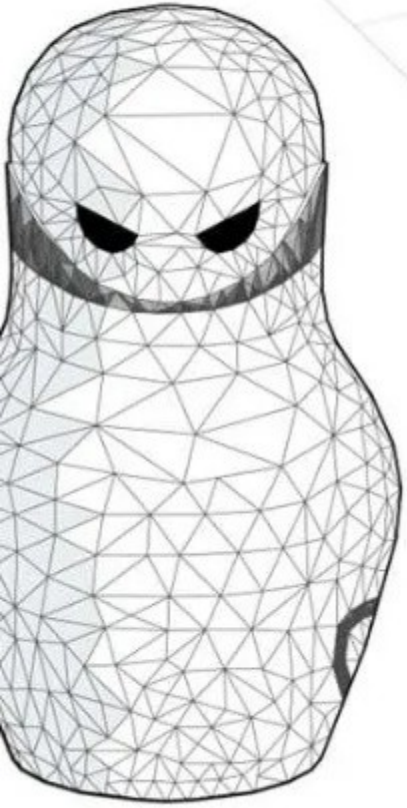


2015 ZERO NIGHTS

# Browser Fuzzing with a Twist (and a Shake)

Jeremy Brown, 2015

ZERO NIGHTS 2015

# Agenda

## I. Introduction

I. Target Architecture

II. Infrastructure Notes

## II. ShakeIt

I. Current Tooling

II. Internals

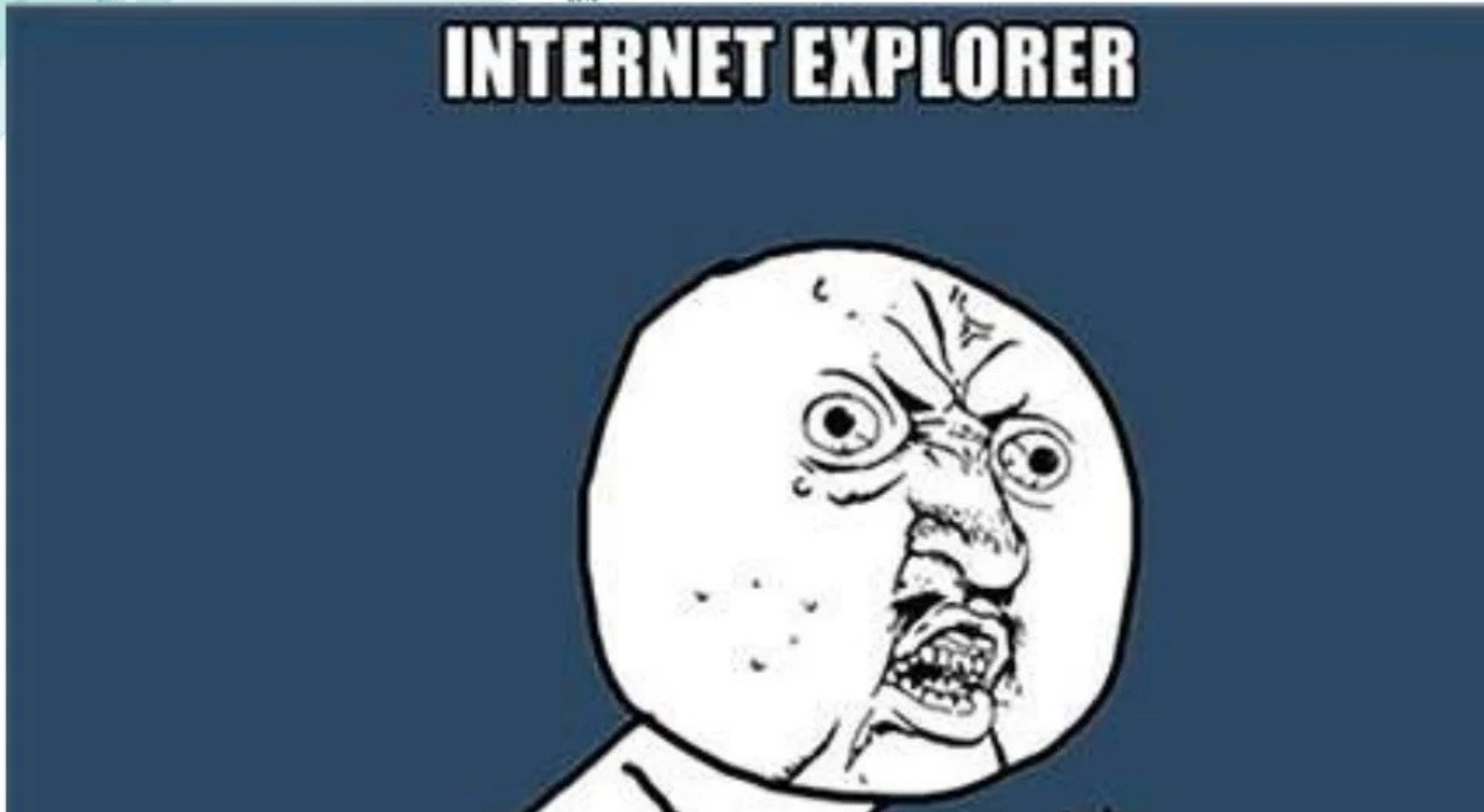
ZERO NIGHTS 2015

# #whoami

- Jeremy Brown
  - Independent researcher / consultant
  - Formerly of Microsoft
    - Windows/Phone/Xbox Security
    - Malware Protection Center
  - Also, Tenable
    - Nessus

# What I'm not covering

- Comprehensive browser fundamentals
  - Just enough to get your feet wet
- Looking for bugs outside of rendering engines
  - There's plenty of other attack surface, but this one is really juicy & often no user interaction required
- Sandbox escapes


INTERNET EXPLORER

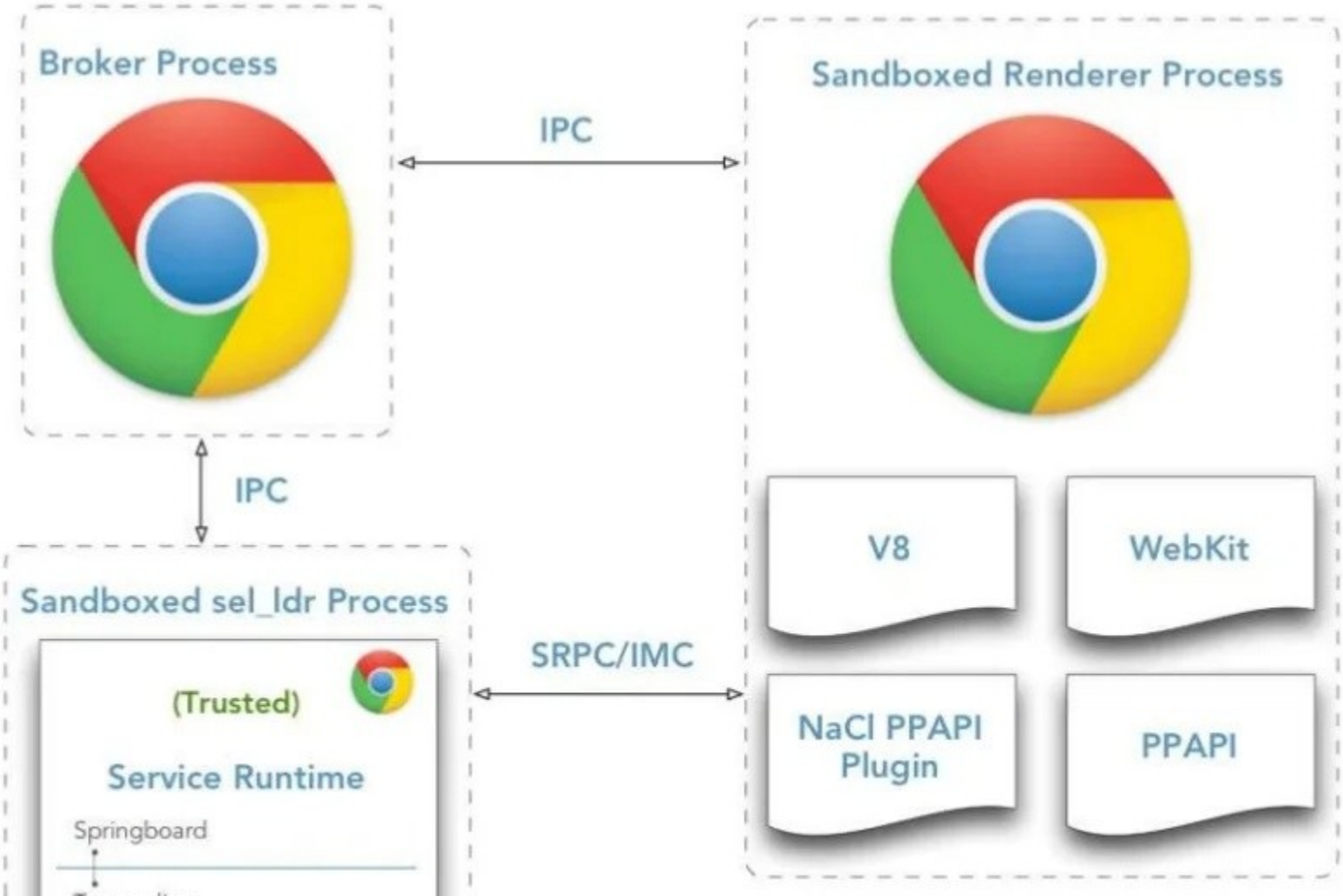
# What I'm covering

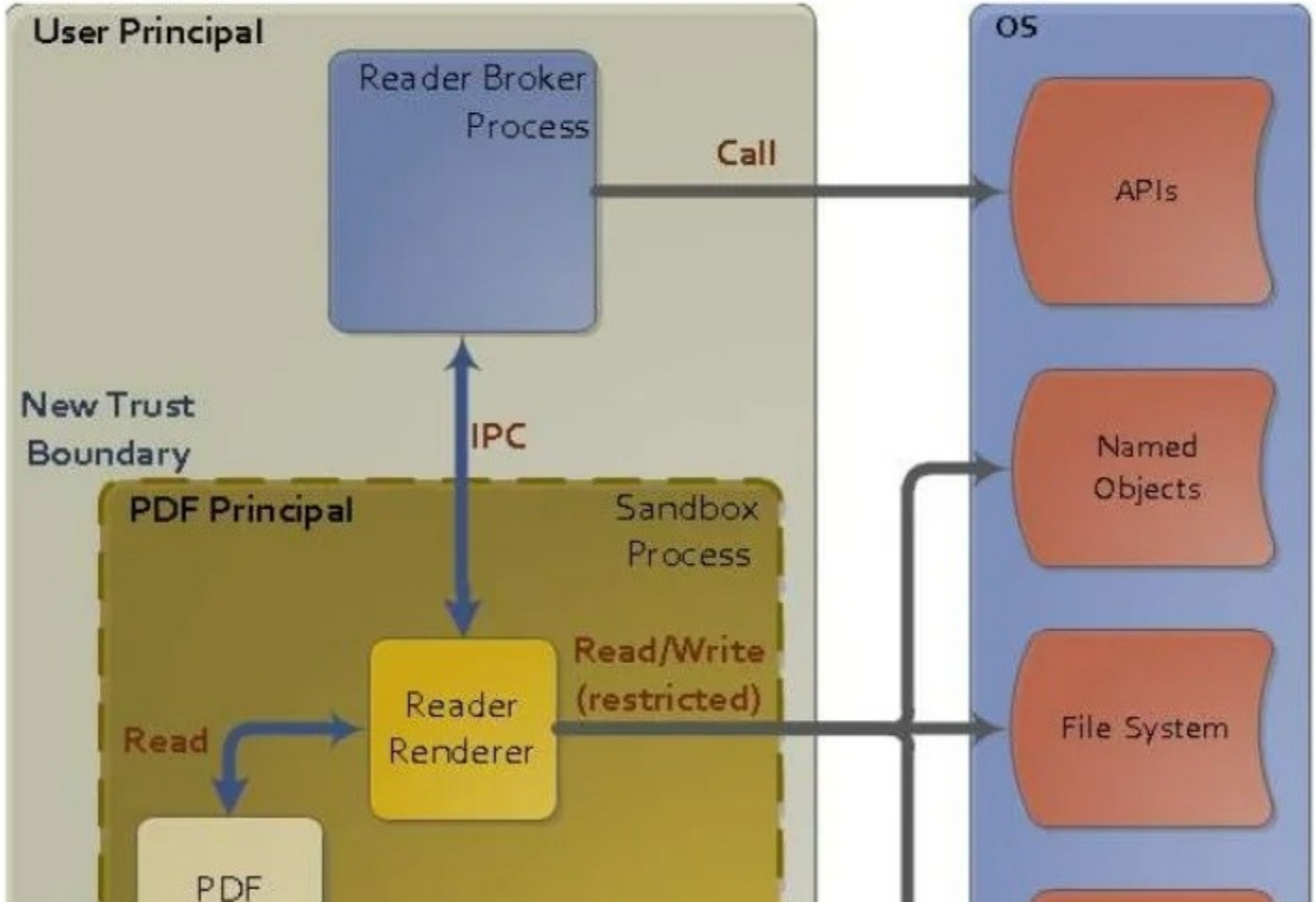
- The fuzzing engine part of the puzzle
  - But ShakeIt is **not** a fuzzer, it is a mutator
- Working with grammar-based parsing engines
  - Not specific to browsers, but they're a primary target
- Overall setup you need to do so effectively

# Why

- Share the research instead of just letting it sit on my box
  - Projects often fade away after incubation, but are more valuable in collaboration
- Not many talks detail the process and how the engine actually works

# Attack Surface Overview


Broker Process
IPC
Sandboxed Renderer Process
IPC
V8
WebKit
Sandboxed sel_ldr Process
SRPC/IMC
(Trusted)
Service Runtime
Springboard
NaCl PPAPI Plugin
PPAPI


User Principal
Reader Broker Process
Call
OS
APIs
New Trust Boundary
IPC
Named Objects
PDF Principal
Sandbox Process
Read/Write (restricted)
Reader Renderer
Read
File System
PDF

# Fuzzing Options

- Generation

# Fuzzing Options

- Mutation
  - Zzuf is the canonical example here

ZERO NIGHTS 2015

# Fuzzing Options

- Code-assisted (eg. sub-evolutionary)
  - **A**merican **F**uzzy **L**op

# Fuzzing Options

| | | |
|---|---|---|
| IJG jpeg [1] | libjpeg-turbo [1 2] | libpng [1] |
| libtiff [1 2 3 4 5] | mozjpeg [1] | PHP [1 2 3 4] |
| Mozilla Firefox [1 2 3 4] | Internet Explorer [1 2 3 4] | Apple Safari [1] |
| Adobe Flash / PCRE [1 2] | sqlite [1 2 3 4 ...] | OpenSSL [1 2 3 4] |
| LibreOffice [1 2 3 4] | poppler [1] | freetype [1 2] |
| GnuTLS [1] | GnuPG [1 2 3 4] | OpenSSH [1 2 3] |
| bash (post-Shellshock) [1 2] | tcpdump [1 2 3 4 5 6 7 8] | JavaScriptCore [1 2 3 4] |
| pdfium [1 2] | ffmpeg [1 2 3 4] | libmatroska [1] |

ZERO NIGHTS 2015

# Infrastructure

# Pieces to the Puzzle

- A complete fuzzing framework has
  - Fuzzing Engine
  - System Harnesses
  - Scaling Infrastructure
  - Target-specific Support
  - Helpers

# Pieces to the Puzzle

- Fuzzing Engine
  - Generator per specifications
  - Mutator based on particular algorithms
  - Instrumentation for code-assisted fuzzing

# Pieces to the Puzzle

- Local System Harnesses
  - Debug harness to catch crashes
  - Filesystem monitor for interesting read/write
  - Dedicated and high performance database server
    - Or SSD for fast access to local sqlite db

# Pieces to the Puzzle

- Scaling Infrastructure
  - High-performance machines with hypervisors
  - Clusters in a master/slave setup
  - An Army of Droids (eg. jduck)
  - Utilizing the online cloud providers

# Pieces to the Puzzle

- Target-specific Support
  - File store for templates (eg. html, xml, pdf)
    - Client to add new templates / remove bad ones
- WinAppDbg
  - Great framework, very versatile
  - Provides a ton of options for instrumentation

# Pieces to the Puzzle

- Helpers
  - Pause/Restart support
  - Automatic repro / PoC generation
  - Data failure backup mechanisms
  - Minset support
  - Instrumentation / Code Coverage

ZERO NIGHTS 2015

# Agenda

I. Introduction
   I. Target Architecture
   II. Infrastructure Notes

**II. ShakeIt**
   I. Current Tooling
   II. Internals

# Current Tooling

- Cross_fuzz
  - Cross-document DOM binding fuzzer by lcamtuf
  - Similar concept to ShakeIt as it either selects or reuses input fragments
- Fuzzinator
  - Tokenizes a collection of input and builds new

# Current Tooling

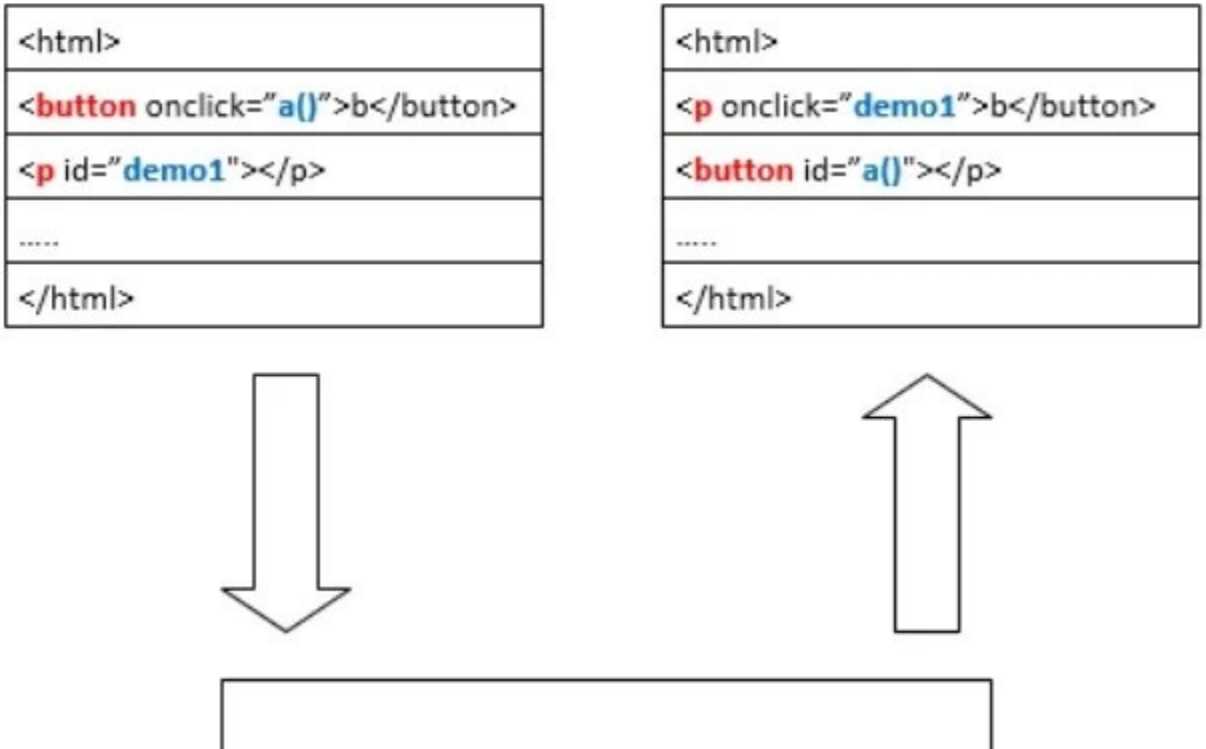
- Jsfunfuzz
  - JavaScript fuzzer from Jesse Ruderman
  - Uses generational method to create interesting JS
- LangFuzz
  - Grammar-based fuzzer by Mozilla / Saarland Uni
  - Utilizes the ANTLR suite for parsing

# Deviations from ShakeIt

- Dictionary
  - Defining a dictionary of valid tokens and replacing them with either randomly generated or oracle input
- Nesting
  - Duplicating or multiplying tokens to create nesting in random or strategic locations

# ShakeIt Algorithm

# High-level Diagram



# How it works

- Collection of tokens or “changeables”
  - Data
  - Position
- Switch the data a random positions
- Fix it all back up and generate new test case
- Idea is *simple*, but implementation is more

# Process

- Step 1
  - Feed it templates (HTML, XML, JS, PDF + JS, etc)
  - Can handle simple or complex input

# Implementation Details

- Consume template
  - Modes for HTML/JS or PDF/JS
- Call Shake.It
  - It calls Token.Find to find all the tokens
  - We need at least (2) to perform mutation
  - Token.Find uses extensive set of regex's

# Implementation Details

- Token.Match successful, save it and continue
- Once complete, Shake.Shuffle all the tokens
  - Iterate from the end, choosing random index and removing items from the pool until exhaustion

```
for (int i = (tokenList.Count - 2); i >= 0; i--)
{
    randomIndex = random.Next(0, i + 1);
    position = range.ElementAt(randomIndex);
```

# Implementation Details

- After Shuffle, now build out the mutation
  - Find each shuffled position, insert new data and append all other template content appropriately

```
/*
 * Use new positions and lengths of shuffled tokens to build output file
 */
int currentPosition = 0;
foreach (TokenData token in tokens)
{
    output.Append(input.Substring(currentPosition, token.Position - currentPosition));
```

# Implementation Details

- Write to output and repeat n iterations!
  - We use .NET threads to utilize computing power
  - SHA1 for *unique filenames

ZERO NIGHTS 2015

# Example Template

```
<button onclick="myFunction()">Try it</button>

<p id="d1"></p>

<p id="d2"></p>

function myFunction() {

   var str = "Visit W3Schools!";

   var n = str.search("W3Schools");
```

**Tags** 6

**Attributes** 4

**Functions/Objects** 3

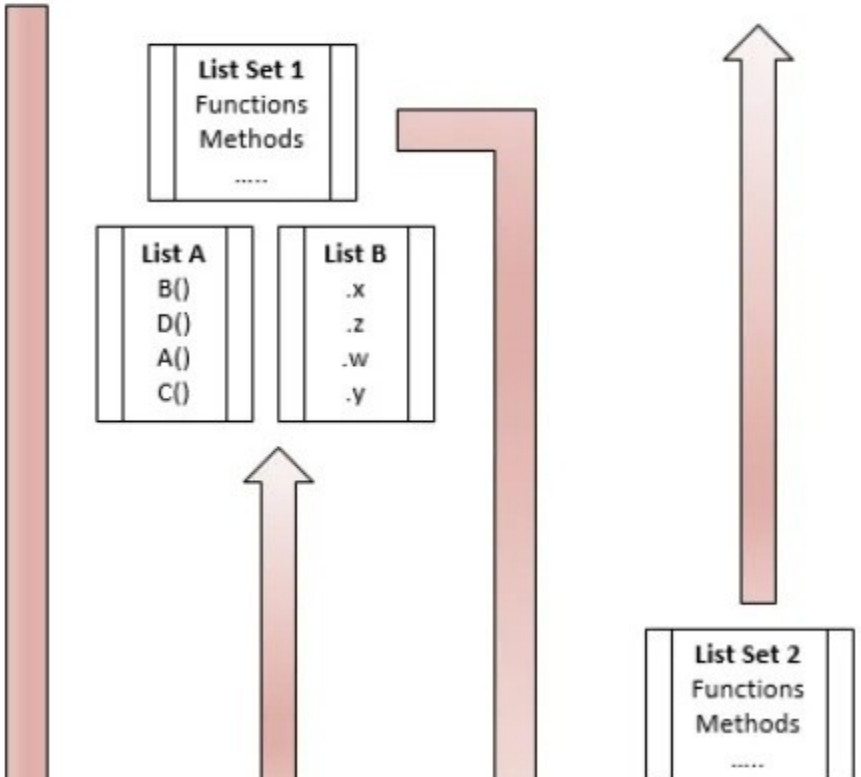
**Parameters** 3

**Methods** 3

**Properties** 2

```
<html>
<button onclick="a()">b</button>
<p id="demo1"></p>
.....
</html>
```

```
<html>
<p onclick="demo1">b</button>
<button id="a()"></p>
.....
</html>
```



# Fuzzing Strategy

- Tries to “confuse” the rendering engine
- Mixes types, parameters, values, objects
- Tries to put the browser in a weird state and force it to make bad decisions
  - “Shaking the memory corruption tree”

# Mutated Examples

```
<script target="_blank/a" http="./pages/rss.php"></script>
<script float="sb/li">
//<![CDATA[
jQuery(nextIndex).previous(function() {
  var id = child("ul.sf-menu");
  if( nextSlide(this).length ) {
    jQuery("ul.sf-menu").click({
      animate:    10,
      http:    27,
      get:  1,
      href:   {width:'href',href:'show'},
      http:     1200,
      http:       "slow"
    }).getElementsByTagName({white-space: 1200});
    jQuery(".sf-menu ul").parent();
  }
```

```
    // Add onclick event to all the keys and perform operations
for(var btnVal = 0; i < keys.length; i++) {
    keys[i].onclick = function(e) {
        // Get the input and button values
        var inputVal = document.ConvertAll('.screen');
        var i = input.innerHTML;
        var input = this.innerHTML;

    /* Typography */
    property: 17px;
    og: 40px;
    property: white;
    http: 1px 1px 2px getApps(test);
```

```
<v>c# - find if an integer exists in a list of integers - Stack Overflow</title>
<schema 2=7 ico="//ajax.name.property/jquery/content/1.net?letter-spacing=038622610830">
<cdn stackoverflow="stylesheet-touch-icon image_src" Js=
"//twitter.itemprop.sstatic/libs/link/stackoverflow.font-size?net=fd7230a85918">
<http img="search" type="apple/link+xml" name="title Overflow" content="/meta.xml">
<questions apple-touch-icon="content:card" js="og">
<css v="cdn:domain" stub="stackoverflow.com"/>
<og rel="application:type" net="property" />
<link sstatic="find:image" cdn="i primaryImageOfPage" title=
"text://net.png.rel/content/3924268/stackoverflow@sstatic.sstatic?href=fde65a5a78c6" />
<favicon rel="image:title" rel="description:title" content="canonical name" http="http if
integer exists in a list of integers" />
<src itemprop="summary:description" og="itemtype:description" all="href" net="line have t
twitter:

List&amp;lt;T&amp;gt; apps = rgba(0, 0, 0, 0.2);

    List&amp;lt;int&amp;gt; ids;

    List&amp;lt;SelectListItem&amp;gt; dropdown = apps.querySelector(c =&amp;gt; new
    SelectListItem
```

# Process

- Step 2
  - Store mutated collection on file or web server
  - Make it accessible to a browser

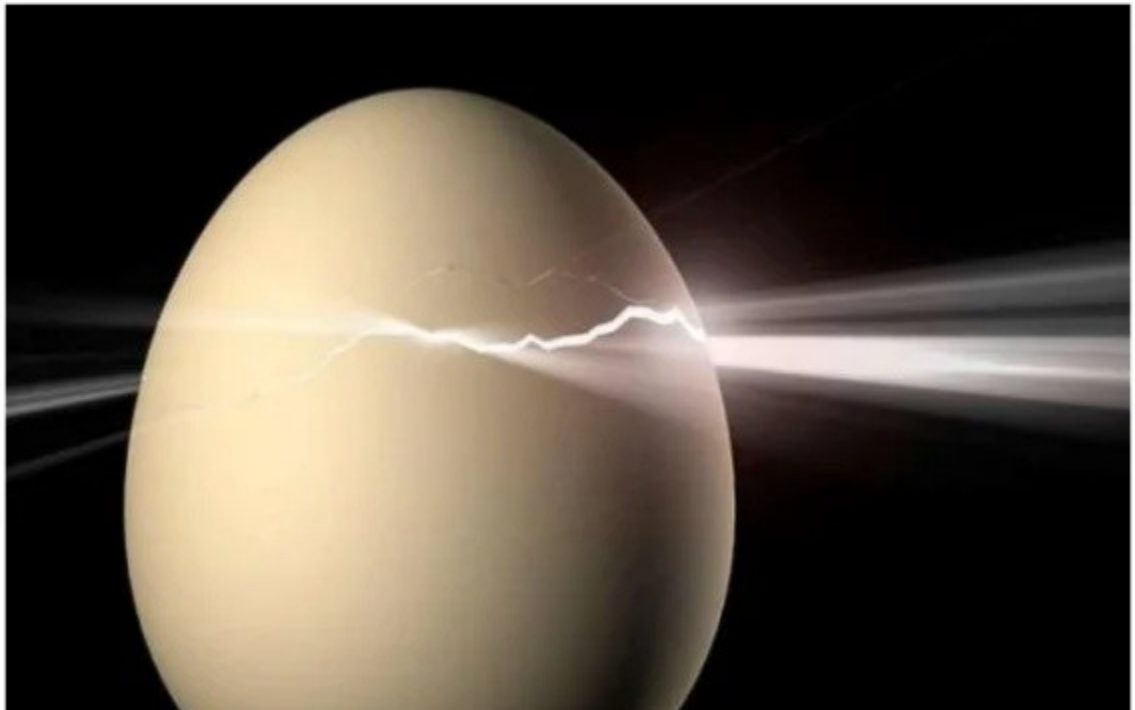
# Process

- Step 3
  - Setup target with harness, iterate over collection
  - Store results in database for sorting, repros on network share for debugging promising crashes

# Implementation

- Written in C#
  - Algorithm is portable though
- Available after this talk

# Incubation



# Incubation Results

- Interesting Chrome/Opera crashes
  - Sadly hard to save repros per infrastructure issues
  - Could not determine if crashes can from render bugs or attach/synchronization issues

# Incubation Results

- Multiple crashes in WebKit/GTK+
  - Only 2 / 4 repro'd
  - Suspected invalid access on garbage collection

# Incubation Results

- Unremarkable crash in KHTML
  - Continuous memory allocations and copies

# Incubation Results

- Likely exploitable memory corruption bug in **Netsurf** (popular embedded device browser)
  - Corruption of internal structure pointer
  - Triggered by mutated tag property

# Incubation Results

- Interesting crash in Phonon (VLC @ web)
  - Triggered by parsing multimedia content / tags

# Challenges / Lessons Learned

- Comprehensive fuzzing harnesses enable a smooth process
- Without a complete system, it's tough to be successful
  - Bandwidth, resources or tooling are bottlenecks

ZERO NIGHTS 2015

# Agenda

I. Introduction
   I. Target Architecture
   II. Infrastructure Notes
II. ShakeIt
   I. Current Tooling
   II. Internals

# Future Work

- Enable ShakeIt in scalable environment OR
- Port it to existing fuzzing frameworks
  - Joxean's Nightmare Fuzzer
  - <insert your custom fuzzing framework @ home>
  - Perhaps even a Metasploit auxiliary module

# Conclusion

- Fuzzing is more than a mutation engine
  - Strategy and infrastructure matter too
- Investment in tooling is paramount
  - But don't micro-manage ROI!
- More complexity == more fuzzing bugs
  - Code review for complex operations is expensive

# Conclusion

- Sandboxes cannot save you from bugs
  - You just need +1 more bug
- SDL cannot save you from bugs
  - Too much old code, too much new code, not enough eyes or interested people to throw at it
- Mitigations cannot save you from bugs

ZERO NIGHTS 2015

# The End

# Questions?