Using Code Coverage to optimise Web Browser Fuzzing

James Fell

Dept of Computer Science

University of York

15th September 2017

Background

- What is Fuzzing?
- Mutation vs Generation
- Code coverage
- Fuzzing web browsers for 0days
- File format vs DOM

Challenges

- How to obtain a corpus of web pages for mutation that gives good code coverage
- Typical method is web crawling. Suspected this misses a lot of HTML tags and hence loses code coverage

Proposed Methods

- Research and build a full HTML specification including obsolete tags and attributes
- Create a Context-Free Grammar that generates syntactically correct web pages using the full spec

Software Suite

- All in Python
- HTMLscan – Evaluate a corpus using spec
- HTMLscrape – Build a corpus by web crawling
- HTMLgen – Generate a corpus from a CFG
- HTMLharness – Deliver a corpus to a browser
- HTMLfuzz – DOM fuzzing using the CFG

Experiments and Results

- Compared 50 page scraped corpus to 50 page generated corpus

Source of Corpus	Missing tags	Missing attributes
HTMLscrape	78	94
HTMLgen	6	0

Experiments and Results

- Different sized corpora were generated with HTMLgen and evaluated with HTMLscan.
- The same corpora were also fed into an instrumented web browser by HTMLharness and code coverage measured.
- Corpus size of 200 pages considered to be optimum when using HTMLgen to seed a mutation fuzzer.

Experiments and Results

- A corpus of 200 valid pages from HTMLgen was used to seed Radamsa and make a corpus of 50,000 mutated pages

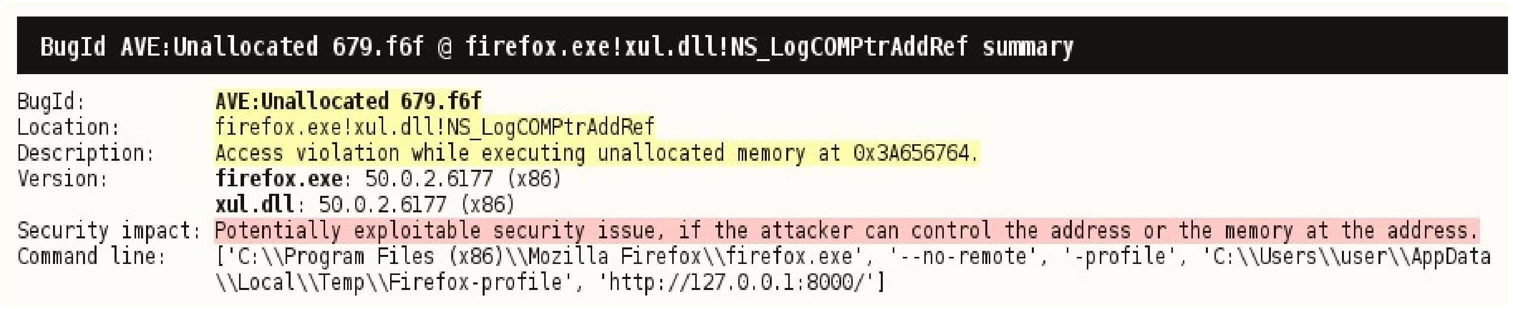
Fuzzing VM	Web Browser	Crashes
Windows 10	Microsoft Edge	2
Windows 10	Internet Explorer 11	1
Windows 10	Firefox	1
Debian 8	Netsurf	1
Debian 8	Xombrero	1

Experiments and Results

- HTMLfuzz was run against several browsers for 24 hours each

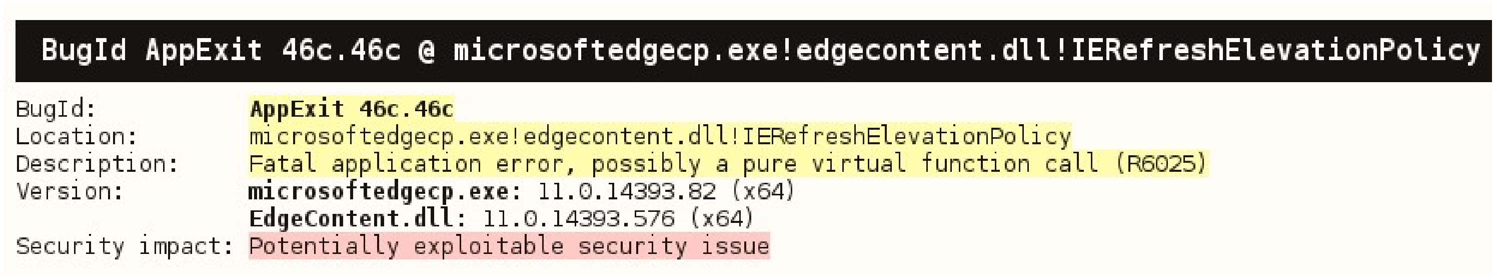
Fuzzing VM	Web Browser	Crashes
Windows 10	Microsoft Edge	1
Windows 10	Internet Explorer 11	3
Windows 10	Midori	2

(Possibly) Exploitable Vulns



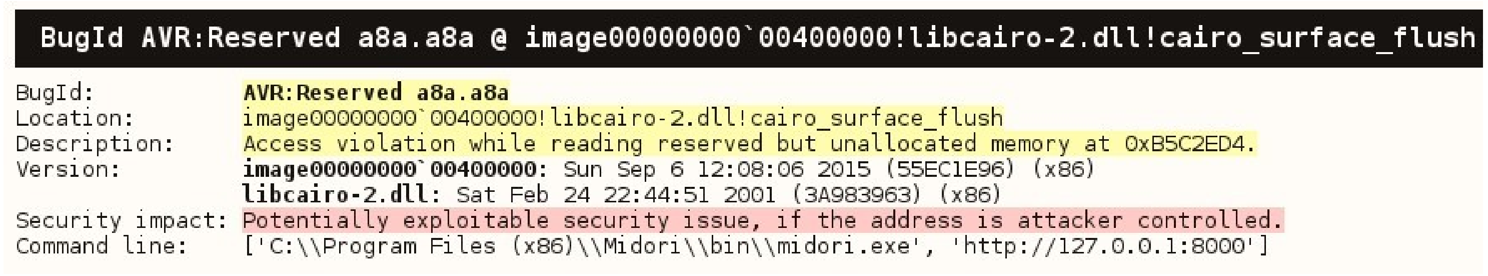
```
BugId AVE:Unallocated 679.f6f @ firefox.exe!xul.dll!NS_LogCOMPtrAddRef summary

BugId:           AVE:Unallocated 679.f6f
Location:        firefox.exe!xul.dll!NS_LogCOMPtrAddRef
Description:     Access violation while executing unallocated memory at 0x3A656764.
Version:         firefox.exe: 50.0.2.6177 (x86)
                 xul.dll: 50.0.2.6177 (x86)
Security impact: Potentially exploitable security issue, if the attacker can control the address or the memory at the address.
Command line:    ['C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe', '--no-remote', '-profile', 'C:\\Users\\user\\AppData
                 \\Local\\Temp\\Firefox-profile', 'http://127.0.0.1:8000/']
```



```
BugId AppExit 46c.46c @ microsoftedgecp.exe!edgecontent.dll!IERefreshElevationPolicy

BugId:           AppExit 46c.46c
Location:        microsoftedgecp.exe!edgecontent.dll!IERefreshElevationPolicy
Description:     Fatal application error, possibly a pure virtual function call (R6025)
Version:         microsoftedgecp.exe: 11.0.14393.82 (x64)
                 EdgeContent.dll: 11.0.14393.576 (x64)
Security impact: Potentially exploitable security issue
```



```
BugId AVR:Reserved a8a.a8a @ image00000000`00400000!libcairo-2.dll!cairo_surface_flush

BugId:           AVR:Reserved a8a.a8a
Location:        image00000000`00400000!libcairo-2.dll!cairo_surface_flush
Description:     Access violation while reading reserved but unallocated memory at 0xB5C2ED4.
Version:         image00000000`00400000: Sun Sep 6 12:08:06 2015 (55EC1E96) (x86)
                 libcairo-2.dll: Sat Feb 24 22:44:51 2001 (3A983963) (x86)
Security impact: Potentially exploitable security issue, if the address is attacker controlled.
Command line:    ['C:\\Program Files (x86)\\Midori\\bin\\midori.exe', 'http://127.0.0.1:8000']
```

Conclusions

- Large number of HTML tags and attributes are rarely found in online web pages
- HTML spec and grammar developed give coverage gains for both file format and DOM fuzzing of browsers
- Found 12 new bugs in browsers with this
- Add CSS and JavaScript to the grammar