# HTTP BASICS

# WHERE ARE WE GOING?

| | |
|---|---|
| **HTTP Basics** | HTTP Request Methods |
| | HTTP Security Response Headers |
| | Sensitive Data In Transit |
| | Intercepting Proxy |
| | Don’t Trust The HTTP Request! |

# WEB APPLICATION BEHAVIOUR

- HTTP is **stateless**. Requests and responses between browsers and servers have no shared memory. Application layer sessions are needed to track state.
- **Dynamic Scripting** can occur on **Server-Side** (e.g. RoR, Django, ASP.NET, JSP, Express, etc) or on **Client-Side** (Javascript, Flash, Applets).
- A web server or an application server can deliver **HTML** to be directly rendered by the web browser. Or, the server might deliver data as **JSON** or **XML** to be processed by a Client-Side application in the browser.
- Requests for data such as images, scripts, and stylesheets are typically retrieved using **HTTP GET**. Requests from HTML forms typically submit data using **HTTP POST**. AJAX requests can additionally submit HTTP requests of types **PUT**, **PATCH**, and **DELETE**.

# WHAT ARE HTTP HEADERS?

- HTTP headers are components of the message header of HTTP **Requests** and **Responses**.
- HTTP headers are used to define **meta-information** for an HTTP transaction.
- HTTP headers are colon-separated name-value pairs in clear-text string format, terminated by a carriage return (\r) and line feed (\n) character sequence.

http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

# EXAMPLES OF HTTP REQUEST HEADERS

**Authorization:**

```
Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

**Accept:**

```
text/plain
```

**Content-Type:**

```
application/x-www-form-urlencoded
```

**User-Agent:**

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:30.0) Gecko/20100101 Firefox/30.0
```

# VALIDATING HTTP REQUEST HEADERS

- ✓ Are the headers themselves known to **IANA**?
- ✓ Are the **number** of headers received appropriate to the application context?
- ✓ Do each of the headers come with a pre-determined regular expression or equivalent for **validation**?
- ✓ What headers are usually seen in **context** with other headers?
- ✓ How do I detect **missing** headers?
- ✓ Some headers occur in context of the **application** and are not global. For example, is a cookie scoped to a domain?
- ✓ Some headers have **time components** to them such as *expires*. Is the header contextually validated by date checks?

*Official standard on HTTP Request Headers*
*https://www.iana.org/assignments/message-headers/message-headers.xhtml*

# HTTP REQUEST: GET VS POST

## HTTP GET Request

```
GET https://example.com/search.jsp?name=foo HTTP/1.0\r\n
User-Agent: Mozilla/4.0\r\n
Host: example.com\r\n
Cookie: SESSIONID=2KDSU72H9GSA289\r\n
\r\n
```

## HTTP POST Request

```
POST https://example.com/search.jsp?data=jim HTTP/1.0\r\n
User-Agent: Mozilla/4.0\r\n
Host: example.com\r\n
Content-Length: 16\r\n
Cookie: SESSIONID=2KDSU72H9GSA289\r\n
\r\n
name=blah&type=1
\r\n
```

# TRIGGERING AN HTTP(S) GET

- Typing into a **URL bar**
- **Bookmark** selection
- **<img>** tag
- Loading a **JS** or **CSS** file
- Loading a **Webfont**
- HTML **Form** submission method="GET"
- **jQuery**.get() *http://api.jquery.com/jQuery.get/*

# HTTP GET REQUEST: PLAINTEXT IMAGE

```
GET /personal/dancing/naked/inebriated/kauaifun.jpg HTTP/1.1\r\n
Host: images.manico.net\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:30.0)
Gecko/20100101 Firefox/30.0\r\n
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
DNT: 1\r\n
Connection: keep-alive\r\n
\r\n
```

# HTTP GET REQUEST: INSECURE FORM SUBMISSION

```
GET
 http://example.com/search?form_name=home&title=security&database=cli
 ents HTTP/1.1\r\n
Host: example.com\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
 rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7 (.NET CLR 3.5.30729)\r\n
Accept:
 text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Proxy-Connection: keep-alive\r\n
Referer: http://company.com?username=Jim&pass=rp2h6jibalice\r\n
Cookie: JSESSIONID=4d9jjtqsr5rba.alice; AxData=; Axxd=clients\r\n
\r\n
```

# HTTP GET SHOULD BE BORING

- Most web frameworks intentionally do not provide CSRF protection for **GET** requests
- A **GET** request should not produce side effects. It should be "Nullipotent".
- A **GET** request should only be used for data retrieval
- A **GET** request should NEVER be used for:
  - Logging out a user
  - Logging in a user
  - Deleting a resource
  - Modifying a resource
  - Creating a resource
  - Sending an email

# HTTP GET PARAMETER LEAKAGE

- Bookmarks
- Browser History
- Proxy Server Logs
- Web Server Logs
- Referrer Request Headers

# TRIGGERING AN HTTP/S POST

## HTML Form POST Submission

```
<form
    action="https://acme-bank.example/payment"
    method="POST"
    id="payment-form">
```

## jQuery.post() *http://api.jquery.com/jQuery.post/*

```
$.post(
  "https://acme-bank.example/payment",
  function () {
    $(".result").html("Payment was successful");
  }
);
```

# HTTP POST REQUEST

```
POST https://login.example.com:443/login.php?loginfail=3 HTTP/1.1\r\n
Host: login.example.com\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
  rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7 (.NET CLR 3.5.30729)\r\n
Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Referer: https://www.example.com/\r\n
Cookie: JSessionID=1263464364617-95d75464239e7\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-length: 224\r\n
\r\n
locale=en_US&email=joe@example.com&pass=letmein123!!Let
\r\n
```

# HTTP PUT REQUEST

- An **HTTP PUT** request is used to replace a resource, or to create a new resource where the identifier of the resource is known.
- The same security precautions that apply to an **HTTP POST** request should also apply to a PUT request.
- **Never** send sensitive data in the **query string** of an HTTP PUT request

```
$.ajax(
  "https://contact-manager.example/contacts/1234",
  dataType: "json",
  type: "PUT",
  data: {
    name: "John Doe",
    email: "john.doe@example.com"
  }
);
```

# HTTP PATCH REQUEST

- An **HTTP PATCH** request is used to apply partial modifications to a resource.
- The same security precautions that apply to an **HTTP POST** request should also apply to a **HTTP PATCH** request.
- **Never** send sensitive data in the **query string** of an **HTTP PATCH** request

```
$.ajax(
  "https://contact-manager.example/contacts/1234",
  dataType: "json",
  type: "PATCH",
  data: {
    email: "john.doe@example.com"
  }
);
```

# HTTP DELETE REQUEST

- An **HTTP DELETE** request is used to delete a resource.
- The same security precautions that apply to an **HTTP POST** request should also apply to a PUT request.
- **Never** send sensitive data in the **query string** of an HTTP PUT request.
- Not all web servers and application frameworks will allow for a message body in an **HTTP DELETE**. Therefore, it is sometimes possible that sensitive cannot be securely sent from an **HTTP DELETE**.

```
$.ajax(
  "https://contact-manager.example/contacts/1234",
  dataType: "json",
  type: "DELETE"
);
```

# TRANSPORTING SENSITIVE DATA

- **Never** transmit sensitive data over HTTP/S GET
- **Always use SSL for everything!**
- In HTML forms, **only** submit sensitive data over **HTTPS POST**
- When using AJAX, submit sensitive data **only** using **POST**, **PUT**, and **PATCH**
- **Only** submit sensitive data only in the **HTTPS REQUEST BODY**
- **Never** submit sensitive data in the HTTP/S query string

BCG RISK ADVISORY

# EXAMPLE HTTP RESPONSE

```
HTTP/1.1 200 OK
Server:  Apache-Coyote/1.1
Cache-Control: no-cache, no-store, must-revalidate
Expires: -1
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 03 Oct 2014 19:55:36 GMT

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>WOOT HTML5</title>
  </head>
  <body>
  <h1>I LOVE HTML</h1>
 </body>
</html>
```

# HTTP RESPONSE Set-Cookie HEADER

```
Set-Cookie: NAME=VALUE; expires=EXPIRES;
            path=PATH; domain=DOMAIN;
            secure; httponly;
```

| | |
|---|---|
| **Name** | The name of the cookie parameter |
| **Value** | The parameter value |
| **Expires** | The date at which to discard the cookie. If absent, the cookie will not be persistent, and will be discarded when the browser is closed. If "-1", the cookie will be discarded immediately. |
| **Domain** | The domain that the cookie applies to |
| **Path** | The path that the cookie applies to |
| **Secure** | Indicates that the cookie can only be used over secure HTTPS. USE THIS! |
| **HttpOnly** | Indicates that the cookie can only be modified and accessed from the server. For example, JavaScript within the browser application will not be able to access the cookie. USE THIS FOR SESSION IDs! |

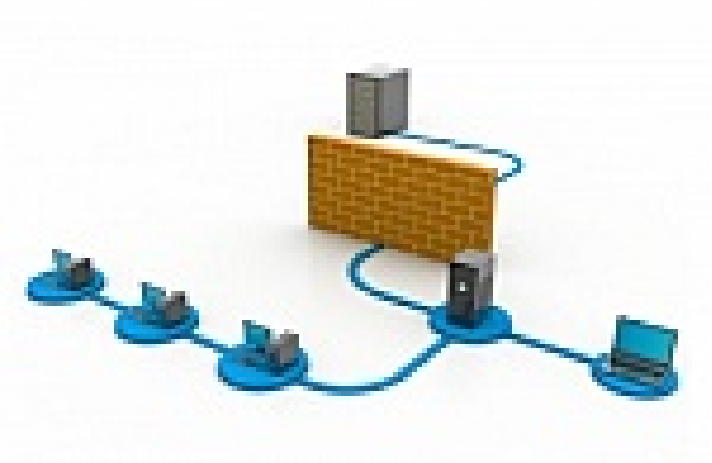
# WHAT ARE HTTP RESPONSE HEADERS?

- HTTP headers are components of the **message header** of HTTP Responses.
- HTTP headers define different aspects of an **HTTP transaction.**
- HTTP headers are colon-separated name-value pairs in clear-text string format, terminated by a carriage return (\r) and line feed (\n) character sequence.

*http://en.wikipedia.org/wiki/List_of_HTTP_header_fields*

# HTTP RESPONSE SECURITY HEADERS SUMMARY

- X-Frame-Options
- X-Xss-Protection
- X-Content-Type-Options
- Content Security Policy
- Access-Control-Allow-Origin
- HTTPS Strict Transport Security
- Cache-Control / Pragma



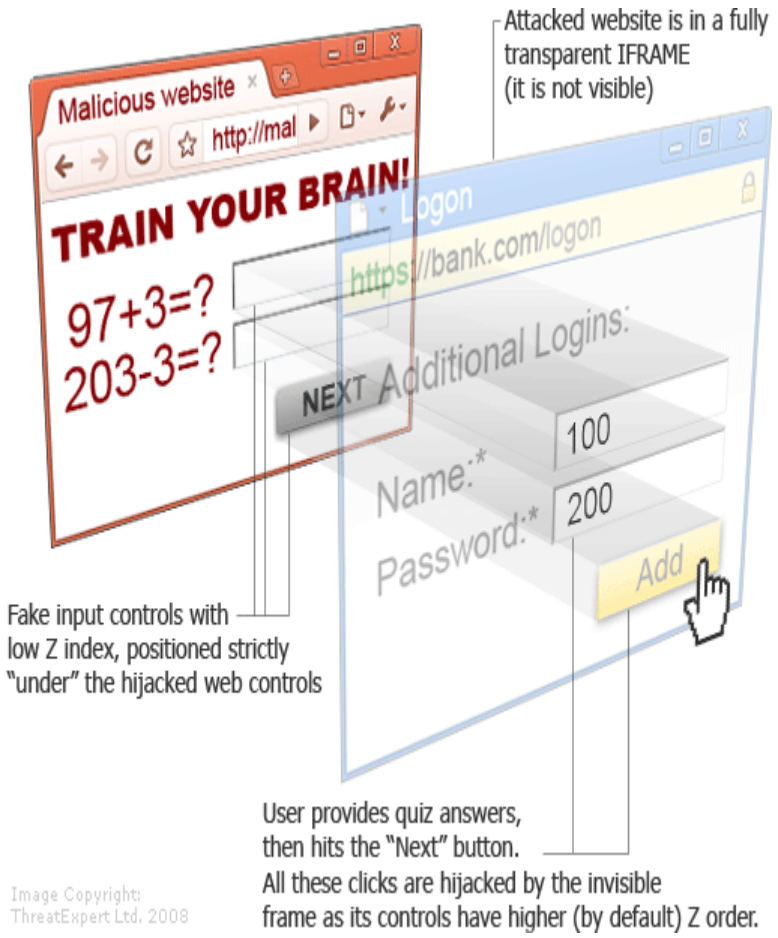
BCG RISK ADVISORY

# HTTP RESPONSE SECURITY HEADERS

| | |
|---|---|
| **X-Frame-Options** | ❖ Set to "SAMEORIGIN" to allow framing on same domain.<br>❖ Set to "DENY" to deny framing at all<br>❖ Set to "ALLOWALL" if you want to allow framing for all website |
| **X-XSS-Protection** | ❖ Set to "1; mode=block" to use XSS Auditor and block page if XSS attack is detected.<br>❖ Set to "0;" if you want to switch XSS Auditor off. This is useful if response contents scripts from request parameters |
| **X-Content-Security-Policy** | ❖ A powerful mechanism for controlling which sites certain content types can be loaded from |
| **Access-Control-Allow-Origin** | ❖ Used to control which sites are allowed to bypass same origin policies and send cross-origin requests. |
| **Strict-Transport-Security** | ❖ Used to control if the browser is allowed to only access a site over a secure connection |
| **Cache-Control** | ❖ Used to control mandatory content caching rules |

# HTTP RESPONSE HEADER: X-Frame-Options

- Protects you from most classes of Clickjacking
- X-Frame-Options: DENY
- X-Frame-Options: SAMEORIGIN
- X-Frame-Options: ALLOW FROM example.com




BCG RISK ADVISORY

# HTTP RESPONSE HEADER: X-Xss-Protection

Use the browser's built-in XSS auditor:

```
X-Xss-Protection: 1; mode=block
```

Disable the browser's built-in XSS auditor:

```
X-Xss-Protection: 0;
```

# CONTENT SECURITY POLICY

- Move all inline script and style into separate files
- Add the X-Content-Security-Policy response header to instruct the browser that CSP is in use
- Define a policy for the site regarding loading of content

*Anti-XSS W3C standard*
*http://www.w3.org/TR/CSP/*

*CSP Support Statistics*
*http://caniuse.com/#feat=contentsecuritypolicy*

*CSP Example Usage*
*http://content-security-policy.com/*

# OTHER SSL FAILS

- Posting passwords or other sensitive data over HTTP
- Using weak version of SSL
- Using weak ciphers
- Terminating SSL early in your infrastructure
- Trusting the CA system ☺

# HTTP RESPONSE HEADER: Strict-Transport-Security

Forces your browser to always use HTTPS

## Base case:

```
Strict-transport-security: max-age=10000000
```

## Do all of your subdomains support SSL?

```
Strict-transport-security: max-age=10000000; includeSubdomains
```

BCG RISK ADVISORY

# DISABLING THE BROWSER CACHE

Add the following as part of your HTTP Response:

**Cache-Control:** no-store, no-cache, must-revalidate
**Expires:** -1

# APPLY ALL THE HEADERS!

**strict-transport-security:** max-age=631138519\r\n

**version:** HTTP/1.1\r\n

**x-frame-options:** SAMEORIGIN\r\n

**x-gitsha:** d814fdf74482e7b82c1d9f0344a59dd1d6a700a6\r\n

**x-rack-cache:** miss\r\n

**x-request-id:** 746d48ca76dc0766ac24e74fa905be11\r\n

**x-runtime:** 0.023473\r\n

**x-ua-compatible:** IE=Edge,chrome=1\r\n

**x-webkit-csp-report-only:** default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'\r\n

**content-security-policy-report-only:** default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'\r\n

**x-content-security-policy-report-only:** default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'\r\n

# ASVS 2 HTTP REQUIREMENTS: EASY

| | |
|---|---|
| **V11.2** | Verify that the application accepts only a defined set of HTTP request methods, such as GET and POST and unused methods are explicitly blocked. |
| **V11.3** | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8). |
| **V11.8** | Verify that HTTP headers and / or other mechanisms for older browsers have been included to protect against clickjacking attacks. |

# ASVS 2 HTTP REQUIREMENTS: INTERMEDIATE

| | |
|---|---|
| **V11.6** | Verify that HTTP headers in both requests and responses contain only printable ASCII characters. |
| **V11.9** | Verify that HTTP headers added by a frontend (such as X-Real-IP), and used by the application, cannot be spoofed by the end user. |
| **V11.10** | Verify that the HTTP header, X-Frame-Options is in use for sites where content should not be viewed in a 3rd-party X-Frame. A common middle ground is to send SAMEORIGIN, meaning only websites of the same origin may frame it. |
| **V11.12** | Verify that the HTTP headers do not expose detailed version information of system components. |

# SUMMARY

| HTTP Basics | |
|---|---|
| | HTTP Request Methods |
| | HTTP Security Response Headers |
| | Sensitive Data In Transit |
| | Intercepting Proxy |
| | Don’t Trust The HTTP Request! |