Agenda

- Necessary Introductions
- Fuzzing Motivation
- Selenium IDE
- Apparatus & Benchmarks
- Building Test Cases
- Oxygen: Scripting Test Cases
- Demos, Videos, Examples
- Conclusions
- Q&A
Necessary Introductions

- Yiannis Pavlosoglou, Seleucus Ltd, London
- OWASP Industry Committee
- Author of JBroFuzz
- PhD, CISSP, ...

Disclaimer: This presentation has nothing to do with selenium as a substance, nor its benefits (got a couple strange emails lately)

Instead, we are discussing Selenium IDE and the security testing of software, namely web applications
Motivation

[Web Application] Flows are hard to define and track in modern applications that use frames and AJAX [1].

- How do we best identify such an issue? (check your job description)
- How do we best automate the identification of such an issue? (perhaps check these slides) 😊
Stateful Fuzzing

- Newly issued cookies
- Cookies / AJAX
- ViewState

Stateless tool examples:
- SqlNinja
- JBroFuzz
- ...

Stateful tools ability:
- Recording of user login
- Chaining of user actions

Stateless: Tools that do not orchestrate state transversal in web applications
Selenium IDE

- Well known tool for:
  - Acceptance testing
  - Regression testing
  - Software testing
  - Penetration testing? (in certain situations)

- Components:
  - Selenium IDE
  - Selenium-RC (Remote Control)
  - Selenium Grid
Selenium IDE UI

- Plug-in for a number of supported browsers
  - O/S Independent

- Records a test case, while user is browsing
  - User clicks, inputs, radio button selections, etc.

- Tests the case for one or more condition
  - e.g. does this text exist?
## Selenium IDE

**Supported Browsers**

<table>
<thead>
<tr>
<th>Browser</th>
<th>Selenium-IDE</th>
<th>Selenium-RC</th>
<th>Operating Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox 3</td>
<td>1.0 Beta-1 &amp; 1.0 Beta-2: Record and playback tests</td>
<td>Start browser, run tests</td>
<td>Windows, Linux, Mac</td>
</tr>
<tr>
<td>Firefox 2</td>
<td>1.0 Beta-1: Record and playback tests</td>
<td>Start browser, run tests</td>
<td>Windows, Linux, Mac</td>
</tr>
<tr>
<td>IE 8</td>
<td></td>
<td>Under development</td>
<td>Windows</td>
</tr>
<tr>
<td>IE 7</td>
<td>Test execution only via Selenium-RC*</td>
<td>Start browser, run tests</td>
<td>Windows</td>
</tr>
<tr>
<td>Safari 3</td>
<td>Test execution only via Selenium-RC</td>
<td>Start browser, run tests</td>
<td>Mac</td>
</tr>
<tr>
<td>Safari 2</td>
<td>Test execution only via Selenium-RC</td>
<td>Start browser, run tests</td>
<td>Mac</td>
</tr>
<tr>
<td>Opera 9</td>
<td>Test execution only via Selenium-RC</td>
<td>Start browser, run tests</td>
<td>Windows, Linux, Mac</td>
</tr>
<tr>
<td>Opera 8</td>
<td>Test execution only via Selenium-RC</td>
<td>Start browser, run tests</td>
<td>Windows, Linux, Mac</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>Test execution only via Selenium-RC(Windows)</td>
<td>Start browser, run tests</td>
<td>Windows</td>
</tr>
<tr>
<td>Others</td>
<td>Test execution only via Selenium-RC</td>
<td>Partial support possible**</td>
<td>As applicable</td>
</tr>
</tbody>
</table>

* Tests developed on Firefox via Selenium-IDE can be executed on any other
Using Selenium IDE: Apparatus

- Operating System of your choice
  - Confirmed operations in: Solaris 10, Windows 7, Fedora 11, Ubuntu 9.10

- Proxy Tool of your choice
  - WebScarab, OWASP Proxy

- Language of your choice
  - Perl, v5.10.0 built for MSWin32-x86-multi-thread

- Selenium IDE
  - Firefox plug-in Selenium IDE 1.0 Beta 2 (June 3, 2008)

- Mozilla Firefox
  - 3.5.7

- Tests herein, performed on: WebGoat 5.3 RC1
  - I know! But recordings from penetration tests performed, are not really an option
  - Unlike a screenshot, with Selenium IDE, you can’t just obfuscate the URL!
Using Selenium IDE: *Benchmarks*

- Assessing Selenium IDE for Web Application Penetration Testing Requirements

- **Benchmark 1:** Can I leave it testing overnight?

- **Benchmark 2:** Can I know all the payloads that passed / failed a particular input field?
Using Selenium IDE: Demo Videos

Demo 1 Video: Login Brute Force
http://www.youtube.com/watch?v=3_LhYkzzN08

Demo 2 Video: SQL Injection
http://www.youtube.com/watch?v=6m0bq5hF_6w

As you’re here, we’ll do the demos live ($%£^&*!) …
Selenium IDE: Benchmark 1

- Given a login prompt:
  - Not necessarily a first landing page
  - A valid user account
  - No lockout present

- Perform a brute-force attack
  - Long list of passwords

- Objective: Quickly assess successful / failed logins
Selenium IDE: Benchmark 2

- Given an input field:
  - A page that you have to browse to
  - Check for all SQL injection payloads

- Objective: Quickly assess which SQL injection payloads succeed
  (don’t just report back a SQL injection vulnerability)
  (We want to know all filter evasion characters & successful payloads) 😊
Building Test Cases: Workflow Process

1. Record Basic Test Case
2. Determine Success/Fail Criterion
3. Decide on Payloads to Test
4. Generate Test Case Suite File
5. Run!
Record Basic Test Case

- Using your browser & Selenium IDE
  - Record your actions

- Select input field to automate testing
  - Specify a unique value
    - Could be: parameter, form field, GET/POST, etc.
    - Could not be: Referrer, Header, etc.*

[*] You could use Selenium-RC for implementing advanced features, outside standard browser operations
Determine Success / Fail Criterion

Something must be present within the page/response that:

- Distinguishes a successful attack from an unsuccessful one
- Is unique

Can be tough!

- Not really a technique for starters in the field:
  - Know your payloads
  - know your platforms
  - know your responses

- Know if this technique can be used for the attack in question
Decide on Payloads to Test
Scale: Generate Test Case Suite File

- For each of the test cases
  - Generate a single suite

- Group together all the test cases
  - Into one entity

- Allows you to obtain success / fail results
  - Batch process all test cases
Scripting Test Cases

To run oxygen.pl, make sure you have the following files:

- 00-challenge-login.xml
- 00-nitro.pl
- 00-oxygen.pl
- 00-payloads.txt

Run nitro.pl, only having executed oxygen.pl successfully, it should generate a file:

- 000-test-case-suite.xml

Another demo ($%£^&*!) …
Example 1: HTTP Form-field Brute-forcing

- Basic Test Case
  - Test Case
  - List of Passwords
  - Test Case Suite

Many other, simpler, ways to perform a brute-force attack
HTTP Form-field Brute-forcing (1/2)

- Basic Test Case
  - Open the URL
  - Type ‘username’
  - Type ‘password’
  - Wait...
  - Verify the text: "* Invalid login"
HTTP Form-field Brute-forcing (2/2)

- Basic Test Case
  - Open the URL
  - Type ‘username’
  - Type ‘password’
  - Wait...
  - Verify the text: “* Invalid login”

- Success if “Invalid login” is obtained...
Lessons Learned

- Timing is Everything
  - Number of hops / Load-balancing
  - Trace route information
  - Delays in the response

In the same way that you *(should)* check for max_rtt_timeouts in nmap

Check for all the above during stateful fuzzing sessions with Selenium IDE
Stateful Vulnerability Format

- Before Selenium, I could give you only a stateless vulnerability in the format of .jbrofuzz files.

"Here is the file, open it, run it, graph the result, see the vulnerability."

- Now, I can just give you a single Selenium IDE xml file with the test case file that is causing all the damage!
When **not** to use Selenium & Oxygen

- Heavy XSRF Protections Present
- CAPTCHA Present
- Threading: Non sequential order fuzzing
- Testing of Headers
  - Referrer Type Fields
  - HTTP Splitting

Read: "*To Automate or Not to Automate? That is the Question!*"[^2]
Conclusions

- It looks very good
- Saves a lot of testing time
- Should be calibrated correctly
- Does not replace human testing

You should have an understanding of:

- What it takes to script up a Selenium Test Case (stateful penetration testing cases)
- How to use Oxygen and Nitro with Selenium IDE (simple Perl scripting, try it in your language!)
- When not to consider using Selenium in Security (when there is more than input validation && state involved)
Questions?

Dr Yiannis Pavlosoglou
Project Leader / Industry Committee
Seleucus Ltd
yiannis@owasp.org
References


Step-by-step Guide (1/2)

1.0 Create a test case: 00-challenge-login.xml

1.1 Within the test case, record the field, parameter, value that you would like to fuzz as:
   sel-oxygen-nitro

1.2 After the response is received, right-click within your browser on something unique (can be tough) and select "Verify Text Present"

1.3 In Selenium IDE, select "Save Test Case"

1.4 Select as name: 00-challenge-login.xml

1.5 Save in a dedicated, clean folder for each test case, e.g. 02-sql-injection

2.0 Folder setup: 02-sql-injection

2.1 Create a 00-payloads.txt file, put inside, one payload per line, each SQL injection payload you would like to test for
Step-by-step Guide (2/2)

2.2 Copy oxygen.pl to the directory, run it by: perl oxygen.pl

2.3 A number of test cases will be generated e.g.

3.0 Bring in Nitro!

3.1 Copy nitro.pl to the directory, run it by: perl nitro.pl

3.2 This will generate the output test case suite in selenium

4.0 Load and run in Selenium IDE

4.1 In Selenium IDE: File -> Open Test Suite: main-test-suite.xml

4.2 Set speed to slow (you can always speed it up during testing)

4.3 Run!
Simple Source Code: oxygen.pl

```perl
#!/usr/local/bin/perl
#
# Program to take a single test case from selenium and substitute the input value marked as 'sel-oxygen-nitro' to a list of potential payloads read from file.
#
$initial_test_case = "00-challenge-login.xml";
$location_to_fuzz = "sel-oxygen-nitro";
$payloads_file = "00-payloads.txt";

# Read file the initial selenium test case file
open(INFO, $initial_test_case) || die "Couldn't read from file: $!\n";
@lines = <INFO>;
close(INFO);
# for later -v .. print @lines;

# Loop through the password files given as a starting brute force
#
open(FILEPWD, "<$payloads_file") || die "Could not find payloads file: $!\n";
$count = 1;
while (<FILEPWD>) {
    chomp;
    $pwd = $_;
    print "Count is: " . $count . " $pwd is: " . $pwd . "\n";
    # for -v later .. print $pwd . "\n";
    open(FILEWRITE, ">$initial_test_case")
    # Loop through the lines of the initial test case generating one file, per password
    foreach $line(@lines){
        $new_line = $line;
        $new_line =~ s/$location_to_fuzz/$pwd/g;
        print FILEWRITE $new_line ;
        # -v -v later print $new_line;
    }
    close FILEWRITE;
    $count++;
}
close FILEPWD;
```
#!/usr/local/bin/perl
#
# Program to generate the output test suite in selenium
# given the original test case and the payloads file
#
# Some notes:
# You need to have executed oxygen.pl before running this
#
# The payloads file must have the same length as when
# running oxygen.pl
#
$initial_test_case = '00-challenge-login.xml';
$payloads_file = '00-payloads.txt';

open(FILEWRITE, "> 000-main-test-suite.xml");
print FILEWRITE "<?xml version="1.0" encoding="UTF-8"?>\n";
print FILEWRITE "<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">\n";
print FILEWRITE "<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">\n";
print FILEWRITE "<head>\n";
print FILEWRITE "<meta content="text/html; charset=UTF-8" http-equiv="content-type" />
";
print FILEWRITE "</head>\n";
print FILEWRITE "<body>\n";
print FILEWRITE "<table id="suiteTable" cellpadding="1" cellspacing="1" border="1" class="selenium">\n";
print FILEWRITE "<tr><td><b>Test Suite</b></td></tr>\n";

open(FILEPWD, "<$payloads_file") || die "Could not find payloads file: $!\n";
$count = 1;
while (<FILEPWD>) {
    print FILEWRITE "<tr><td><a href="" . $count . $initial_test_case . "">" . $count . $initial_test_case . "</a></td></tr>\n";
    $count++;
}

print FILEWRITE "</tbody></table>\n";
print FILEWRITE "</body>\n";
print FILEWRITE "</html>\n";
close(FILEWRITE);