CONTENT SECURITY POLICY
IN MODERN INTERNET BROWSERS

Presenter mgr inż. Darek Łysyszyn

08.05.2013, Seminarium z Kryptologii i Ochrony Informacji

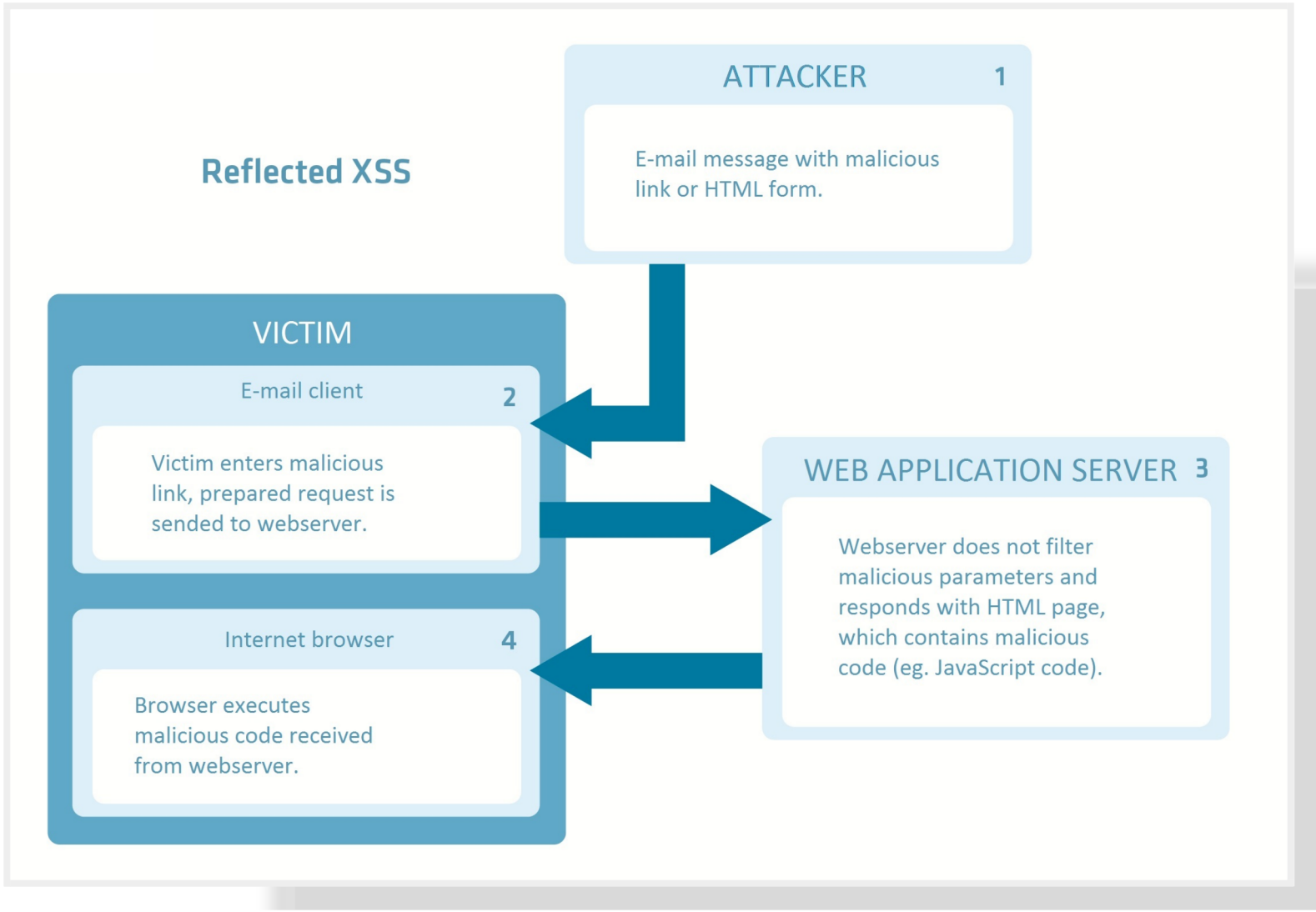
AGENDA

- Cross-Site Scripting
- CSP: Definition
- CSP: Features
- CSP: Examples
- Future of CSP
- Conclusions

CROSS-SITE SCRIPTING (XSS) - 1/3

*" **Cross-site scripting (XSS)** is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same origin policy. "* - via Wikipedia

CROSS-SITE SCRIPTING (XSS) - 2/3



CROSS-SITE SCRIPTING (XSS) - 3/3

According to OWASP XSS is one of the most common type of web application vulnerability.

There is promising new defense that can significantly reduce the risk and impact of XSS attacks in modern browsers:

Content Security Policy (CSP).

CSP: DEFINITION

Content Security Policy is an W3C specification offering the possibility to instruct the client browser from which location and/or which type of resources are allowed to be loaded. To define a loading behavior, the CSP specification use “directive” where a directive define a loading behavior for a target resource type.

CSP: HOW IT WORKS?

CSP defines the Content-Security-Policy **HTTP header** that allows you to create a whitelist of sources of trusted content, and instructs the browser to only execute or render resources from those sources. Even if an attacker can find a hole through which to inject script, the script won't match the whitelist, and therefore won't be executed.

CSP: CAN I USE IT?

	IE	Firefox	Chrome	Safari	Opera
5 versions back	5.5	15.0 moz	21.0 webkit	3.1	11.0
4 versions back	6.0	16.0 moz	22.0 webkit	3.2	11.1
3 versions back	7.0	17.0 moz	23.0 webkit	4.0	11.5
2 versions back	8.0	18.0 moz	24.0 webkit	5.0	11.6
Previous version	9.0	19.0 moz	25.0	5.1 webkit	12.0
Current	10.0 ms	20.0 moz	26.0	6.0 webkit	12.1
Near future		21.0 moz	27.0		
Farther future		22.0 moz	28.0		

Source: http://caniuse.com/contentsecuritypolicy

CSP: DEFINITION

Content Security Policy is an W3C specification offering the possibility to instruct the client browser from which **location** and/or which type of **resources** are allowed to be loaded. To define a loading behavior, the CSP specification use **directive** where a directive define a loading behavior for a target resource type.

CSP: USAGE - FIRST VIEW

Pattern:

```
Content-Security-Policy: directive locations/resources
```

Examples:

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

CSP: BROWSER BEHAVIOR

Policy:

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

Script inclusion:

```
<script src="http://evil.com/evil.js"></script>
```

Result:

CSP: AVAILABLE DIRECTIVES

- **script-src**: limits the origins from which scripts can be loaded,
- **frame-src**: lists the origins that can be embedded as frames,
- **connect-src**: limits the origins to which you can connect (via XHR, WebSockets, and EventSource).
- **img-src**: defines the origins from which images can be loaded.
- **object-src**: allows control over Flash and other plugins.
- **style-src**: limits the origins from which images can be loaded,
- **media-src**: restricts the origins allowed to deliver video and audio.
- **font-src**: specifies the origins that can serve web fonts.
- **default-src**: default policy if case some of the above is not set.

CSP: RESOURCE LIST KEYWORDS

- **none**: matches nothing,
- **self**: matches the current origin, but not its subdomains,
- **unsafe-inline**: allows inline JavaScript and CSS,
- **unsafe-eval**: allows text-to-JavaScript mechanisms like JS `eval` function.

CSP: BLOCKING INLINE SCRIPTS

```
<button onclick="inlineEvilFoo();">Click me!</button>
```

Content Security Policy by default blocks this kind of inclusion - perfect XSS defence!

CSP: BLOCKING JS EVAL FUNCTION

Let assume we have a chunk of code like this:

```
var parsed_obj = eval("(" + json_text + ")");
```

And we serve JSON object like this:

```
{ "name": alert("Hi everyone!") }
```

Content Security Policy by default blocks eval function calls.

`JSON.parse() >> eval()`

REPORTING 1/2

POLICY VIOLATION REPORTING

Policy with reposting:

```
Content-Security-Policy:
    default-src 'self';
    report-uri /csp_report_parser;
```

Report example:

```
{
  "csp-report": {
    "document-uri": "http://example.org/page.html",
    "referrer": "http://evil.example.com/",
    "blocked-uri": "http://evil.example.com/evil.js",
    "violated-directive": "script-src 'self' https://apis.google.com",
    "original-policy": "(...) report-uri /csp_report_parser"
  }
}
```

REPORTING 2/2

CONTENT-SECURITY-POLICY-REPORT-ONLY HEADER

The policy specified in report-only mode won't block restricted resources, but it will send violation reports to the location you specify. This is a great way to evaluate the effect of changes to an application's CSP.

```
Content-Security-Policy-Report-Only:
    default-src 'self';
    report-uri /csp_report_parser;
```

CSP: USAGE - SECOND VIEW

Examples:

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

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

Content-Security-Policy: frame-src *://cdn.mydomain.com:*

Content-Security-Policy:
    script-src
        'self'
        http://cdn.mydomain.com:*
        http://adserver..example.com;
    frame-src
        *://yourdomain.com:*;
    object-src
        'none';
    connect-src
        https:;
```

CSP: EXAMPLES

CASE#1: SOCIAL MEDIA BUTTONS - 1/3

CSP: EXAMPLES

CASE#1: SOCIAL MEDIA BUTTONS · 2/3

- **Google +1 button**:
  includes a `script` from https://apis.google.com and
  embeds an `iframe` from https://plusone.google.com

- **Facebook Like button**:
  embeds an `iframe` from https://facebook.com

- **Twitter Tweet button**:
  includes a `script` from https://platform.twitter.com and
  embeds an `iframe` from https://platform.twitter.com

CSP: EXAMPLES

CASE#1: SOCIAL MEDIA BUTTONS · 3/3

The final policy:

```
Content-Security-Policy:
    script-src
        'self'
        https://apis.google.com
        https://platform.twitter.com;
    frame-src
        https://plusone.google.com
        https://facebook.com
        https://platform.twitter.com
```

CSP: EXAMPLES

CASE#2: SSL ONLY · 1/2

Let assume some webpage admin wants to load all external resources via secure channel. Rewriting large parts of code could be very difficult and – in some cases – even impossible.

CSP: EXAMPLES

CASE#2: SSL ONLY · 2/2

The final policy:

```
Content-Security-Policy:
    default-src
        https:;
    script-src
        https: 'unsafe-inline';
    style-src
        https: 'unsafe-inline'
```

FUTURE OF CSP

CSP 1.1 DRAFT

New version of document is being actively discussed and browser vendors are hard at work solidifying and improving their implementations.

FUTURE OF CSP

DOM API

The ability to query a page's current policy via JavaScript, which will enable you to make runtime decisions about implementations, and gracefully settle on something that will work for the environment in which your code finds itself.

Eg. if `eval` function is unavailable code might implement some feature differently.

FUTURE OF CSP

NEW DIRECTIVES

- **script-nonce**: enables inline script only for explicitly specified script elements,
- **plugin-types**: limits the MIME types of content for which plugins could be loaded,
- **form-action**: allows form submission to only specific origins.

CONCLUSIONS

1. Allows to control script and style inclusions
2. Blocks inline scripts and styles
3. Easy to implement
4. Reporting mode as a tool for developers
5. Can be implemented in some part of application
6. Available in main browsers (~68% of Internet users)

LINKS

http://www.w3.org/TR/CSP/

http://www.w3.org/TR/CSP11/

https://www.owasp.org/index.php/Content_Security_Policy

http://www.html5rocks.com/en/tutorials/security/content-security-policy/

https://developer.mozilla.org/en-US/docs/Security/CSP/Introducing_Content_Security_Policy

http://developer.chrome.com/extensions/contentSecurityPolicy.html

http://lists.w3.org/Archives/Public/public-webappsec/

http://lists.w3.org/Archives/Public/public-webappsec/2012Nov/att-0112
/Web_Application_Security_Working_Group.htm

http://engineering.twitter.com/2011/03/improving-browser-security-with-csp.html

http://falcon80.com/JSON/parsingJSON.html

https://en.wikipedia.org/wiki/Cross-site_scripting

http://erlend.oftedal.no/blog/csp/readiness/

http://caniuse.com/contentsecuritypolicy

THE END

Questions?

Presenter **mgr inż. Darek Łysyszyn**