OWASP Shanghai (March 2014)

Web Security – New Browser Security Technologies

Tobias Gondrom
OWASP London
OWASP Project Leader
Chair of IETF Web Security WG

tobias.gondrom@gondrom.org

Tobias Gondrom

- 15 years information security experience
  (Global Head of Security, CISO, CTO)
  CISSP, CSSLP, CCISO
- 12 years management of application development experience
- Sloan Fellow M.Sc. In Leadership and Strategy, London Business School
- Thames Stanley: Managing Director,
  CISO Advisory, Information Security & Risk Management, Research and Advisory
- OWASP Global Board member, OWASP Project Leader for the CISO Survey,
  www.owasp.org
- Author of Internet Standards on Secure Archiving, CISO training and co-author of the OWASP CISO guide
- Chair of IETF Web Security Working Group
  http://datatracker.ietf.org/wg/websec/charter/
  Member of the IETF Security Directorate
  IETF Administrative Oversight Committee (IAOC),
  Chair of the IETF Trust
- Cloud Security Alliance, Hong Kong chapter, Vice Chairman

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

The Past: CA breaches March 15th 2011: Comodo breach

- Nine fake certificates for seven domains were issued: mail.google.com, login.live.com, www.google.com, login.yahoo.com (three certificates), login.skype.com, addons.mozilla.org, and global trustee
- Hacked several times afterwards

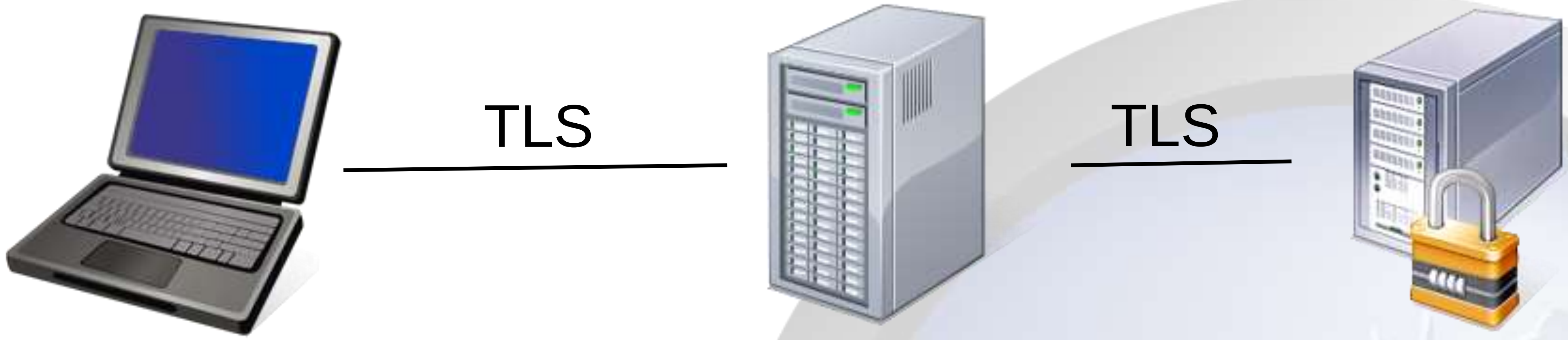
The Past: CA breaches
June (?) 2011: DigiNotar breach

- Discovered on June 19th
- July 10, 2011: wildcard cert issued for Google, subsequently used by unknown persons in Iran to conduct a man-in-the-middle attack against Google services
- August 28, 2011, certificate problems were observed on multiple Internet service providers in Iran
- Tor Project has published extensive updates on the scope of the attack, including a list of 531 fraudulent certificates issued by DigiNotar

The Past: CA breaches June (?) 2011: DigiNotar breach

- All browser vendors remove trust of DigiNotar swiftly, e.g. August 30, 2011: Mozilla removed DigiNotar certificates from their list of trusted CAs (via patches etc.)
- September 20, 2011 – DigiNotar filed for bankruptcy
- Remark: Google Chrome users were protected from this attack because Chrome was able to detect the fraudulent certificate due to pinning.
- Statements have appeared that the DigiNotar attacker is the same person who attacked Comodo earlier
- The attacker claims to be an individual Iranian who has chosen to help the government monitor individuals' communications. Additionally, he claims to have compromised four additional as-yet-unspecified certificate authorities.

MITMA - TLS attack



Attacker replaced Server cert with own compromised cert and could read all communication (incl. passwords) in the clear

The situation

- Browsers trust CA certificates for all domains equally (any trusted CA can sign for any identity, true or fake, e.g. google.com, paypal.com, …)
- hundreds of CAs
- From 46 countries/jurisdictions
- If a single one is broken, all TLS/SSL domains are prone to attacks

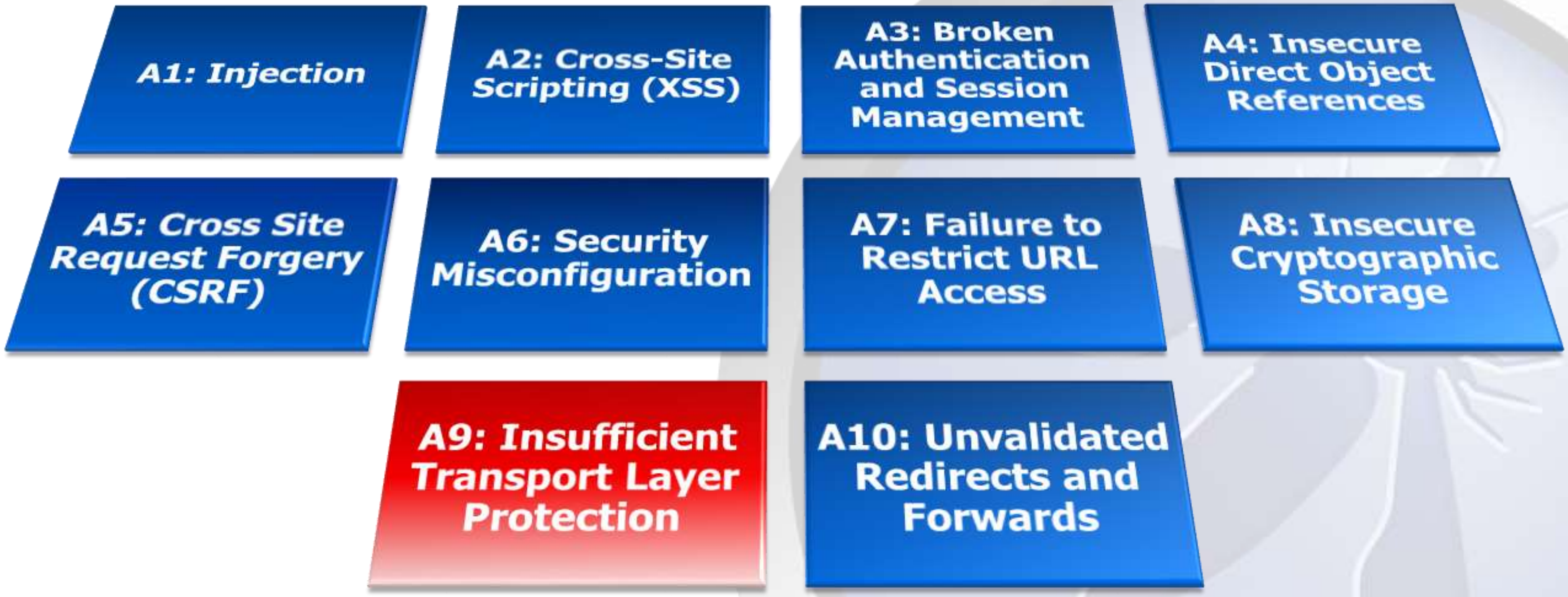
From EFF: SSL Observatory

- 1,482 CA Certificates trustable by Windows or Firefox
- 1,167 distinct issuer strings
- 651 organizations, but ownerships & jurisdictions overlap
- (If a CA can sign for one domain, it can sign for any domain.)

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

OWASP Top 10 – Insufficient Transport Layer Protection


A1: Injection
A2: Cross-Site Scripting (XSS)
A3: Broken Authentication and Session Management
A4: Insecure Direct Object References
A5: Cross Site Request Forgery (CSRF)
A6: Security Misconfiguration
A7: Failure to Restrict URL Access
A8: Insecure Cryptographic Storage
A9: Insufficient Transport Layer Protection
A10: Unvalidated Redirects and Forwards

What’s the problem

- Some are not using / not mandating TLS/SSL
- Relies on trust relationships (trust on first use / trusted source)
- Weak channel protection
- Authentication & leakage of credentials

=> Today, Web Applications try to fix this on the Application level with little support of the underlying infrastructure

A9 – Insufficient Transport Layer Protection

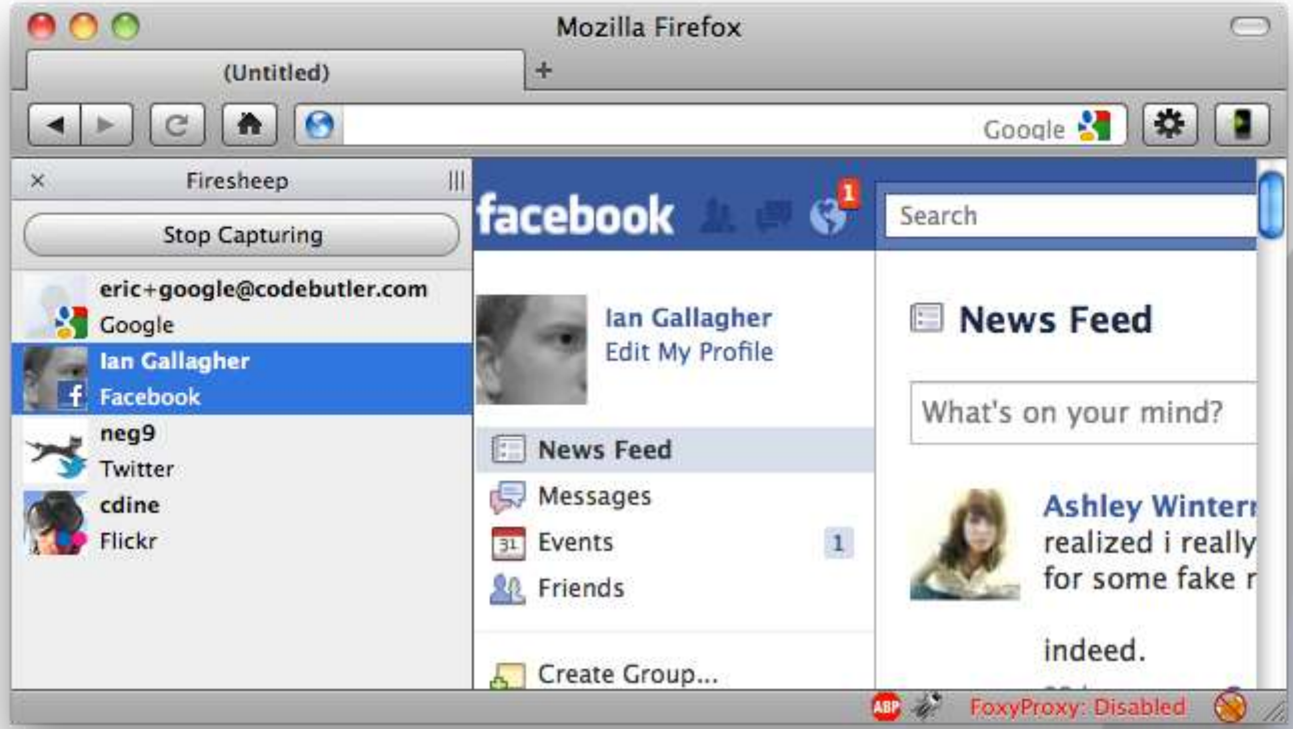
Transmitting sensitive data insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

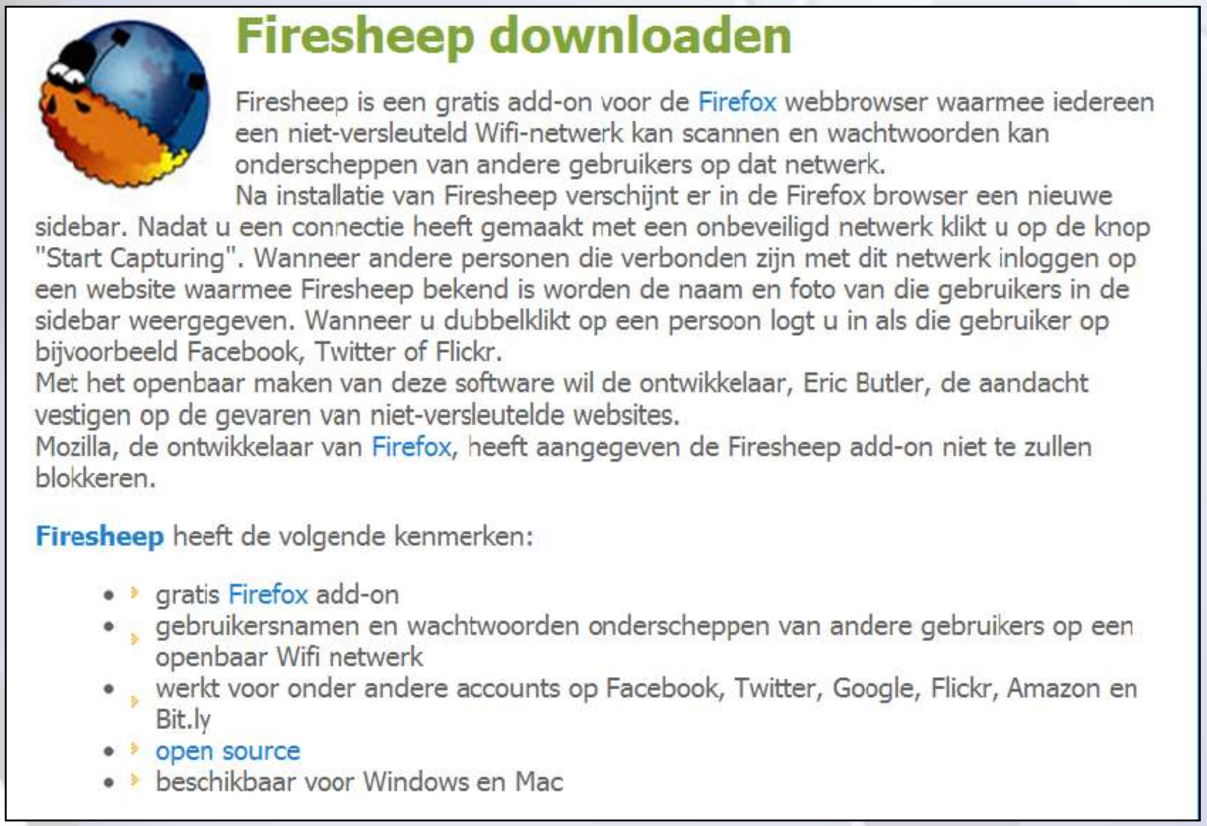
Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident
- Business gets sued and/or fined

Still not using SSL?



Now:
Redirect to https
before login



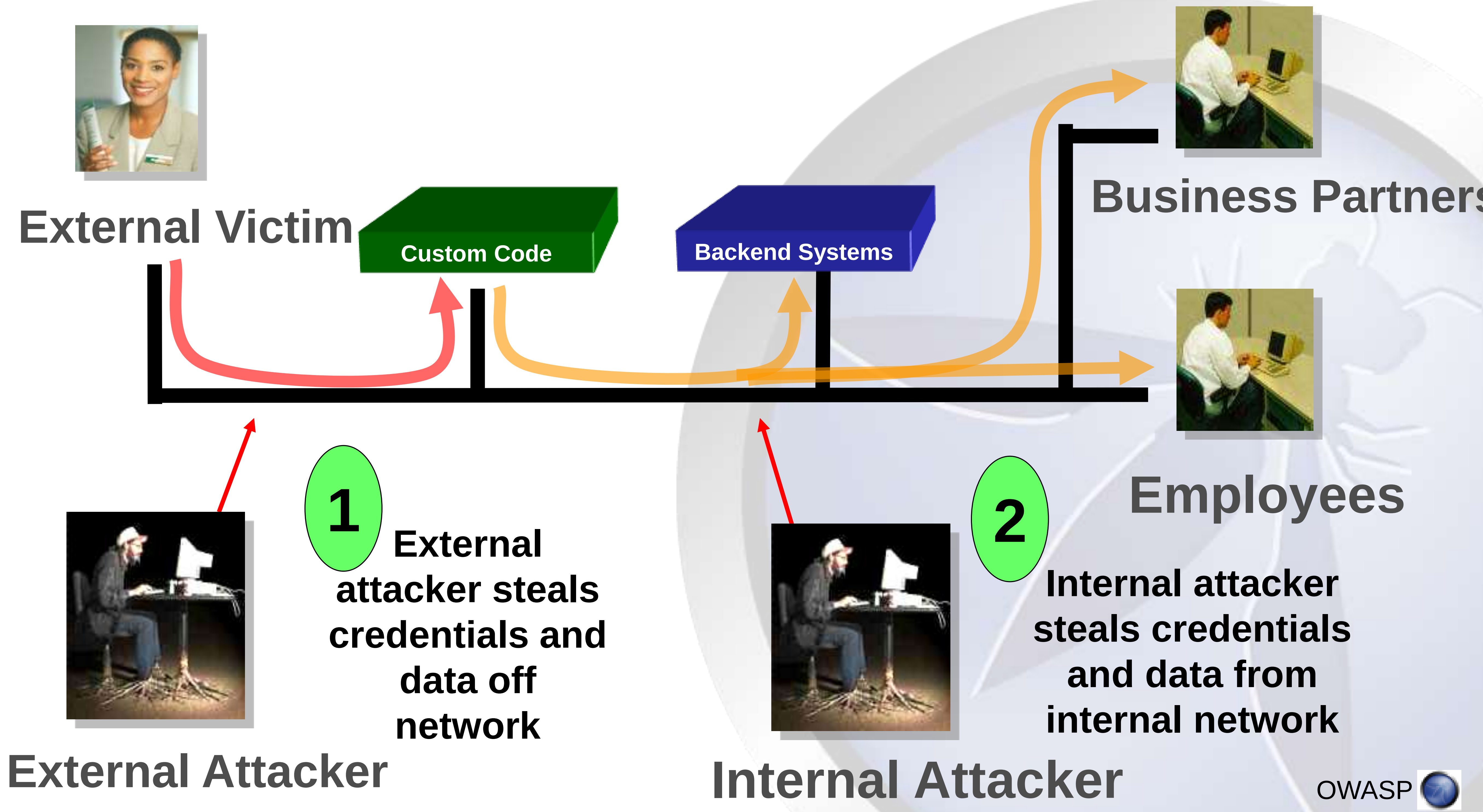
Firesheep downloaden

Firesheep is een gratis add-on voor de Firefox webbrowser waarmee iedereen een niet-versleuteld Wifi-netwerk kan scannen en wachtwoorden kan onderscheppen van andere gebruikers op dat netwerk.
Na installatie van Firesheep verschijnt er in de Firefox browser een nieuwe sidebar. Nadat u een connectie heeft gemaakt met een onbeveiligd netwerk klikt u op de knop "Start Capturing". Wanneer andere personen die verbonden zijn met dit netwerk inloggen op een website waarmee Firesheep bekend is worden de naam en foto van die gebruikers in de sidebar weergegeven. Wanneer u dubbelklikt op een persoon logt u in als die gebruiker op bijvoorbeeld Facebook, Twitter of Flickr.
Met het openbaar maken van deze software wil de ontwikkelaar, Eric Butler, de aandacht vestigen op de gevaren van niet-versleutelde websites.
Mozilla, de ontwikkelaar van Firefox, heeft aangegeven de Firesheep add-on niet te zullen blokkeren.

Firesheep heeft de volgende kenmerken:

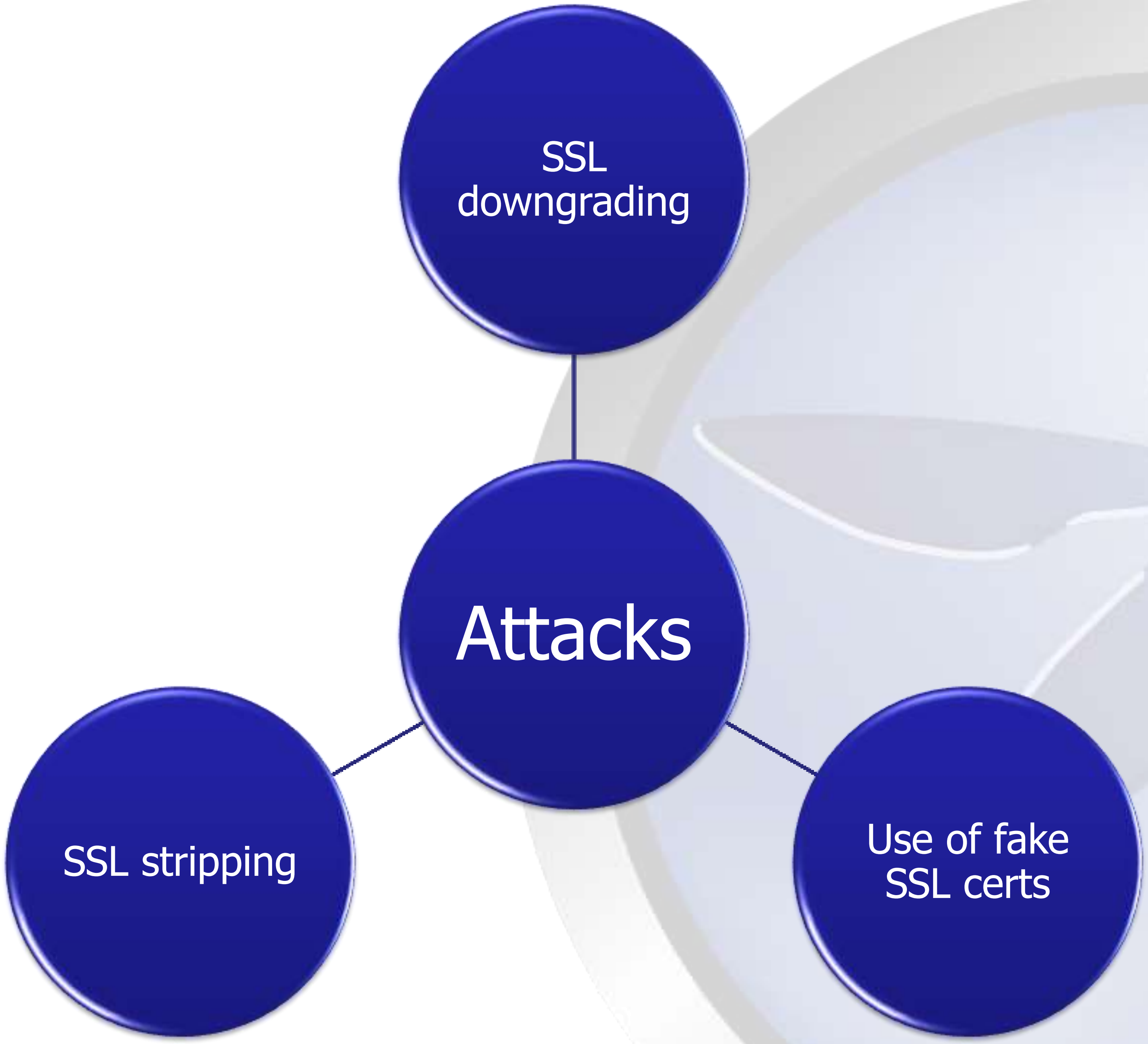
- gratis Firefox add-on
- gebruikersnamen en wachtwoorden onderscheppen van andere gebruikers op een openbaar Wifi netwerk
- werkt voor onder andere accounts op Facebook, Twitter, Google, Flickr, Amazon en Bit.ly
- open source
- beschikbaar voor Windows en Mac

Insufficient Transport Layer Protection

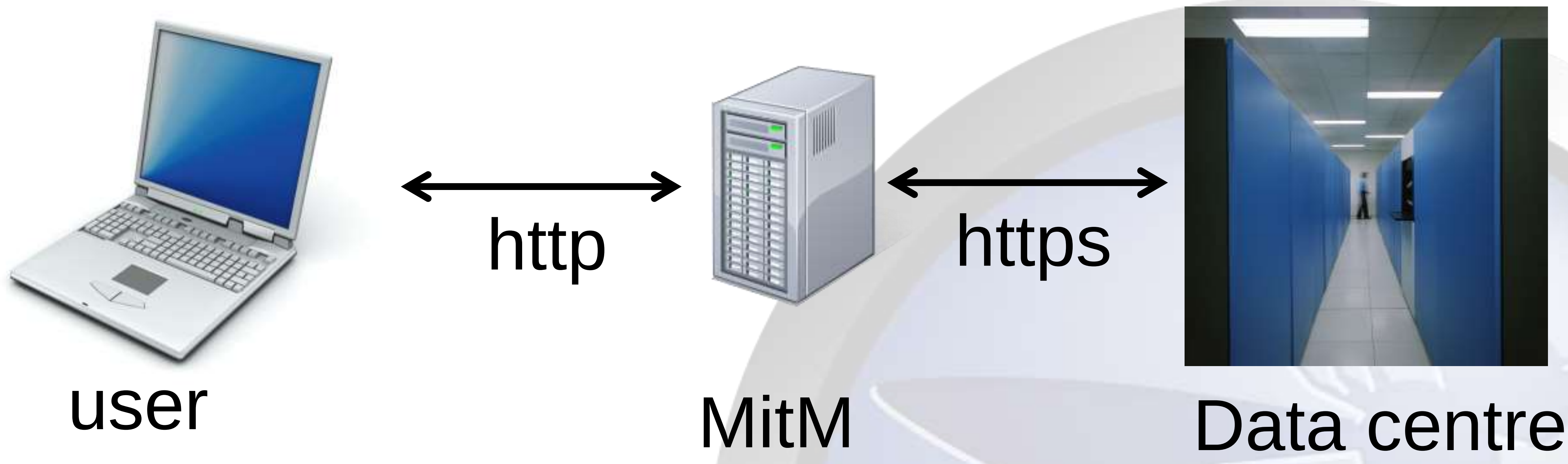


External Victim
Custom Code
Backend Systems
Business Partners
Employees
1
External attacker steals credentials and data off network
2
Internal attacker steals credentials and data from internal network
External Attacker
Internal Attacker

Common attack vectors



Moxie's SSL Strip



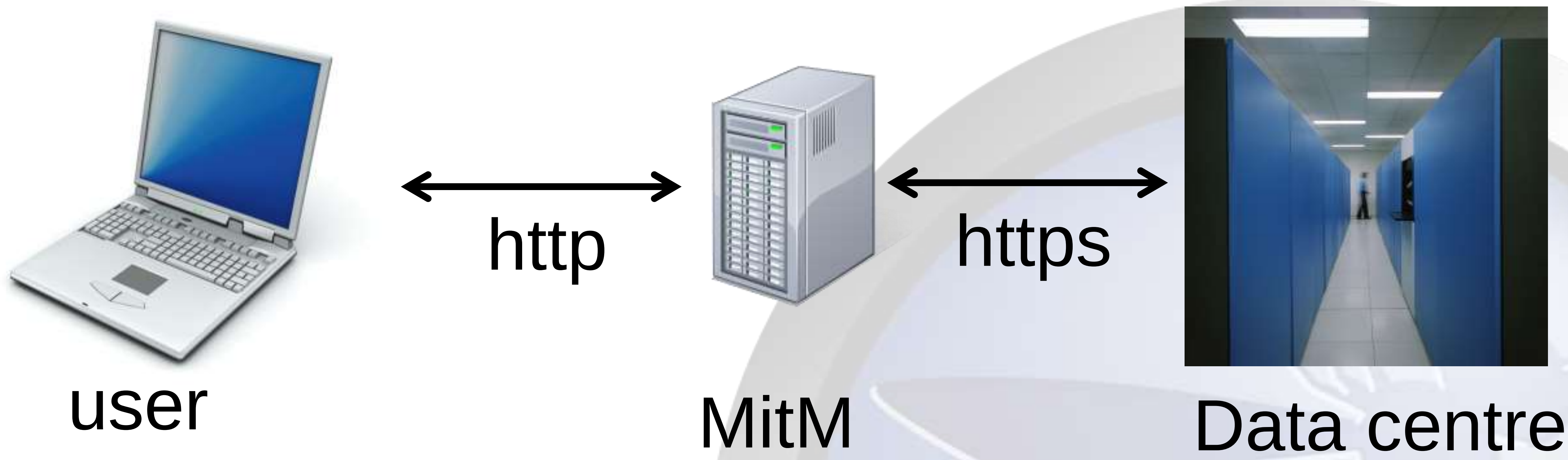
Terminates SSL

Changes https to http

Normal https to the server

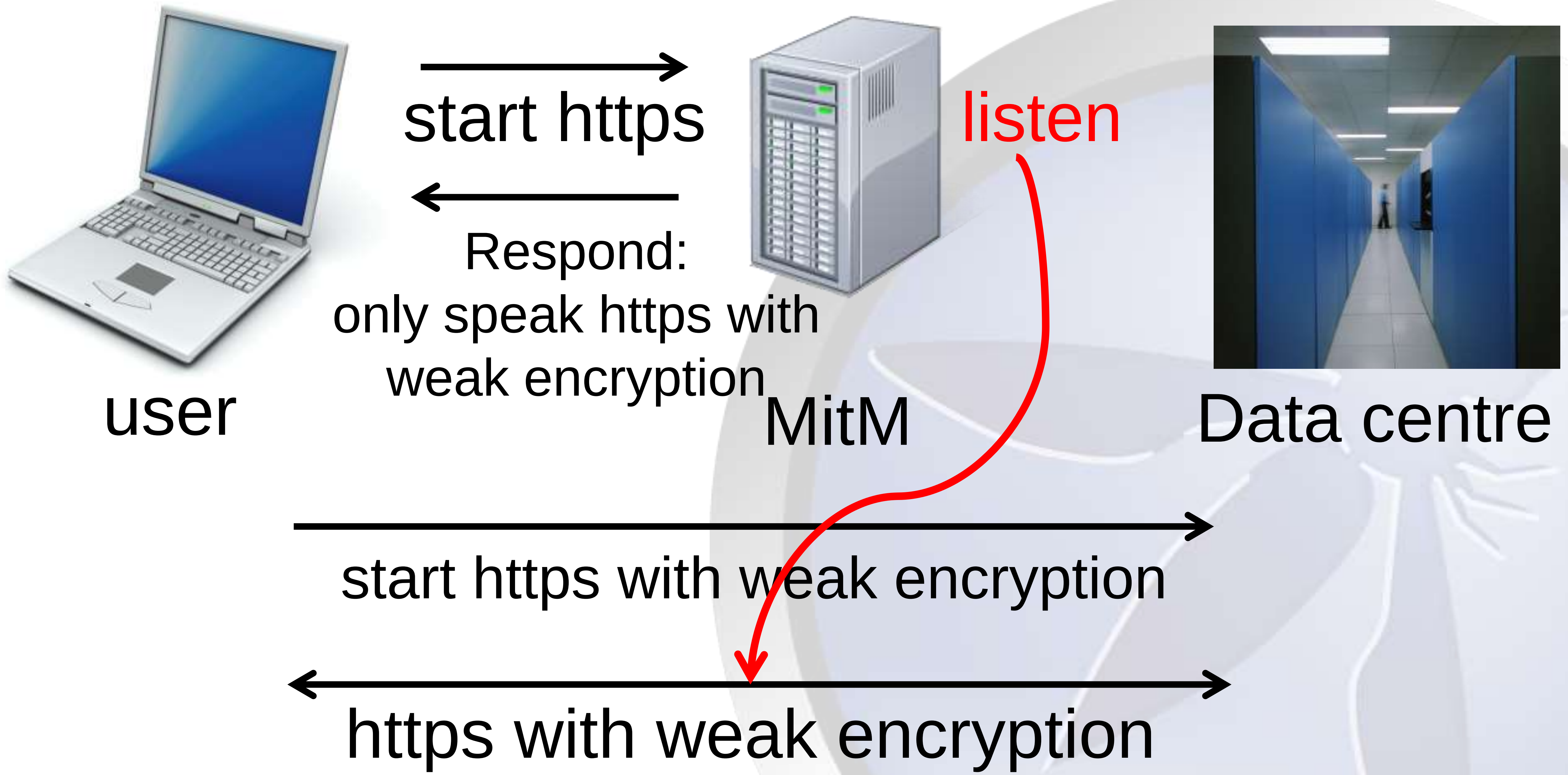
Acts as client

Moxie’s SSL Strip



Secure cookie?	Strip the secure attribute off all cookies.
Encoding, gzip?	Strip all encodings in the request.
Cached content?	Strip all if-modified-since in the request.
Sessions?	Redirect to same page, set-cookie expired

SSL dowgrading



A9 – Avoiding Insufficient Transport Layer Protection

Use the mechanisms correctly

- Use TLS on **all** connections with sensitive data
- Use standard strong algorithms (disable old SSL algorithms)
- Manage keys/certificates properly
- Verify SSL certificates before using them
- Use proven mechanisms when sufficient
  - E.g., SSL vs. XML-Encryption

See: http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for more details

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

Who – Introducing the Players

- OWASP
  - Top Ten
  - Browser Security Day at OWASP Summit
- IETF
  - Web Security WG
- Browser Vendors
- Secure Web-sites of critical information and payment systems (e.g. paypal, google, ebay, …)
- Security Researchers and Plug-in developers for browsers

What’s been done / what’s coming

- Secure the Channel:
  - HSTS - HTTP Strict Transport Security
  - Cert Pinning
  - TLS cert pinning in DNSSEC

HSTS - Secure Channels: Strict Transport Security

- Server declares **"I only talk TLS"**
  - Example:
    HTTP(S) Response Header:
    Strict-Transport-Security: max-age=15768000; includeSubDomains
- Header can be cached and also prevents leakage via subdomain-content through non-TLS links in content
- Weakness: "Trust on first use"
  - Possible pre-loaded HSTS in browsers
- Already first deployments

Cert Pinning (1)

`draft-ietf-websec-key-pinning-01`

- Server identities tend to be long-lived, but clients have to re-establish the server's identity on every TLS session.
- How could Google/Chrome be resilient to DigiNotar attack?
  - Google built-in in Chrome "preloaded" fingerprints for the known public keys in the certificate chains of Google properties. Thereby exposed the false *.google.com certificate DigiNotar signed.

Cert Pinning (2)

But….

…..preloading does not scale, so we need something dynamic:

=> Could use an HTTP header

i.e. transmit the SHA1 or SHA256 hash of the Subject Public Key Info structure of the X.509 certificate. (You could pin to end entity, intermediary, root. Select your degree of precision.)

Cert Pinning - Syntax

```
Header add Public-Key-Pins
  "max-age=10000; pin-
  sha1=\"ObT42aoSpAqWdY9WfRfL7i
  0HsVk=\"; pin-
  sha1=\"hvfkN/qlp/zhXR3cuerq6j
  d2Z7g=\"“
```

Cert Pinning - parameters

- List at least 2 certs: 1 live pin (a hash of an SPKI in the current cert chain) and at least one backup pin (a hash of an SPKI not in the current cert chain).
- Clients remember the most recently seen set of pins for max-age seconds after it was most recently seen.
- Clients drop TLS connections if not using the listed certs.

Cert Pinning – possible problems

Possible Problems:

- Bootstrap – “trust on first use”
  - Pre-loaded browser
- Servers might accidently "brick" themselves (pin for a long time to an SPKI which is later lost, for example) – reason why backup cert is mandatory
- Attackers with ISP capabilities / man-in-the-middle access may try to “brick” domains for users even when outside of their reach (imagine: Iranian travelling abroad and no longer able to access Google, etc.)
  - Recovery / cache flush mechanisms

Other Methods: Secure Channels: DNSSEC for TLS

- DNSSEC can be used to declare supported protocols for domains
- DNSSEC can be used to declare server certificate for domain

- Advantage: Advantage of trusted signed source
- Disadvantage: long time to deploy

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

Content Security Policy (CSP) and X-Frame-Options - New Technologies to protect against XSS and Clickjacking

A1: Injection

A2: Cross-Site Scripting (XSS)

A3: Broken Authentication and Session Management

A4: Insecure Direct Object References

A5: Cross Site Request Forgery (CSRF)

A6: Security Misconfiguration

A7: Failure to Restrict URL Access

A8: Insecure Cryptographic Storage

A9: Insufficient Transport Layer Protection

A10: Unvalidated Redirects and Forwards

A2 – Cross-Site Scripting (XSS)

Occurs any time…

- Raw data from attacker is sent to an innocent user's browser

Raw data…

- Stored in database
- Reflected from web input (form field, hidden field, URL, etc…)
- Sent directly into rich JavaScript client

Virtually every web application has this problem

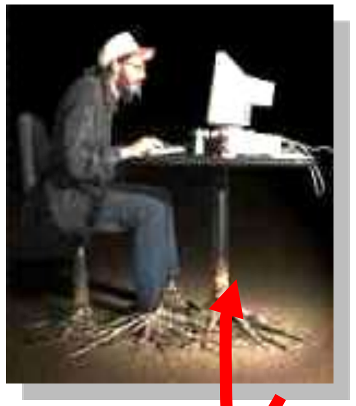
- Try this in your browser – javascript:alert(document.cookie)

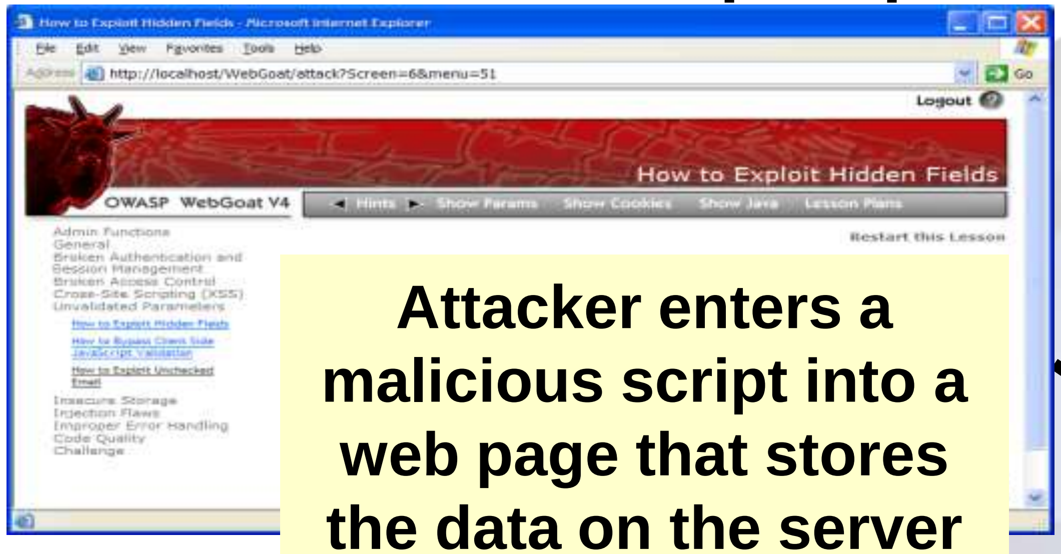
Typical Impact

- Steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user's behavior on vulnerable site and force user to other sites

Cross-Site Scripting Illustrated

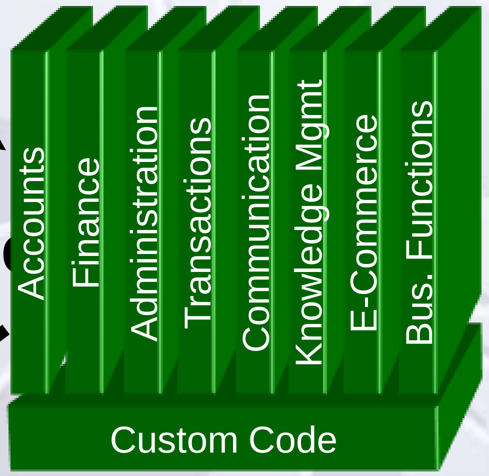
1 **Attacker sets the trap – update my profile**



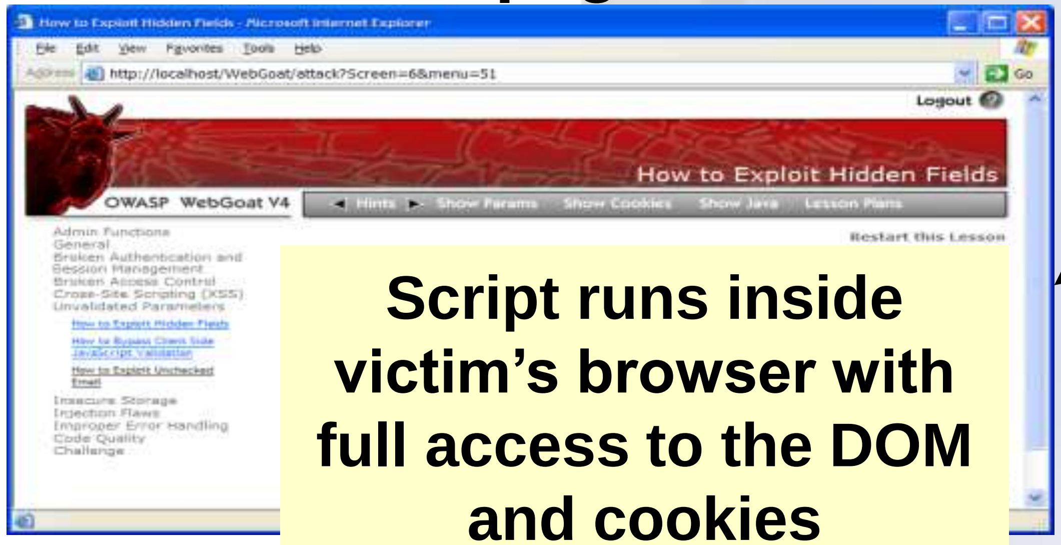

How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints Show Params Show Cookies Show Java Lesson Plans
Restart this Lesson
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge

Attacker enters a malicious script into a web page that stores the data on the server

Application with stored XSS vulnerability



2 **Victim views page – sees attacker profile**


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints Show Params Show Cookies Show Java Lesson Plans
Restart this Lesson
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge

Script runs inside victim's browser with full access to the DOM and cookies

3 **Script silently sends attacker Victim's session cookie**

A2 – Avoiding XSS Flaws

Recommendations

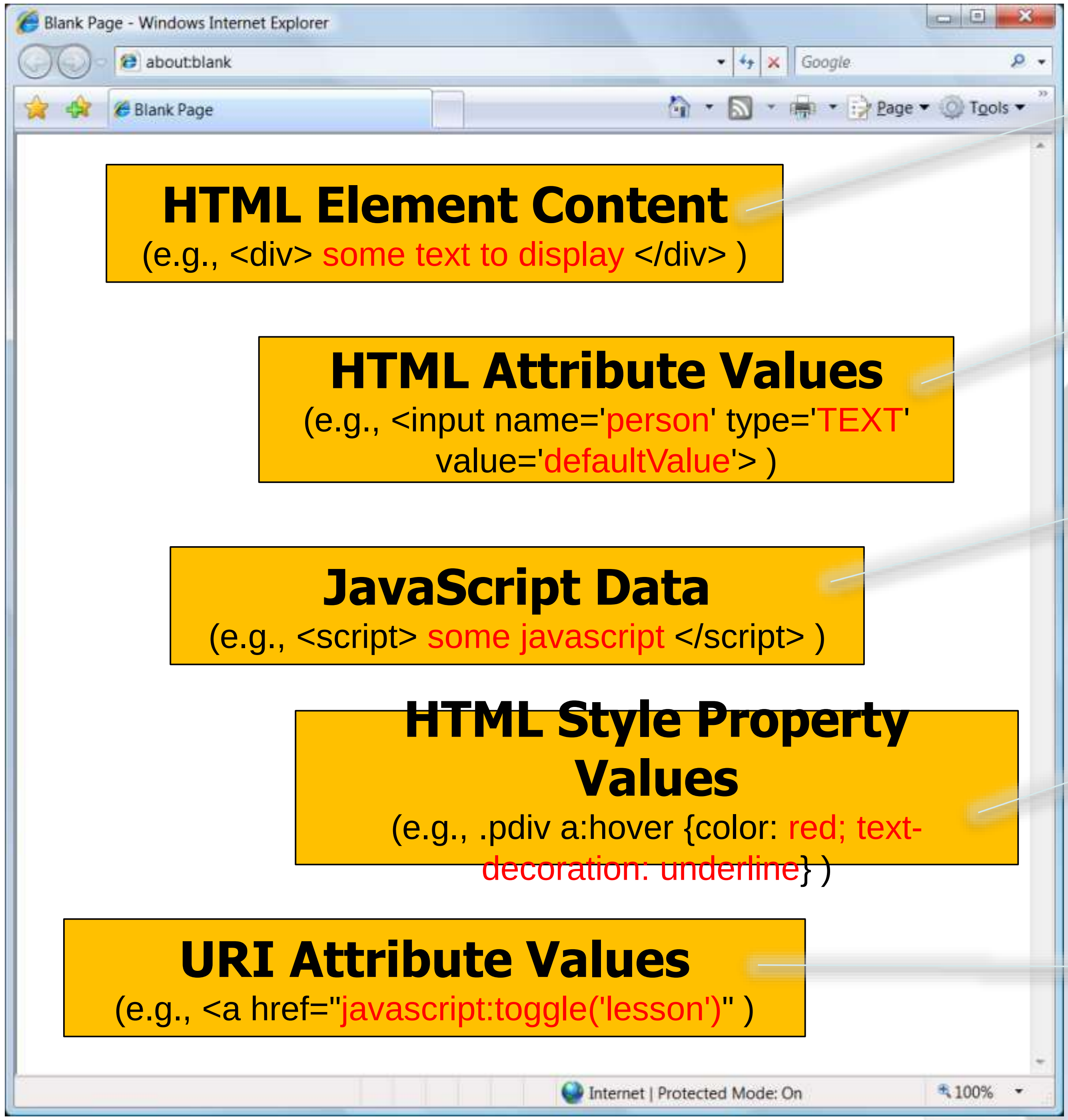
- Eliminate Flaw
  - Don't include user supplied input in the output page
- Defend Against the Flaw
  - Primary Recommendation: Output encode all user supplied input (Use OWASP's ESAPI to output encode:

    http://www.owasp.org/index.php/ESAPI
  - Perform 'white list' input validation on all user input to be included in page
  - For large chunks of user supplied HTML, use OWASP's AntiSamy to sanitize this HTML to make it safe

    See: http://www.owasp.org/index.php/AntiSamy
  - **Use Content Security Policy!**

References: For how to output encode properly, read the new http://www.owasp.org/index.php/XSS_(Cross Site Scripting) Prevention Cheat Sheet

Safe Escaping Schemes in Various HTML Execution Contexts


Blank Page - Windows Internet Explorer
about:blank
Google
Blank Page
Page
Tools
Internet | Protected Mode: On
100%

HTML Element Content
(e.g., <div> some text to display </div>)

HTML Attribute Values
(e.g., <input name='person' type='TEXT' value='defaultValue'>)

JavaScript Data
(e.g., <script> some javascript </script>)

HTML Style Property Values
(e.g., .pdiv a:hover {color: red; text-decoration: underline})

URI Attribute Values
(e.g., <a href="javascript:toggle('lesson')")

#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

ALL other contexts CANNOT include Untrusted Data

Recommendation: Only allow #1 and #2 and disallow all others

See: www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet for more details

CSP and X-Frame-Options
New Protection against XSS and Clickjacking

- `"Content-Security-Policy:" 1#policy`
- `"Content-Security-Policy-Report-Only:" 1#policy`

```
directive-name    = "default-src"

directive-value   = source-list
```

Example:

```
Content-Security-Policy: default-src 'self';
   script-src example.com
```

New Protection against XSS and Clickjacking X-Frame-Options and CSP

Content Security Policy (CSP)
new Browser Security Model for
Separation of Data and Program Code

Directives:

`script-src`	`media-src`
`object-src`	`frame-src`
`style-src`	`font-src`
`img-src`	`connect-src`

Frame-Options - History

X-Frame-Options widely deployed/used to prevent Click-jacking

- Running code and (some) consensus by implementers in using X-FRAME-OPTIONS

HTTP-Header:

- DENY: cannot be displayed in a frame, regardless of the site attempting to do so.
- SAMEORIGIN: can only be displayed if the top-frame is of the same “origin” as the page itself.

Frame-Options – Example Use-Cases

A.1. Shop

- An Internet Marketplace/Shop link/button to "Buy this" Gadget, wants their affiliates to be able to stick the "Buy such-and-such from XYZ" IFRAMES into their pages.

A.2. Confirm Purchase Page

- Onlineshop "Confirm purchase" anti-Click-Jacking page. The Confirm Purchase page must be shown to the end user without possibility of overlay or misuse by an attacker.

Frame-Options

In EBNF:

```
Frame-Options = "Frame-Options" ":"
"DENY"/ "SAMEORIGIN" / ("ALLOW-FROM"
":"URI)
```

- DENY: The page cannot be displayed in a frame, regardless of the site attempting to do so.
- SAMEORIGIN: can only be displayed in a frame on the same origin as the page itself.
- ALLOW-FROM: can only be displayed in a frame on the specified origin

Protection against Clickjacking with Frame-Options in CSP1.1

```
directive-name  = "frame-options"

directive-value = 'deny' / 'self' ['top-only']
   / 1*1<host-source> ['self'] /  1*1<host-
   source> 'top-only'
```

CSP – Content Security Policy

And special thanks to Sendsafely for donating the following slides and info about their experiences when implementing CSP.

History of Content Security Policy

Started at Mozilla, and has since grown into its own W3C specification

v1.0 – Release Candidate

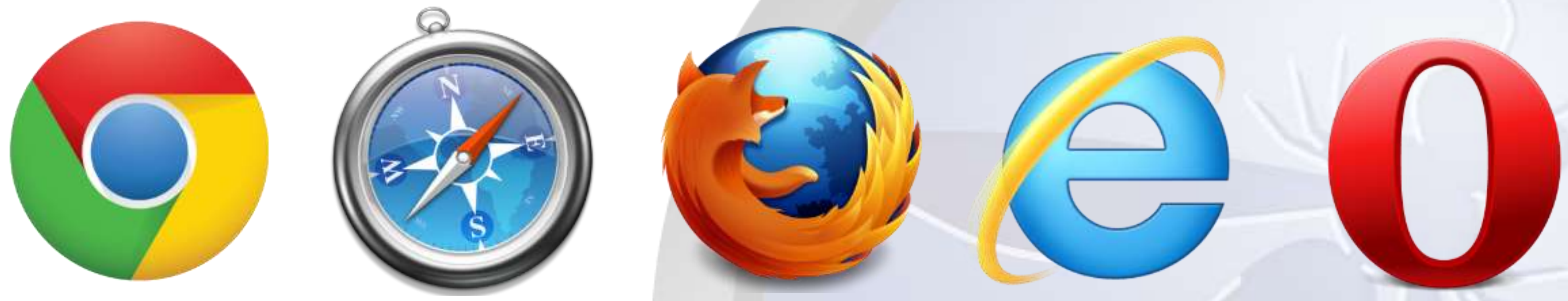
Support is pretty good across most major browsers

v1.1 – Draft

Browser support is lacking / spotty

What we will be talking about today.

Browser Support for CSP



DESKTOP	v14+	v6+	v4+	N/A	v15+

Who’s using CSP?

Only 191 out of Alexa top 1,000,000 sites

Source: Veracode - http://www.veracode.com/blog/2013/03/security-headers-on-the-top-1000000-websites-march-2013-report/

List includes some high-profile websites

- Facebook, Twitter, GitHub

What types of content can be restricted?

Content Type	CSP Attribute
JavaScript	script-src
CSS	style-src
Connections (XHR, WebSockets, etc)	connect-src
iFrame	frame-src
Images	img-src
Web Fonts	font-src
Video and Audio Sources	media-src
Embedded elements, applets and plug-ins	object-src

How is content restricted?

By Source (per directive)

- 'none', 'self', specific hostnames, *

By Category

- unsafe-inline (script-src & style-src only)
- unsafe-eval (script-src only)

Missing directives inherit from default-src

- Assumed to be * if missing (open by default)

Objectives of their CSP Implementation

Be as Strict as Possible

Deny everything by default, only allow what is needed

Monitor Attempts to Bypass

Leverage built in CSP error reporting

Maximize Coverage Across Browsers

Handle browser-specific nuances

We want to be as strict as possible

Our policy uses a "Deny by Default" model

- default-src= "none"

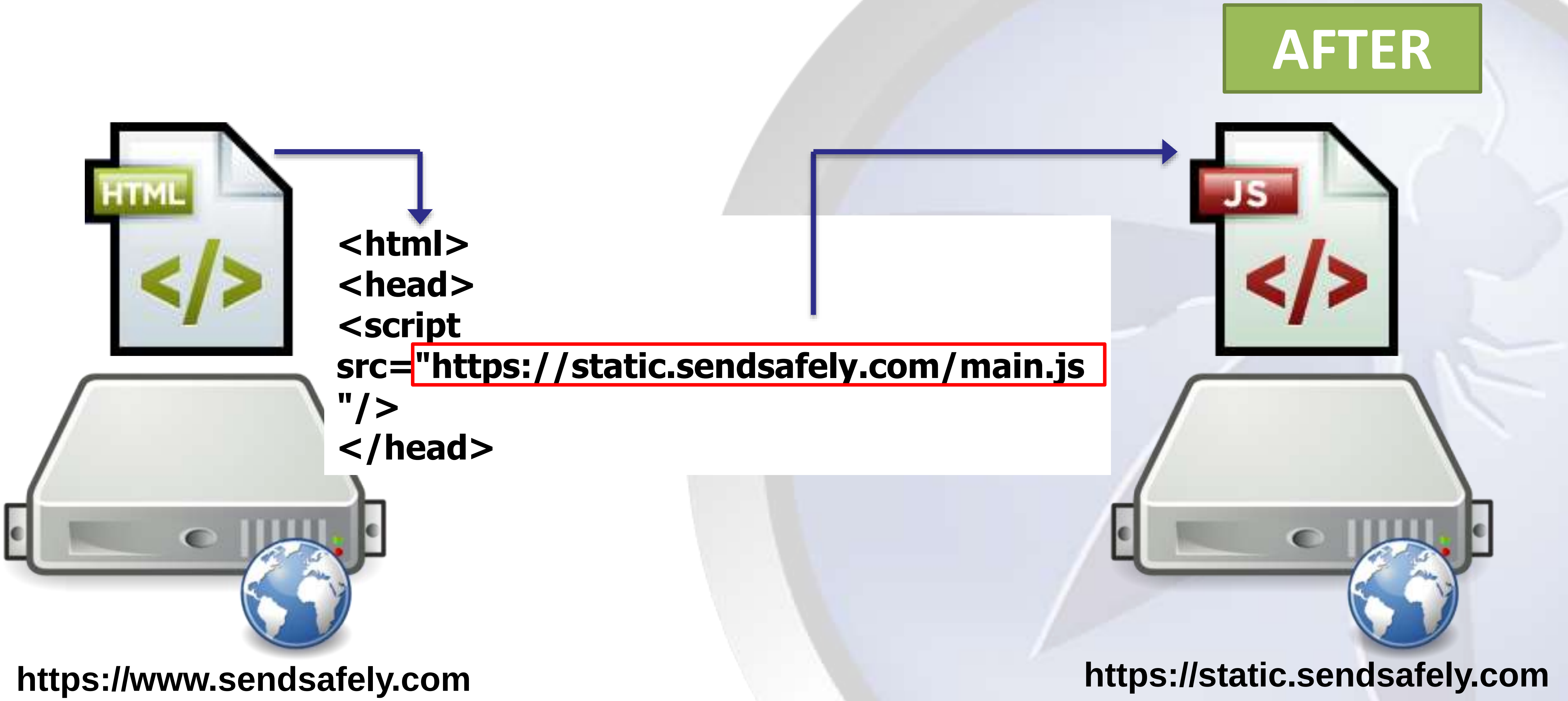
Allow JavaScript only from our static server

- static.sendsafely.com

No string-to-code functions or in-line scripts

- No use of unsafe-eval or unsafe-inline

Dedicated Site for Authorized Scripts



```
<html>
<head>
<script
src="https://static.sendsafely.com/main.js
"/>
</head>
```

And now for the harder part…

Removal of patterns banned by unsafe-eval

- Covers text-to-Javascript functions like eval()
- Luckily they had already banned these

Removal of all in-line JavaScript

- This was pervasive within the code
- Most of your code will likely look like our examples

Eliminating In-line Scripting (Example #1)

Before (HTML w/ In-line JavaScript)

```
<a href="javascript:doSomething()">Do something</a>
```

Find the link based on it’s ID

After (HTML without JavaScript)

```
<a id="my-link">Do something</a>
```

After (JavaScript loaded from static.sendsafely.com)

```
var link = document.getElementById("my-link");
link.addEventListener("click", doSomething, false);
```

Eliminating In-line Scripting (Example #1)

Before (HTML w/ In-line JavaScript)

```
<a href="javascript:doSomething()">Do something</a>
```

Wire up the "click" event to "doSomething()"

After (HTML without JavaScript)

```
<a id="my-link">Do something</a>
```

After (JavaScript loaded from static.sendsafely.com)

```
var link = document.getElementById("my-link");
link.addEventListener("click", doSomething, false);
```

Eliminating In-line Scripting (Example #2)

Before (HTML w/ In-line JavaScript)

```
<type="text" name="email" onkeyup="doSomethingElse(arg1, arg2)">
```

Embed argument values within the HTML as data-vars

After (HTML without JavaScript)

```
<type="text" id="my-email-field" name="email"
data-arg-one="arg1" data-arg-two="arg2">
```

After (JavaScript loaded from static.sendsafely.com)

```
$(“#my-email-field”).keyup(function()
{   doSomethingElse(this.getAttribute(“data-arg-one”),
this.getAttribute(“data-arg-two”));  });
```

Eliminating In-line Scripting (Example #2)

Before (HTML w/ In-line JavaScript)

```
<type="text" name="email" onkeyup="doSomethingElse(arg1, arg2)">
```

Use JQuery to wireup event and pass data-args to function

After (HTML without JavaScript)

```
<type="text" id="my-email-field" name="email"
data-arg-one="arg1" data-arg-two="arg2">
```

After (JavaScript loaded from static.sendsafely.com)

```
$(“#my-email-field”).keyup(function()
{   doSomethingElse(this.getAttribute(“data-arg-one”),
this.getAttribute(“data-arg-two”));  });
```

Working with Third Party Scripts

We try to minimize reliance on third party scripts

- Occasionally the cost/benefit is worth using them
- Examples: JQuery, reCAPTCHA

Our policy is to host all JavaScript on our servers

- Re-factoring to make CSP compliant is not desirable

Content Type	Directive	
Default	**default-src**	✓
JavaScript	**script-src**	✓
Stylesheets	**style-src**	✓
Connections	**connect-src**	✓
iFrame	**frame-src**	✓
Images	**img-src**	✓
Web Fonts	**font-src**	**N/A**
Video and Audio	**media-src**	**N/A**
Embedded Objects	**object-src**	**N/A**

CSP

default-src 'none'; script-src https://static.sendsafely.com https://www.google.com; style-src 'self' 'unsafe-inline'; connect-src 'self'; frame-src 'self'; img-src 'self' https://www.google.com

Allowing Java Applets

The same pages that use Web Workers use a Java Applet when using a browser without HTML5

- Initially we planned for an object-src directive to allow

As it turns out...

None of the browsers we tested block the <applet> tag (even with object-src: 'none')

- Chrome, FireFox, Safari

Were we missing something?

- Thread started on public-webappsec@w3.org

CSP Error Reporting

Built-in feature of CSP, used to report violations

- Browser sends error reports when a directive is violated
- Source URL, offending directive and code fragment

▪ If CSP kicks in for a user, you will know

- Great for development and testing
- CSP also works in report-only mode

Example CSP Error Report

Report

document-uri: https://mysite.com/vulnerable/
referrer: https://evil.com/attack
blocked-uri: self
violated-directive: inline script base restriction
source-file: https://mysite.com/vulnerable/
script-sample: alert(document.cookie)

The URI of the page where the error occurred

Concluding Thoughts

CSP works, especially with regards to XSS

- But you will likely not end up with a bullet proof CSP

Error Reporting causes A LOT of noise

- Mainly due to browser add-ons

Browser support currently useable for 1.0 only

- 1.0 features are fairly stable, not much coverage for 1.1

Web Security – New Browser Security Technologies

- Past Attacks/Breaches
- Insufficient Transport Layer Protection
  - Solutions
    - HSTS - HTTP Strict Transport Security
    - Cert Pinning
- New Protection against XSS and Clickjacking
  - X-Frame-Options and CSP
- When

When - Timeframes

- HSTS Strict Transport Security – now (RFC 6797)
- Cert Pinning Q2 2014
- TLS in DNSSEC – 201?
- X-Frame-Options – now (RFC 7034)
- CSP 1.0 – now - published as a W3C Candidate Recommendation. – Q4 2012
- CSP 1.1 – Q2 2014

Join the discussion

Ideas / feedback / participation welcome

IETF Websec:
http://tools.ietf.org/wg/websec/charters

W3C WebAppSec:

http://www.w3.org/2011/webappsec/

Or drop me an email:
tobias.gondrom@gondrom.org

???

Questions?

Thank you