Notes From The field

OWASP tools and usage experiences

Jarkko Holappa
Antti Laulajainen

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

The OWASP Foundation
http://www.owasp.org
The OWASP Tools Project's goal is to provide unbiased, practical information and guidance about application security tools.

Tools are to help:

- Protect Applications
- Find Vulnerabilities in Applications
- Test Other Application Security Tools
- Educate People on Application Security Topics

Categories of tools:

- Web Application Firewalls (WAFs)
- Application Vulnerability Scanning Tools
- Application Penetration Testing Tools
- Source Code Analysis Tools
- Test and Educational Applications
- Application Security Analysis Support Tools
Tools from OWASP

- OWASP .NET Project
- OWASP AntiSamy Project
- OWASP CAL9000 Project
- OWASP DirBuster Project
- OWASP Encoding Project
- OWASP Flash Security Project
- OWASP Insecure Web App Project
- OWASP Interceptor Project
- OWASP JBroFuzz
- OWASP LAPSE Project

- OWASP Live CD Project
- OWASP Orizon Project
- OWASP Pantera Web Assessment Studio Project
- OWASP SQLiX Project
- OWASP Sprajax Project
- OWASP Validation Project
- OWASP WSFuzzer Project
- OWASP WebGoat Project
- OWASP WebScarab Project
OWASP: CAL9000

- CAL9000 is a collection of web application security testing tools that complement the feature set of current web proxies and automated scanners.
- CAL9000 is written in Javascript
- Features e.g.
  - XSS Attacks
  - Character Encoder/Decoder
  - HTTP Requests - Manually craft and send HTTP requests to servers.
  - String Generator - Create character strings of almost any length.
  - Testing Tips - Collection of testing ideas for assessments.
  - Testing Checklist - Track the progress of your testing efforts and record your findings.
  - AutoAttack Editor - Create/edit/save/delete the AutoAttack Lists that are used to drive the automated multiple-request capabilities on the HTTP Requests page.
  - Store/Restore - Temporarily hold and retrieve textarea and text field contents.
CAL9000 Demo
OWASP: WebScarab

- WebScarab is a framework for analysing applications that communicate using the HTTP and HTTPS protocols.
- It is written in Java
- WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser.
- WebScarab is able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have passed through WebScarab.
<table>
<thead>
<tr>
<th>ID</th>
<th>Date</th>
<th>Method</th>
<th>Host</th>
<th>Path</th>
<th>Parameters</th>
<th>Status</th>
<th>Origin</th>
<th>Referrer-Header</th>
<th>Comments</th>
<th>Scripts</th>
</tr>
</thead>
</table>
**OWASP**

---

**HTTP Response**

**Method:** GET

**URL:** http://www.owasp.org/index.php/Main_Page

**Version:** HTTP/1.1

**Header** | **Value**
--- | ---
Accept | *
Referer | http://www.owasp.org/index.php/Main_Page
Accept-Language | en
User-Agent | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727; .NET CLR 1.1.4322; .NET CLR 3.0.4506.30; .NET CLR 3.0.4506.84; InfoPath 1)
Connection | Keep-Alive
Host | www.owasp.org

---

**Message**

**HTTP/1.1 200 OK**

**Header**

---

**Text**

This code is generated from an HTML file and contains no visible text. The code format suggests it might be related to a task management or workflow system, possibly using some proprietary or custom HTML/CSS for styling.

---

**For some reason it does not inherit the padding top into every div,**

```html
#column-content { margin-top: 12.2em; margin-top: 3em; height: 1%; } 
```

**Problematic**

```html
/* For some reason it does not inherit the padding-top into every div, */
/* and I can't figure out how to get it back off */
/* Margin works correctly for this use, though. */
/* padding-top: 6; margin-top: 16px; */
```

---

**For the future**

```html
#actions { z-index: 3; } 
```
WebScarab - example

Try to purchase the HDTV for less than the purchase price, if you have not done so already.

Shopping Cart

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Quantity</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 inch HDTV (model KTV-551)</td>
<td>2999.99</td>
<td>1</td>
<td>$2999.99</td>
</tr>
</tbody>
</table>

The total charged to your credit card: $2999.99

Update Cart | Purchase
WebScarab - example

![WebScarab Interface](image-url)

### Edit Request

**Method**: POST  
**URL**: http://192.168.0.20:8080/WScoat/attack/menu=110

<table>
<thead>
<tr>
<th>Header</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>192.168.0.20:8080</td>
</tr>
</tbody>
</table>

#### URLEncoded

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTY</td>
<td>1</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>Purchase</td>
</tr>
<tr>
<td>Price</td>
<td>2999.99</td>
</tr>
</tbody>
</table>

#### Hex

<table>
<thead>
<tr>
<th>Position</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WebScarab - example
WebScarab - example

Try to purchase the HDTV for less than the purchase price, if you have not done so already.

* Congratulations, You have successfully completed this lesson.*

Your total price is: $1.0

This amount will be charged to your credit card immediately.
**OWASP: WebGoat**

- WebGoat is a deliberately insecure J2EE web application
- It is designed to teach web application security lessons. For example, in one of the lessons the user must use SQL injection to steal fake credit card numbers.
- Lessons include e.g.:

  - Cross Site Scripting
  - Access Control
  - Thread Safety
  - Hidden Form Field Manipulation
  - Parameter Manipulation
  - Weak Session Cookies
  - Blind SQL Injection
  - Numeric SQL Injection
  - String SQL Injection
  - Web Services
  - Fail Open Authentication
  - Dangers of HTML Comments
OWASP: Sprajax

- Open source black box security scanner used to assess the security of AJAX-enabled (Asynchronous JavaScript and XML) applications.

- The goal of the Sprajax project is to create a useful tool for assessing the security of AJAX-enabled web application.
OWASP: Insecure Web App

- Web application that includes common web application vulnerabilities.

- Targeted for automated and manual penetration testing, source code analysis, vulnerability assessments and threat modeling.

- The goals of this tool:
  - Demonstrate how dangerous application vulnerabilities can be.
  - Close the gap between the theory of web application security and the actual code that we design and build.
  - Learn how these vulnerabilities can be fixed.
**OWASP LiveCD Project: LabRat**

- Beta Version 2.1 is focused on providing all of OWASP tools and documents on a bootable CD.
- At this point the OWASP tools and documents are on the CD but they are not all configured.
General notes about testing

- Manual testing: a human being *attacks* a web application using his experience, knowledge and tools

- Automated testing: a human being *uses* an automated vulnerability scanner to *attack* a web application

- Many tools generate extensive reports but results need to be verified → Testing can be automated, analysis never
Functional versus Technical Issues

Different vulnerabilities, different kind of testing:

- **Technical Vulnerabilities**
  - Data Handling
  - Dealing with Strong Typing, Encoding, and Validation of Data that the application handles
  - Can be tested fairly effectively by automated tools

- **Functional or Logical Vulnerabilities**
  - Deals with issues allowed by design, but not foreseen by the designers as a risk
  - Issues in human-eye visible forms: Comments revealing information, inconsistent business logic, etc.
What are they good for?

- Automated → Saves time and resources.
- In-built knowledge about vulnerabilities and threats against various technologies.
- Quite good reporting features
- Tools provide reliable, repeatable tests that produce accurate quantifiable information about software defects with security implications.
Click & Scan and send report to customer?

- Many tools generate extensive reports but results need to be verified → Testing can be automated, analysis never

- Tools are not short-cut to success, there is still need for security awareness and good insight to software development issues to be able to use these tools efficiently. → “A Fool with a Tool is still a Fool”

- Testers must be aware of limitations of tools

- There are issues that cannot be automated reliably:
  - business logic flaws
  - workflow bypasses
  - human eye visible errors.
Next step:

Restaurant Kaisla, Vilhonkatu 4, 00100 Helsinki
Thank You!