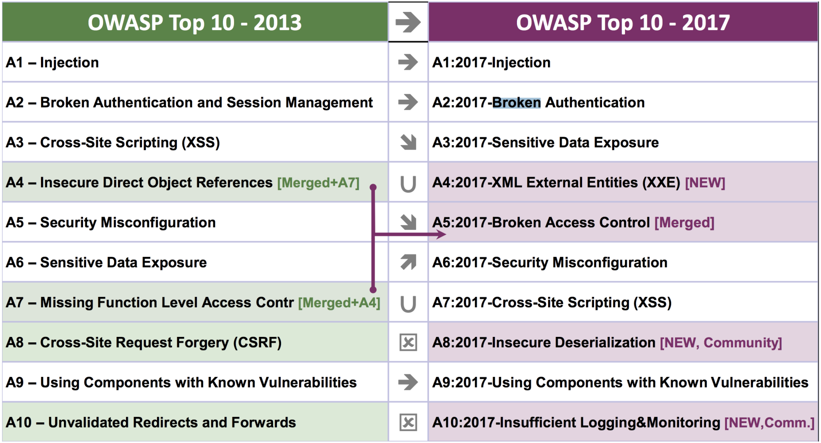
Security of web applications

Tomáš Pevný

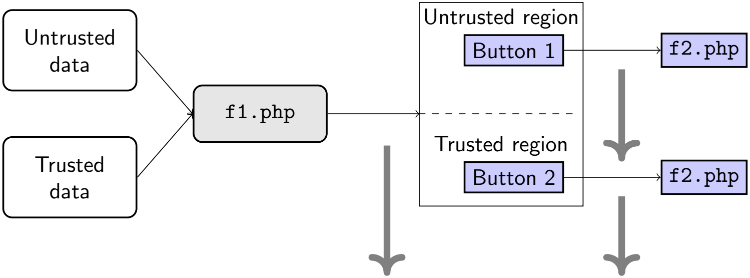
December 1, 2022

OWASP top ten vulnerabilities



OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Preservation of trust state



Untrusted data

Trusted data

`f1.php`

Untrusted region

Button 1

`f2.php`

Trusted region

Button 2

`f2.php`

the trust state lost: led to same-origin policy

the trust state lost: led to same-session policy

Prototypical XSS

```
<script> var x = 'INPUT_FROM_USER'; </script>
```

- Single quote breaks out of JS string, context into JS context
- `</script>` breaks out of JS context into HTML context

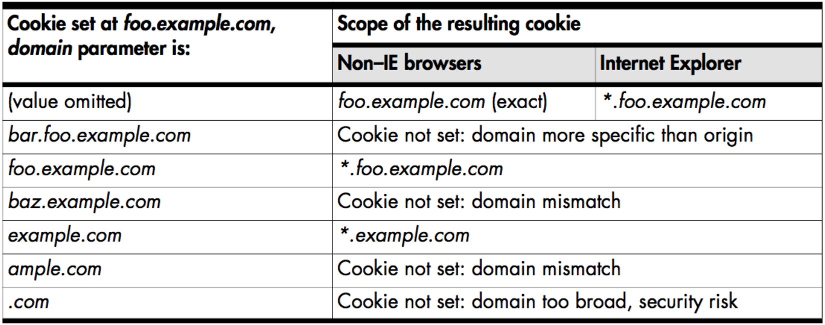
Mash-up

```
+--------------------------------------------+
| +--------------------------------------+   |
| | ad.gif from ads.com                  |   |
| +--------------------------------------+   |
| +-----------------+ +------------------+   |
| | Analytics.js    | | jQuery.js from   |   |
| | from google.com | | from cdn.foo.com |   |
| +-----------------+ +------------------+   |
|                                            |
| HTML (text inputs, buttons)                |
|                                            |
| +--------------------------------------+   |
| | Inline .js from foo.com (defines     |   |
| | event handlers for HTML GUI inputs)  |   |
| +--------------------------------------+   |
|+------------------------------------------+|
|| frame: https://facebook.com/likeThis.html||
||                                          ||
|| +-----------------+ +-----------------+ ||
|| | Inline .js from | | f.jpg           | ||
|| |https://fb.com   | | https://fb.com  | ||
|| +-----------------+ +-----------------+ ||
||                                          ||
|+------------------------------------------+|
|                                            |
+--------------------------------------------+
```

Same origin policy

originating document	accessed document	non-IE	IE
http://example.com/a/	http://example.com/**b**/	OK	Ok
http://example.com/	http://**www**.example.com/	—	—
http://example.com/	**https**://example.com/	—	—
http://example.com:**81**/	http://example.com/	—	OK

Cookies



Cookie set at *foo.example.com*, *domain* parameter is:	**Scope of the resulting cookie**	
	Non–IE browsers	**Internet Explorer**
(value omitted)	*foo.example.com* (exact)	**.foo.example.com*
bar.foo.example.com	Cookie not set: domain more specific than origin	
foo.example.com	**.foo.example.com*	
baz.example.com	Cookie not set: domain mismatch	
example.com	**.example.com*	
ample.com	Cookie not set: domain mismatch	
.com	Cookie not set: domain too broad, security risk	

Cross-origin-request-forgery

Imagine a following sequence

1. You log to your bank `https://bank.com` and perform transaction
2. You close the tab and continue other work
3. You visit some totally unrelated site `https://notsoobviousattacker.com`
4. There you click on link

```
<a
  href="https://bank.com/xfer?amount=500\&to=attacker">
  win free ipad
</a>
```

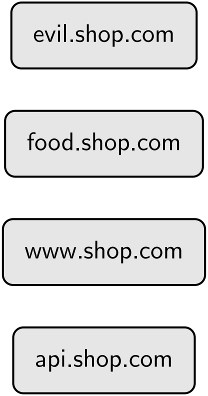
Cookies — SameSite

- `SameSite` attribute allow to specify, if cookie should be served to third parties
- options:
  - **None**
  - **Lax**
  - **Strict**

`https://web.dev/samesite-cookies-explained/`

Attack on cookie integrity: Related domain attacker

1. User create *secure* cookie on `food`.
   Sent only to `food.shop.com` over HTTPS.



Attack on cookie integrity: Related domain attacker

1. User create *secure* cookie on `food`.
   Sent only to `food.shop.com` over HTTPS.
2. User visits `evil.shop.com`.
   Set cookie for `*.shop.com`.

evil.shop.com

food.shop.com

www.shop.com

api.shop.com

Attack on cookie integrity: Related domain attacker

1. User create *secure* cookie on `food`.
   Sent only to `food.shop.com` over HTTPS.
2. User visits `evil.shop.com`.
   Set cookie for `*.shop.com`.
3. `food.shop.com` receives cookie set by `evil.shop.com`.

evil.shop.com

food.shop.com

www.shop.com

api.shop.com

Content security policy

White-list sources of trusted content.

Example: Google we trust

```
Content-Security-Policy: script-src ’self’
                         https://apis.google.com
```

Content security policy

- base-uri
- child-src
- connect-src
- font-src
- form-action
- frame-ancestors
- img-src
- media-src
- object-src
- plugin-types
- report-uri
- style-src
- upgrade-insecure-requests

Example: white-listing more resources

```
Content-Security-Policy: default-src https://cdn.example.net;
                         child-src ’none’; object-src ’none’
```

Keywords

- `none`
- `self`
- `unsafe-inline`
- `unsafe-eval`

Example: insecure embedding of javascript

```
<script>
  function doAmazingThings() {
    alert(’Hello!’);
  }
</script>
<button onclick=’sayHello();’>Say Hello.</button>
```

Example: secure embedding javascript

```
<!-- Hello.html -->
<script src=’Hello.js’></script>
<button id=’Hello’>Am I Hello?</button>
```

```
// Hello.js
function sayHello() {
  alert(’Hello!’);
}
document.addEventListener(’DOMContentReady’, function () {
  document.getElementById(’Hello’)
          .addEventListener(’click’, sayHello);
});
```

"Safely" enabling inline scripts

```
Content-Security-Policy: script-src 'nonce-EDNnf03nceIOfn39f'
```

```
<script nonce=EDNnf03nceIOfn39f>
  // Some inline code I can't remove yet, but need to asap.
</script>
```

`'strict-dynamic'` requires nonce for inline scripts but not for scripts included from external sources.

DOM-based cross-site scripting

```
el.innerHTML = ’<img src=xyz.jpg>’;
```

- Script manipulation: `<script src>` and setting text content of `<script>` elements.
- Generating HTML from a string: `innerHTML`, `outerHTML`,`insertAdjacentHTML`, `<iframe> srcdoc`, `document.write`, `document.writeln`, and `DOMParser.parseFromString`
- Executing plugin content: `<embed src>`, `<object data>` and `<object codebase>`
- Runtime JavaScript code compilation: `eval`, `setTimeout`, `setInterval`, `new Function()`

Trusted Types

```
Content-Security-Policy: require-trusted-types-for ’script’;

const escapeHTMLPolicy = trustedTypes.createPolicy(’myEscapePoli
    createHTML: string => string.replace(/\</g, ’&lt;’)
  });

const escaped = escapeHTMLPolicy.createHTML(’<img src=x onerror=
el.innerHTML = escaped;  // ’&lt;img src=x onerror=alert(1)>’
```

Dealing with untrusted content?

- Static or dynamic validation of all 3rd party data (user-supplied data and extensions).
- Mark-down language
- Use `<sandbox>` tag in `HTML5`.
- Use content security policy.

Example: embedding twitter button

```
<iframe
  src="https://platform.twitter.com/widgets/tweet_button.html"
  style="border: 0; width:130px; height:20px;">
</iframe>
```

Example: embedding twitter button

```
<iframe
  sandbox="allow-same-origin allow-scripts
           allow-popups allow-forms"
  src="https://platform.twitter.com/widgets/tweet_button.html"
  style="border: 0; width:130px; height:20px;">
</iframe>
```

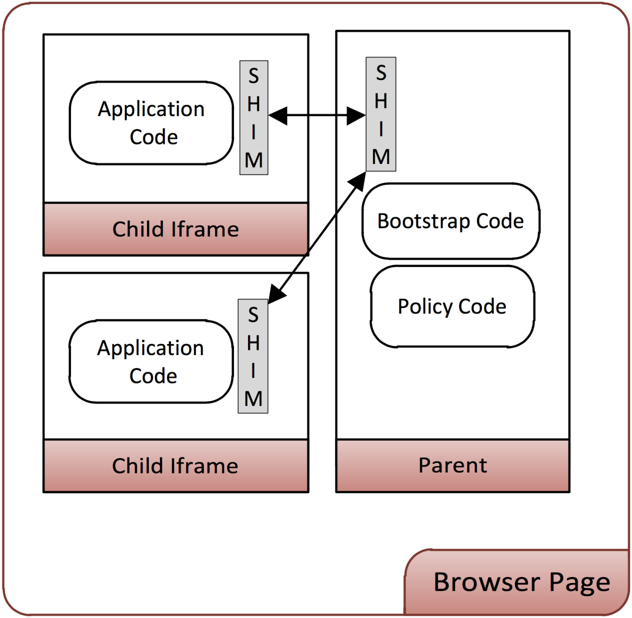
Example: Turning page into static content

```
<iframe sandbox src="example.com">
```

Sandbox options

- `allow-forms`
- `allow-popups`
- `allow-pointer-lock`
- `allow-same-origin`
- `allow-scripts`
- `allow-top`
- `allow-scripts`
- `allow-popups`
- `allow-forms`

Example: separation of privileges



Plan

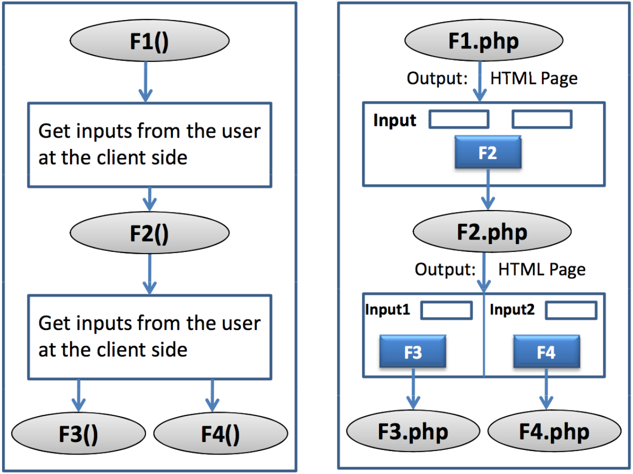
Preserving code integrity

Example of synchronous application

Warehouse application:

1. choose goods to buy
2. go to checkout
3. pay
4. send notification to release the goods.

Synchronous application with asynchronous mechanisms

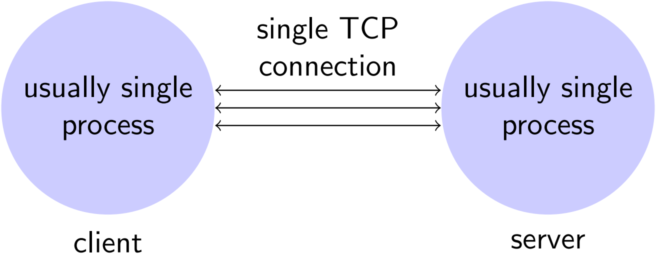


(a) Traditional Application

(b) Web Application

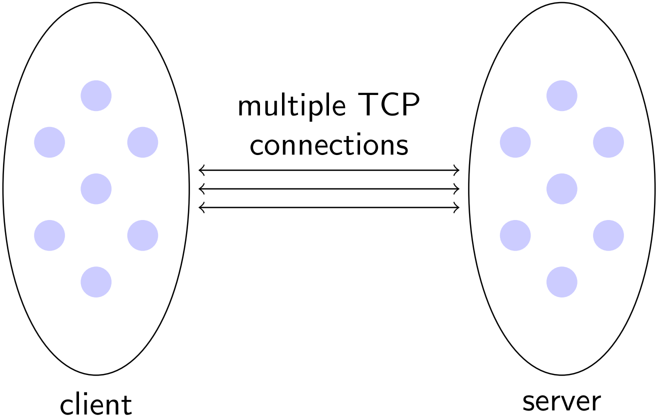
Synchronous vs. Asynchronous models

Stateful client/server framework



Synchronous vs. Asynchronous models

Web client-server model



Attack: session hijacking

If an attacker discover session ID he has free access to the session.

- Some applications do not protect session ID sufficiently.
- Some applications exploit session ID for functionality, such as sharing.

Attack: session fixation on PHP

Provide the parameter when `session_start()` is called

- In GET request as `http://targeted_server.com/logon.php?SID=12345`.
- In cookie when `http://targeted_server.com/logon.php?SID=12345`