Efficient Browser Identification with JavaScript Engine Fingerprinting
Philipp Reschl, Martin Mulazzani, Markus Huber, Edgar Weippl
SBA Research, Vienna

Browser Fingerprinting

Motivation
- Web browsers become more and more powerful & prevalent
- Revenue generated from and with web browsers increases steadily (e.g., cloud computing, online advertisements)
- User Agent String can be easily modified
- Similar to nmap for OS fingerprinting
- Goal: fast and reliable fingerprinting method

Legit and illegit use cases
1. User tracking: Detect different browsers behind NAT
2. Privacy: Decrease anonymity set in Tor
3. Attack surface: Malware and drive-by downloads
4. Appearance: To render websites correctly
5. Hijacking detection: Change of web browser during session

JavaScript Basics (Browsers, Engines, Features)

Fingerprinting Information
- JavaScript engines vary in features and performance
  - JavaScript compilation
  - GPU utilization
- Steadily decreasing time for new major versions
- We use ECMAScript (JavaScript) conformance tests:
  - Google Sputnik (http://sputnik.googlelabs.com)
  - ecmascript test262 (http://test262.ecmascript.org)
- Related work uses time execution patterns (see Mowery et al., W2SP 2011)

Current Browsers and ECMAScript Compliance
- ecmascript test262 includes Sputnik tests
- 11108 test cases, retrieved 2011-12-01:

<table>
<thead>
<tr>
<th>Webbrowser</th>
<th>Engine</th>
<th># of failed tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla Firefox</td>
<td>SpiderMonkey</td>
<td>164</td>
</tr>
<tr>
<td>Microsoft IE 9</td>
<td>Chakra</td>
<td>394</td>
</tr>
<tr>
<td>Google Chrome V8</td>
<td></td>
<td>418</td>
</tr>
<tr>
<td>Apple Safari Nitro</td>
<td></td>
<td>851</td>
</tr>
<tr>
<td>Opera Carakan</td>
<td></td>
<td>3821</td>
</tr>
</tbody>
</table>

Error Distribution with Sputnik in Detail

Our Implementation for Browser Identification
- We used the open source Sputnik testsuite (5246 tests)
- At best, every test uniquely identifies a particular web browser
- Tradeoff needed:
  - Number of tests increases with number of supported browsers
  - Too few tests could result in false-positives
- Manual selection of 10 erroneous tests for each web browser
- Linear ranking function for final results with conservative thresholds
- Ground truth used for verification:
  - UA-String
  - User manually identifies his/her browser

Results Experiment

Representative Survey with 189 participants
- Optimized for most prevalent web browsers:
  - Chrome, IE 8 & 9, Firefox
  - test.js 24KB in size
- Results:
  - 90 ms on average for PC
  - 200 ms on Smartphones
  - Multiple runs prevented by technical means

Results

<table>
<thead>
<tr>
<th>Result</th>
<th>Percentage</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection of supported web browsers</td>
<td>92.59 %</td>
<td>175</td>
</tr>
<tr>
<td>Unsupported web browsers (e.g., smartphones)</td>
<td>7.04 %</td>
<td>14</td>
</tr>
<tr>
<td>Correctly identified supported web browsers</td>
<td>100 %</td>
<td>174</td>
</tr>
<tr>
<td>Correctly identified UA-String manipulation</td>
<td>100 %</td>
<td>1</td>
</tr>
<tr>
<td>User survey error</td>
<td>7.94 %</td>
<td>15</td>
</tr>
</tbody>
</table>

No false-positives in the survey, 100% correct detection!

Conclusion

Future Research
- Increase number of web browsers in the test suite
- Enrich fingerprinting with HTML5 and other features
- Build a privacy-preserving normalizing proxy
- Further research if this is already used by advertisement companies or malware authors

Conclusion
- JavaScript engine fingerprinting can be done fast and efficiently
- Test survey among 189 users showed feasibility
- Runtime < 100ms
- Test suite very small, only 24KB in size
- No false-positives