The Devils Behind Web Application Vulnerabilities

“Defending against Web Application Vulnerabilities”
IEEE Computer, February 2012

Nuno Antunes, Marco Vieira
{\text{nmsa, mvieira}@dei.uc.pt}

Software and Systems Engineering
Centre for Informatics and Systems of the University of Coimbra
Outline

- Background and Motivation
- Developing Secure Code
- Detecting Vulnerabilities
- Detecting Attacks
- What are we doing about it?
- Conclusions
Security in Web Applications
Security Vulnerabilities...

- **Faults** that leave space to an exploitation or a corruption of a system

- Web applications are widely exposed

- Hackers moved their focus from the network to application’s code

- Injection and Cross-Site Scripting (XSS) are the two most common vulnerabilities
... are 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...
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]
Security in Software Development Lifecycle

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

- White-box analysis
- Black-box testing
- Limitations of Vulnerability Detection
White-Box Analysis

- Analyze the code without actually executing it
- Looks for potential vulnerabilities
  - Among other types of software defects
- Requires access to the source code or bytecode
- Automated tools provide an automatic way for highlighting possible coding errors

✗ Ignore the runtime perspective
Black-Box Testing

- A specialization of Robustness Testing
  - Analyzes the program execution in the presence of **malicious** inputs, searching for vulnerabilities.

- **Does NOT** require access to the source code or bytecode

- Automated tools provide an automatic way to search for vulnerabilities
  - Avoid a large number of manual tests

- Ignore the internals of the application
Limitations of Vulnerability Detection

- [Antunes09a]

**% Coverage**

**% False Positives**
Detecting Attacks

- Consists of identifying deviations from the correct behavior
  - In 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
What are we doing about it?

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

- **Benchmarking vulnerability detection tools**

- **Focused in Web Services**
Penetration Testing does not require access to the code

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

**Solution:** Using 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!
## 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>54.46%</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>
A new Runtime Anomaly Detection Approach

To find SQL/XPath Injection Vulnerabilities

Combine the analysis of services responses with the analysis of the runtime behavior

Two phases: Profiling and Detection

Vulnerabilities are identified by comparing the structure of commands executed in the presence of attacks to the ones learned in the absence of attacks
Experimental Evaluation

Using the same set of Web Services

Penetration testing

<table>
<thead>
<tr>
<th>Tool</th>
<th>False Positives %</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>14%</td>
<td>47.7%</td>
</tr>
<tr>
<td>VS2</td>
<td>4%</td>
<td>33.8%</td>
</tr>
<tr>
<td>VS3</td>
<td>0%</td>
<td>9.2%</td>
</tr>
<tr>
<td>CIVS-WS</td>
<td>0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

Static Code Analysis

<table>
<thead>
<tr>
<th>Tool</th>
<th>False Positives %</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>23%</td>
<td>82.1%</td>
</tr>
<tr>
<td>SA2</td>
<td>26%</td>
<td>100.0%</td>
</tr>
<tr>
<td>SA3</td>
<td>27%</td>
<td>39.3%</td>
</tr>
<tr>
<td>CIVS-WS</td>
<td>0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>
Benchmarking Vuln. Detection Tools

- [Antunes10]
- Proposed an approach to benchmark the effectiveness of V. D. tools in web services
  - 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
Benchmarking Vuln. Detection Tools

Results for CIVS-WS and static analysis

Benchmarked Tools Ranking

<table>
<thead>
<tr>
<th>Criteria</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F-Measure</td>
<td>VS1</td>
<td>VS4</td>
<td>VS2</td>
<td>VS3</td>
</tr>
<tr>
<td>Precision</td>
<td>VS3</td>
<td>VS4</td>
<td>VS1</td>
<td>VS2</td>
</tr>
<tr>
<td>Recall</td>
<td>VS1</td>
<td>VS2/VS4</td>
<td>VS3</td>
<td></td>
</tr>
<tr>
<td>Queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F-Measure</td>
<td>CIVS</td>
<td>SA2</td>
<td>SA1</td>
<td>SA3</td>
</tr>
<tr>
<td>Precision</td>
<td>CIVS</td>
<td>SA1</td>
<td>SA2</td>
<td>SA3</td>
</tr>
<tr>
<td>Recall</td>
<td>SA2</td>
<td>CIVS</td>
<td>SA1</td>
<td>SA3</td>
</tr>
</tbody>
</table>
Conclusions

- Developers must always consider security:
  - Use best practices in coding;
  - security testing;
  - use attack-detection systems;

- Developers need help with training and the tools

- Researchers should propose new tools:
  - New security testing tools
  - Possibly compile-time fixing of vulnerabilities
Questions

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

Nuno Antunes
Department of Informatics Engineering
University of Coimbra
nmsa@dei.uc.pt
References


