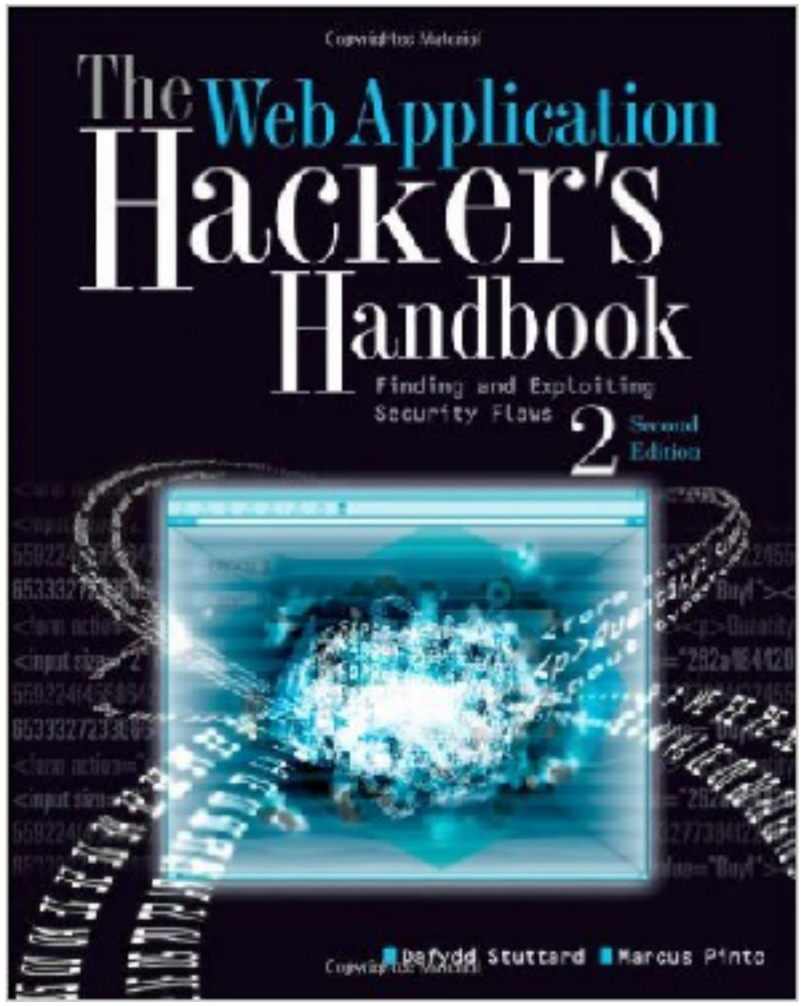
CNIT 129S: Securing Web Applications

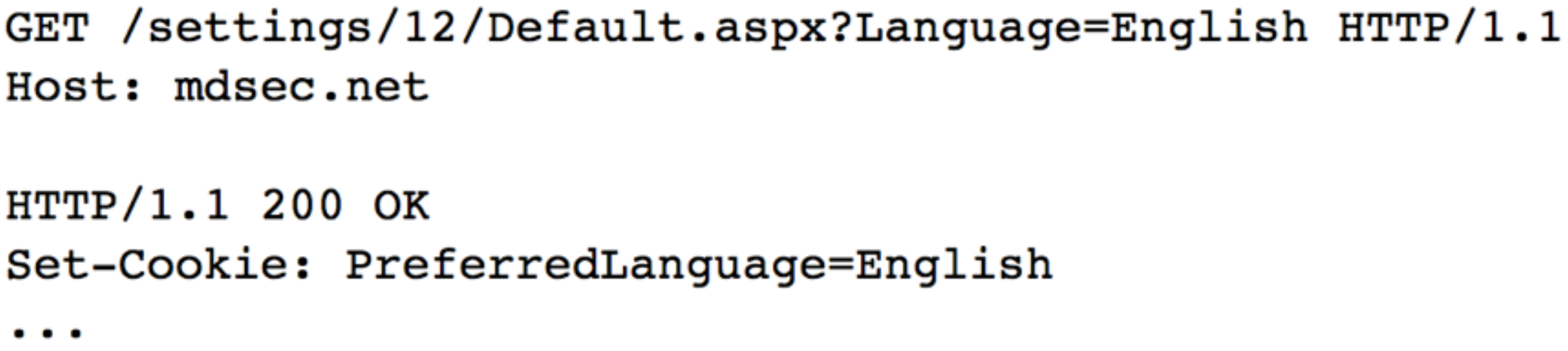
Ch 13: Attacking Users: Other Techniques (Part 2)


Copyrighted Material
The Web Application Hacker's Handbook
Finding and Exploiting Security Flaws
2 Second Edition
Dafydd Stuttard
Marcus Pinto

Other Client-Side Injection Attacks

HTTP Header Injection

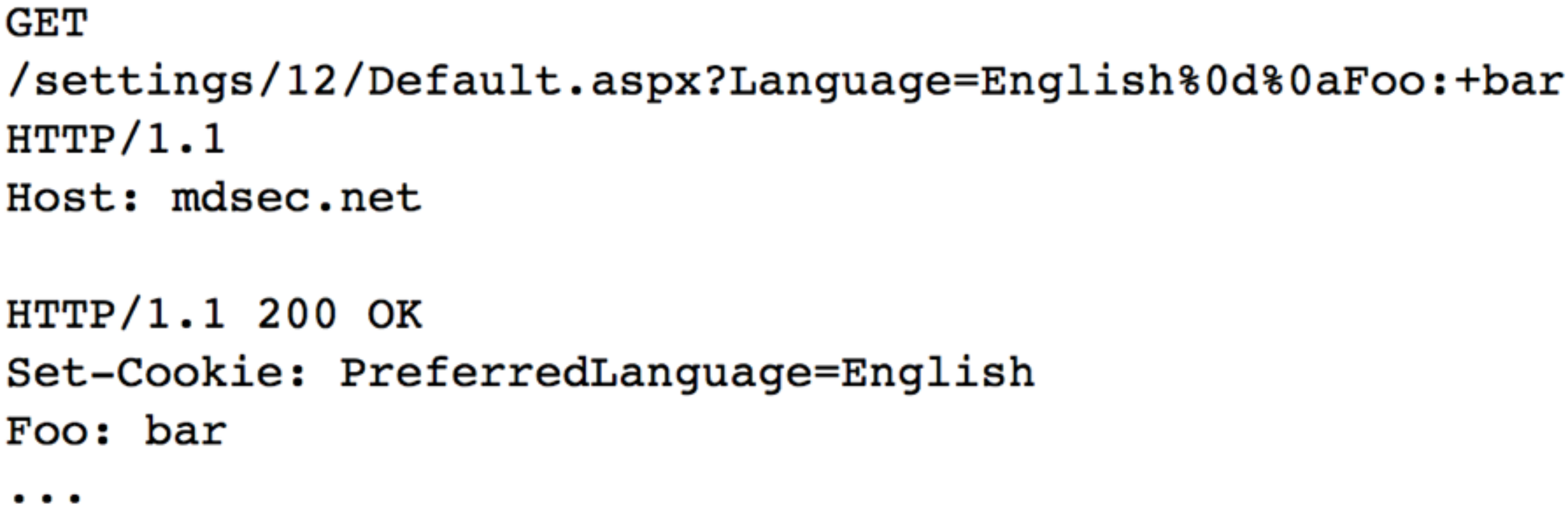
- **User-controlled data in an HTTP header**
- **Most commonly the Location and Set-Cookie headers**



```
GET /settings/12/Default.aspx?Language=English HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
...
```

Injecting Another Header

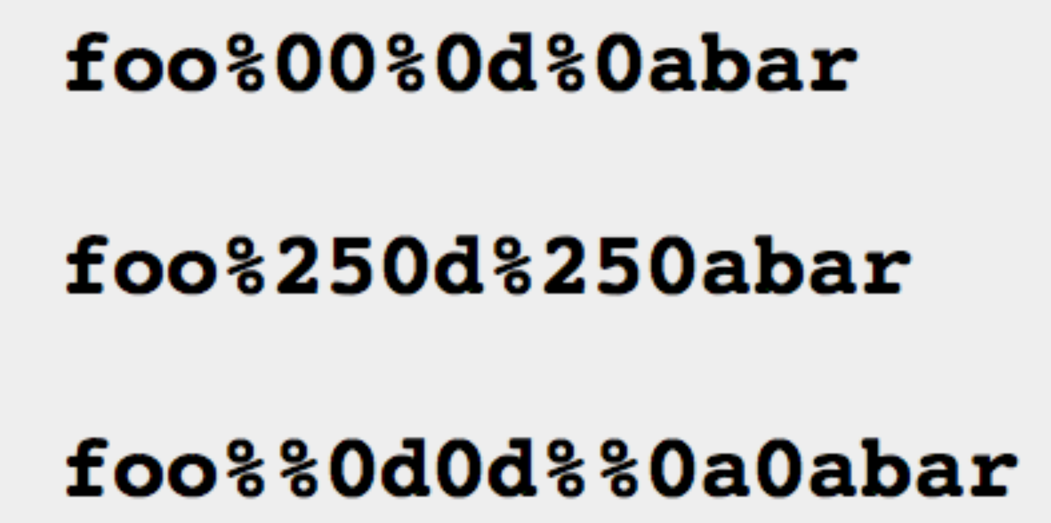


```
GET
/settings/12/Default.aspx?Language=English%0d%0aFoo:+bar
HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
Foo: bar
...
```

Exploiting Header Injection

- **See if %0d and %0a return decoded as carriage-return and line-feed**
  - **If only one works, you may still be able to exploit it**
- **If they are blocked or sanitized, try these bypasses**

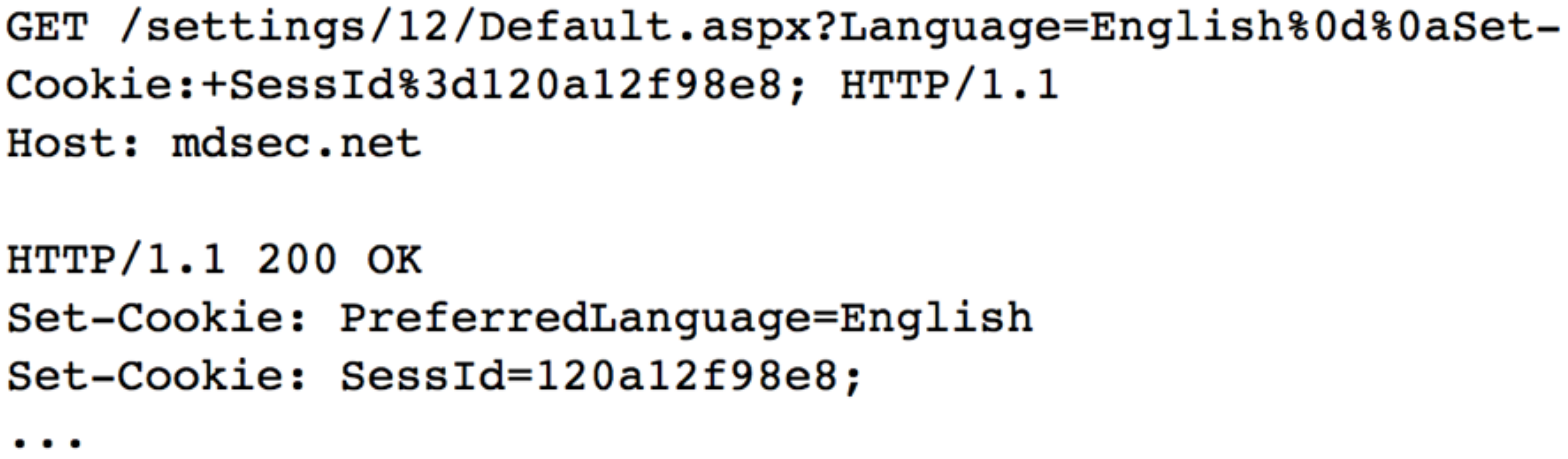


```
foo%00%0d%0abar

foo%250d%250abar

foo%%0d0d%%0a0abar
```

Injecting Cookies



```
GET /settings/12/Default.aspx?Language=English%0d%0aSet-
Cookie:+SessId%3d120a12f98e8; HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
Set-Cookie: SessId=120a12f98e8;
...
```

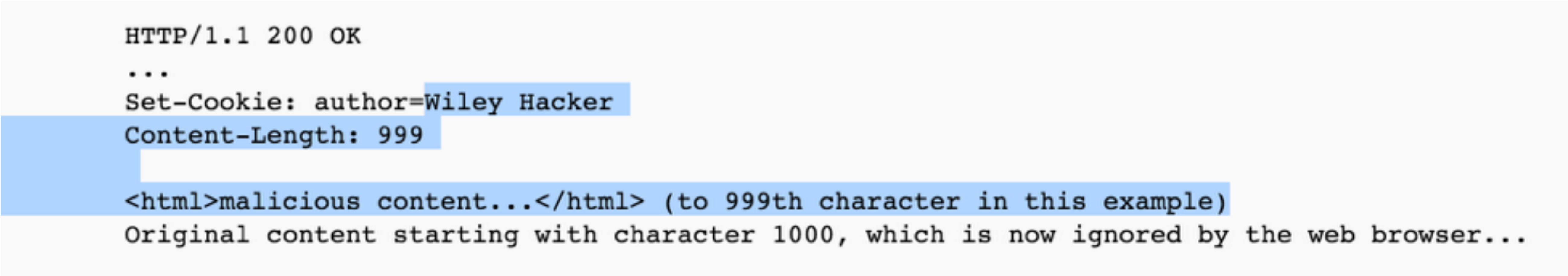
- **Cookies may persist across browser sessions**

Delivering Other Attacks

- **HTTP header injection allows an attacker to control the entire body of a response**
- **Can deliver almost any attack**
  - **Virtual website defacement**
  - **Script injection**
  - **Redirection**

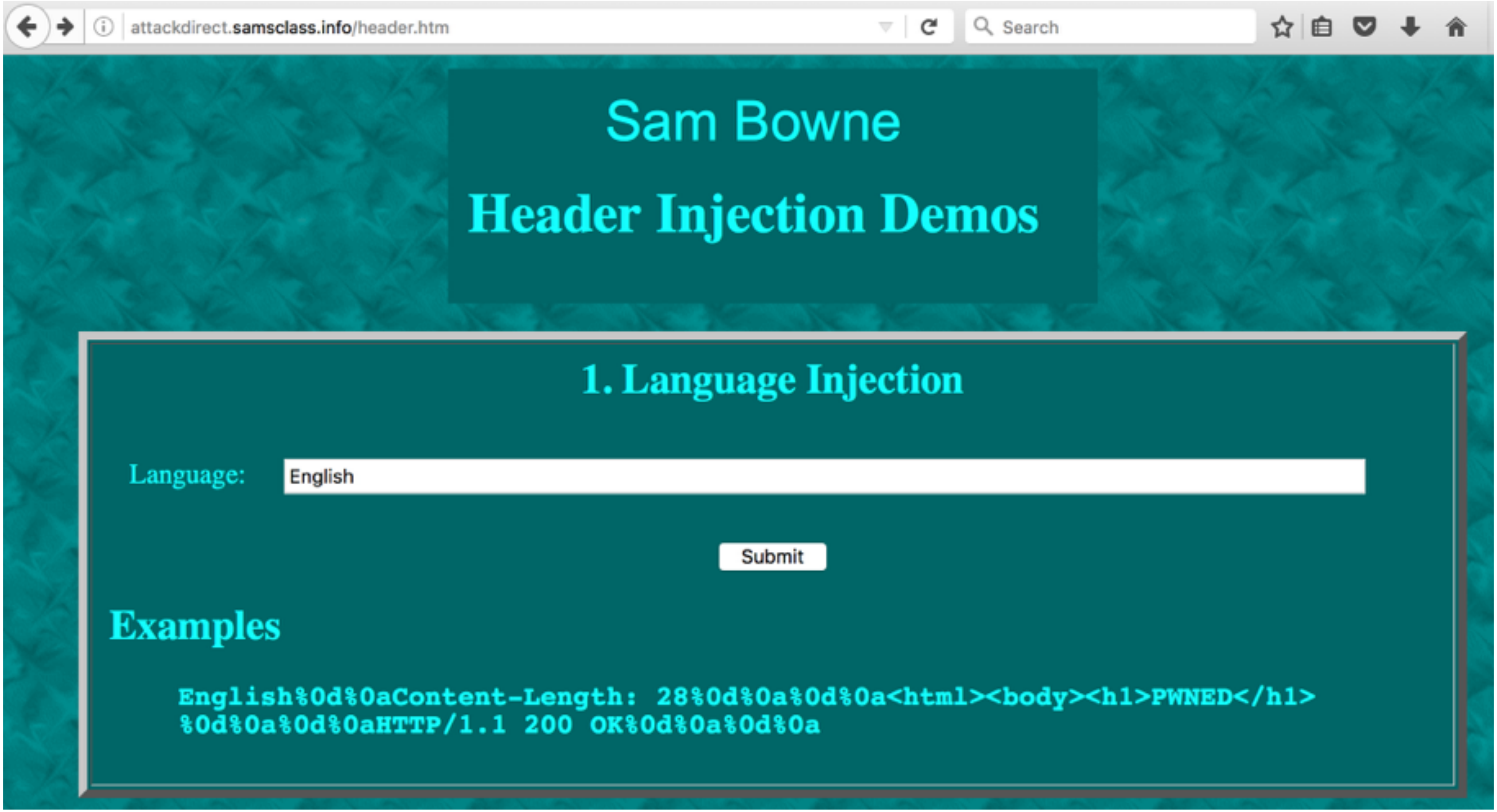
HTTP Response Splitting

- **Inject a second complete page into the headers**
- **Must inject carriage returns and line feeds**
- **Fixed in modern servers (link Ch 13d)**



```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker
Content-Length: 999

<html>malicious content...</html> (to 999th character in this example)
Original content starting with character 1000, which is now ignored by the web browser...
```



Sam Bowne

Header Injection Demos

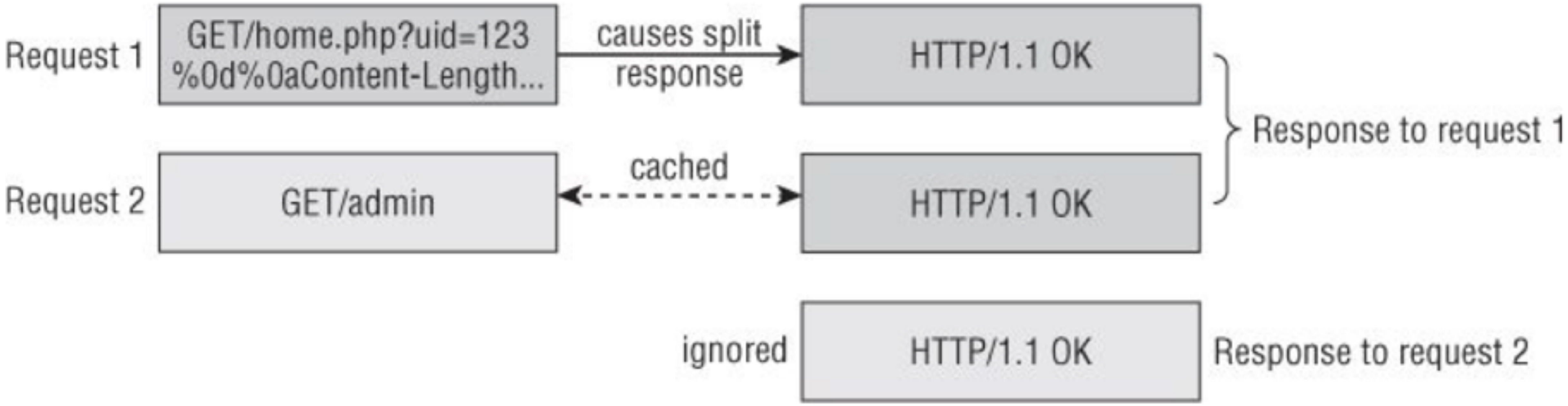
1. Language Injection

Language: English

Examples

```
English%0d%0aContent-Length: 28%0d%0a%0d%0a<html><body><h1>PWNED</h1>
%0d%0a%0d%0aHTTP/1.1 200 OK%0d%0a%0d%0a
```

Poisoning the Cache on a Proxy Server



Preventing Header Injection

- **Don't insert user-controllable input into headers**
- **If you must, use**
  - **Input validation (context-dependent)**
  - **Output validation: block all ASCII characters below 0x20**

Cookie Injection

- **Attacker sets or modifies a cookie in the victim user's browser**
- **This may be possible if:**
  - **App has functionality that takes a name and value from parameters and sets those within a cookie, such as "Save user preferences"**
  - **HTTP header injection vulnerability**

Cookie Injection

- **Setting a malicious cookie via XSS**
  - **XSS in related domains can be leveraged to set a cookie on the targeted domain, from any of these:**
    - **Any subdomain of the target domain, any of its parents and their subdomains**

Cookie Injection

- **Setting a malicious cookie via a Man-in-the-middle (MITM) attack**
  - **MITM attacker can set cookies for arbitrary domains**
  - **Even if the targeted app uses only HTTP and all its cookies are flagged as "secure"**

Consequences of Setting a Cookie

- **Some apps may change their logic in response to cookie values, such as `UseHttps=false`**
- **Client-side code may trust cookie values and use them in dangerous ways, leading to DOM-based XSS or JaScript injection**
- **Some apps implement anti-CSRF tokens by placing the token into both a cookie and a request parameter and comparing them**
  - **If an attacker controls both, this defense can be bypassed**

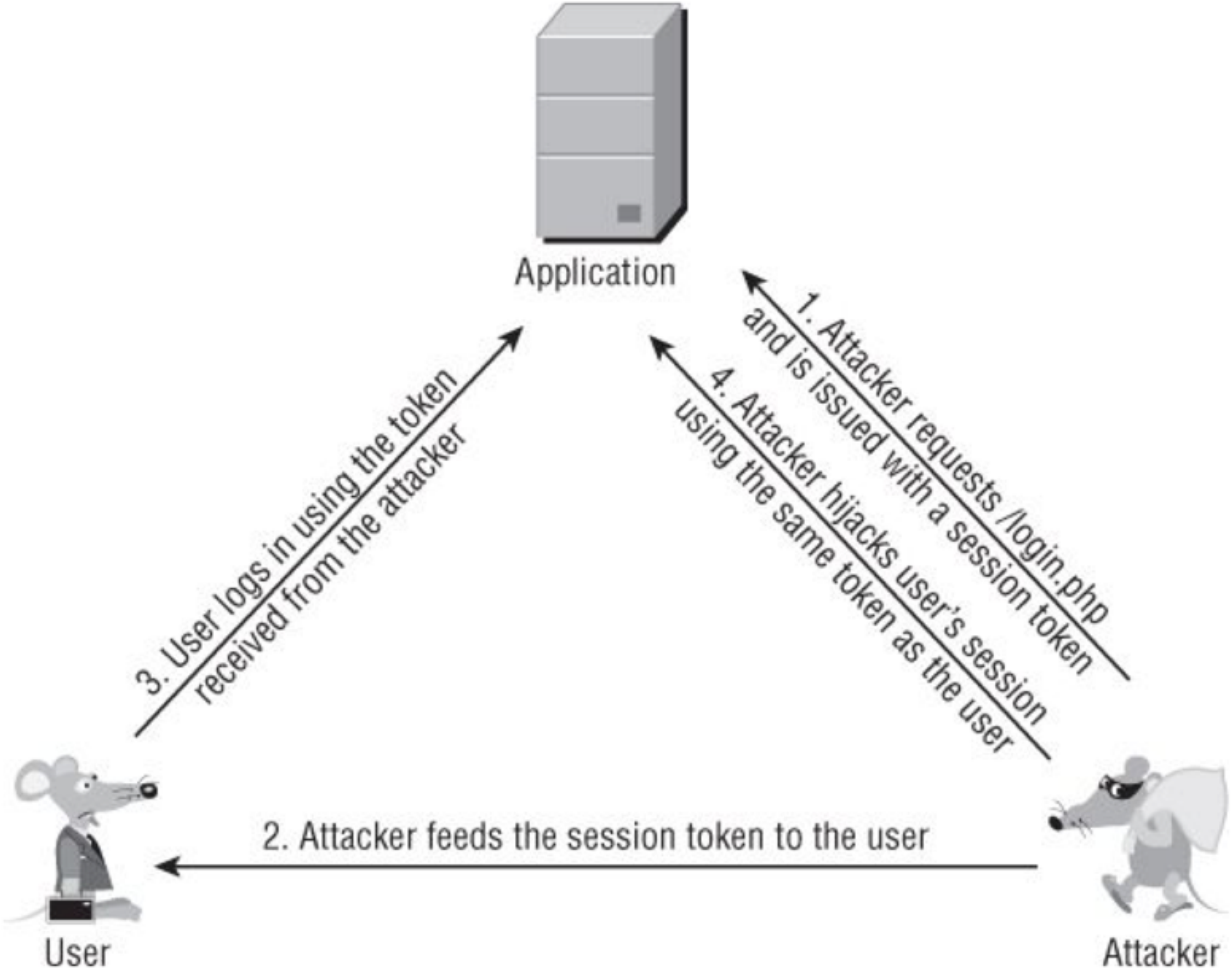
Consequences of Setting a Cookie

- **If an app has a same-user persistent XSS vuln**
  - **You can use CSRF to trick the user into loading the script, but you can perform the same attack by putting your own session token into the user's cookie**
- **Exploit session fixation vulnerabilities**

Session Fixation

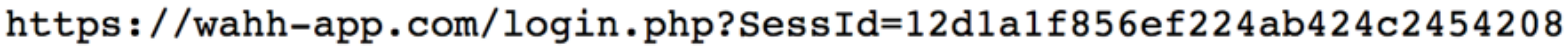
- **Suppose an app creates an anonymous session for each user before login**
  - **When the user logs in, the session is upgraded to an authenticated session**
  - **Using the same token**
- **In *session fixation*, attacker gets an anonymous token and fixes it within the victim's browser**
  - **When victim logs in, the token gains privileges**

Figure 13.4 The steps involved in a session fixation attack


Application
1. Attacker requests /login.php and is issued with a session token
2. Attacker feeds the session token to the user
3. User logs in using the token received from the attacker
4. Attacker hijacks user's session using the same token as the user
User
Attacker

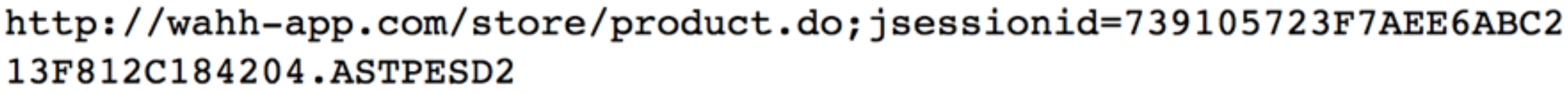
How to Inject the Token

- **Cookie injection (if token is in a cookie)**
- **If session token is in the URL, feed victim a URL like this**



```
https://wahh-app.com/login.php?SessId=12d1a1f856ef224ab424c2454208
```

- **Some apps let you add a token in the URL after a semicolon, even if this isn't the default**



```
http://wahh-app.com/store/product.do;jsessionid=739105723F7AEE6ABC2
13F812C184204.ASTPESD2
```

- **If session token is in a hidden HTML field, use CSRF**

Session Fixation Without Login

- **Anonymous user browses products**
  - **Places items into a shopping cart**
  - **Checks out by submitting personal data and payment details**
  - **Reviews data on a Confirm Order page**
- **Attacker fixes an anonymous toke in target's browser and views the Confirm Order page to steal data**

Arbitrary Tokens

- **Some apps accept arbitrary tokens submitted by users**
  - **Even if they were not issued by the server itself**
  - **App creates a new session using the token**
  - **Microsoft IIS and Allaire ColdFusion did this in the past**
- **So attacker can just send target a link with an arbitrary token**

Finding and Exploiting Session Fixation Vulnerabilities

- **Review handling of session tokens in relation to login**
- **Two vulnerabilities**
    - **App assigns token to anonymous user and upgrades its privileges upon login**
    - **User who logs in, then logs in again to a different account, retains the same token**

Finding and Exploiting Session Fixation Vulnerabilities

- **In either case, an attacker can obtain a valid session token and feed it to the target user**
- **When that user logs in, the attacker can hijack the session**
- **Even without a login, the app may reveal sensitive information to an attacker with the target's session token**

Preventing Session Fixation

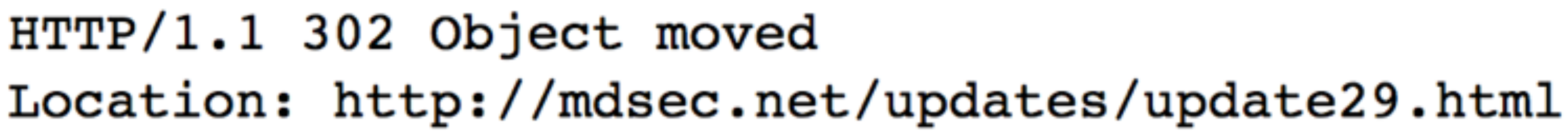
- **Whenever a user transitions from being anonymous to being identified, issue a fresh session token**
  - **This applies to both login and when a user first submits personal or other sensitive information**
- **For defense-in-depth, employ per-page tokens to supplement the main session token**
- **App should not accept arbitrary session tokens that it does not recognize as being issued itself**

Open Redirection

- App takes user-controllable input and uses it to redirect to a different URL
  - Commonly used for Rickrolling
- Useful in phishing attacks, to make a fake page appear to be in the target domain
- Most real-world phishing attacks use other techniques
  - Registering similar domain names, using official-looking subdomains, or using anchor text that doesn't match the URL

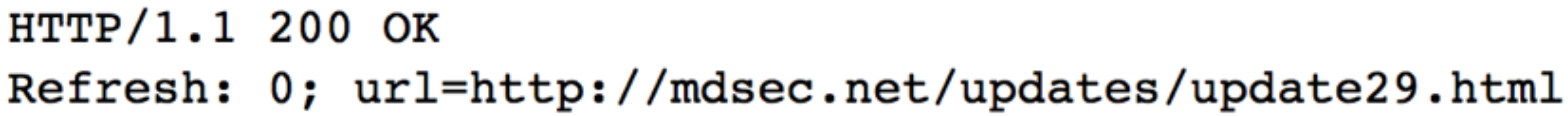
Finding Open Redirects

- **First identify redirects within the app (3xx status code)**



```
HTTP/1.1 302 Object moved
Location: http://mdsec.net/updates/update29.html
```

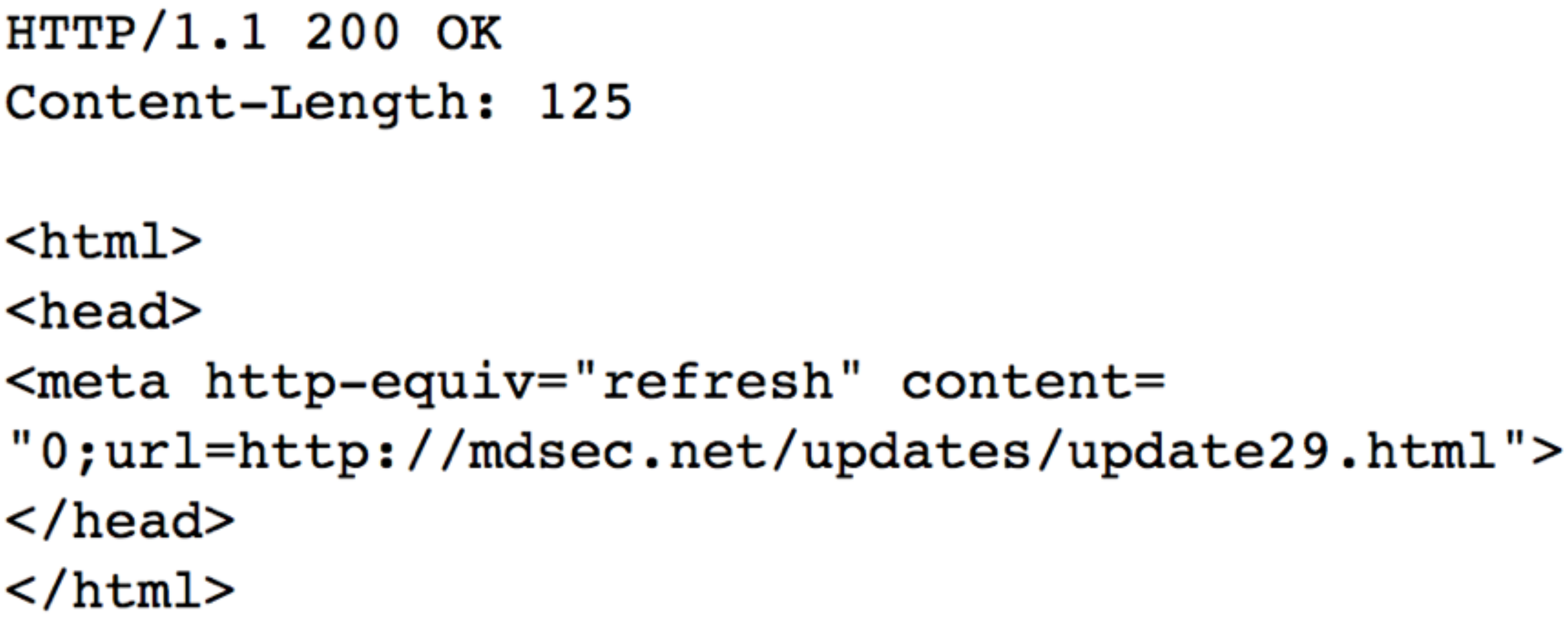
- **HTTP Refresh header can trigger a redirect (number is delay in seconds)**



```
HTTP/1.1 200 OK
Refresh: 0; url=http://mdsec.net/updates/update29.html
```

Finding Open Redirects

- **HTML <meta> tag**

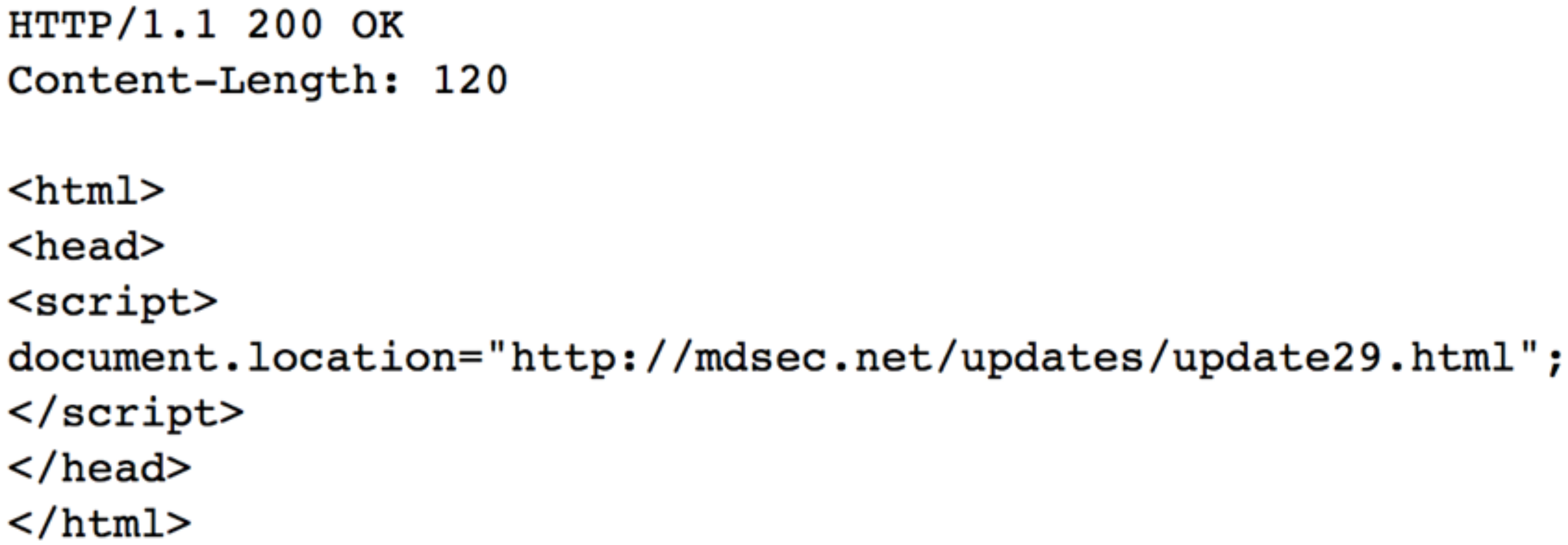


```
HTTP/1.1 200 OK
Content-Length: 125

<html>
<head>
<meta http-equiv="refresh" content=
"0;url=http://mdsec.net/updates/update29.html">
</head>
</html>
```

Finding Open Redirects

- **JavaScript API**

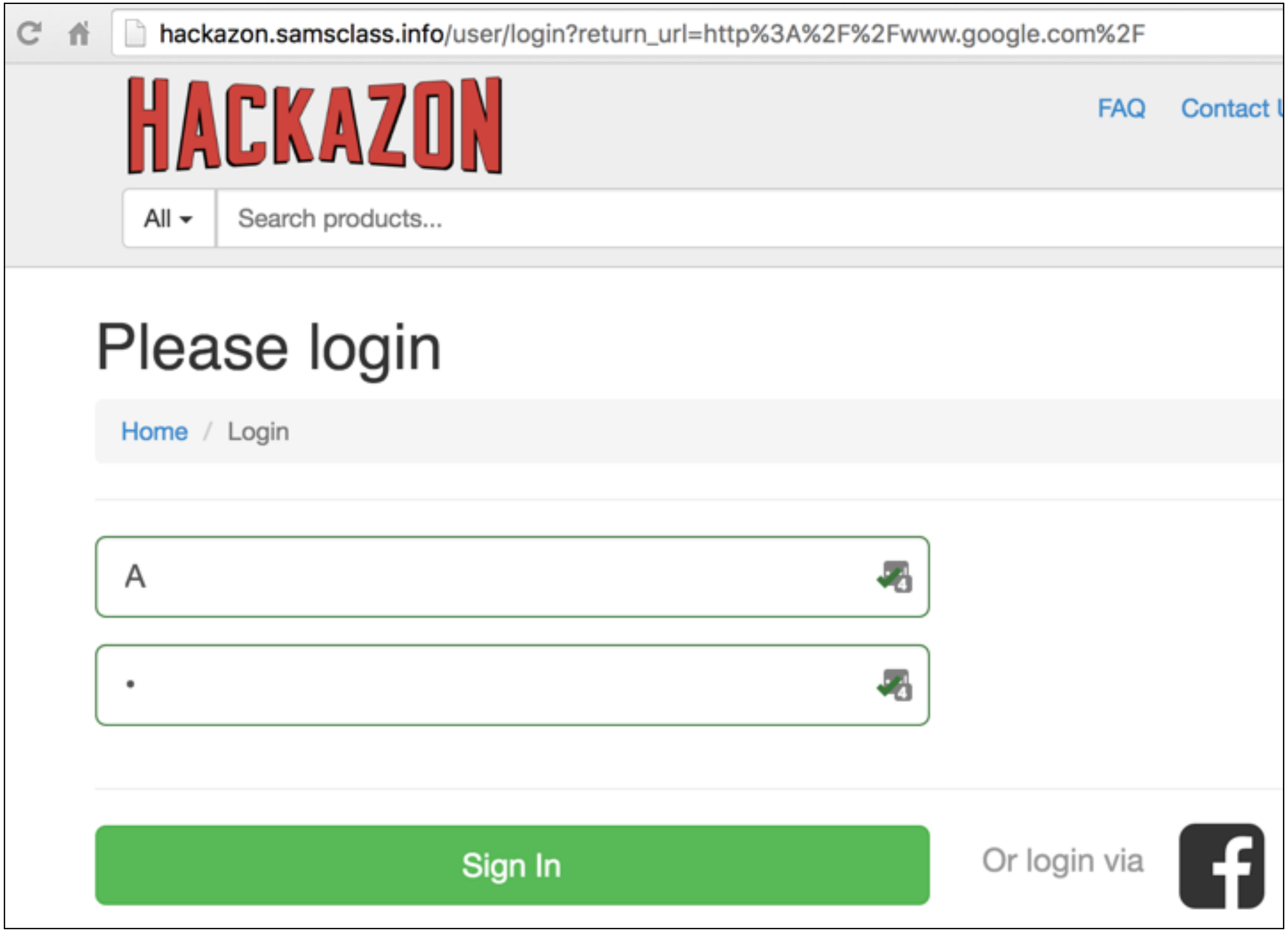


```
HTTP/1.1 200 OK
Content-Length: 120

<html>
<head>
<script>
document.location="http://mdsec.net/updates/update29.html";
</script>
</head>
</html>
```

Finding Open Redirects

- **Most redirects are not user-controllable**
- **One common place they are is when app has "return to original page" functionality**
  - **For example, after a timeout and re-login**
- **Look for URLs that contain a domain name and try changing it**



hackazon.samsclass.info/user/login?return_url=http%3A%2F%2Fwww.google.com%2F

Please login

- **Open redirect misidentified by ZAP as RFI**

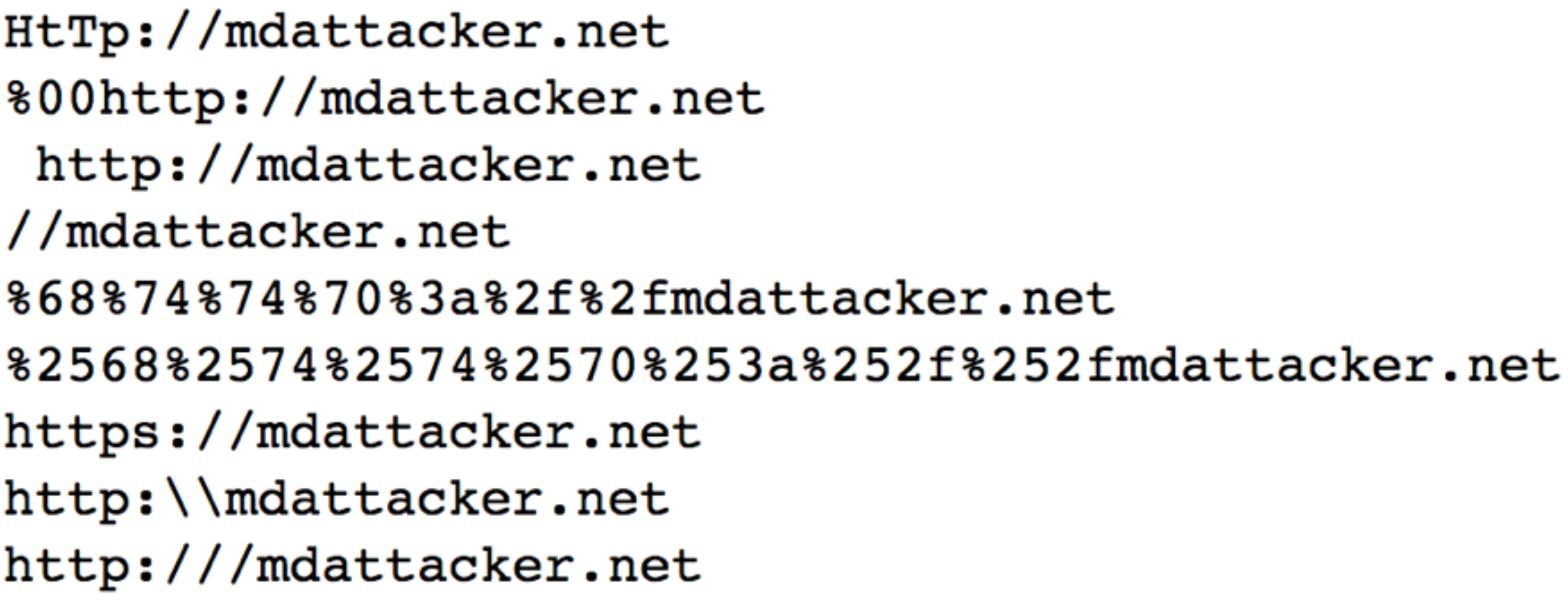
Filtering or Sanitizing URLs

- **Some apps try to prevent redirection attacks by**
  - **Blocking absolute URLs**
  - **Adding a specific absolute URL prefix**

Blocking Absolute URLs

- **Block user-supplied strings that starts with "http://"**

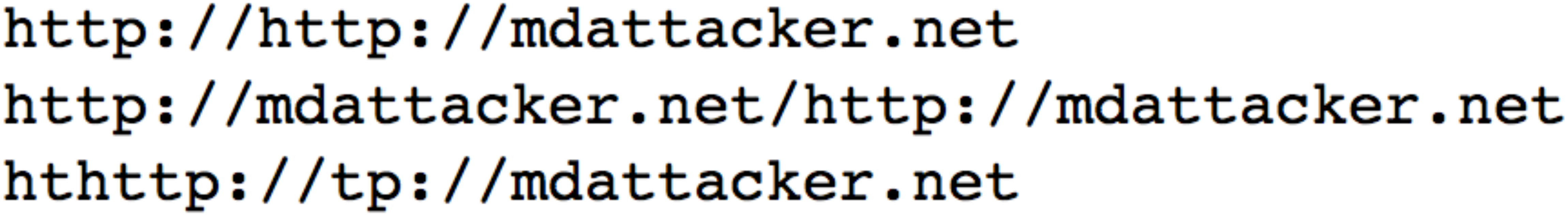
- **These tricks might work**



```
HtTp://mdattacker.net
%00http://mdattacker.net
 http://mdattacker.net
//mdattacker.net
%68%74%74%70%3a%2f%2fmdattacker.net
%2568%2574%2574%2570%253a%252f%252fmdattacker.net
https://mdattacker.net
http:\\mdattacker.net
http:///mdattacker.net
```

Sanitizing Absolute URLs

- **Remove "http://" and any external domain**
- **Previous tricks might work, and these:**

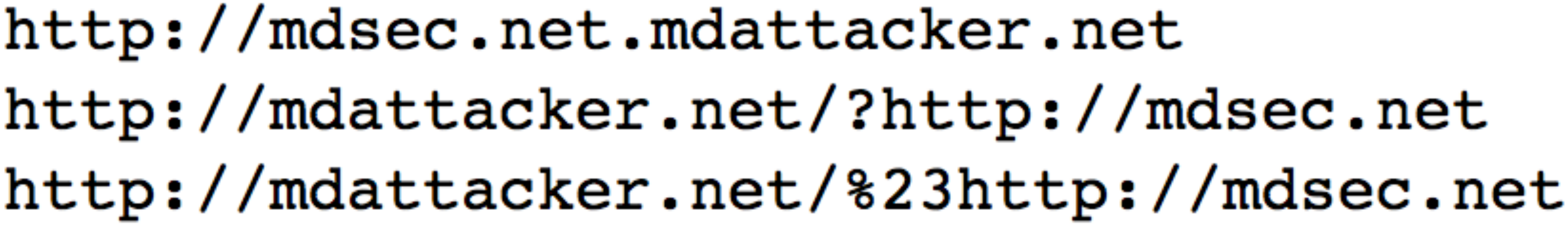


```
http://http://mdattacker.net
http://mdattacker.net/http://mdattacker.net
hthttp://tp://mdattacker.net
```

Sanitizing Absolute URLs

- **App may verify that the user-supplied string starts with, or contains, an absolute URL to its own domain name**

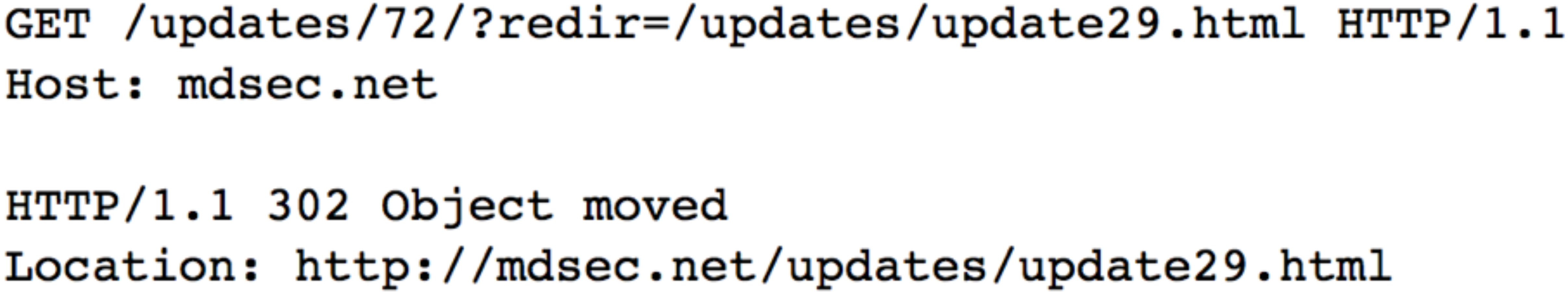
- **Try these:**



```
http://mdsec.net.mdattacker.net
http://mdattacker.net/?http://mdsec.net
http://mdattacker.net/%23http://mdsec.net
```

Adding an Absolute Prefix

- **App forms target of redirect by appending the user-controlled string to an absolute URL prefix**

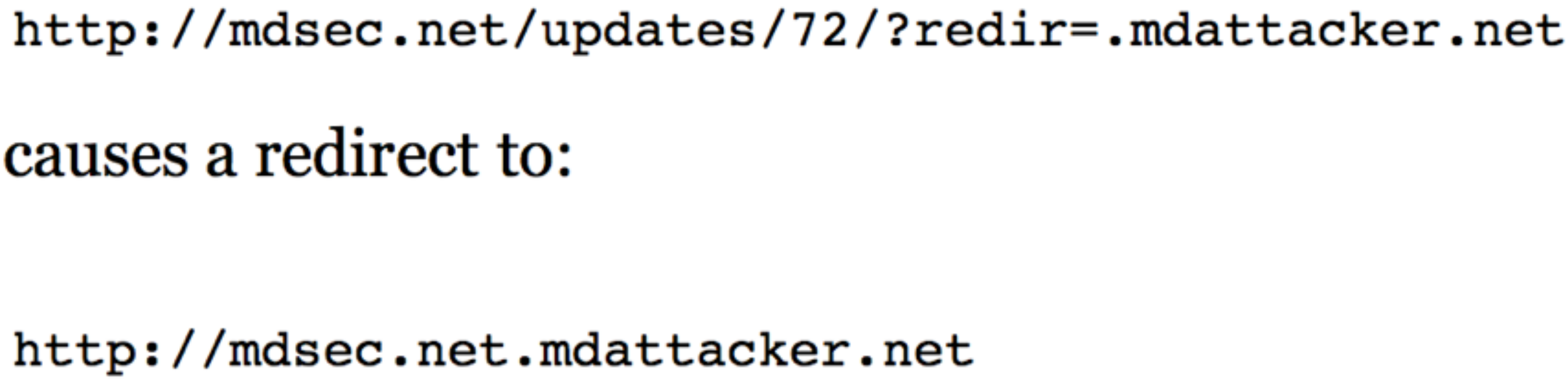


```
GET /updates/72/?redir=/updates/update29.html HTTP/1.1
Host: mdsec.net

HTTP/1.1 302 Object moved
Location: http://mdsec.net/updates/update29.html
```

Adding an Absolute Prefix

- **If the added prefix is "http://mdsec.net" instead of "http://mdsec.net/", it's vulnerable**



```
http://mdsec.net/updates/72/?redir=.mdattacker.net
```

causes a redirect to:

```
http://mdsec.net.mdattacker.net
```

Preventing Open Redirection Vulnerabilities

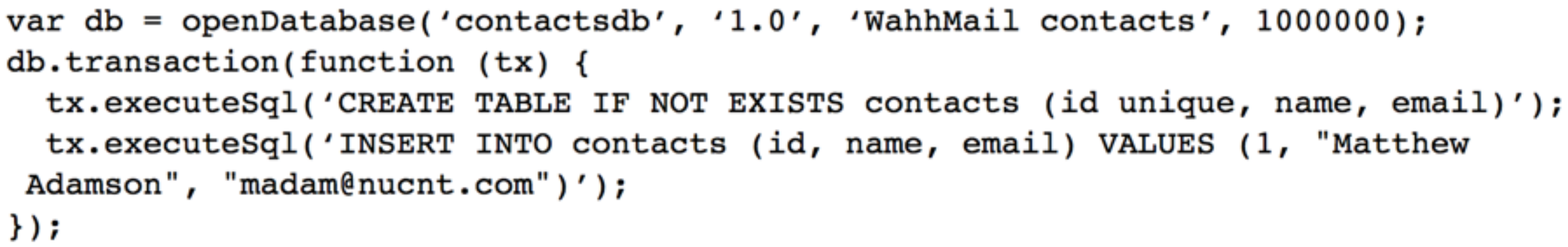
- **Don't incorporate user-supplied data onto the target of a redirect**
- **It's better to have a list of allowed redirection targets, and only allow known good choices**

Preventing Open Redirection Vulnerabilities

- **If you must use user-controlled data:**
  - **Use relative URLS in all redirects, and the redirect page should verify that the user-supplied URL begins with a single slash followed by a letter, or begins with a letter and does not have a colon before the first slash**
  - **Prepend every URL with http://domain.com/**
  - **Verify that every URL starts with http://domain.com/**

Client-Side SQL Injection

- **HTML5 supports client-side SQL databases**
- **Accessed through JavaScript, like this**



```
var db = openDatabase(‘contactsdb’, ‘1.0’, ‘WahhMail contacts’, 1000000);
db.transaction(function (tx) {
  tx.executeSql(‘CREATE TABLE IF NOT EXISTS contacts (id unique, name, email)’);
  tx.executeSql(‘INSERT INTO contacts (id, name, email) VALUES (1, "Matthew
 Adamson", "madam@nucnt.com")’);
});
```

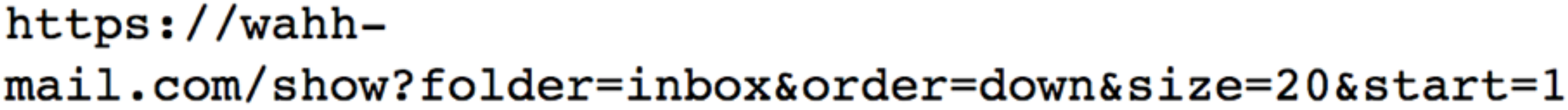
- **Allows apps to store data on the client side**
- **Allows apps to run in "offline mode"**

Client-Side SQL Injection

- **Attacker may be able to steal data such as**
  - **User's contact information from social networking apps**
  - **Comments from news apps**
  - **Email from web mail apps**
- **Attacks such as sending SQLi in the subject of an email**

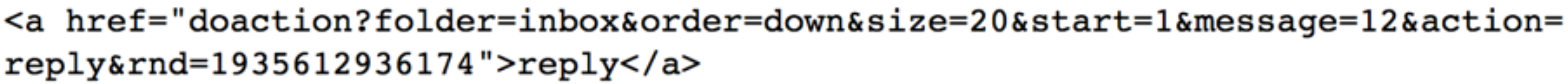
Client-Side HTTP Parameter Pollution

- **A web mail app loads the inbox with this URL:**



```
https://wahh-
mail.com/show?folder=inbox&order=down&size=20&start=1
```

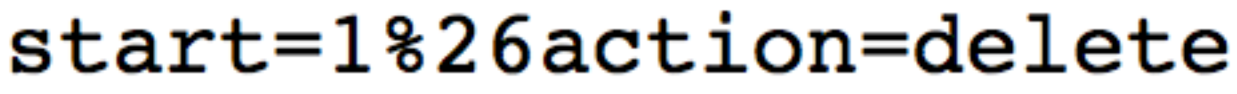
- **This link allows the user to reply to a message, and it uses several parameters from the inbox URL:**



```
<a href="doaction?folder=inbox&order=down&size=20&start=1&message=12&action=
reply&rnd=1935612936174">reply</a>
```

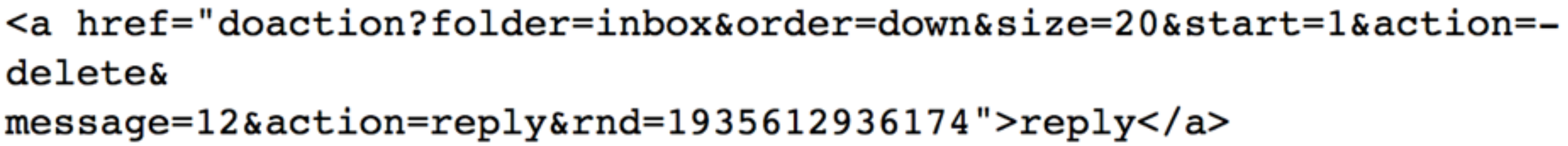
Client-Side HTTP Parameter Pollution

- **Attacker tricks target into opening an inbox with this parameter:**



```
start=1%26action=delete
```

- **This makes the "Reply" link look like this, so it deletes messages instead:**



```
<a href="doaction?folder=inbox&order=down&size=20&start=1&action=-
delete&
message=12&action=reply&rnd=1935612936174">reply</a>
```

Local Privacy Attacks

Shared Machines

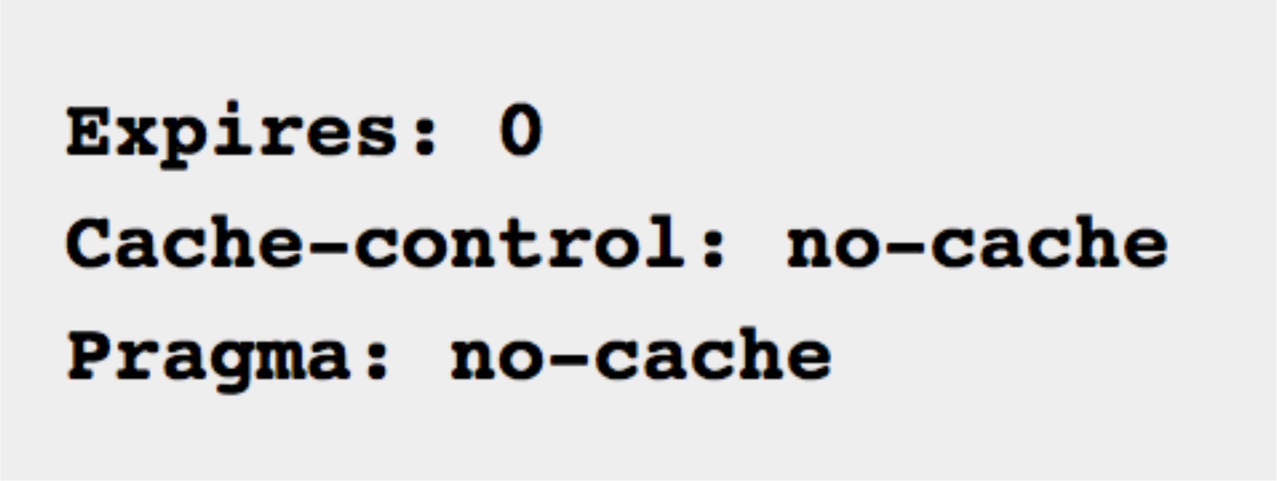
- **Attacker has access to the same computer as the target user**
- **Similar situation: a stolen cell phone or laptop**

Persistent Cookies

- **Cookies often have expiration dates far in the future**
- **Especially on mobile devices**

Cached Web Content

- **Browsers typically cache non-SSL content unless told not to, by HTTP response headers or HTML metatags**



```
Expires: 0
Cache-control: no-cache
Pragma: no-cache
```

Browsing History & AutoComplete

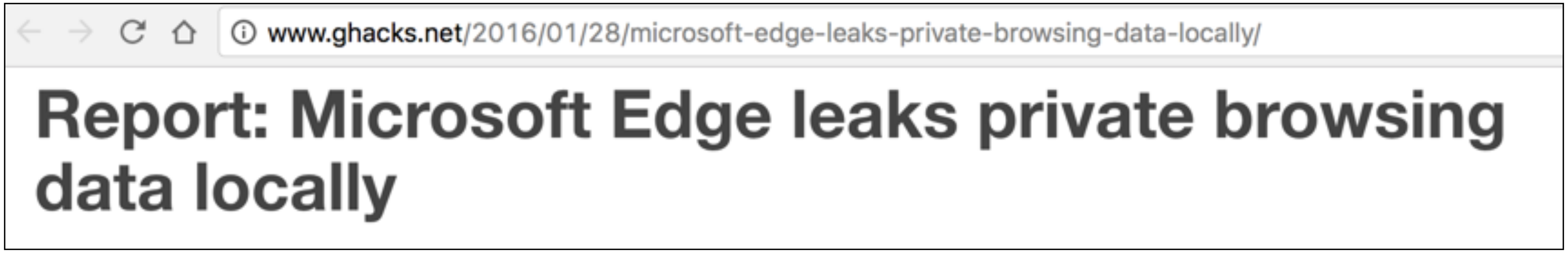
- **Browsing history may include sensitive data in URL parameters**
- **Autocomplete often stores passwords, credit card numbers, etc.**
    - **IE stores autocomplete data in the registry, Firefox stores it in the file system**
- **Autocomplete data can be stolen by XSS under some circumstances**

Flash Local Stored Objects

- **Also called "flash cookies"**
- **Shared between different browsers, if they have the Flash extension installed**
- **Used by Google and other companies to mark your computer in a way that's difficult to erase**

Internet Explorer userData

- **IE's custom user data storage system**
- **Edge stores local data even in Private Browsing mode**
  - **Link Ch 13e**



www.ghacks.net/2016/01/28/microsoft-edge-leaks-private-browsing-data-locally/

Report: Microsoft Edge leaks private browsing data locally

HTML5 Local Storage Mechanisms

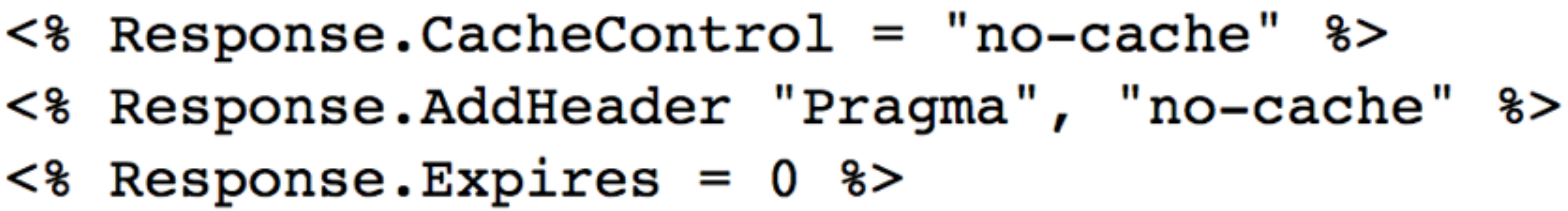
- **HTML5 introduced a range of new local storage mechanisms, including:**
  - **Session storage**
  - **Local storage**
  - **Database storage**
- **The specifications are still evolving; privacy implications are not clear**

Preventing Local Privacy Attacks

- **Apps shouldn't store anything sensitive in a persistent cookie**
  - **Even if it's encrypted, because the attacker could replay it**
- **Apps should use cache directives to prevent sensitive data being stored by browsers**

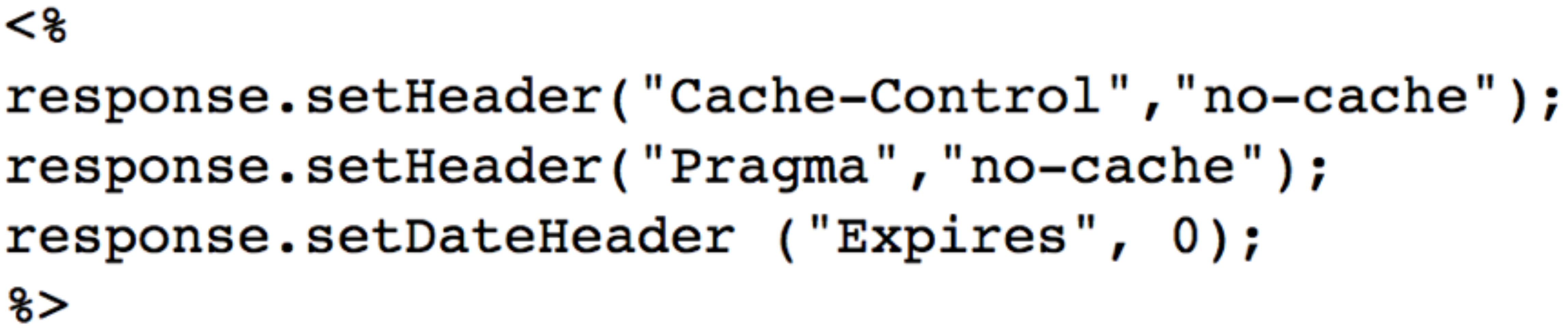
Preventing Local Privacy Attacks

- **ASP instructions to prevent caching**



```
<% Response.CacheControl = "no-cache" %>
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = 0 %>
```

- **Java commands:**



```
<%
response.setHeader("Cache-Control","no-cache");
response.setHeader("Pragma","no-cache");
response.setDateHeader ("Expires", 0);
%>
```

Preventing Local Privacy Attacks

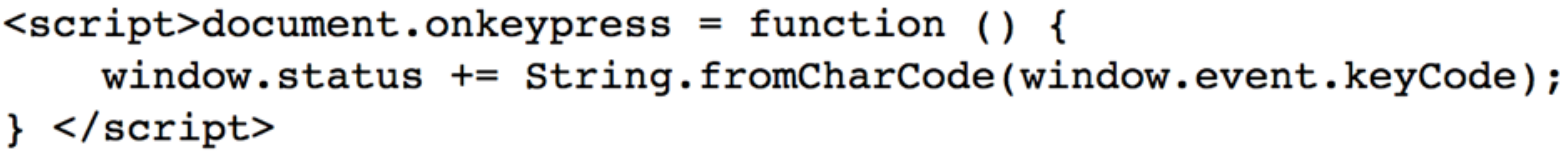
- **Apps shouldn't use URLs to transmit sensitive data**
    - **Because URLs are logged in numerous locations**
- **All sensitive data should be transmitted with POST**
- **Sensitive fields should use the "autocomplete=off" attribute**

Attacking the Browser

Logging Keystrokes

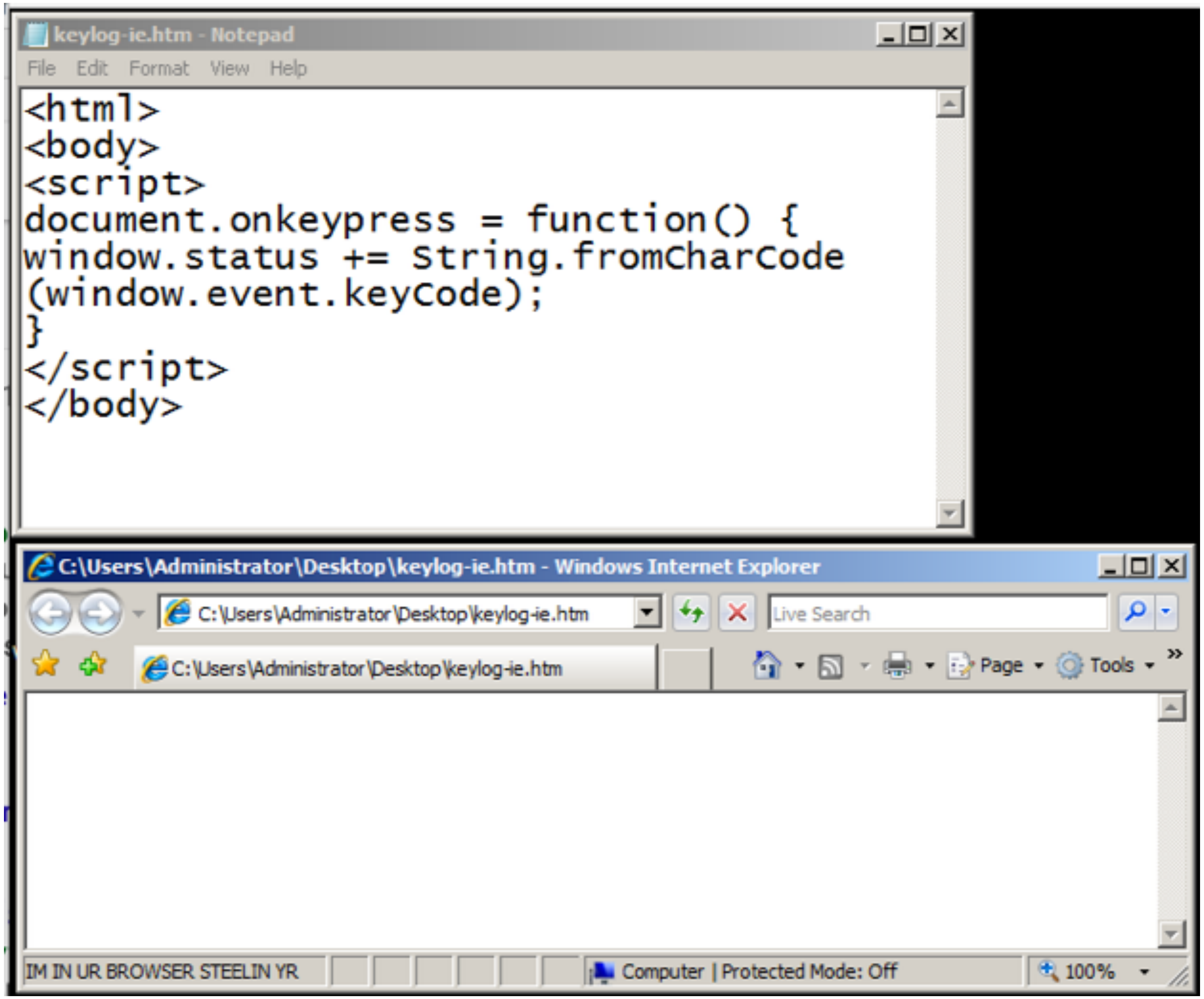
- **JavaScript can monitor all keys pressed while the browser window has the focus**

- **This script capture all keystrokes in Internet Explorer and displays them in the status bar**



```
<script>document.onkeypress = function () {
    window.status += String.fromCharCode(window.event.keyCode);
} </script>
```

Demo in Win 2008



keylog-ie.htm - Notepad
File Edit Format View Help

```
<html>
<body>
<script>
document.onkeypress = function() {
window.status += String.fromCharCode
(window.event.keyCode);
}
</script>
</body>
```

C:\Users\Administrator\Desktop\keylog-ie.htm - Windows Internet Explorer
C:\Users\Administrator\Desktop\keylog-ie.htm
Live Search
C:\Users\Administrator\Desktop\keylog-ie.htm
Page
Tools

IM IN UR BROWSER STEELIN YR

Computer | Protected Mode: Off
100%

Logging Keystrokes

- **Can only capture keystrokes while the frame running the code is in focus**
- **Apps are vulnerable when they embed a third-party widget or advertising applet in a frame within the app's own pages**
- **In "reverse strokejacking", malicious code in a child frame can grab the focus from the top-level window**
    - **It can echo the keypresses to the top-level window so the app appears to be working correctly**

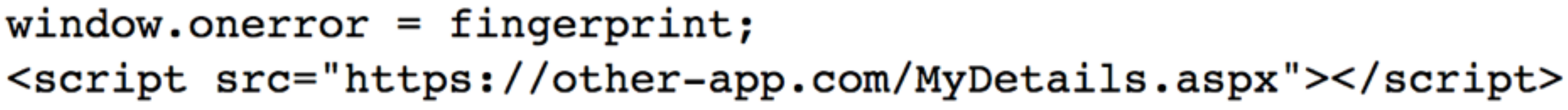
Stealing Browser History and Search Queries

- **JavaScript can brute-force common websites to see which ones have been visited via the "getComputerStyle" API**

- **This also works for query strings because they're in the URL**

Enumerating Currently Used Applications

- **JavaScript can determine whether the user is currently logged in to third-party applications**
- **By requesting a page that can only be viewed by logged-in users, such as "My Details"**
- **This script uses a custom error handler to process scripting errors**
- **And then makes a cross-domain request**



```
window.onerror = fingerprint;
<script src="https://other-app.com/MyDetails.aspx"></script>
```

Enumerating Currently Used Applications

- **Since the page contains HTML, not script, the request always causes a JavaScript error**
- **But the error will have a different line number and error type**
- **So the attacker can deduce whether the user is logged in or not**

Port Scanning

- **Browser-based port scanning from a Java applet**
- **BUT same-origin policy means browser can't see the response**
  - **Attempt to dynamically load and execute script from each targeted host and port**
  - **If a Web server is running on that port, it returns HTML or some other content**
  - **Resulting in a JavaScript error the port-scanning script can detect**

Port Scanning

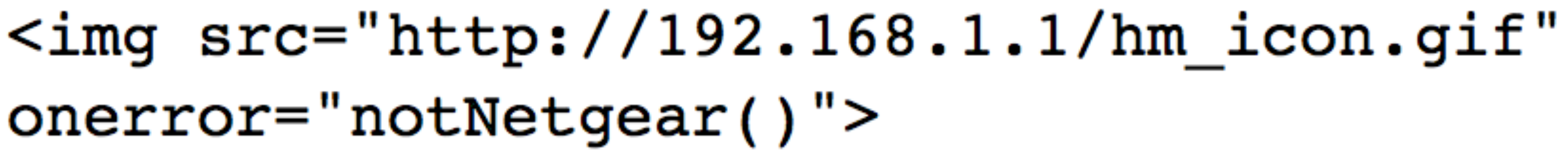
- **Most browsers implement restrictions on the ports that can be accessed using HTTP requests**
  - **Ports commonly used by other well-known services, such as port 25, are blocked**

Attacking Other Network Hosts

- **After a port scan identifies other hosts running HTTP servers**

- **A script can attempt to fingerprint them by looking for known files**

- **This image is present on a certain brand of DSL routers:**



```
<img src="http://192.168.1.1/hm_icon.gif"
onerror="notNetgear()">
```

Attacking Other Network Hosts

- **After identifying the device, attacker can try default username and password**
  - **Or exploit known vulnerabilities**
- **Even if attacker can only issue requests but not see responses, many attacks are possible**

Exploiting Non-HTTP Services

- **Attacker can send arbitrary binary content to a port**
    - **But it will always start with an HTTP header**
- **Many network services do tolerate unrecognized input and still process subsequent input that is correctly formed**

XSS Attacks from Non-HTTP Services

- **Non-HTTP service running on a port that is not blocked by browsers**

- **Non-HTTP service tolerates unexpected HTTP headers**

- **Non-HTTP service echoes part of the request content in its response, such as an error message**

- **Browser tolerates responses that don't have valid HTTP headers, and process part of the response as HTML (all browsers do this for backward compatibility)**

- **Browser must ignore port number when segregating cross-domain access to cookies (they do)**

XSS Attacks from Non-HTTP Services

- **Under those conditions, attacker can send script to the non-HTTP service, read cookies for the domain, and transmit those to the attacker**

Exploiting Browser Bugs

- **Bugs in browser or extensions may be exploitable with JavaScript or HTML**
- **Java bugs have enabled attackers to perform two-way binary communication with non-HTTP services on the local computer or elsewhere**

DNS Rebinding

- **A way to evade the same-origin policy**
- **Attacker has a malicious website and a malicious authoritative DNS server**
- **User visits a malicious page on the attacker's server**
- **That page makes Ajax requests to the attacker's domain, which resolves them to the target domain's IP address**

DNS Rebinding

- **Subsequent requests to the attacker's domain name are sent to the targeted application**
- **Browser thinks the target app is in the attacker's domain, so the same-origin policy doesn't block responses**

Limitations of DNS Rebinding

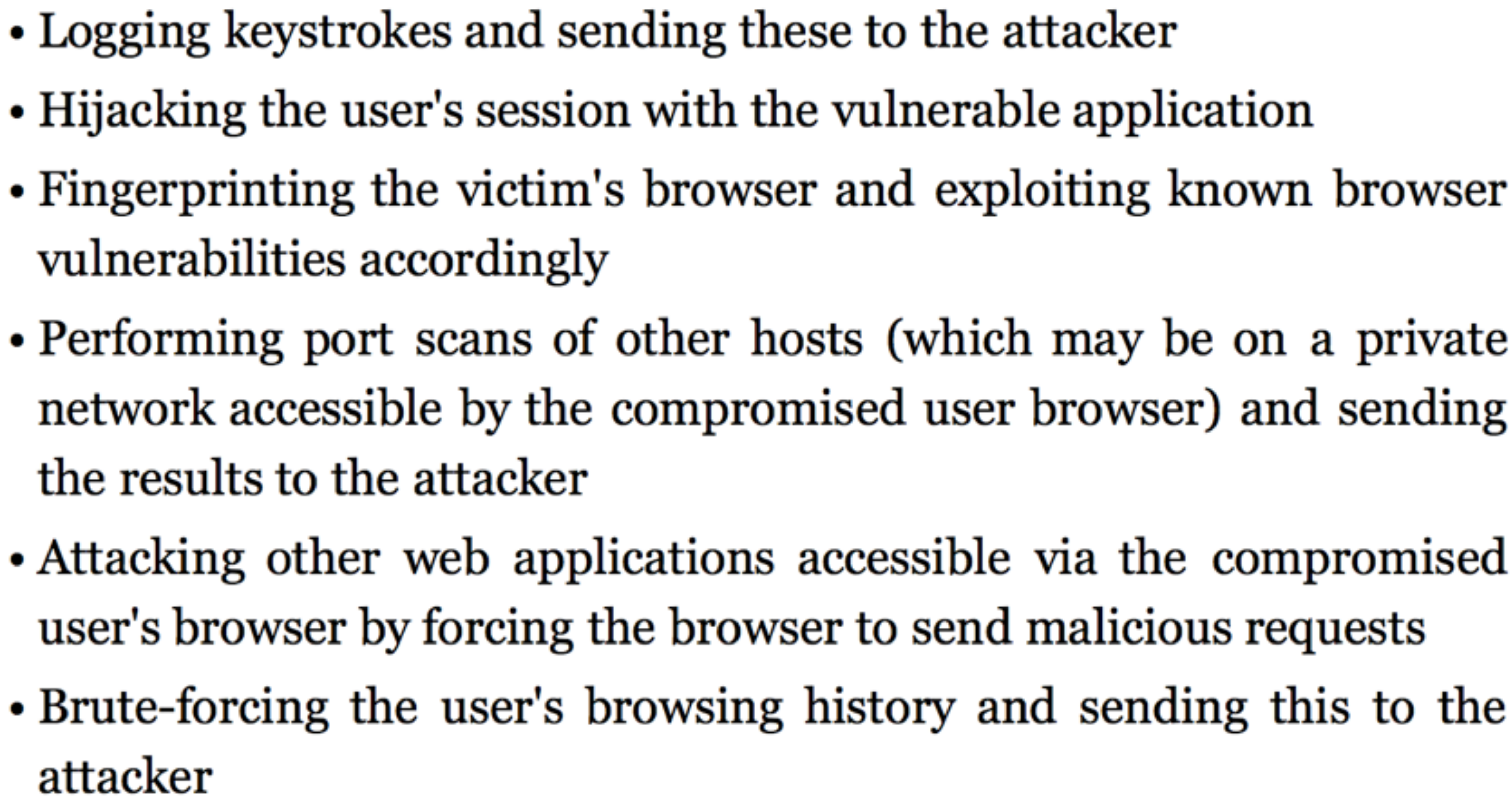
- **Host: parameter will point to the attacker's domain**
- **Requests won't contain the target domain's cookies**
- **This attack is only useful in special situations, when other controls prevent the attacker from directly accessing the target**

Browser Exploitation Frameworks

- **Such as BeEF or XSS Shell**
- **Use a Javascript hook placed in the victim's browser**
  - **By tricking them into visiting a malicious page, or using a vulnerability such as XSS**

Browser Exploitation Frameworks

- **Possible attacks**



- Logging keystrokes and sending these to the attacker
- Hijacking the user's session with the vulnerable application
- Fingerprinting the victim's browser and exploiting known browser vulnerabilities accordingly
- Performing port scans of other hosts (which may be on a private network accessible by the compromised user browser) and sending the results to the attacker
- Attacking other web applications accessible via the compromised user's browser by forcing the browser to send malicious requests
- Brute-forcing the user's browsing history and sending this to the attacker

Man-in-the-Middle Attacks

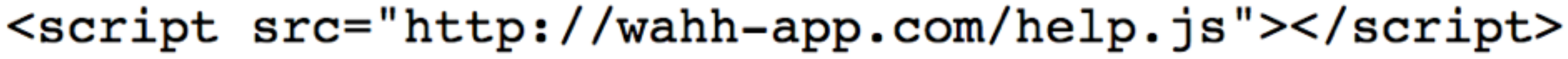
- **If app uses unencrypted communications, an attacker in the middle can intercept sensitive data like tokens and passwords**

- **But apps that use HTTPS can be attacked as well, if it loads any content over HTTP**

  - **Or even if it doesn't**

Separation of HTTP and HTTPS

- **Many apps, like Amazon, use both HTTP and HTTPS**
- **Browser separates HTTP cookies from HTTPS cookies, even for the same domain**
- **But consider a page that loads script over HTTP**



```
<script src="http://wahh-app.com/help.js"></script>
```

MITM Attack

- **MITM can modify any HTTP response to force user to reload that page over HTTPS**
  - **The script will still load over HTTP**
  - **Without a warning message (in some browsers)**
- **Attacker can inject script into the response, which has access to the HTTPS cookies**

HTTPS-Only Domains Like Google

- **Attacker can still induce the user to make requests for the target domain over HTTP**
  - **By returning a redirection from an HTTP request to a different domain**
- **Even if servers don't listen on port 80, MITM attacker can intercept those requests and respond to them**

HTTPS-Only Domains Like Google

- **Ways to escalate HTTP to HTTPS access**
  - **Set or update a cookie that is used in HTTPS requests**
  - **This is allowed even for cookies that were originally set over HTTPS and flagged as secure**
  - **Cookie injection can deliver an XSS exploit**

HTTPS-Only Domains Like Google

- **Ways to escalate HTTP to HTTPS access**
  - **Some browser extensions don't separate HTTP and HTTPS content**
  - **Script can leverage such an extension to read or write the contents of pages that the user accessed using HTTPS**