Efficient Browser Identification with JavaScript Engine Fingerprinting

Philipp Reschl, Martin Mulazzani, Markus Huber, Edgar Weippl
SBA Research, Vienna

secure sba-research.org

Browser Fingerprinting

Motivation

- Webbrowsers become more and more powerful & prevalent
- Revenue generated from and with webbrowsers 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 webbrowser 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)

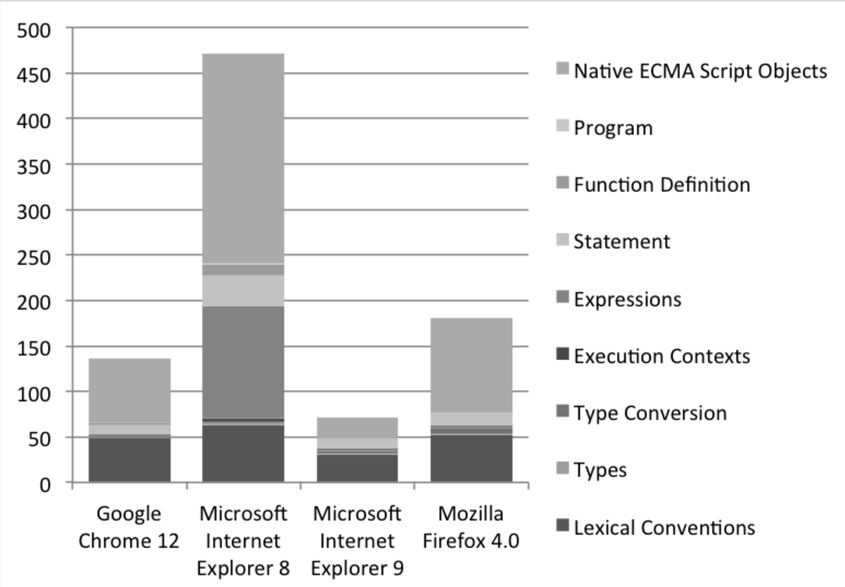
Current Browsers and ECMAScript Compliance

- ecmascript test262 includes Sputnik tests
- 11108 test cases, retrieved 2011-12-01:

Webbrowser	Engine	# of failed tests
Mozilla Firefox	SpiderMonkey	164
Microsoft IE 9	Chakra	394
Google Chrome	V8	418
Apple Safari	Nitro	851
Opera	Carakan	3821

JavaScript Engine Identifikation

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 webbrowser
- 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 webbrowser
- 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 webbrowsers:
  - 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

Result	Percentage	Number
Detection of supported webbrowsers	92.59 %	175
Unsupported webbrowsers (e.g., smartphones)	7.04 %	14
Correctly identified supported webbrowsers	100 %	174
Correctly identified UA-String manipulation	100 %	1
User survey error	7.94 %	15

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

Conclusion

Future Research

- Increase number of webbrowsers in the testset
- 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
- Testsurvey among 189 users showed feasability
- Runtime $<$ 100ms
- Testsuite very small, only 24KB in size
- No false-positives