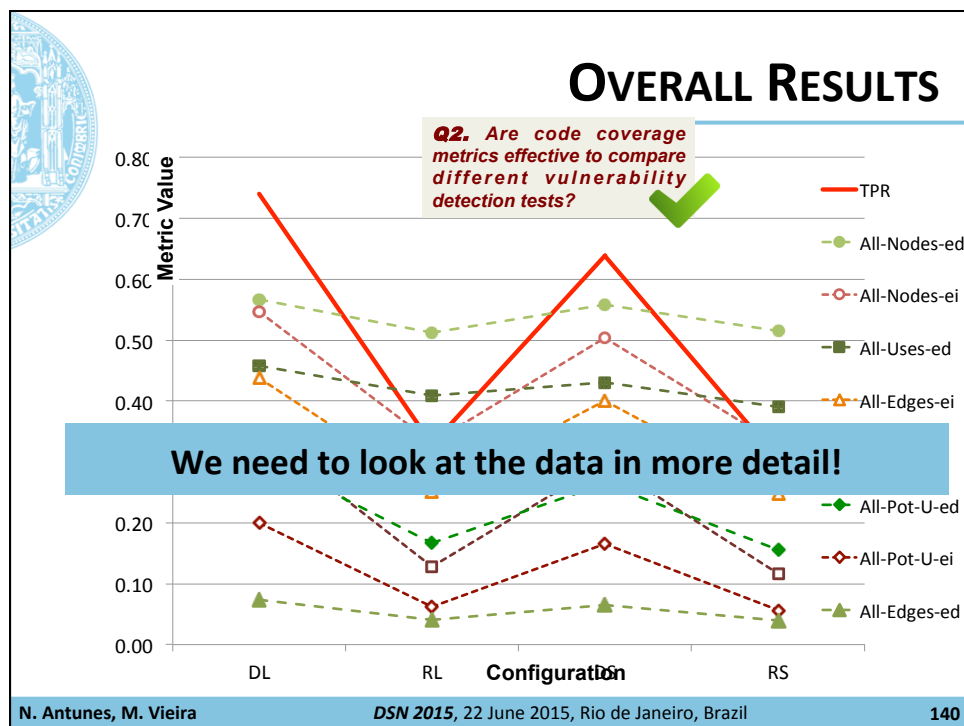
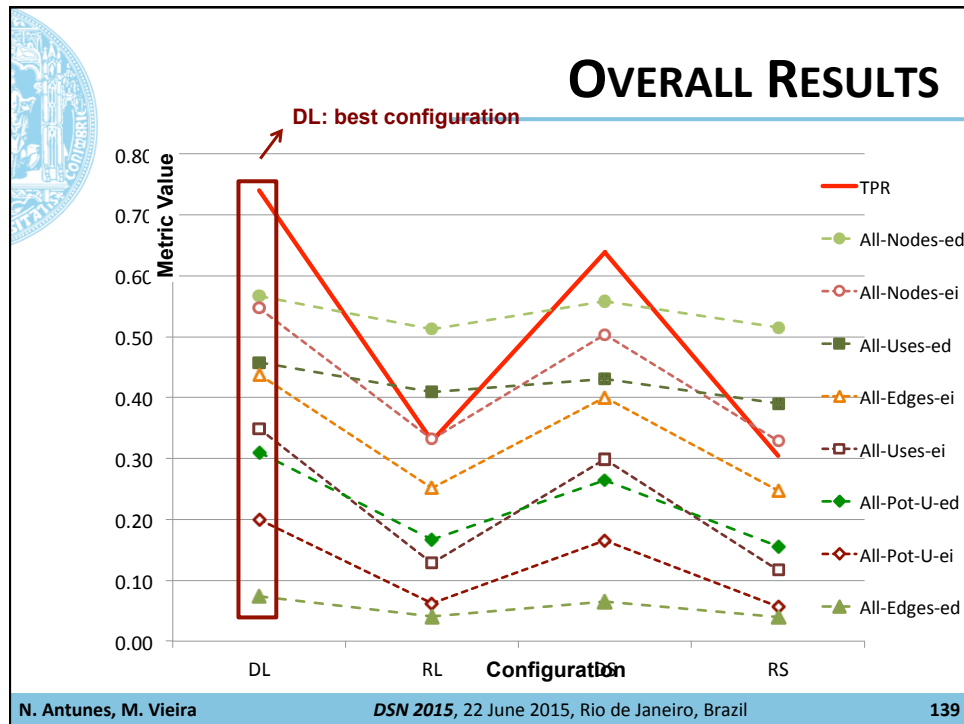


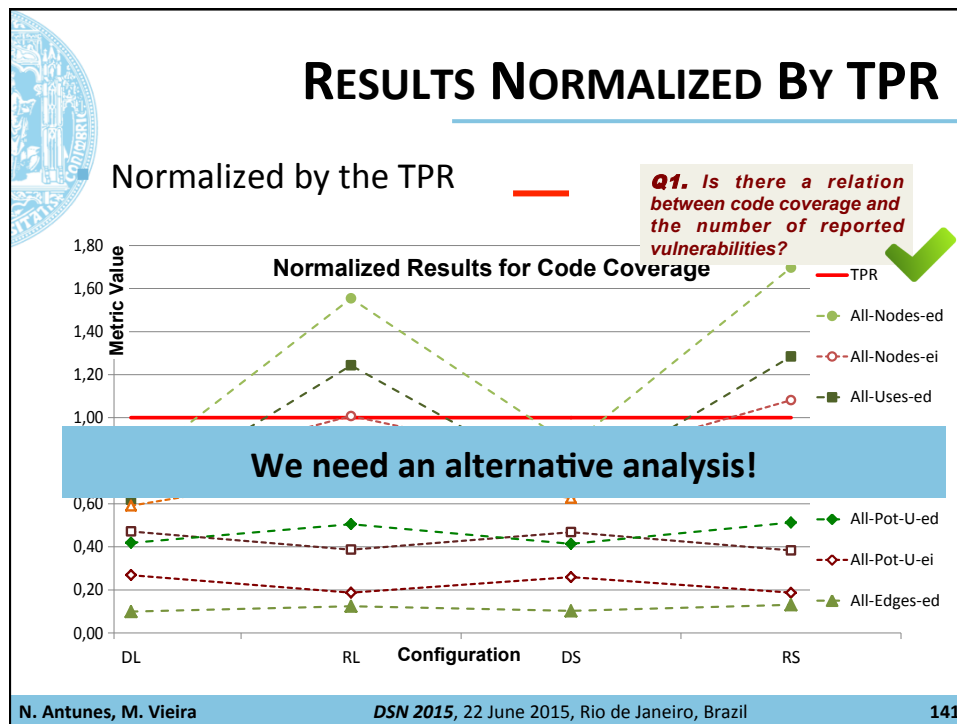
## CODE COVERAGE METRICS

Structural testing criteria

- Includes exception dependent variants (ed)
  - Originated because of exception handling mechanisms
- Ordered from the Weaker $\zeta$  to the Stronger+
- Control flow criteria [Rapps85]
  - All-Nodes-ei $\zeta$  All-Nodes-ed $\zeta$
  - All-Edges-ei All-Edges-ed
- Data flow criteria [Maldonado97]
  - All-Uses-ei All-Uses-ed
  - All-Pot-Uses-ei+ All-Pot-Uses-ed+

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 138






## CORRELATION ANALYSIS

Simple correlation analysis, for each metrics we computed the **Pearson's correlation coefficient (r)** between it and TPR and also the slope (m) of the fitted line of regression

	All-Nodes		All-Edges		All-Uses		All-Pot-Uses	
	-ei	-ed	-ei	-ed	-ei	-ed	-ei	-ed
<b>r</b>	0,9988	0,9951	0,9989	<b>0,9998</b>	<b>0,9998</b>	<b>0,9481</b>	0,9997	0,9979
<b>m</b>	0,5114	0,1246	0,4470	0,0768	<b>0,5302</b>	0,1108	0,3259	0,3356

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      142




## CORRELATION ANALYSIS

Simple correlation analysis, for each metrics we computed the **Pearson's correlation coefficient (r)** between it and TPR and also the slope (m) of the fitted line of regression

	All-Nodes		All-Edges		All-Uses		All-Pot-Uses	
	-ei	-ed	-ei	-ed	-ei	-ed	-ei	-ed
<b>r</b>	0,9988	0,9951	0,9989	<b>0,9998</b>	<b>0,9998</b>	0,9481	0,9997	0,9979
<b>m</b>	0,5114	0,1246	0,4470	0,0768	<b>0,5302</b>	0,1108	0,3259	0,3356

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 143



## TO SUMMARISE


Code coverage results presents a clear relation with the vulnerabilities reported

- Coverage metrics may be useful to compare different sets of tests
- We need better metrics to have more information
  - Potentially to estimate of the vulnerabilities left undetected

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 144

**QUESTIONS?**

*CODE COVERAGE  
ANALYSIS*




N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 145

**TUTORIAL STRUCTURE**

- Introduction
- Developing Secure Code
  - Defense in depth
    - Common defense lines in web apps
  - Roots of insecurity
    - Best coding practices for each line
- Detecting Software Vulnerabilities
  - Techniques Overview
  - Manual
    - White Box Analysis
    - Black Box Testing
  - Automated Tools
    - Static Code Analysis
    - Penetration Testing
    - Understand Effectiveness: Benchmarking!
- Improving Vulnerability Detection
  - Developments in Vulnerability detection
    - New approaches, new tools UC
    - Merits and limitations
  - Advanced analysis
    - Code coverage and detection coverage
- Closing Remarks
  - Research Directions
    - Future improvements
  - Conclusions

N. Antunes, M. Vieira DSN 2015, 22 June 2015, Rio de Janeiro, Brazil 146




## CLOSING REMARKS

---

### *RESEARCH DIRECTIONS*

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      147




## RESEARCH DIRECTIONS (1)

---

- **Applications and services are deployed with problems**
- Runtime V&V for service-based infrastructures
  - Verification and Validation
- Improve the software development process?
  - More effective testing?
  - More effective code reviews and inspections?
  - Use targeted checklists?
- Measure the trust that an application deserves?

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      148




## RESEARCH DIRECTIONS (2)

**Applications and services are deployed with vulnerabilities**

- Research vulnerability removal and attack detection mechanisms
- Extend techniques to other types of services and vulnerabilities
- What about attack detection?
  - Can we include attack detection mechanisms in the applications?

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      149




## RESEARCH DIRECTIONS (3)

**Effectiveness of detection tools is very low**

- How to improve penetration testing?
  - Increase representativeness of the workload
  - Guarantee high coverage
  - Improve the attacks performed
  - Improve the vulnerability detection algorithms
- How to improve static analysis?
  - Include new vulnerable code patterns
    - [How to identify those patterns?](#)
- Merge penetration testing and static code analysis?
  - Combining different techniques may be effective...
- Code coverage analysis to improve testing techniques

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      150




## RESEARCH DIRECTIONS (4)

### How to select the best tools to use?

- Benchmark vulnerability detection tools
- What are the best metrics to portray effectiveness of these tools?
- Use vulnerability injection to improve current benchmarks
- Use code coverage analysis

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      151




## CONCLUSIONS

We discussed the basics of vulnerabilities in web applications

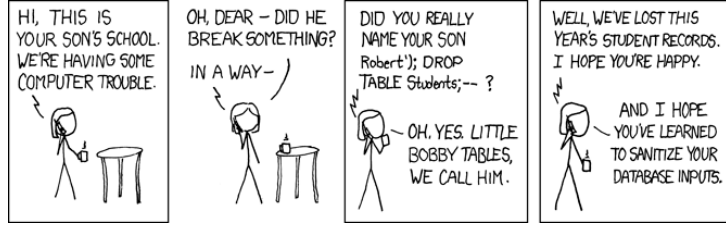
- We discussed how to protect against them and why many times this is not done properly
- We showed how some techniques can be improved to detect more vulnerabilities
- A lot of work is yet to be done to improve the current situations!

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      152





## QUESTIONS?



HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR - DID HE BREAK SOMETHING? IN A WAY--


DID YOU REALLY NAME YOUR SON Robert?); DROP TABLE Students;-- ?


OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.


<http://xkcd.com/327/>





**Nuno Antunes, Marco Vieira**  
Department of Informatics Engineering  
University of Coimbra  
[nmsa@dei.uc.pt](mailto:nmsa@dei.uc.pt) | <http://eden.dei.uc.pt/~nmsa>  
[mvieira@dei.uc.pt](mailto:mvieira@dei.uc.pt) | <http://eden.dei.uc.pt/~mvieira>

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      153



## REFERENCES

[Antunes09] N. Antunes, N. Laranjeiro, M. Vieira, and H. Madeira, "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services," in IEEE International Conference on Services Computing (SCC 2009), Bangalore, India, 2009, pp. 260–267.

[Antunes11] N. Antunes and M. Vieira, "Enhancing Penetration Testing with Attack Signatures and Interface Monitoring for the Detection of Injection Vulnerabilities in Web Services," in IEEE International Conference on Services Computing (SCC), 2011, pp. 104–111.

[Antunes15] N. Antunes and M. Vieira, "Assessing and Comparing Vulnerability Detection Tools for Web Services: Benchmarking Approach and Examples", *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 269–283, 2015.

[Howard02] M. Howard and D. E. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, WA: Microsoft Press, 2002.

[Lyu94] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Computer*, vol. 27, no. 9, pp. 60–69, 1994.

[McCabe] McCabe Software, Inc., "Combining McCabe IQ with Fuzz Testing," 2009.

[Maldonado97] A. N. Crespo, A. Pasquini, M. Jino, and J. C. Maldonado, "Cobertura dos critérios potenciais-usos e a confiabilidade do software," 9th SBES, pp. 379–394, 1997.

[Neves06] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using Attack Injection to Discover New Vulnerabilities," in International Conference on Dependable Systems and Networks, 2006. DSN 2006, 2006, pp. 457–466.

[Stock07] A. Stock, J. Williams, and D. Wichers, "OWASP Top 10," OWASP, 2007.

[Stuttard07] D. Stuttard and M. Pinto, *The web application hacker's handbook: discovering and exploiting security flaws*. Wiley Publishing, Inc., 2007.

N. Antunes, M. Vieira      DSN 2015, 22 June 2015, Rio de Janeiro, Brazil      154