

Computer Security List 6

Bartosz Michalak

9 June 2024

1 Task 1

1.1

We enter HTTPS website, we click on locker icon, then "connection secure", "more information" and "view certificate" to get certificate info. To download it we scroll down till "download" section.

```

(base) bartek@bartek-Aspire-A515-45:~/Desktop/sen6/Semester-6/CS/lista6$ openssl x509 -in pwr-edu-pl-chain.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            59:a4:51:a2:db:2f:66:8f:e5:01:06:99:40:12:30:0b
        Signature Algorithm: sha384WithRSAEncryption
        Issuer: C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
        Validity
            Not Before: Feb 18 00:00:00 2024 GMT
            Not After : Feb 17 23:59:59 2025 GMT
        Subject: C = PL, ST = Dolno(C5\9Bl\C4\8Sskie, O = Politechnika Wroc\C5\82awska, CN = *.pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public-Key: (4096 bit)
            Modulus:
                00:da:e5:0e:de:aa:c4:3e:35:b9:b6:e4:ab:43:42:
                b5:99:c0:51:09:00:e4:11:ac:61:49:9d:91:3d:60:
                6c:5f:17:e0:07:3f:fa:40:89:0c:af:93:be:4c:37:
                21:3a:25:17:66:dd:8d:88:e4:77:db:54:33:e9:19:
                82:9f:dd:c0:74:95:85:19:3c:e9:4f:22:3e:be:76:
                8a:9f:9c:06:6f:67:c0:b1:4b:7f:9a:8e:6f:50:04:
                d5:98:1a:6f:4c:70:b3:27:bd:5e:0a:5e:f1:75:2b:
                bd:f1:a7:fb:bd:02:45:7c:1d:39:39:11:a2:4a:5c:
                c1:c4:d6:fd:52:16:c6:40:e1:9c:e9:3d:f0:65:75:
                3f:80:75:7f:c2:11:0d:bc:9f:ca:b0:0f:09:cf:ed:
                54:f3:59:83:05:61:66:06:0f:f9:f7:36:c7:18:d2:
                b9:32:7b:33:00:3e:f5:d4:3e:5c:bb:94:04:10:56:
                e4:14:99:2d:46:24:b2:0a:44:df:2e:cc:9d:8a:a5:
                6b:82:1f:bb:07:59:1d:f1:6d:23:08:3c:d9:79:56:
                42:3e:0e:87:d3:37:23:b7:b9:8d:b8:90:db:90:
                dd:c3:04:b8:7a:cc:20:c0:ec:89:3b:35:14:2d:97:
                ef:a8:c2:6c:88:f1:9c:23:1b:78:83:26:bc:c6:b5:
                0c:6c:bb:82:01:57:26:35:de:e4:0f:dd:0b:42:fa:
                ec:e1:c4:37:b4:bb:e4:3f:0f:47:70:c1:b1:1f:24:
                04:8e:5f:3f:fd:c1:57:e5:c4:ea:dc:14:fc:96:c3:
                0e:95:f5:c2:19:db:2b:7a:c6:17:62:7d:44:dc:2d:
                10:af:b9:b4:91:8c:4a:7b:9b:bf:6a:71:05:9c:cd:
                9c:1b:2f:27:0a:34:77:22:6f:b4:b9:f7:82:81:c9:
                ea:29:ff:08:9f:f6:ab:80:ed:da:83:5f:0d:5a:74:
                6d:09:55:16:33:2d:13:0e:e3:be:2a:c9:e2:cd:97:
                fc:ca:c6:e8:9e:c8:a6:f2:ad:ba:1d:38:9d:81:14:
                dd:f2:65:55:1d:49:8b:29:03:66:2f:f4:bf:21:
                b2:2c:51:e8:a2:ba:37:04:9e:88:0d:0d:bc:fc:03:
                67:2f:e4:be:46:40:27:c2:e1:2b:47:ca:3f:5b:1d:
                ef:a4:20:2e:ab:e0:4d:0b:4b:6d:5d:8d:93:5d:c0:
                d7:e7:f6:fb:ea:b3:70:b4:93:ba:63:82:36:d4:cb:
                29:a5:b9:ea:66:79:25:5d:54:de:b6:fd:e2:47:b6:
                ee:7f:16:15:b6:1f:ce:2b:63:d1:9d:83:f5:68:6b:
                89:12:89:0d:6d:e3:85:c0:4f:e2:88:a8:22:a5:f6:
                f1:6f:99
            Exponent: 65537 (0x10001)
        X509v3 extensions:

```

Figure 1: ssl info

```

X509v3 extensions:
  X509v3 Authority Key Identifier:
    keyid:6F:1D:35:49:10:6C:32:FA:59:A0:9E:BC:8A:E8:1F:95:BE:71:7A:0C

  X509v3 Subject Key Identifier:
    40:39:A6:1D:52:C6:67:81:14:06:84:5A:5D:83:48:C5:FF:4E:C4:60
  X509v3 Key Usage: critical
    Digital Signature, Key Encipherment
  X509v3 Basic Constraints: critical
    CA:FALSE
  X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
  X509v3 Certificate Policies:
    Policy: 1.3.6.1.4.1.6449.1.2.2.79
    CPS: https://sectigo.com/CPS
    Policy: 2.23.140.1.2.2

  X509v3 CRL Distribution Points:

    Full Name:
      URI:http://GEANT.crl.sectigo.com/GEANTOVRSA4.crl

  Authority Information Access:
    CA Issuers - URI:http://GEANT.crt.sectigo.com/GEANTOVRSA4.crt
    OCSP - URI:http://GEANT.ocsp.sectigo.com

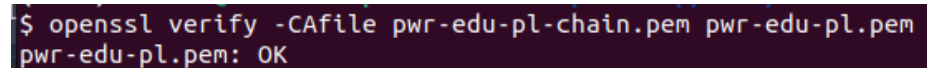
  X509v3 Subject Alternative Name:
    DNS:*.pwr.edu.pl, DNS:pwr.edu.pl
  CT Precertificate SCTs:
    Signed Certificate Timestamp:
      Version : v1 (0x0)
      Log ID : CF:11:56:EE:D5:2E:7C:AF:F3:87:5B:D9:69:2E:9B:E9:
        1A:71:67:4A:B0:17:EC:AC:01:D2:5B:77:CE:CC:3B:08
      Timestamp : Feb 18 00:29:07.158 2024 GMT
      Extensions: none
      Signature : ecdsa-with-SHA256
        30:45:02:21:00:C7:AD:4D:A2:B1:53:61:5A:CF:65:A0:
        94:A8:13:95:59:DF:C2:7E:8F:5E:CE:8B:0F:79:B3:42:
        7B:F2:B2:30:22:02:20:33:55:81:C1:9E:50:24:F4:14:
        42:65:F1:C5:57:4D:2C:CC:92:FB:DB:90:EB:AB:DF:0D:
        D7:E5:4C:BB:E4:59:6E
    Signed Certificate Timestamp:
      Version : v1 (0x0)
      Log ID : A2:E3:0A:E4:45:EF:BD:AD:9B:7E:38:ED:47:67:77:53:
        D7:82:5B:84:94:D7:2B:5E:1B:2C:C4:B9:50:A4:47:E7
      Timestamp : Feb 18 00:29:07.348 2024 GMT
      Extensions: none
      Signature : ecdsa-with-SHA256
        30:46:02:21:00:F0:DC:33:4F:0A:DA:8F:60:0F:F7:5C:
        74:18:53:5E:AC:65:2F:55:DF:A2:47:AB:DA:C0:72:58:
        61:54:E9:75:BA:02:21:00:DF:4A:6B:33:A6:DE:A7:48:
        3C:F9:C8:94:2D:44:0B:3C:89:73:47:CD:67:76:C8:C6:
        4C:46:78:C0:A2:27:B3:60
    Signed Certificate Timestamp:
      Version : v1 (0x0)
      Log ID : 4E:75:A3:27:5C:9A:10:C3:38:5B:6C:D4:DF:3F:52:EB:
        1D:F0:E0:8E:1B:8D:69:C0:B1:FA:64:B1:62:9A:39:DF
      Timestamp : Feb 18 00:29:07.204 2024 GMT
      Extensions: none
      Signature : ecdsa-with-SHA256
        30:44:02:20:2B:BD:6E:B4:5D:19:99:D4:7F:22:CB:7D:
        75:F2:CF:49:F8:7F:E8:63:07:5F:AC:CE:0B:C7:76:E0:
        23:D3:EE:17:02:20:0B:0D:FA:B5:DB:32:6C:9C:9A:5E:
        F9:EF:19:E6:F1:14:06:AC:6A:2F:3D:7F:05:06:C0:17:
        48:DF:C3:28:C5:A2
  Signature Algorithm: sha384WithRSAEncryption

```

Figure 2: ssl info 2

For <https://pwr.edu.pl/>:
 Issuer: C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
 Signature Algorithm: sha384WithRSAEncryption
 Validity
 Not Before: Feb 18 00:00:00 2024 GMT
 Not After : Feb 17 23:59:59 2025 GMT
 Subject: C = PL, ST = Dolno\C5\9Bl\C4\85skie, O = Politechnika Wroc\C5\82awska,
 CN = *.pwr.edu.pl
 X509v3 Subject Alternative Name:
 DNS:*.pwr.edu.pl, DNS:pwr.edu.pl
 X509v3 CRL Distribution Points:
 Full Name:
 URI:<http://GEANT.crl.sectigo.com/GEANTOVRSA4.crl>
 Authority Information Access:
 CA Issuers - URI:<http://GEANT.crt.sectigo.com/GEANTOVRSA4.crt>
 OCSP - URI:<http://GEANT.ocsp.sectigo.com>

1.2



```
$ openssl verify -CAfile pwr-edu-pl-chain.pem pwr-edu-pl.pem
pwr-edu-pl.pem: OK
```

Figure 3: verified cert

1.3

What is OCSP?

OCSP (Online Certificate Status Protocol) - a standard describing the communication protocol between the IT system of the recipient of certification services and the service server. This protocol specifies the format and structure of the query (request) about the certificate status and the format and structure of the response (token), which contains the verification result in the form of the status: "correct", "revoked", "unknown".

Issuing a certificate with the "correct" status means that the certificate is issued by one of the entities whose certificates are covered by the OCSP service and that this entity has not effectively revoked the checked certificate at the time of responding.

Using a trusted OCSP service is a more practical and "secure" way to verify certificate validity than searching certificate revocation lists (CRLs). We need intermediate chain (no pwr cert) so we delete it and run:

```
$ openssl ocsf -issuer intermediate.pem -cert
pwr-edu-pl.pem -url http://GEANT.ocsp.sectigo.com
```

2 Task 2



Figure 4: Basic Website

Run to gen private key:

```
$ openssl genpkey -algorithm RSA -out private.key
```

Run to gen public key:

```
$ openssl rsa -pubout -in private.key -out public.key
```

Run to gen CRT:

```
$ openssl req -new -key private.key -out domain.csr
```

Cert bot didn't work for me as I don't have domain. So I signed my website myself:

```
$ openssl req -x509 -newkey rsa:4096 -nodes -out  
cert.pem -keyout key.pem -days 365
```

```

(base) bartek@bartek-Aspire-A515-45:~/Desktop/sem6/Semester-6/CS/listan$ openssl req -new -key privat
e.key -out domain.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:Wielkopolska
Locality Name (eg, city) []:Kalisz
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Student
Organizational Unit Name (eg, section) []:Alg
Common Name (e.g. server FQDN or YOUR name) []:Bartosz
Email Address []:nowyfacebook67@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:12345678
An optional company name []:

```

Figure 5: CRT creation process

Firefox browser window showing a certificate page. The address bar displays: `about:certificate?cert=MIIGAzCCA%2BugAwIBAgIUFlyLvtUuEB`.

Certificate

Bartosz

Subject Name

Country	PL
State/Province	WielkoPolska
Locality	Kalisz
Organization	Student
Organizational Unit	Alg
Common Name	Bartosz
Email Address	nowyfacebook67@gmail.com

Issuer Name

Country	PL
State/Province	WielkoPolska
Locality	Kalisz
Organization	Student
Organizational Unit	Alg
Common Name	Bartosz
Email Address	nowyfacebook67@gmail.com

Validity

Not Before	Mon, 10 Jun 2024 08:12:52 GMT
Not After	Tue, 10 Jun 2025 08:12:52 GMT

Public Key Info

Algorithm	RSA
Key Size	4096
Exponent	65537

Figure 6: My Sign

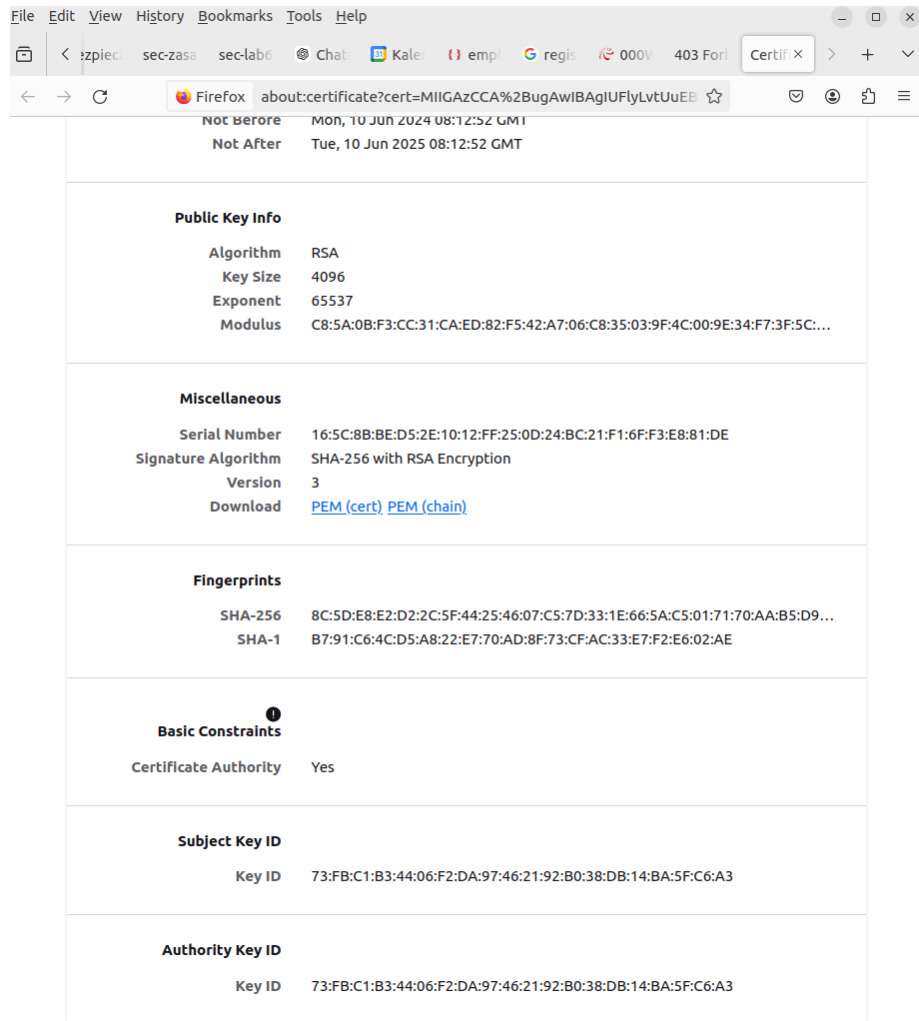


Figure 7: My Sign Pt2

3 Task 3 - XSS

After setting up webGoat we need to use "http://127.0.0.1:8080/WebGoat/login" to loginto it.

3.1 Cross Site Scripting 1

XSS is used to perform tasks that were not the developer's original intent.

3.2 Cross Site Scripting 2

By using "alert(document.cookie);" we display document.cookie, it is the same for both instances of the same URL on WebGoat.

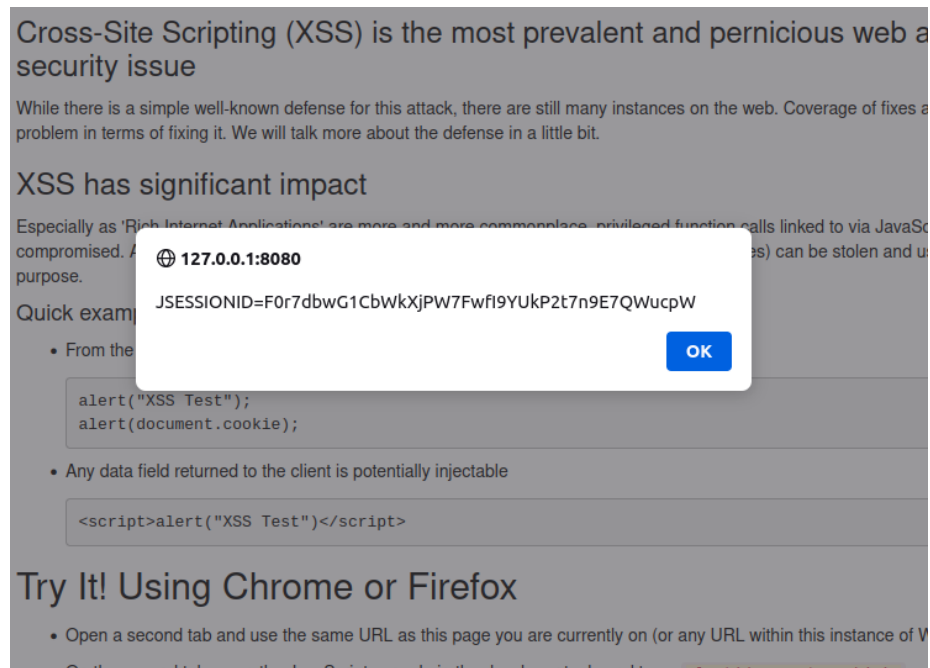


Figure 8: WebGoat Cookie

3.3 Cross Site Scripting 3

Common places where we may get users' data:

1. Search fields that echo a search string back to the user
2. Input fields that echo user data
3. Error messages that return user-supplied text
4. Hidden fields that contain user-supplied data

5. Any page that displays user-supplied data e.g. Message boards, Free form comments
6. HTTP Headers

3.4 Cross Site Scripting 4

Results of XSS:

1. Stealing session cookies
2. Creating false requests
3. Creating false fields on a page to collect credentials
4. Redirecting your page to a "non-friendly" site
5. Creating requests that masquerade as a valid user
6. Stealing of confidential information
7. Execution of malicious code on an end-user system (active scripting)
8. Insertion of hostile and inappropriate content e.g.: ``
"GoodYear recommends buying BridgeStone tires..."

It is also worth noting that XSS attacks add validity to phishing attacks by using a valid domain in the URL.

3.5 Cross Site Scripting 5

Types of XSS:

3.5.1 Reflected

1. Malicious content from a user request is displayed to the user in a web browser
2. Malicious content is written into the page after from server response
3. Social engineering is required
4. Runs with browser privileges inherited from the user in a browser

3.5.2 DOM-based (also technically reflected)

1. Client-side scripts use malicious content from a user request to write HTML to its page
2. Similar to reflected XSS
3. Runs with browser privileges inherited from the user in a browser

3.5.3 Stored or persistent

1. Malicious content is stored on the server (in a database, file system, or other objects) and later displayed to users in a web browser
2. Social engineering is not required

3.6 Cross Site Scripting 6

Reflected XSS scenario:

1. Attacker sends a malicious URL to the victim
2. Victim clicks on the link that loads a malicious web page
3. The malicious script embedded in the URL executes in the victim's browser. The script steals sensitive information, like the session id, and releases it to the attacker

Victim does not realize attack occurred.

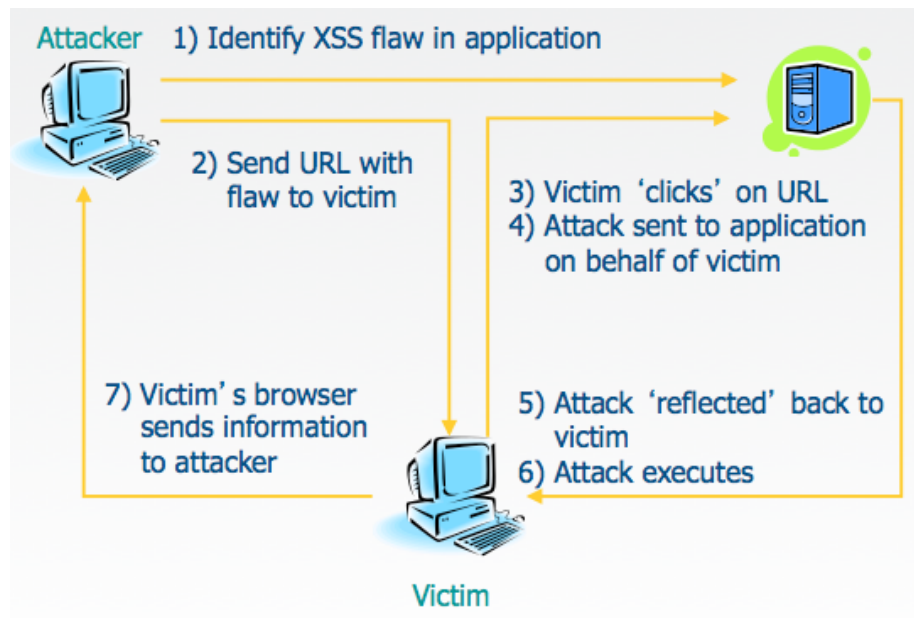


Figure 9: XSS scenario

3.7 Cross Site Scripting 7

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Putting `"<script>alert('XSS');</script>"` into "Enter your credit card number:" field was the solution.

3.8 Cross Site Scripting 8

Apparently in previous task we performed self XSS because no link can activate it.

3.9 Cross Site Scripting 9

DOM-based XSS is another form of reflected XSS. Both are triggered by sending a link with inputs reflected in the browser. The difference between DOM and 'traditional' reflected XSS is that, with DOM, the payload will never go to the server. The client will only ever process it.

1. Attacker sends a malicious URL to the victim
2. Victim clicks on the link

3. That link may load a malicious web page or a web page they use (are logged into?) that has a vulnerable route/handler
4. If it's a malicious web page, it may use its own JavaScript to attack another page/URL with a vulnerable route/handler
5. The vulnerable page renders the payload and executes an attack in the user's context on that page/site
6. Attacker's malicious script may run commands with the privileges of local account

Victim does not realize attack occurred ... Malicious attackers don't use `<script>alert('xss')</script>`

3.10 Cross Site Scripting 10

Trying to run test route instead of lesson route. Go to js code of this app Go-Router.js handles routes. Go there and you will find:

```
"routes:
  'welcome': 'welcomeRoute',
  'lesson/:name': 'lessonRoute',
  'lesson/:name/:pageNum': 'lessonPageRoute',
  'test/:param': 'testRoute',
  'reportCard': 'reportCard'
,"
```

Correct answer is: start.mvctest/

3.11 Cross Site Scripting 11

Now by opening:

`http://127.0.0.1:8080/WebGoat/start.mvc?username=ttreesstest/%3Cscript%3Ewebgoat%2Ecustomjs%2EphoneHome%28%29%3B%3C%2Fscript%3E`
we have made this website run `phoneHome()`; function, which returns "-76283620".

3.12 Cross Site Scripting 12

It doesn't load :(

4 Task 3 - XSS (stored)

4.1

Closer look at another form of attack: Stored XSS.

4.2 Scenario

Stored Cross-Site Scripting is different in that the payload is persisted (stored) instead of passed/injected via a link.

Stored XSS Scenario:

1. Attacker posts malicious script to a message board
2. Message is stored in a server database
3. Victim reads the message
4. The malicious script embedded in the message board post executes in the victim's browser. The script steals sensitive information, like the session id, and releases it to the attacker

Victim does not realize attack occurred.

4.3 Comment

No idea how to do that :(

5 Task 3 - XSS(mitigation)

5.1

How do we defend against XSS?

5.2

5.2.1 What to encode?

The basic premise of defending against XSS is **output encoding** any untrusted input to the screen. That may be changing with more sophisticated attacks, but it is still the best defense we currently have. **AND ... context matters**

Another word on 'untrusted input.' If in doubt, treat everything (even data you populated in your DB as untrusted). Sometimes data is shared across multiple systems, and what you think is your data may not have been created by you/your team.

5.2.2 When/Where?

Encode **as the data is sent to the browser** (not in your persisted data). In the case of Single Page Apps (SPA's), you will need to encode in the client. Consult your framework/library for details, but some resources will be provided on the next page.

5.2.3 How?

1. Encode as HTML Entities in HTML Body
2. Encode as HTML Entities in HTML Attribute
3. Encode for JavaScript if outputting user input to JavaScript (but think about that ... you are outputting user input into JavaScript on your page!!)

DO NOT try to blacklist/negative filter on strings like '`script;`' and so forth.

5.3

5.3.1 What is encoding?

Not trusting user input means validating data for type, length, format, and range whenever it passes through a trust boundary, say from a web form to an application script, then encode it before redisplay in a dynamic page.

In practice, this means that you need to review every point on your site where user-supplied data is handled and processed and ensure that, before being passed back to the user, any values accepted from the client side are checked, filtered, and encoded.

Client-side validation cannot be relied upon, but user input can be forced down to a minimal alphanumeric set with server-side processing before being used by a web application in any way.

5.3.2 Escaping

Escaping means that you convert (or mark) key characters of the data to prevent it from being interpreted in a dangerous context. In the case of HTML output, you need to convert the `<` and `&` characters (among others) to prevent any malicious code from rendering. Escaping these characters involves turning them into their entity equivalents `<` and `&`, which will not be interpreted as HTML tags by a browser.

5.3.3 Special characters

You need to encode special characters like `"` and `'` before they are redisplayed if they are received from user input. For example, encoding `"` and `'` ensures a browser will display `"` but not execute it. In conjunction with encoding, your web pages must always define their character set so the browser will not interpret special character encodings from other character sets.

Cross-site scripting attacks usually occur when you manage to sneak a script (usually javascript) onto someone else's website, where it can run maliciously.

5.4 XSS defense resources

1. [Java OWASP Encoder](#)
2. [General XSS prevention Cheat Sheet](#)
3. [DOM XSS Prevention Cheat Sheet](#)
4. [XSS Filter Evasion](#)

5.5

IDK

5.6

IDK