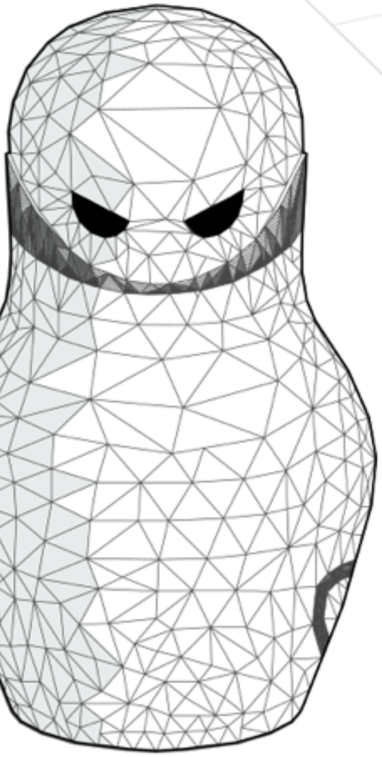


ZERO NIGHTS 2015

Browser Fuzzing with a Twist (and a Shake)

Jeremy Brown, 2015

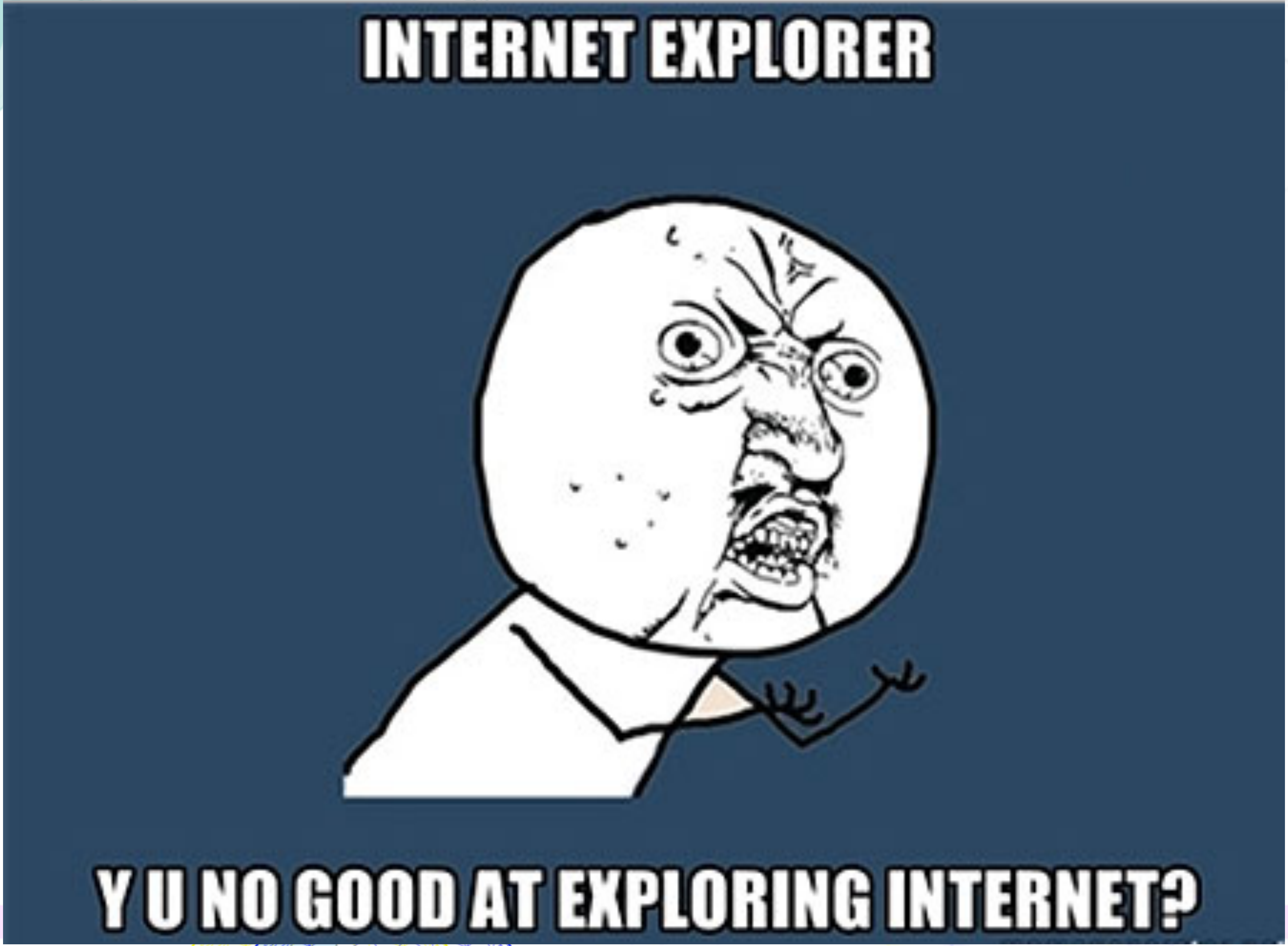
Agenda

#whoami

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

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
  - This is needed post-compromise of renderer


INTERNET EXPLORER
Y U NO GOOD AT EXPLORING INTERNET?

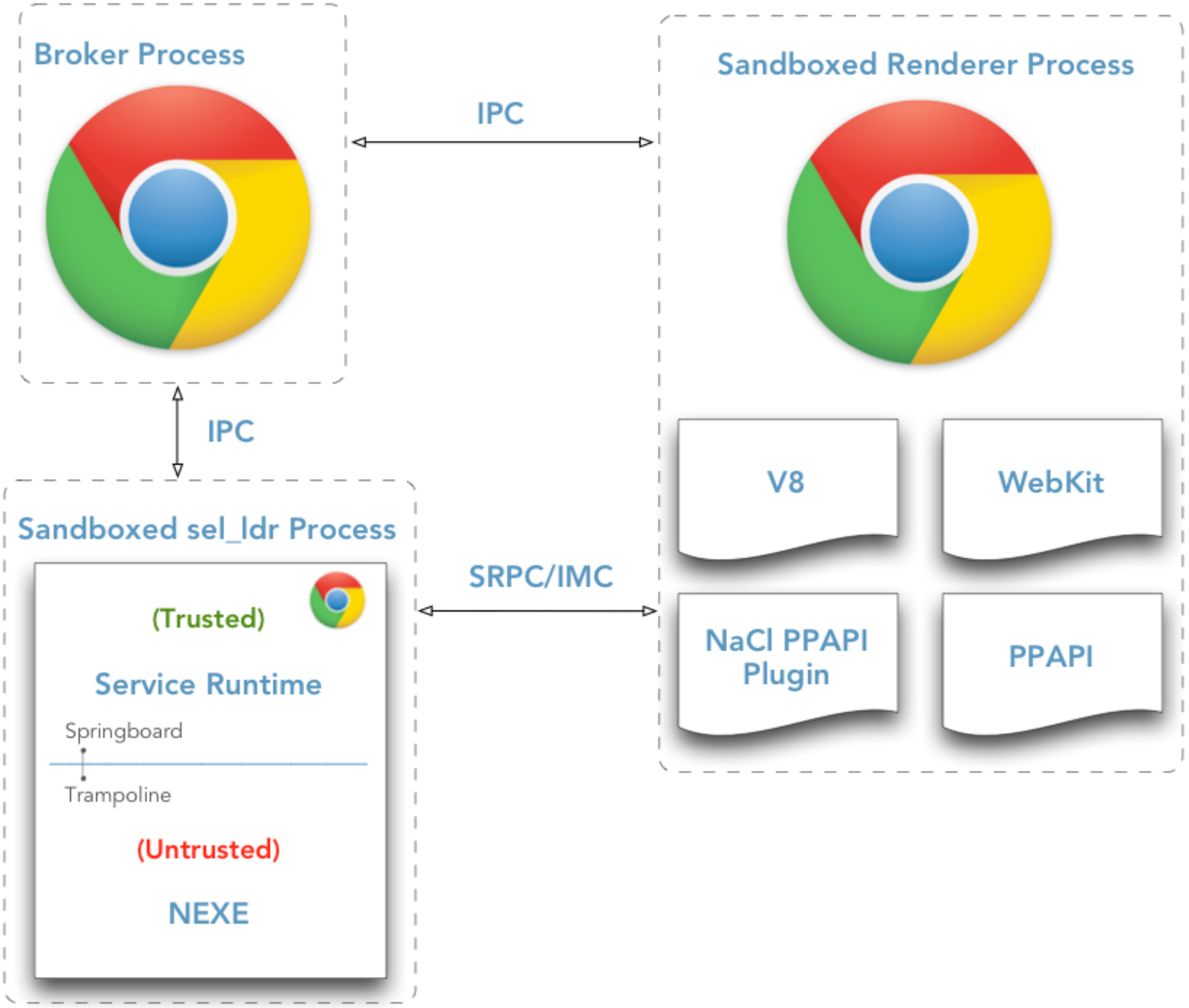
What I’m covering

- 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
  - But not claiming I fuzz as well as Ben Nagy
  - A lot of hard lessons learned

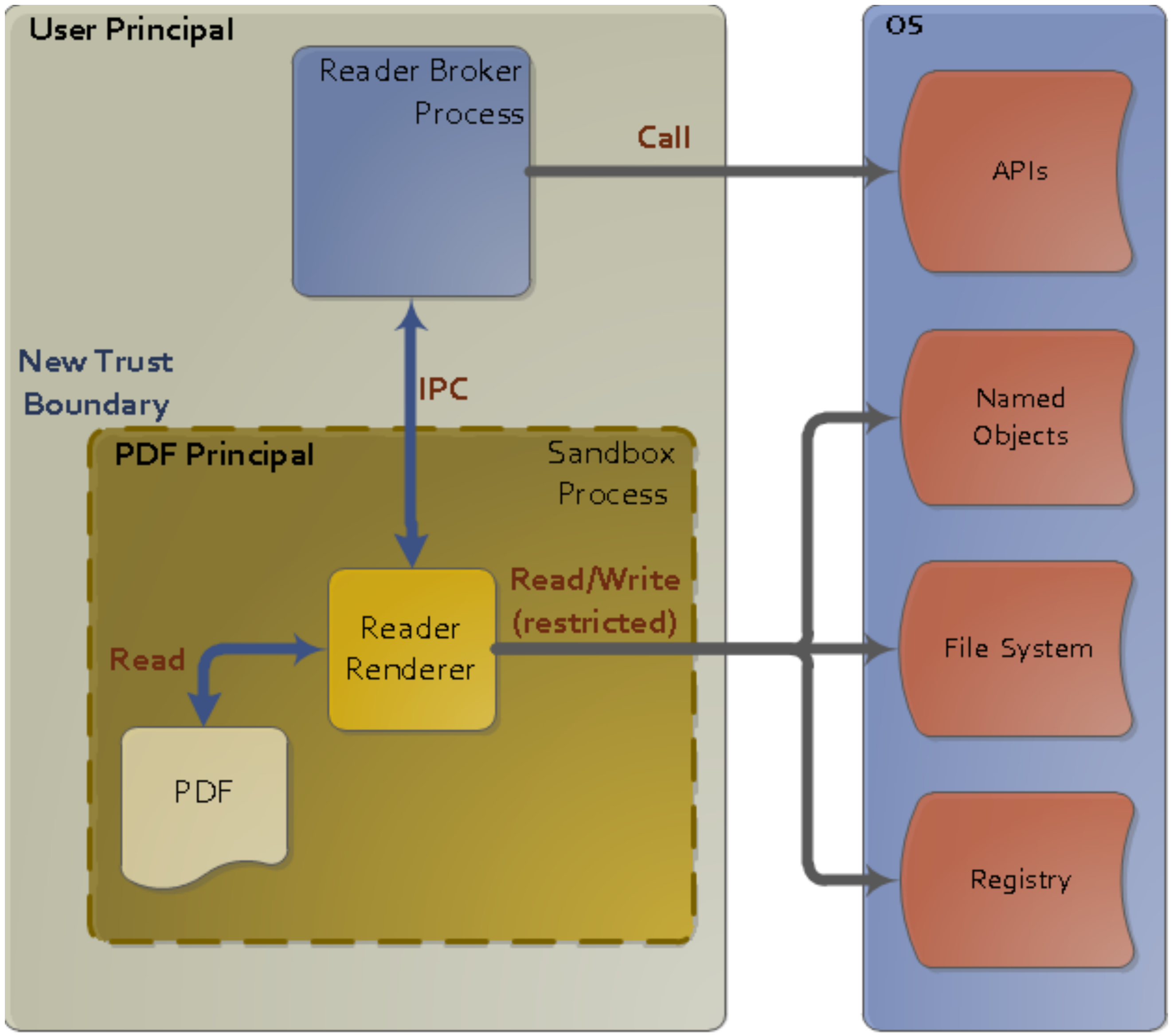
Why

- 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
  - Most engines are not rocket science
  - Fuzzing really has no rules, any method fair game

Attack Surface Overview



Credit:
Chris Rolf / LeafSR (now Yahoo!)
http://blog.leafsr.com/2012/09/09/google-native-client-attack-surface-and-vulnerabilities-part-4/



Reference:
"Inside Adobe Reader Protected Mode - Part 1 - Design" – Security @ Adobe
http://blogs.adobe.com/security/2010/10/inside-adobe-reader-protected-mode-part-1-design.html

Fuzzing Options

- Generation

PEACH FUZZER

Reference: http://www.peachfuzzer.com

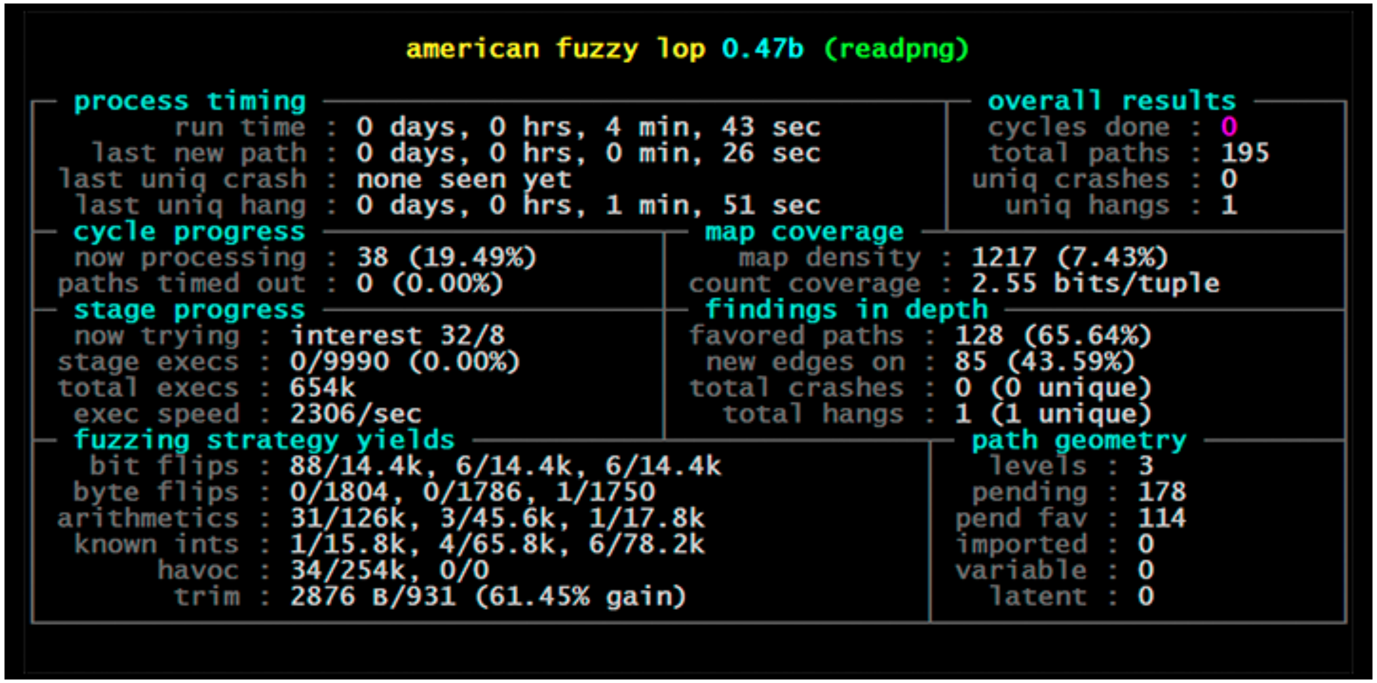
Fuzzing Options

- Mutation
  - Zzuf is the canonical example here

Reference: http://caca.zoy.org/wiki/zzuf

Fuzzing Options

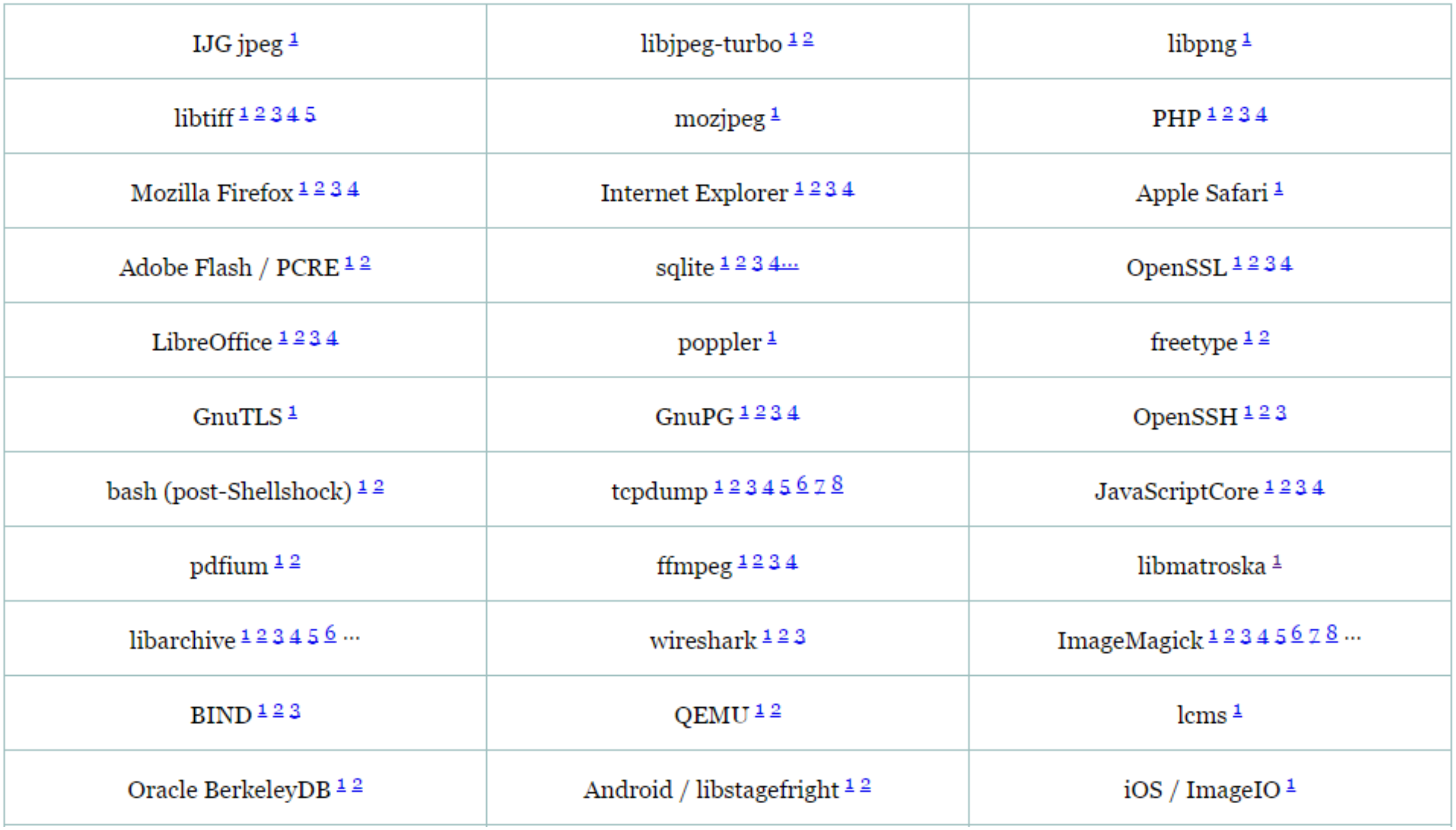
- Code-assisted (eg. sub-evolutionary)
  - **A**merican **F**uzzy **L**op



```
                 american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec       │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec       │  total paths : 195    │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec       │   uniq hangs : 1      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 38 (19.49%)       │    map density : 1217 (7.43%)          │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.55 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : interest 32/8         │ favored paths : 128 (65.64%)           │
│ stage execs : 0/9990 (0.00%)        │  new edges on : 85 (43.59%)            │
│ total execs : 654k                  │ total crashes : 0 (0 unique)           │
│  exec speed : 2306/sec              │   total hangs : 1 (1 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k            │    levels : 3          │
│  byte flips : 0/1804, 0/1786, 1/1750                │   pending : 178        │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k             │  pend fav : 114        │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k             │  imported : 0          │
│       havoc : 34/254k, 0/0                          │  variable : 0          │
│        trim : 2876 B/931 (61.45% gain)              │    latent : 0          │
└─────────────────────────────────────────────────────┴────────────────────────┘
```

Reference: http://lcamtuf.coredump.cx/afl/

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
libarchive 1 2 3 4 5 6 ...	wireshark 1 2 3	ImageMagick 1 2 3 4 5 6 7 8 ...
BIND 1 2 3	QEMU 1 2	lcms 1
Oracle BerkeleyDB 1 2	Android / libstagefright 1 2	iOS / ImageIO 1

Reference: http://lcamtuf.coredump.cx/afl/

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
  - Run into interesting issues sometimes, eg. bottleneck with db server / attach memory errors

Pieces to the Puzzle

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

Agenda

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 tests from those

References:
http://lcamtuf.coredump.cx/cross_fuzz/
http://browser.sed.hu/blog/20141023/fuzzinator-reloaded

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
  - Like Cross_fuzz, it can reuse input fragments

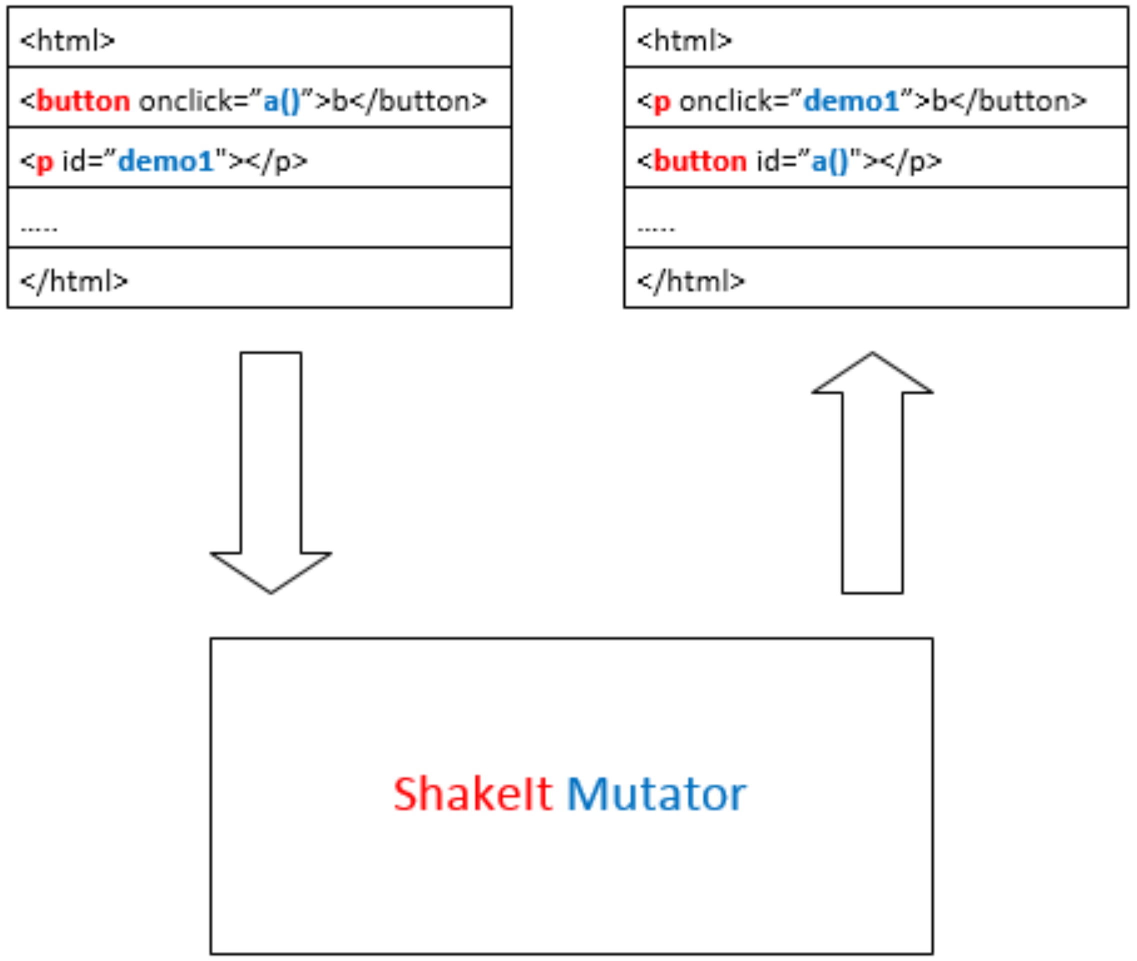
References:
https://github.com/MozillaSecurity/funfuzz/blob/master/js/jsfunfuzz/README.md
https://www.st.cs.uni-saarland.de/publications/files/holler-usenix-2012.pdf

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


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

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

How it works

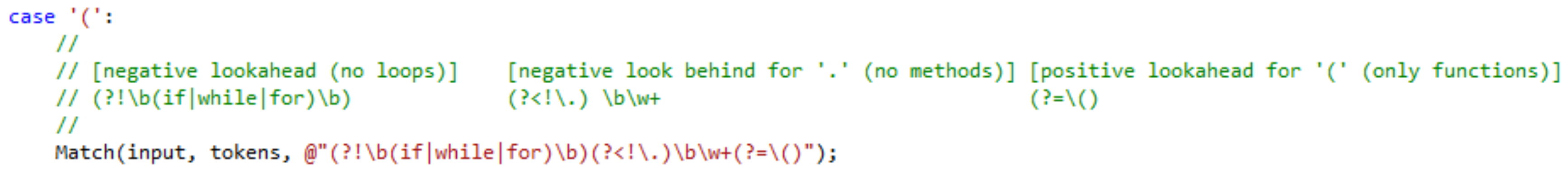
- Collection of tokens or “changeables”
  - Data
  - Position
- Switch the data a random positions
- Fix it all back up and generate new test case
- Idea is *simple*, but implementation is more complex

Process

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

Implementation Details

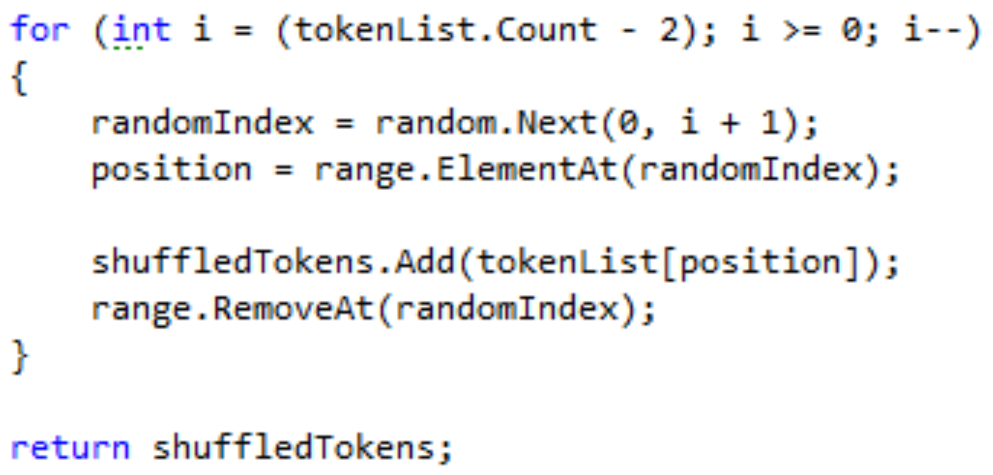
- Consume template
  - Modes for HTML/JS or PDF/JS
- Call Shake.It
  - It calls Token.Find to find all the tokens
  - We need at least (2) to perform mutation
  - Token.Find uses extensive set of regex's



```
case '(':
    //
    // [negative lookahead (no loops)]    [negative look behind for '.' (no methods)] [positive lookahead for '(' (only functions)]
    // (?!\b(if|while|for)\b)             (?<!\.) \b\w+                               (?=\()
    //
    Match(input, tokens, @"(?!\b(if|while|for)\b)(?<!\.)\b\w+(?=\()");
```

Implementation Details

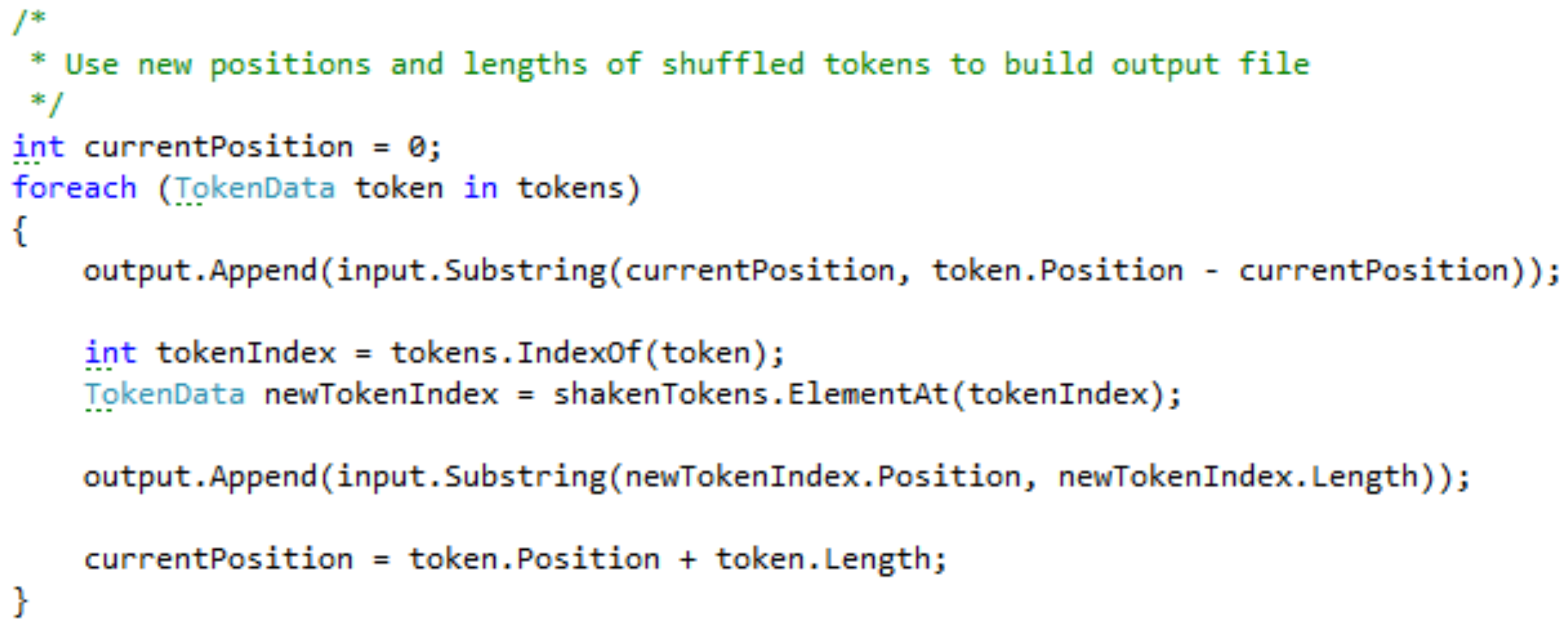
- Token.Match successful, save it and continue
- Once complete, Shake.Shuffle all the tokens
  - Iterate from the end, choosing random index and removing items from the pool until exhaustion



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

    shuffledTokens.Add(tokenList[position]);
    range.RemoveAt(randomIndex);
}

return shuffledTokens;
```

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));

    int tokenIndex = tokens.IndexOf(token);
    TokenData newTokenIndex = shakenTokens.ElementAt(tokenIndex);

    output.Append(input.Substring(newTokenIndex.Position, newTokenIndex.Length));

    currentPosition = token.Position + token.Length;
}
```

Implementation Details

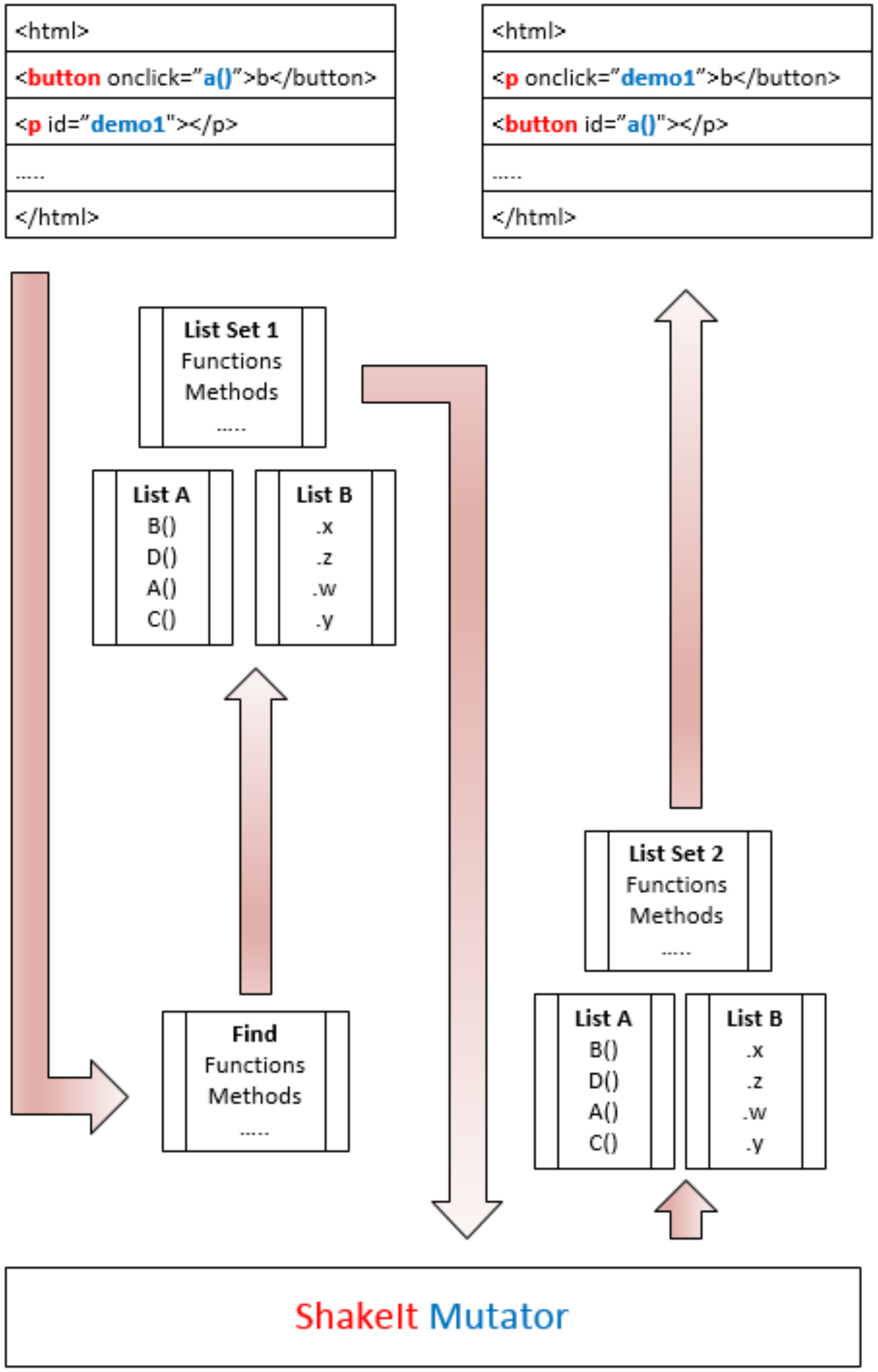
- Write to output and repeat n iterations!
  - We use .NET threads to utilize computing power
  - SHA1 for *unique filenames

- * We don't care about collisions here ☺

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");

document.getElementById("d1").innerHTML = n;

document.getElementById("d2").target = "_blank"; }
```

Tags	6
Attributes	4
Functions/Objects	3
Parameters	3
Methods	3
Properties	2
Variables	5
Values	6

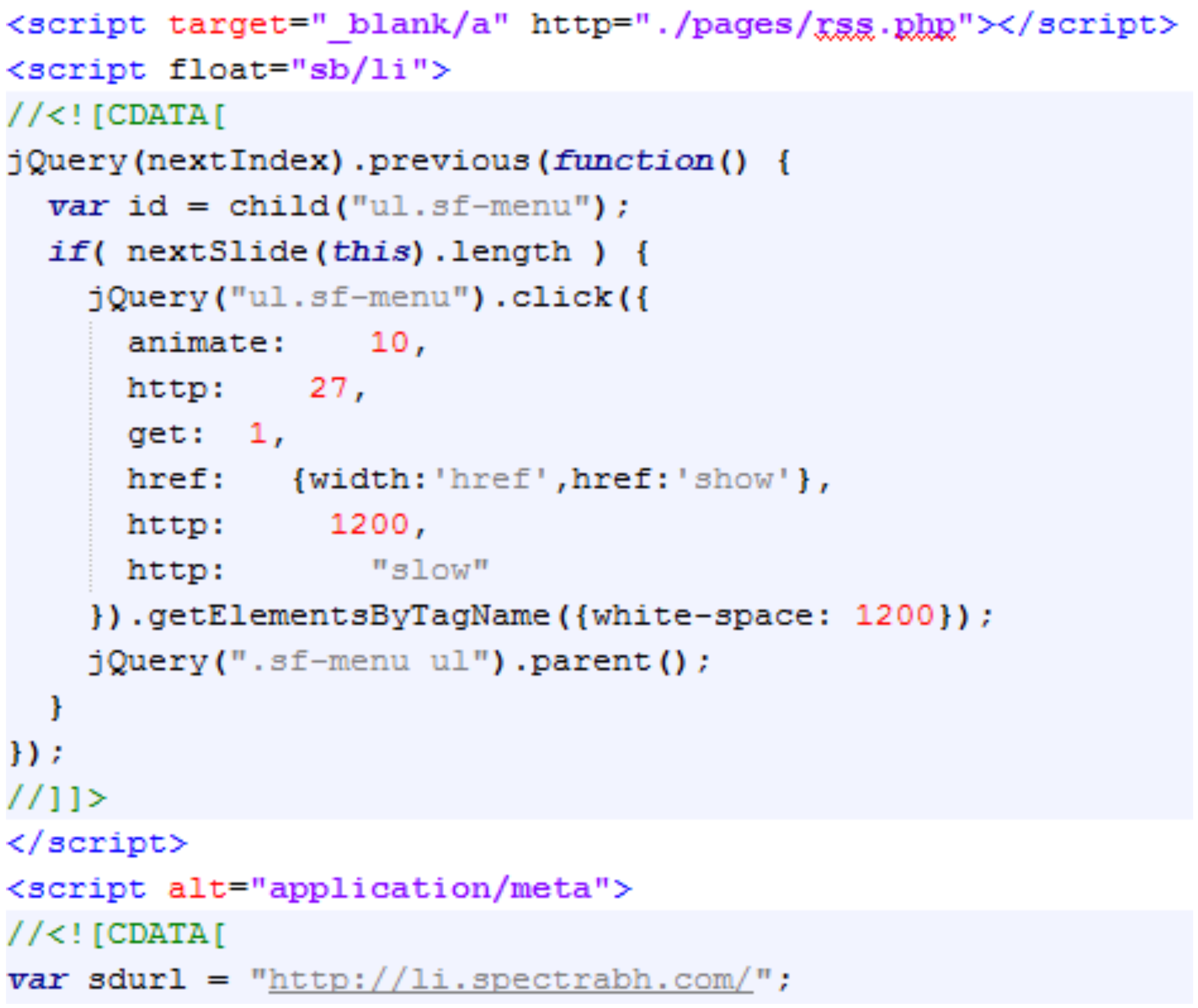


<html>
<button onclick="a()">b</button>
<p id="demo1"></p>
.....
</html>
<html>
<p onclick="demo1">b</button>
<button id="a()"></p>
.....
</html>
List Set 1
Functions
Methods
.....
List A
B()
D()
A()
C()
List B
.x
.z
.w
.y
Find
Functions
Methods
.....
List Set 2
Functions
Methods
.....
List A
B()
D()
A()
C()
List B
.x
.z
.w
.y
ShakeIt Mutator

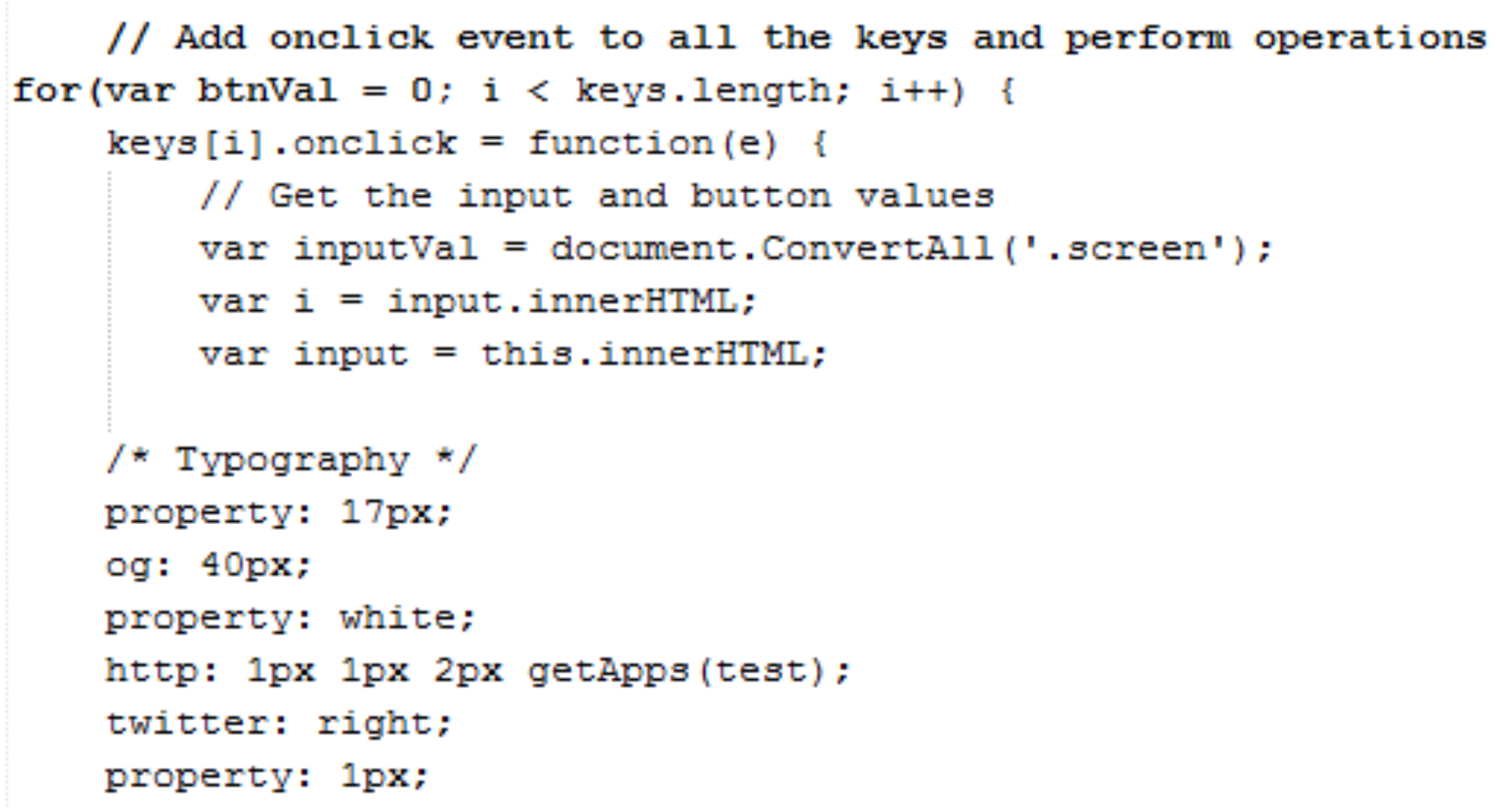
Fuzzing Strategy

- 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();
  }
});
//]]>
</script>
<script alt="application/meta">
//<![CDATA[
var sdurl = "http://li.spectrabh.com/";
```



```
    // 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);
    twitter: right;
    property: 1px;
```

```
<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=03862261O830">
<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 an
integer exists in a list of integers" />
<src itemprop="summary:description" og="itemtype:description" all="href" net="line have this
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
    {
        Se..." />
<meta color="text-shadow:url" png=
"text-height://stackoverflow.com/meta/en/meta-if-an-integer-exists-in-a-list-of-integers"/>
<apple href="name" content=
"twitter-align://opensearchdescription.com/meta/find/meta-if-an-integer-exists-in-a-list-of-int
egers" />
```

Process

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

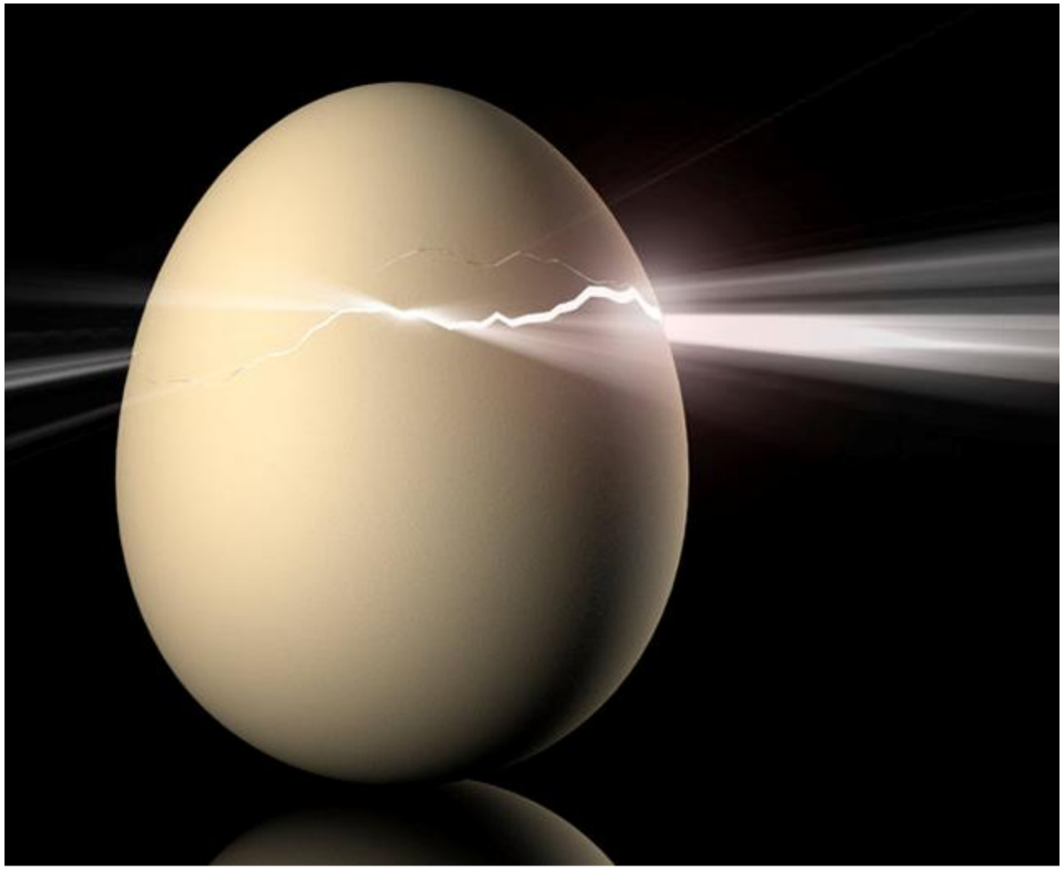
Process

- 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

Agenda

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
  - Manually pen-testing is great *for logic bugs*
  - **Does anyone seeing software becoming simpler?**

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
  - They only make them +n days harder to exploit
- **Managed code is a positive step forward**

The End

Questions?