Defending against Web Application Vulnerabilities

Nuno Antunes
nmsa@dei.uc.pt
University of Coimbra – Portugal

Nuno Antunes...

- From Coimbra, Portugal 😊
  - Marco Vieira will present it properly

- BSc and MSc in Informatics Engineering

- Researcher with CISUC since 2008
  - Software and Systems Engineering group

- Ph. D. student since 2009
  - Supervised by Prof. Marco Vieira
  - It’s time to finish it... 😊
Research Topics

- Dependable and Secure Software
- Vulnerability Detection
- Security Vulnerabilities
- Service Oriented Architectures
- Benchmarking
- Verification and Validation
- Intrusion Detection Systems

Vulnerability Detection

- [Antunes09b, Antunes09c, Antunes11]
- Improving Vulnerability Detection tools
- Focusing on Service Oriented Architectures
- Focusing on Injection Vulnerabilities

- Many of the concepts can be applied to other applications and vulnerabilities

...that’s what we’ll talk about today!
Benchmarking Vulnerability Detection Tools

- Proposed an approach to benchmark the effectiveness of V. D. tools in web services
  - [Antunes10]
  - Inspired in measurement-based techniques
  - Procedures and measures were specified

- A concrete benchmark was implemented
  - Targeting tools able to detect SQL Injection
  - A benchmarking example was conducted

- Results show that the benchmark can be used to assess and compare different tools

Intrusion Detection Systems Benchmarking

- Starting with **IDS** for Cloud Environments
  - Applying vulnerability and attack injection to that domain
  - Working on Security- and performance-related IDS benchmarking metrics

- Working Group of **SPEC Research**
- In collaboration with:
  - **Karlsruhe Institute of Technology**, Germany
  - **SIEMENS Research**, USA
  - **SESM S.c.a.r.l.**, Italy
Runtime V&V in SOAs

- **SOAs**
  - Complex and Highly dynamic
  - Multiple services deployed, interconnected and updated
  - Services outside control

- **V&V in Critical Applications**
  - Detailed checking of a system prior to its deployment

- **Very important problem in Business-Critical**
  - Cristina will present in future secondments 😊
    - For now, the work is just starting!

CECRIS & PROLAN

- **Industry-Academia Partnerships and Pathways**
  - Transfer of Knowledge!

- **In Prolan as Seconded Researcher**
  - 2 weeks now (June 2013)
  - A lot to plan for the future

- **Secondment of 2 weeks in late 2013**
- **Secondment of 2 months in 2014**
Outline

- What are code vulnerabilities?
- Developing Secure Code
- Detecting Vulnerabilities
  - Case study: How effective are vulnerability detection tools?
  - Can we do better?
- Detecting Attacks
- Conclusions
What is a vulnerability?

- **A fault** that leave space for malicious exploitation of a system
  - 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
  - ...

Exploiting Vulnerabilities (1)

**US man 'stole 130m card numbers'**

Officials say it is the biggest case of identity theft in American history.
They say Albert Gonzalez, 28, and two un-named Russian co-conspirators hacked into the payment systems of retailers, including the 7-Eleven chain.

Prosecutors say they aimed to sell the data on. If convicted, Mr. Gonzalez faces up to 20 years in jail for wire fraud and five years for conspiracy.
He would also have to pay a fine of $250,000 (£150,000) for each of the two charges.

**'Standard' attack**

Mr. Gonzalez used a technique known as an "SQL injection attack" to access the databases and steal information, the US Department of Justice (DoJ) said.

**SQL INJECTION ATTACK**

- This is a fairly common way that fraudsters try to gain access to consumers’ card details.
- They scour the internet for vulnerabilities.
- They use the vulnerable software to execute theirSQL injection attack.
- They steal credit card numbers and sell them on the black market.

...
Exploiting Vulnerabilities (2)

Exploiting Vulnerabilities (3)

News

Hacker Defaces Microsoft U.K. Web Page

By Keith Ward • June 29, 2007

A hacker manages a rare feat Wednesday, successfully attacking a Web page within Microsoft's U.K. domain and replacing the page with several graphics related to Saudi Arabia.

The hacked page was a U.K. events page here. It has since been fixed. According to the security site Zone-h, a SQL injection attack is the likely culprit. Zone-h reported the hack methodology: "Most probably, the attacker exploited the site by means of SQL injection to insert the HTML code " in a field belonging to the table which gets read every time a new page is generated." This would work on a page utilizing Microsoft's SQL Server.

The defaced page had three images: a child waving a green and white flag of Saudi Arabia, a woman with a green scarf over her face and a stand-alone image of the Saudi Arabian flag. Beneath the flag is a message that reads "HACKED BY 1Em0Er".

U.K. website The Register quoted a Microsoft spokesperson as saying that although the attack was embarrassing, it didn't appear to be serious. "There is no reason to believe customer data or any other sensitive information has been compromised."
Web Applications and Security

- Are **widely exposed** to attacks
- Any existing vulnerability will most probably be uncovered/exploited
- Hackers moved the focus to applications’ **code**
  - Use specially tampered values
  - Take advantage of improperly coded applications
- What about Firewalls, Encryption and IDSs?
  - The attacker uses the inputs of the application
  - The ports used are the same for regular web traffic
- **...thus, can’t solve the problem!**

Developing Teams

- Developers should
  - Apply best security practices
    - Don’t forget: the presence of security features doesn’t imply that the system is secure
  - Perform security testing
- Security concerns are often disregarded
  - Focus on satisfying customer’s requirements
  - Hard time-to-market constraints
  - Not specialized on security
- **Automated tools have a fundamental role**
  - Increase productivity
  - Help developers that are not specialized in security
Vulnerabilities in Web Applications (1)

- Buffer Overflow
  - **Very dangerous** vulnerability in any application written in languages as C/C++
  - It is possible to manipulate inputs in such a way that causes buffer allocation problems
  - An attacker can exploit this causing Denial of Service
  - In worst cases, alter application flow forcing unintended actions
  - Lost importance with the advent of languages running in VMs, because boundaries are enforced
    - Java, C#, python...

Vulnerabilities in Web Applications (2)

- 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

- XSS
  - It is possible to inject client-side script into web pages viewed by other users
  - Can be used by attackers to bypass access controls
Vulnerabilities in Web Applications (3)

- **Injection**
  - **Very dangerous** vulnerability when the application uses external resources
    - Databases, XML, OS, etc.
  - It is possible to alter the construction of backend commands or queries
  - An attacker can trick the interpreter into executing unintended commands or accessing unauthorized data
  - Classical example: **SQL Injection**

- Our focus... later we’ll understand why!

Example of SQL Injection Vulnerability

```java
String sql = "SELECT * FROM users WHERE " + 
  "username='" + Email + "' AND " + 
  "password='" + Password + "'";

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

… that is not getting better!

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

- According to WhiteHat Security Website Security Statistics Report, *63%* of assessed websites are vulnerable

- Something is *wrong* in the development of web applications!
The Solution

- "...a defense-in-depth approach, with overlapping protections, can help secure Web applications" [Howard02]
Developing Secure Code

- The characteristics of Web applications suggest the use of three distinct lines of defense:
  - Input validation
  - Hotspot protection
  - Output validation

Input Validation

- Reduce an application’s input domain
- All inputs are malicious until proven otherwise

- Starts with normalization of the inputs
- Uses filtering strategies to reject values outside the domain

- Domain can allow malicious data:
  - e.g. ‘ in the case of SQL injection
Hotspot Protection

- Each type of attack targets a hotspot:
  - Hotspot: a set of statements that is prone to specific types of vulnerabilities.

- This line of defense focuses on protecting only key hotspots

- e.g. SQL injection attacks use quotes (' or `):
  - Character Escaping
    - Data length problems 😐
    - Parameterized commands/queries ✔

Output Validation

- Prevents users from receiving restricted information as:
  - Internal Exceptions that can lead to other attacks
  - Credit card numbers

- Encoding is a example of output validation
  - Avoids XSS vulnerabilities
Why don’t developers follow these practices?

- Training and education
- Security is boring and uninteresting
- Someone else should “take care” of security
- Security “limits” application functionality

Outline Revisited

- What are code vulnerabilities?
- Developing Secure Code
- Detecting Vulnerabilities
  - Case study: How effective are vulnerability detection tools?
  - Can we do better?
- Detecting Attacks
- Conclusions
Detecting Vulnerabilities

- Penetration testing
- Static code analysis
- Limitations of Vulnerability Detection
Penetration testing

- A specialization of Robustness Testing
  - Analyzes the program execution in the presence of malicious inputs, searching for vulnerabilities.
- Widely used by developers
- Does NOT require access to the source code or bytecode
- Can be performed manually or automatically

Penetration Testing

- Consists in stressing the application from the point of view of an attacker
  - “black-box” approach
  - Uses specific malicious inputs
    - e.g., for SQL Injection: ‘ or 1=1
- Automated tools provide an automatic way to search for vulnerabilities
  - Avoid a large number of manual tests
  - Ignore the internals of the application
Examples of penetration testing tools

- HP WebInspect™
- WSDigger
- WSFuzzer

Static code analysis

- An white-box approach
- Consists in analyzing the source code of the application
  - Without execution it
- Does require access to the source code (or bytecode)
- Can be performed manually or automatically
Static code analysis

- Looks for potential vulnerabilities
  - Among other types of software defects

- Automated tools provide an automatic way for highlighting possible coding errors

  x Ignore the runtime perspective

Examples of static analysis tools

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

Nuno Antunes
Prolan Zrt., Budapest, 19th June, 2013
Using vulnerability detection tools…

- Tools are often expensive
- Many tools can generate conflicting results
- Due to time constraints or resource limitations
  - Developers have to select a tool from the set of tools available
  - Rely on that tool to detect vulnerabilities
- However...
  - Previous work shows that the effectiveness of many of these tools is low

**How effective are V. D. Tools?**

Experimental study [Antunes9a]

- Evaluate several automatic penetration testing tools and static analysis tools
  - In a controlled environment
- Focus on two key measures of interest:
  - Coverage
    - Portrays the percentage of existing vulnerabilities that are detected by a given tool
  - False positives
    - Represents the number of reported vulnerabilities that in fact do not exist
- Target only SQL Injection vulnerabilities
  - Extremely relevant in Web Services
Web Services tested

<table>
<thead>
<tr>
<th>Service</th>
<th>Short Description</th>
<th>#Op</th>
<th>LoC</th>
<th>LoC/Op</th>
<th>Avg. C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDetail</td>
<td>Get details about a product</td>
<td>1</td>
<td>105</td>
<td>105.0</td>
<td>6.0</td>
</tr>
<tr>
<td>NewProducts</td>
<td>Add new product to the database</td>
<td>1</td>
<td>136</td>
<td>136.0</td>
<td>6.0</td>
</tr>
<tr>
<td>NewCustomer</td>
<td>Add new customer to the database</td>
<td>1</td>
<td>184</td>
<td>184.0</td>
<td>9.0</td>
</tr>
<tr>
<td>Change/Payment Method</td>
<td>Change customer’s payment method</td>
<td>1</td>
<td>97</td>
<td>97.0</td>
<td>11.0</td>
</tr>
<tr>
<td>JamesSmith</td>
<td>Manages personal data about students</td>
<td>5</td>
<td>270</td>
<td>54.0</td>
<td>6.0</td>
</tr>
<tr>
<td>PhoneDir</td>
<td>Phone book</td>
<td>5</td>
<td>132</td>
<td>26.4</td>
<td>2.8</td>
</tr>
<tr>
<td>Bank</td>
<td>Manages bank operations</td>
<td>5</td>
<td>175</td>
<td>35.0</td>
<td>3.4</td>
</tr>
<tr>
<td>Bank3</td>
<td>Manages bank operations (different from the Bank service)</td>
<td>6</td>
<td>377</td>
<td>62.8</td>
<td>9.0</td>
</tr>
</tbody>
</table>

Tools studied

- Penetration testing
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner

- Static code analysis
  - FindBugs
  - Yasca
  - IntelliJ IDEA

- Decided not to mention the brand of the tools to assure neutrality
  - VS1, VS2, VS3 (without any order in particular)
  - SA1, SA2, SA3 (without any order in particular)
Web Services manual inspection

<table>
<thead>
<tr>
<th>Service</th>
<th>#Vuln. Inputs</th>
<th>#Vuln. Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDetail</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NewProducts</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NewCustomer</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>ChangePaymentMethod</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>JamesSmith</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>PhoneDir</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Bank</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bank3</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>61</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>

Penetration testing results

<table>
<thead>
<tr>
<th>Tool</th>
<th>% F. P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>14.0%</td>
</tr>
<tr>
<td>VS2</td>
<td>4.0%</td>
</tr>
<tr>
<td>VS3</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>50.8%</td>
</tr>
<tr>
<td>VS2</td>
<td>36.1%</td>
</tr>
<tr>
<td>VS3</td>
<td>9.8%</td>
</tr>
</tbody>
</table>
Static code analysis results

<table>
<thead>
<tr>
<th>Tool</th>
<th>% F. P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>23.3%</td>
</tr>
<tr>
<td>SA2</td>
<td>26.3%</td>
</tr>
<tr>
<td>SA3</td>
<td>26.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>82.1%</td>
</tr>
<tr>
<td>SA2</td>
<td>100.0%</td>
</tr>
<tr>
<td>SA3</td>
<td>39.3%</td>
</tr>
</tbody>
</table>

Penetration testing vs Static analysis (1)  

- Coverage
Penetration testing vs Static analysis (2)

- False positives

Key observations

- The coverage of static code analysis is typically higher than of penetration testing
- False positives are a problem for both approaches
  - But have more impact in the case of static analysis
- Different tools report different vulnerabilities in the same piece of code
  - Even tools implementing the same approach frequently
- Poor results!
  - Can’t we do better?
Can we do better?

- **Focus on Service Based Infrastructures**
  - Complex and Highly dynamic
  - Services being deployed, interconnected and updated
  - Services under control of multiple Providers
  - Costumers also need to test

Yes, we can!!

- **New vulnerability detection tools**
  - Penetration Testing
    - [Antunes09b]
  - Attack Signatures & Interface Monitoring
  - Runtime Anomaly Detection
    - [Antunes09c]
  - Integrated Approach
    - [Antunes13]
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

What is necessary to overcome this?

- Add visibility to the process
  - Yet, keep it as black-box as possible

- Solution: Use 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
Attack Signatures & Interface Monitoring

A 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
Perform Signed Injection Attacks

- Reversed token used to confirm vulnerabilities
  - Reinforce **unambiguity** and avoid false positives and

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

<table>
<thead>
<tr>
<th>Regular</th>
<th>Reversed</th>
</tr>
</thead>
<tbody>
<tr>
<td>_12_p</td>
<td>_21_o</td>
</tr>
</tbody>
</table>

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

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 ; |
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!

Sign-WS: Configuration

- A configuration file allow the definition of attacks with signatures
- A set of attacks is defined in the prototype
- Examples of attacks:
- 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
Experimental Evaluation

- Used a “Benchmark for SQL Injection Vulnerability Detection Tools” [Antunes10]
  - 21 services, 80 operations, 158 known vulnerabilities

- Two well-known metrics were considered:
  - **detection coverage**: percentage of existing vulnerabilities detected by the tool
  - **false positives rate**: percentage of vulnerabilities detected by the tool but that do not exist

- Compared with three commercial scanners:
  - Acunetix Web Vulnerability Scanner, HP WebInspect, IBM Rational AppScan
  - Named VS1, VS2, VS3

Experimental Results

<table>
<thead>
<tr>
<th>Tool</th>
<th>Detection Coverage</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign-WS</td>
<td>74.05%</td>
<td>0.00%</td>
</tr>
<tr>
<td>VS1</td>
<td>32.28%</td>
<td>54.40%</td>
</tr>
<tr>
<td>VS2</td>
<td>24.05%</td>
<td>61.22%</td>
</tr>
<tr>
<td>VS3</td>
<td>1.90%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Results (2)

Security in Software Development Lifecycle

- Initialization
- Specification and Design
- Implementation
- Testing
- Decommissioning
- Deployment
Detecting Attacks

- Consists of identifying deviations from the correct behavior
  - At runtime

- **Anomaly detection** tools usually require a training phase with non-malicious requests

- **Signature-based** tools look for patterns of a predefined set of rules or signatures

Limitations of Attack Detection [Elia10]

- Tools only perform well in specific scenarios:
  - Anomaly-detection better for simpler applications
  - Signature-based better for complex applications

- Achieve low detection coverage
  - Less than 20 percent in many cases

- Report many false alarms
  - As high as 50 percent of the alarms generated

- Developers often lack the training required to create adequate configurations
Conclusions

- Web Applications and Services are many times deployed with vulnerabilities
- Developers must always consider security:
  - Use best development practices
  - Preform security assessment
  - Use attack-detection systems
- Developers need help with training and tools
  - But the existing tools are very limited
- Researchers should propose new tools:
  - New and improved tools
  - Possibly compile-time fixing of vulnerabilities?

Questions

More about this in:
http://eden.dei.uc.pt/~nmsa
References