Formal Security Analysis of Mobile and Web Applications

Matteo Maffei

12/7/2016

Purdue University
Coming here…
Coming here…
Coming here…

<table>
<thead>
<tr>
<th>Desktop</th>
<th>Mobile</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="British Airways logo" /></td>
<td><img src="image2.png" alt="Booking.com logo" /></td>
<td><img src="image3.png" alt="Checkmark" /></td>
</tr>
</tbody>
</table>
Coming here…

Desktop | Mobile | Web

- | - | -

- | - | -

- | - | -

- | - | -

- | - | -

- | - | -

- | - | -

- | - | -
Coming here…

BRITISH AIRWAYS

Booking.com

Desktop   Mobile   Web

[Checkmarks for each platform]
Coming here…

<table>
<thead>
<tr>
<th>Desktop</th>
<th>Mobile</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="logo.png" alt="British Airways" /></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><img src="logo.png" alt="Booking.com" /></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><img src="logo.png" alt="Google Maps" /></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><img src="logo.png" alt="Weather.com" /></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><img src="logo.png" alt="Deutsche Bank" /></td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Coming here…
Great work on *practical*, best-effort, security-enhancement solutions (sandboxing, bug finding, etc.)

much less attention to *provable* security guarantees…

Can we get the best of the two worlds?
HornDroid: Practical and Sound Static Analysis of Android Applications by SMT Solving

Stefano Calzavara
Università Ca’ Foscari Venezia
calzavara@dais.unive.it

Ilya Grishchenko
CISPA, Saarland University
grishchenko@cs.uni-saarland.de

Matteo Maffei
CISPA, Saarland University
maffei@cs.uni-saarland.de

Abstract—We present HornDroid, a new tool for the static analysis of information flow properties in Android applications. The core idea underlying HornDroid is to use Horn clauses for soundly abstracting the semantics of Android applications and to express security properties as a set of proof obligations that are automatically discharged by an off-the-shelf SMT solver. This approach makes it possible to fine-tune the analysis in order to achieve a high degree of precision while still using off-the-shelf verification tools, thereby leveraging the recent advances in this field. As a matter of fact, HornDroid outperforms state-of-the-art Android static analysis tools on benchmarks proposed by the community. Moreover, HornDroid is the first static analysis tool for Android to come with a formal proof of soundness, which covers the core of the analysis technique: besides yielding correctness assurances, this proof allowed us to identify some critical corner-cases that affect the soundness guarantees provided by some of the previous static analysis tools for Android.

1. Introduction

The Android platform is by far the most popular choice for mobile devices nowadays, with billions of applications routinely installed on a massive number of different phones and tablets. Given this increasing popularity, personal information and other sensitive data stored on Android devices constitute an attractive target for breaching users’ privacy at scale by malicious application developers. Information flow control frameworks for Android have thus emerged as a prominent research direction, with several different proposals spanning from dynamic analysis [11], [19], [34], [17] to static analysis [39], [38], [24], [14], [20], [23], [2], [15], [21]. Static analysis is particularly appealing for information flow control, given its ability to provide full coverage of all the possible execution paths and the possibility to be employed in the vetting phase, i.e., before the application is uploaded to the Google Play store.

The most recent works in this area [2], [15], [21], [36] are impressive in their efforts to support a significant fragment of the Android platform. Most of them leverage existing static analysers by encoding Android applications in a suitable format, e.g., FlowDroid [2], DroidSafe [15], and lccTA [21] use Soot [35], while CHEX [23] uses Wala [13].

Observing that existing static analysers come with intrinsic limitations that limit the precision of the analysis (e.g., Soot and Wala do not calculate all objects’ points-to information in a both flow- and context-sensitive way), Amandroid [36] relies on a dedicated data-flow analysis algorithm.

Despite all this progress and sophisticated machinery, none of these tools achieves a satisfactory degree of soundness: even on benchmarks written by the community and consisting of simple programs (i.e., Droidbench [2]), for which the ground truth is known, all existing tools miss several malicious leaks (false negatives). This, along with the fact that none of these tools comes with a formal model or soundness proof, makes one wonder how accurately these analyses capture all the subtleties of the Android execution model, which is far from being trivial [26], and to which extent their results are reliable on real-life applications, for which the ground truth is not known.

Furthermore, the lack of precise and fully documented analysis definitions complicates the comparison between different approaches: for instance, there is no universal agreement on a single notion of object-sensitivity [28], though object-sensitivity has been recognized as crucial to support a precise analysis of real-world Android applications [2]. Hence, at the time of writing, the only way to grasp the relative strengths and weaknesses of different static analysis tools for Android applications relies on a hands-on testing on some common benchmark and a source code inspection of their implementation.

Our contributions. We present a fresh approach to the static analysis of Android applications, i.e., a data-flow analysis based on Horn clause resolution [5]. The core idea is to soundly abstract the semantics of Android applications into a set of Horn clauses and to formulate security properties as a set of proof obligations, which can be automatically discharged by off-the-shelf SMT solvers. In particular:

- We prove the soundness of our analysis against a rigorous formal model of a large fragment of the Android ecosystem, covering Dalvik bytecode, the event-driven nature of the activity lifecycle, and inter-component communication. While elaborating the proof, we identified a few critical corner-cases that affect the soundness guarantees provided by some of previous static analysis tools for Android.
Analysis of mobile apps
Analysis of mobile apps

- Services that cost you money directly call phone numbers
- Your location: coarse (network-based) location, fine (GPS) location
- Network communication: full Internet access
- Your accounts: Google Maps, manage the accounts list, use the authentication credentials of an account
- Storage: modify/delete USB storage contents

Install | Cancel

<= 5.0

6.0 (Marshmallow)
Analysis of mobile apps

≤ 5.0

6.0 (Marshmallow)

Coarse-grained permissions: easy to understand, but not informative enough
(e.g., Internet permission: to download map or to leak my position?)
Technical challenges
Technical challenges

Bytecode + Native (no source code)
Technical challenges

Bytecode + Native  
(no source code)

Multithreading  
(shared memory)
Technical challenges

Bytecode + Native
(no source code)

Multithreading
(shared memory)

Intents
(message passing)
Technical challenges

Bytecode + Native
(no source code)

Multithreading
(shared memory)

Intents
(message passing)

User-driven lifecycle
(multiple entrypoints)
AppGuard [TACAS’13, DPM’13]

• First dynamic monitoring tool that does not require rooting the phone or modifying the firmware
• Allows users to dynamically grant or remove fine-grained permissions
• Based on inline reference monitoring
  • App rewriting in order to wrap sensitive calls
• Technology transfer
  • Maintained by a spin-off, installed on more than 3 million devices in Germany
From dynamic monitoring to static analysis

• Dynamic monitoring stops attacks but…
  ✓ Best effort, no security guarantee
  ✘ Monitoring of the current execution, unsuitable for vetting purposes
  ✘ Run-time overhead (often negligible, but still)
  ✘ Possible crashes

• Static analysis
  ✓ Security by design (security guarantees for any possible program executions)
  ✓ Useful to prevent malicious apps from even entering into stores or being downloaded on user devices
Prior work

• Lots of research in the last two years. Typically,
  • Encode Dalvik bytecode into an intermediate representation compatible with existing static analyzers for Java (e.g., Soot or Wala)
  • Taint analysis (direct flows from sensitive sources to public sinks, e.g., address book to the Internet)

• Shortcomings:
  • Rather inflexible, tailored to one specific analyzer: not easy to refine the precision of the analysis or to handle different properties
    E.g., the standard taint query “is the credit card sent over the Internet?” is not very informative, better a value-sensitive query like “is the credit card sent only over https to a trusted website?”
  • No formal specification or proof of soundness
    What does the analysis actually handle?
    Are the reported results trustworthy?
  • Unsatisfactory performance and scalability
HornDroid [EuroS&P’16]

- **Predicate abstraction:**
  - Semantics of app’s bytecode abstracted into a set of Horn clauses
  - Security property as a query (reachability analysis), automatically verified with Z3 (state-of-the-art SMT solver)

- **Precision**
  - Value-, flow-, and context-sensitive

- **Performance**
  - Fast, scales to real life apps

- **Flexibility and expressiveness**
  - can leverage any SMT solver
  - refine precision by tweaking Horn clause generation, without touching SMT solver
  - can handle any FOL query

- **Formal proof of soundness** within a precise semantic model of the Android ecosystem
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c’ m’ r_3 , r_2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c' m' r_3, r_2
13 ...

// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

\[
R_{c,m,12}(args_{\text{caller}},v_1,\ldots,v_k,res_{old}) \Rightarrow R_{c',m',0}(r_3,r_2;0,\ldots,0,r_3,r_2;0)
\]

\[
R_{c',m}(args_{\text{caller}},res_{\text{new}}) \Rightarrow R_{c,m,13}(args_{\text{caller}},v_1,\ldots,v_k,res_{\text{new}})
\]
A sneak peek into the abstraction: method invocation

```
// c.m
...
12 invoke c' m' r_3, r_2,
13 ...
```

```
// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return
```

Register abstraction per program counter (flow sensitivity)

\[ R_{c,m,12}(\text{args}_{\text{caller}}, v_1, ..., v_k; \text{res}_{\text{old}}) \Rightarrow R_{c';m',0}(r_3, r_2; 0, ..., 0, r_3, r_2; 0) \]

\[ R_{c',m}(\text{args}_{\text{caller}}, \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, ..., v_k; \text{res}_{\text{new}}) \]
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c' m' r_3, r_2
13 ...

// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

Arguments at call time (context sensitivity)

R_{c,m,12}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; \text{res}_{\text{old}}) \Rightarrow R_{c';m',0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0)

R_{c',m}(\text{args}_{\text{caller}}; \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; \text{res}_{\text{new}})

Register abstraction per program counter (flow sensitivity)
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c’ m’ r_3, r_2
13 ...
12 invoke c’ m’ r_3, r_2

Arguments at call time
(context sensitivity)

Content of registers
(value sensitivity)

Register abstraction
per program counter
(flow sensitivity)

// c’..m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

R_{c,m,12}(args_{caller}, v_1, ..., v_k, res_{old}) \Rightarrow R_{c’,m’,0}(r_3, r_2, 0, ..., 0, r_3, r_2, 0)

R_{c’,m}(args_{caller}, res_{new}) \Rightarrow R_{c,m,13}(args_{caller}, v_1, ..., v_k, res_{new})
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c’ m’ r₃, r₂
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r₁ r₂
28 return

Arguments at call time (context sensitivity)
Content of registers (value sensitivity)
Content of result register

Register abstraction per program counter (flow sensitivity)

\[
R_{c,m,12}(\text{args}_{\text{caller}}, v₁, ..., vₖ; r; \text{res}_\text{old}) \Rightarrow R_{c’;m’;0}(r₃, r₂; 0, ..., 0, r₃, r₂; 0)
\]

\[
R_{c’;m}(\text{args}_{\text{caller}}; r; \text{res}_\text{new}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}; v₁, ..., vₖ; \text{res}_\text{new})
\]
A sneak peek into the abstraction: method invocation

```
// c.m
...
12 invoke c' m' r_3, r_2
13 ...
```

Arguments at call time (context sensitivity)

```
R_{c,m,12}(args_{caller}; v_1, ..., v_k; res_{old}) \Rightarrow R_{c',m',0}(r_3, r_2; 0, ..., 0, r_3, r_2; 0)
```

Content of registers (value sensitivity)

```
R_{c',m}(args_{caller}; res_{new}) \Rightarrow R_{c,m,13}(args_{caller}; v_1, ..., v_k; res_{new})
```

Content of result register

Register abstraction per program counter (flow sensitivity)

```
// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return
```
A sneak peek into the abstraction: method invocation

// c.m
...
12 invoke c’ m’ r₃ , r₂
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r₁ r₂
28 return

Arguments at call time (context sensitivity)
Content of registers (value sensitivity)
Content of result register

Register abstraction per program counter (flow sensitivity)

Rₐ,m,₁₂(argscaller,v₁,..., vₗ,resold) ⇒ Rₐ’,m’,₀(r₃,r₂; 0,...0,r₃,r₂; 0)
Rₐ’,m(argscaller; resnew) ⇒ Rₐ,m,₁₃(argscaller; v₁,..., vₗ; resnew)

Update register result at return time

Rₐ’,m’,₂₄(regsc,v₁,...) ⇒ Rₐ’,m’,₂₇(regsc,v₁,...)

Rₐ’,m’,₂₇(regsc,v₁,v₂,...) ⇒ Rₐ’,m’,₂₈(regsc,v₁,v₁,...)
Rₐ’,m’,₂₈(regsc; v₁,...; vres) ⇒ Rₐ’,m(regsc; vres)
A sneak peek into the abstraction: method invocation

// c'.m
...
12 invoke c' m' r_3, r_2
13 ...

// c.m
...
12 invoke c' m' r_3, r_2
13 ...

Arguments at call time (context sensitivity)

Content of registers (value sensitivity)

Content of result register

Register abstraction per program counter (flow sensitivity)

Update register result at return time

R_{c,m,12}(args_{caller}; v_1, ..., v_k; res_{old}) \Rightarrow R_{c',m',0}(r_3, r_2; 0, ..., 0, r_3, r_2; 0)

R_{c',m}(args_{caller}; res_{new}) \Rightarrow R_{c,m,13}(args_{caller}; v_1, ..., v_k; res_{new})

R_{c',m,24}(reg_{c}; v_1, ...) \Rightarrow R_{c',m,27}(reg_{c}; v_1, ...)

... ...

R_{c',m,27}(reg_{c}; v_1, v_2, ...) \Rightarrow R_{c',m,28}(reg_{c}; v_1, v_1, ...)

R_{c',m,28}(reg_{c}; v_1, ..., v_{res}) \Rightarrow R_{c',m}(reg_{c}; v_{res})

Set result register at return time
Results on DroidBench

<table>
<thead>
<tr>
<th></th>
<th>HornDroid</th>
<th>FlowDroid</th>
<th>DroidSafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness tp/(tp+fn)</td>
<td>100</td>
<td>67</td>
<td>93</td>
</tr>
<tr>
<td>Precision tn/(tn+fp)</td>
<td>94</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>Average analysis time</td>
<td>1s</td>
<td>19s</td>
<td>176s</td>
</tr>
</tbody>
</table>

Take Home Message: 
*Sound* (within a precise semantic model), and yet more *precise* and orders of magnitude *faster*!
Ongoing and future work
Ongoing and future work

- Implicit flows
  - Encoding of non-interference in Horn clauses (for handling implicit flows)
Ongoing and future work

• Implicit flows
  • Encoding of non-interference in Horn clauses (for handling implicit flows)

• Fine-tune the abstraction to find the sweet spot between precision and efficiency
  • flow-sensitive heap abstraction
Ongoing and future work

- Implicit flows
  - Encoding of non-interference in Horn clauses (for handling implicit flows)
- Fine-tune the abstraction to find the sweet spot between precision and efficiency
  - flow-sensitive heap abstraction
- Automated attack reconstruction
Ongoing and future work

- Implicit flows
  - Encoding of non-interference in Horn clauses (for handling implicit flows)
- Fine-tune the abstraction to find the sweet spot between precision and efficiency
  - flow-sensitive heap abstraction
- Automated attack reconstruction
- Handle sources of unsoundness (currently, out of the model)
  - native code, threads, reflection
Micro-Policies for Web Session Security

Stefano Calzavara
Università Ca' Foscari Venezia
calzavara@dais.unive.it
Riccardo Focardi
Università Ca' Foscari Venezia
focardi@dais.unive.it
Niklas Grimm
Saarland University
grimm@cs.uni-saarland.de
Matteo Maffei
Saarland University
maffei@cs.uni-saarland.de

Abstract—Micro-policies, originally proposed to implement hardware-level security monitors, constitute a flexible and general enforcement technique, based on assigning security tags to system components and taking security actions based on dynamic checks over these tags. In this paper, we present the first application of micro-policies to web security, by proposing a core browser model supporting them and studying their effectiveness at securing web sessions. In our view, web session security requirements are expressed in terms of a simple, purely declarative information flow policy, which is then automatically translated into a micro-policy implementing it. This leads to a browser-side enforcement mechanism which is elegant, sound and flexible, while being accessible to web developers. We show how a large class of attacks against web sessions can be uniformly and effectively prevented by the adoption of this approach. Since we carefully design micro-policies with ease of deployment in mind, we are also able to implement our proposal as a Google Chrome extension. Our experiments show that Microchrome can be easily configured to enforce strong security policies without breaking the websites functionality.

1. INTRODUCTION

The Web is nowadays the primary means of access to a plethora of online services with strict security requirements. Electronic health records and online statements of income are a well-established reality as of now, and more and more security-sensitive services are going to be supplied online in the next few years. Despite the critical importance of securing these online services, web applications and, more specifically, web sessions are notoriously hard to protect, since they can be attacked at many different layers.

At the network layer, man-in-the-middle attacks can break both the confidentiality and the integrity of web sessions running (at least partially) over HTTP. The standard solution against these attacks is deploying the entire web application over HTTPS with trusted certificates and, possibly, making use of HSTS [17] to prevent subtle attacks like SSL stripping. At the session implementation layer, code injection attacks (or again network attacks) can be exploited to steal authentication cookies and hijack a web session, or to compromise the integrity of the cookie jar and mount dangerous attacks like session fixation [19]. This is particularly problematic because, though the standard HtppOnly and Secure cookie attributes [3] are effective at protecting cookie confidentiality, no effective countermeasure exists as of now to ensure cookie integrity on the Web [32]. Finally, web sessions can also be attacked at the application layer: for instance, since browsers automatically attach cookies set by a website to all the requests sent to it, cross-site request forgery (CSRF) attacks can be mounted by a malicious web page to harness the integrity of the user session with a trusted web application and inject attacker-controlled messages inside it. Standard solutions against this problem include the usage of secret tokens and the validation of the Origin header attached by the browser to filter out malicious web requests [4].

In principle, it is possible to achieve a reasonable degree of security for web sessions using the current technologies, but the overall picture still exhibits several important shortcomings and it is far from being satisfactory. First, there are mechanisms like the HtppOnly cookie attribute which are easy to use, popular and effective, but lack flexibility: a cookie may either be HtppOnly or not, hence JavaScript may either be able to access it or be prevented from doing any kind of computation over the cookie value. There is no way, for instance, to let JavaScript access a cookie for legitimate computations, but at the cost of disciplining its communication behaviour to prevent the cookie leakage. Then, there are defenses which are sub-optimal and not always easy to implement: this is the case for the token-based protection against CSRF. Not only this approach must be directly implemented into the APIs of a web development framework to ensure that it is convenient to use, but also it is not very robust, since it fails in presence of code injection vulnerabilities which disclose the token value to the attacker. Finally, we observe that some attacks and attack vectors against web sessions are underestimated by existing standards and no effective solution against them can be deployed as of now: this is the case for many threats to cookie integrity [32]. These issues will likely be rectified with ad-hoc solutions in future standards, whenever browser vendors and web application developers become more concerned about their importance, and find a proper way to patch them while preserving the compatibility with existing websites.

In this paper, we advocate that a large class of attacks harming the security of web sessions can be provably, uniformly, and effectively prevented by the adoption of browser-enforced security policies, reminiscent of a dynamic typing discipline for the browser. In particular, we argue for the adoption of micro-policies [14] as a convenient tool to improve the security of web sessions, by disciplining the browser behaviour when interacting with security-sensitive web applications. Roughly, the specification of a micro-policy involves: (1) the definition of a set of tags, used to label selected elements of the web ecosystem, like URLs, cookies, network connections, etc., and (2) the definition of a transfer function, defining which operations are permitted by the browser based on the tags and how tags are assigned to browser elements after a successful operation. This kind of security policies has already proved
Introduction

Web gives access to many security sensitive online services
Introduction

Web gives access to many security sensitive online services.

Squashed bug opened EVERY PayPal account to hijacking

Yet another tale of incredibly crooked software
Web Browser as a Reactive System
Web Browser as a Reactive System

① load(https://www.facebook.com)
Web Browser as a Reactive System

1. `load(https://www.facebook.com)`

2. `request(https://www.facebook.com; )`
Web Browser as a Reactive System

1. load(https://www.facebook.com)

2. request(https://www.facebook.com; )

3. response(https://www.facebook.com; ; xhr(http://facebook.com))
Web Browser as a Reactive System

1. load(https://www.facebook.com)
2. request(https://www.facebook.com;)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com; )
Web Browser as a Reactive System

Input Stream

1. load(https://www.facebook.com)

2. ?

3. response(https://www.facebook.com; ; xhr(http://facebook.com))

Output Stream

2. request(https://www.facebook.com; )

4. request(http://www.facebook.com; )
Indistinguishable inputs should always produce indistinguishable outputs
Reactive Non-Interference
Reactive Non-Interference
Reactive Non-Interference

- Confidentiality: High confidentiality inputs do not influence low confidentiality outputs
Reactive Non-Interference

- Confidentiality: High confidentiality inputs do not influence low confidentiality outputs
Reactive Non-Interference

- Confidentiality: High confidentiality inputs do not influence low confidentiality outputs
- Integrity: Low integrity inputs do not influence high integrity outputs
Reactive Non-Interference

- Confidentiality: High confidentiality inputs do not influence low confidentiality outputs
- Integrity: Low integrity inputs do not influence high integrity outputs
Cookie Leakage

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; xhr(http://facebook.com))
4. request(http://www.facebook.com; )

?
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; 
   xhr(http://facebook.com))
4. request(http://www.facebook.com)

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; 
   xhr(http://facebook.com))
4. request(http://www.facebook.com)
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com;)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com;)

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com;)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com;)

(requests and responses between Firefox and Facebook)

(requests and responses between Firefox and Facebook)

(requests and responses between Firefox and Facebook)

(requests and responses between Firefox and Facebook)
Cookie Leakage (Confidentiality)

Defined by desired policy

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com; )

Passive Network Attacker
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))

defined by desired policy

2. request(https://www.facebook.com)
4. request(http://www.facebook.com)

defined by attacker capabilities

Attacker
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)

Web Attacker at evil.com

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; unit)
4. load(https://www.evil.com)

Web Attacker at evil.com

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; unit)
4. load(https://www.evil.com)
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )

Web Attacker at evil.com
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)

Web Attacker
at evil.com

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
   xhr(https://www.facebook.com/like)
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. response(https://www.facebook.com; ; unit)
8. xhr(https://www.facebook.com/like)

Web Attacker at evil.com
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)  
3. response(https://www.facebook.com; , ; unit)  
4. load(https://www.evile.com)  
6. response(https://www.evile.com; , ; unit)

7. request(https://www.facebook.com/like; , )

Web Attacker at evil.com

Defined by attacker capabilities
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)

Defined by attacker capabilities

7. request(https://www.facebook.com/like; )

Defined by desired policy
Existing Defences

• Secure flag for cookies
  • Cookie may only be sent over https
  • Lack of integrity guarantees

• HttpOnly flag for cookies
  • Cookie may not be directly read/set/overwritten by scripts
  • Scripts can still set/overwrite cookies indirectly via network communication
  • Lack of flexibility

• CSRF Defences (secret tokens, origin/referer header validation)
  • Require server side modifications
  • Not very robust
Existing Defences

- Secure flag for cookies
  - Cookie may only be sent over https
  - Lack of integrity guarantees

- HttpOnly flag for cookies
  - Cookie may not be directly read/set/overwritten by scripts
  - Scripts can still set/overwrite cookies indirectly via network communication
  - Lack of flexibility

- CSRF Defences (secret tokens, origin/referer header validation)
  - Require server side modifications
  - Not very robust

Some reasonable defences exist, but
- Many have downsides
- Each defence targeted at very specific attack
Our Contributions

- Browser enforced security policies
  - Simple declarative policies
  - Uniform enforcement mechanism
  - Enforcement by monitor on interfaces
Our Contributions

• Browser enforced security policies
  • Simple declarative policies
  • Uniform enforcement mechanism
  • Enforcement by monitor on interfaces

• Formalisation of web browser as reactive system
  • Assign labels to elements of browser ecosystem
  • Define *transfer* function (i.e., monitoring rules) for these labels
Our Contributions

- Browser enforced security policies
  - Simple declarative policies
  - Uniform enforcement mechanism
  - Enforcement by monitor on interfaces

- Formalisation of web browser as reactive system
  - Assign labels to elements of browser ecosystem
  - Define *transfer* function (i.e., monitoring rules) for these labels

- Application to common attack scenarios
Our Contributions

- Browser enforced security policies
  - Simple declarative policies
  - Uniform enforcement mechanism
  - Enforcement by monitor on interfaces

- Formalisation of web browser as reactive system
  - Assign labels to elements of browser ecosystem
  - Define \textit{transfer} function (i.e., monitoring rules) for these labels

- Application to common attack scenarios
- Prototype implementation
Security Labels

• Simple Labels:

\[ l ::= \emptyset \mid \{\text{http}(d)\} \mid \{\text{https}(d)\} \mid l \cup l. \]

• Security labels are pairs of simple labels

  • **Confidentiality**: To whom may the value/presence of the object be disclosed
  
  • **Integrity**: By whom may the object be influenced or by whom has the object been influenced

• Example for a cookie label:

\[
(\{\text{https}(\text{facebook.com})\}, \{\text{http}(\text{facebook.com}), \text{https}(\text{facebook.com})\})
\]

• We label URLs, network connections, cookies and scripts
Formal modeling of attackers

- **Confidentiality Component**: Reading Capabilities
- **Integrity Component**: Injection Capabilities

Web attacker on domain \(d\):

\[\ell_w(d) \triangleq (\{\text{http}(d), \text{https}(d)\}, \{\text{http}(d), \text{https}(d)\})\]

Passive network attacker:

\[\ell_{pn} \triangleq (\{\text{http}(d) \mid d \in \mathcal{D}\}, \emptyset)\]

Active network attacker

\[\ell_{an} \triangleq (\{\text{http}(d) \mid d \in \mathcal{D}\}, \{\text{http}(d) \mid d \in \mathcal{D}\})\]
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com)

Passive Network Attacker
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com)

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com)
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; xhr(http://facebook.com))
4. request(http://www.facebook.com; )

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; xhr(http://facebook.com))
4. request(http://www.facebook.com; )
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(http://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com; xhr(http://facebook.com))

Passive Network Attacker

{http://facebook.com} ∉ {https://facebook.com}

{https://facebook.com}
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com;)

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com;)
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com;)

{https://facebook.com}

{http://facebook.com}
Cookie Leakage (Confidentiality)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com)

Passive Network Attacker

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; xhr(http://facebook.com))
4. request(http://www.facebook.com)
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. request(https://www.facebook.com/like; )
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)

Web Attacker at evil.com

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
7. request(https://www.facebook.com/like; )

{https(facebook.com)}

{https(facebook.com)}
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. request(https://www.facebook.com/like; 

Web Attacker at evil.com

facebook.com/like
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. request(https://www.facebook.com/like; )

{https(facebook.com)}  {https(evil.com)}  {https(facebook.com)}

{https(evil.com)}  {https(facebook.com)}

{https(evil.com)}  {https(facebook.com)}
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. request(https://www.facebook.com/like; ; unit)

{https(facebook.com)}

Web Attacker at evil.com

{https(evil.com)}

{https(facebook.com)}
Cross Site Request Forgery (Integrity)

1. load(https://www.facebook.com)
2. request(https://www.facebook.com; )
3. response(https://www.facebook.com; ; unit)
4. load(https://www.evil.com)
5. request(https://www.evil.com; )
6. response(https://www.evil.com; ; unit)
7. request(https://www.facebook.com/like; )

Web Attacker at evil.com
MiChrome

- Prototype implemented as a Google Chrome extension
  - No modification of browser source code
- Successfully tested prevention of CSRF on google gruyere
- Defined strict security policies for Paypal and made a successful purchase
- No noticeable overhead
Formal Result

- We model a web browser as a reactive system enhanced with a *transfer* function (i.e., monitoring rules).
- We automatically generate all entries of the *transfer* function when given labels for URLs and cookies.
- We define confidentiality and integrity for web sessions and cookies as reactive non-interference properties.

**Theorem.** A browser enhanced with our transfer function guarantees confidentiality and integrity for cookies and web sessions.
Formal Result

- We model a web browser as reactive system enhanced with a \textit{transfer} function (i.e., monitoring rules)
- We automatically generate all entries of the \textit{transfer} function when given labels for URLs and cookies
- We define confidentiality and integrity for web sessions and cookies as reactive non-interference properties

\textbf{Theorem.} A browser enhanced with our transfer function guarantees confidentiality and integrity for cookies and web sessions.

\begin{center}
\textbf{Take Home Message:} \\
\textit{Formal Web session security guarantees via a practical client-side security enforcement tool}
\end{center}
Future Work

• More comprehensive browser model (DOM, Extensions, etc.)

• Complete JavaScript sandboxing via rewriting (performed by our extension)

• Static analysis of Web protocols, reasoning about the security import of our extension on the Web application as a whole

• Machine-checked security proofs
Interested in a post-doc in Vienna?