Detection of Browser Fingerprinting by Static JavaScript Code Classification

Sjors Haanen & Tim van Zalingen

UvA

February 6, 2018

Supervisors (KPMG):
Aidan Barrington & Ruben de Vries

Research Project 82

Tracking users on the Web

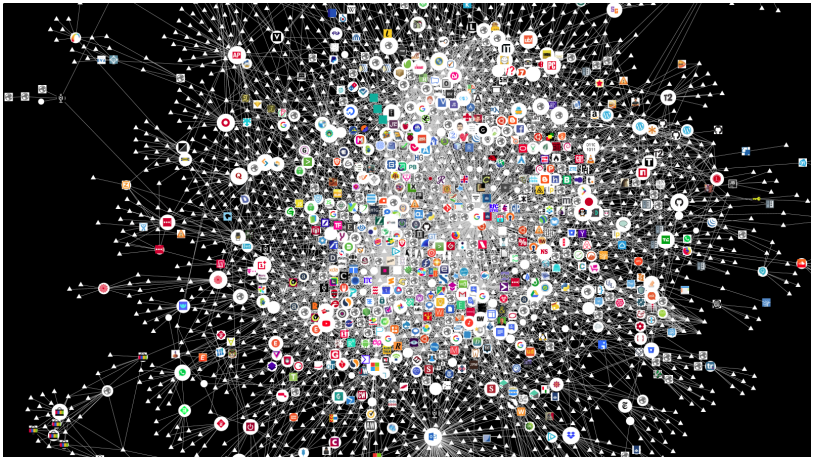


Figure 1: Third party cookies
source: Mozilla - Lightbeam for Firefox

Browser fingerprinting

- Browser settings
- Hardware characteristics
- OS characteristics

} Unique fingerprint

- Stateless
- Often even unnoticed by user
- Recent study could uniquely identify 89.4% out of 118,934 browsers[1]

[1] Laperdrix, Pierre 2017.

Fingerprint example

Table 1: Excerpt fingerprinting results from *https://amiunique.org*

Attribute	**Similarity ratio**	**Value**
User agent	<0.1%	"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:58.0) Gecko..."
Accept	54.78%	"text/html,application/xhtml+xml,application/..."
Content encoding	40.54%	"gzip, deflate, br"
Content language	27.53%	"en-US,en;q=0.5"
List of plugins	25.61%	""
Platform	10.64%	"Linux x86_64"
Cookies enabled	79.63%	"yes"
Do Not Track	30.51%	"yes"
Timezone	20.66%	"-60"
Screen resolution	21.29%	"1920x1080x24"

Defences against browser fingerprinting

- Disable functionality
- N:1 - Many Browsers, One Configuration (Tor)
- 1:N - One Browser, Many Configurations
  - Randomise data per request/session

Motivation

- **Privacy**
- Existing detection and prevention solutions often criticised

Prior work

Previous attempts to detect fingerprinting:

- Blacklists[2]
- Dynamic analysis: detection at runtime[3]
- Static analysis: counting[4]

[2] Kontaxis, Georgios and Chew, Monica 2015.

[3] Acar, Gunes and Juarez, Marc and Nikiforakis, Nick and Diaz, Claudia and Gürses, Seda and Piessens, Frank and Preneel, Bart 2013; FaizKhademi, Amin and Zulkernine, Mohammad and Weldemariam, Komminist 2015.

[4] Rausch, Michael and Good, Nathan and Hoofnagle, Chris Jay 2014.

Research question

Can the action of browser fingerprinting be detected before execution by analysing JavaScript code with machine learning?

Method overview

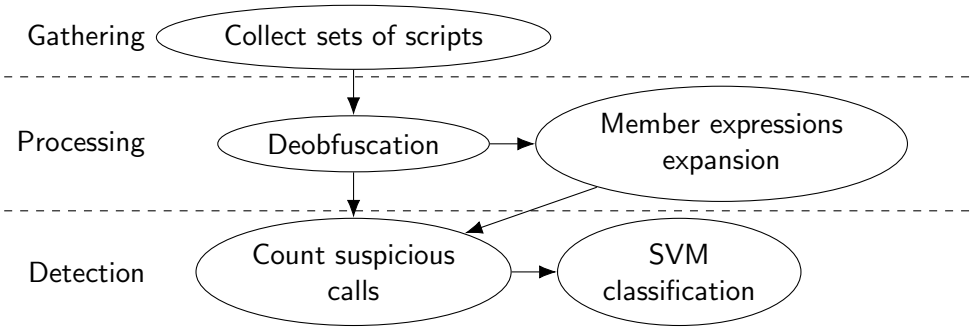
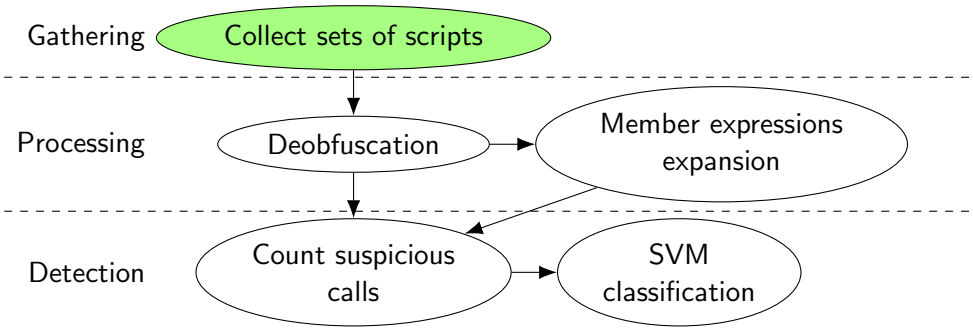


Figure 2: Process of analysing JavaScript (JS) source code for a given set of websites to find fingerprinting practices

Method overview

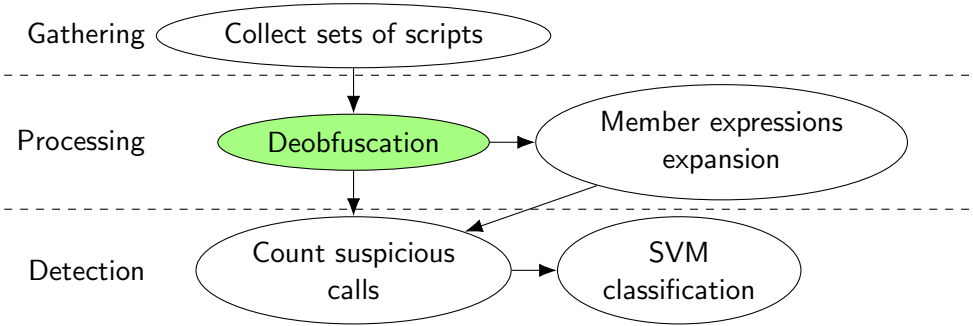


Collect sets of scripts

Predefined sets (by manual search):

- Set of 12 fingerprinting scripts
- Set of 20 non-fingerprinting scripts

Method overview



Deobfuscation: The problem

```
eval(function(p,a,c,k,e,d){e=function(c){return c.toString(36)};
if(!''.replace(/^/,String)){while(c--){d[c.toString(a)]=k[c]
||c.toString(a)}k=[function(e){return d[e]}];
e=function(){return'\\w+'};
c=1};while(c--){if(k[c]){p=p.replace(new RegExp(
'\\b'+e(c)+'\\b','g'),k[c])}}return p}('0 1=3;8 4(){0 a=1.2;
0 b=a;0 5=b.6;0 7=1.9}',12,12,'var|nav|plugins|navigator|
fingerprint|c|length|d|function|userAgent||'.split('|'),0,{}))
```

Figure 3: An example of JS code obfuscated by *www.danstools.com/javascript-obfuscate/*

Who can tell us what this piece of code does?

Deobfuscation: JSBeautifier

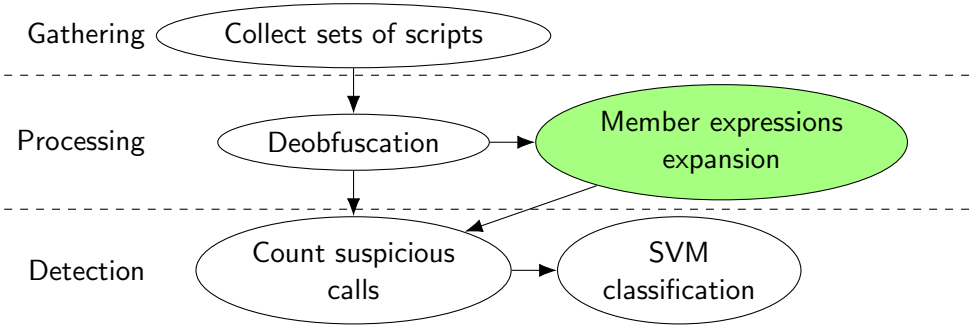
Requirements:

- Counter obfuscation
- Counter minification
- Counter packing

```
var nav = navigator;
function fingerprint() {
    var a = nav.plugins;
    var b = a;
    var c = b.length;
    var d = nav.userAgent
}
```

Figure 4: The JS code in figure 3 deobfuscated by *http://jsbeautifier.org/*

Method overview



Expanding member expressions: The problem

```
var nav = navigator;
function fingerprint() {
  var a = nav.plugins;
  var b = a;
  var c = b.length;
  var d = nav.userAgent;
}
```

Figure 5: Example JS code with split member expressions

```
navigator.plugins
navigator.plugins.length
navigator.userAgent
```

Figure 6: Expanded member expressions for the code in figure 5

Expanding member expressions: Abstract Syntax Tree (AST)

- Parse code
- Traverse AST
- Analyse scope

```
var nav = navigator;
function fingerprint() {
  var a = nav.plugins;
}
```

Figure 7: Example JS code with split member expressions

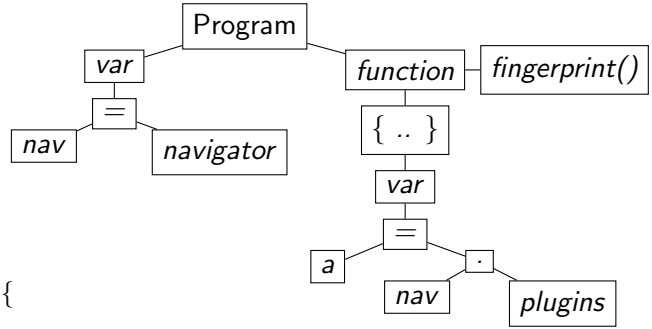
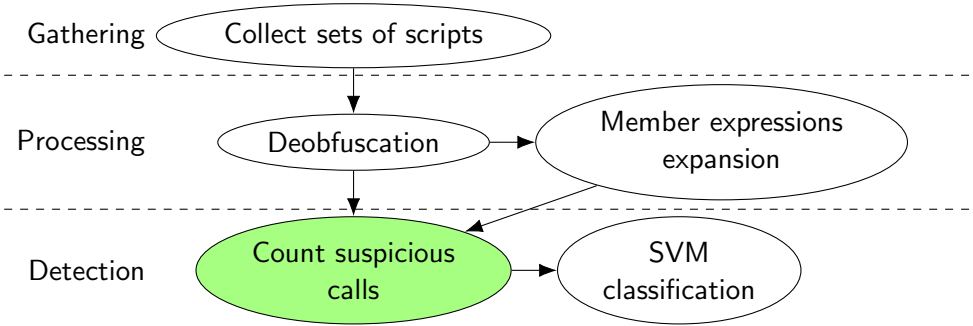


Figure 8: The Abstract Syntax Tree of the code in figure 7

Method overview



Count suspicious calls

Counting calls in processed files aggregated per domain

Examples of suspicious JS calls:

- navigator.userAgent
- navigator.plugins.name
- navigator.javaEnabled()
- window.screen.colorDepth
- Date().getTimezoneOffset()

Inspecting JS calls

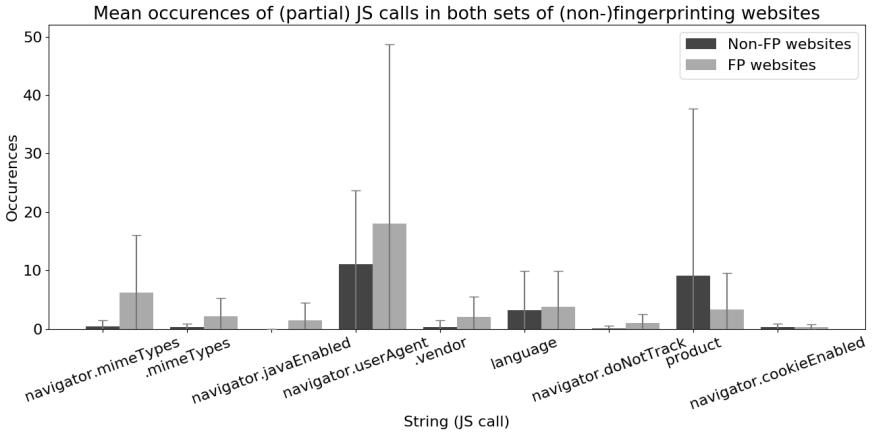
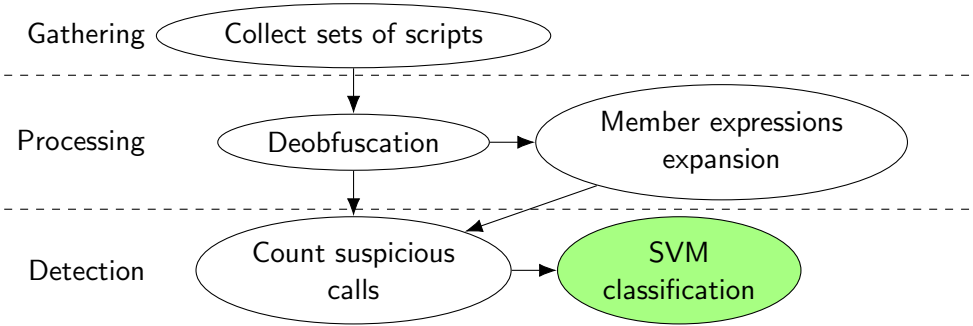


Figure 9: Comparing different JS calls that can be used as a feature to differentiate scripts

Method overview



Support Vector Machine (SVM)

- Supervised learning methods
- Classification
- Relevant advantages:
  - Effective in high dimensional spaces
  - Effective with more dimensions than samples
- Avoid over-fitting with small number of samples

2D SVM Classification

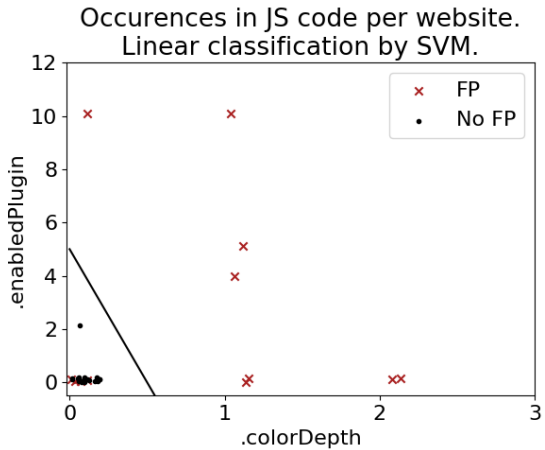


Figure 10: SVM Classification example for two features

2D SVM Classification (Cont'd)

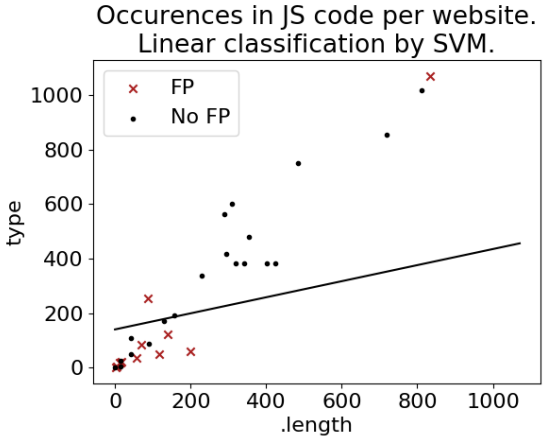


Figure 11: SVM Classification example for two features. These two features are not easily distinguishable

Support Vector Machine: Prevent overfitting

- Partition data into training and test set
- Cross-validation
- Stratified k-fold preserves positive and negative ratio

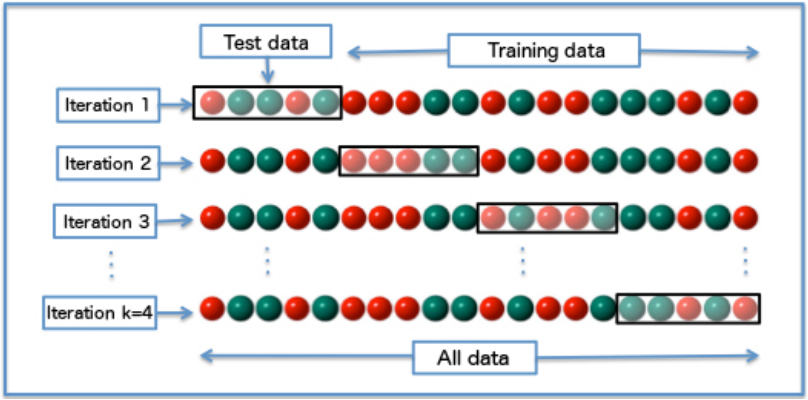


Figure 12: Visualised example of k-fold cross-validation with k=4 (*source: Wikipedia - Cross-validation (statistics)*)

Results: Full dimensional classification

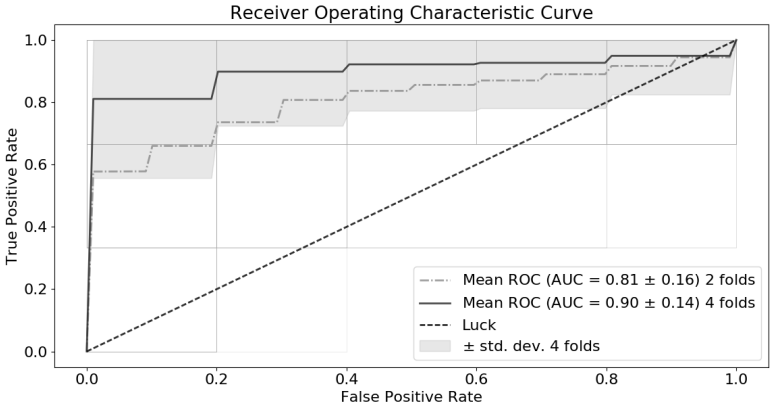


Figure 13: Receiver Operating Characteristic curve to illustrate the performance of the classifier F_1-score=0.80

Discussion

Observable difference, SVM can detect fingerprinting scripts

- Combining features and using a classifier improves on earlier research
- Future implementation of proposed method might aid in detection
- False positives

Future work

- Refine list of suspicious JS calls
- Include other signs of fingerprinting in the analysis, e.g.:
  - Hashing values
  - Sending fingerprintable data to a remote server
- Bigger dataset
- Other machine learning algorithms

References I

- Acar, Gunes and Juarez, Marc and Nikiforakis, Nick and Diaz, Claudia and Gürses, Seda and Piessens, Frank and Preneel, Bart (2013). "FPDetective: dusting the web for fingerprinters". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1129–1140.
- FaizKhademi, Amin and Zulkernine, Mohammad and Weldemariam, Komminist (2015). "FPGuard: Detection and prevention of browser fingerprinting". In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 293–308.
- Kontaxis, Georgios and Chew, Monica (2015). "Tracking protection in firefox for privacy and performance". In: *arXiv preprint arXiv:1506.04104*.
- Laperdrix, Pierre (2017). "Browser Fingerprinting: Exploring Device Diversity to Augment Authentication and Build Client-Side Countermeasures". PhD thesis. INSA Rennes.

References II

Rausch, Michael and Good, Nathan and Hoofnagle, Chris Jay (2014). "Searching for Indicators of Device Fingerprinting in the JavaScript Code of Popular Websites". In: *Proceedings, Midewest Instruction and Computing Symposium*.