Secure Programming Lecture 12: Web Application Security III

David Aspinall

Informatics @ Edinburgh

Outline

Roadmap

We’re continuing to look at security in web application programming.

Some basics:

- Cookies and sessions

Some technical attacks:

- Session hijacking
- CSRF
- Unvalidated redirects
- XML External Entities
- Insecure deserialization

Outline

Cookies: state in a stateless world

Recall that HTTP is a **stateless** protocol.

- in principle, HTTP response depends only on request

But saving state is highly desirable between requests:

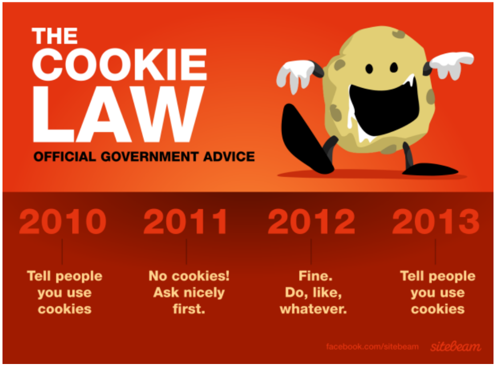
- remember user's preferences, navigation point, . . .
- web applications: **user logged in** for a session

However, also the less desirable:

- advertising network **tracking ids**
- may be shared between websites
- thus can profile user browsing behaviour
- hence **compromise privacy**
- also **risk of theft**
  - if browser/machine compromised, or
  - if cookies passed in clear

Outline

Cookies and the law (pre 2018)


THE COOKIE LAW
OFFICIAL GOVERNMENT ADVICE
2010
Tell people you use cookies
2011
No cookies! Ask nicely first.
2012
Fine. Do, like, whatever.
2013
Tell people you use cookies
facebook.com/sitebeam
sitebeam

This was a cheeky infographic claiming in 2013 “the stupid cookie law is dead at last”.

Cookies and the e-Privacy law (2003-)

The Privacy and Electronic Communications Regulations (EC Directive) apply in Europe and (currently) UK as well. These cover the storage of cookies particularly.

PECR Regulations 2003, Regulation 6 (paraphrased)

A person shall not use an electronic communications network to store information, or to gain access to information stored, in the terminal equipment of a subscriber or user unless:

- The purpose is explained and the user can refuse;
- The only reason is to enable a communication;
- or, storage or access is strictly necessary to provide a service requested by the user.

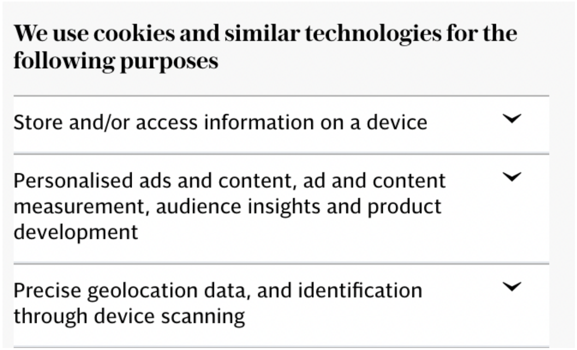
Cookies and the GDPR laws (2018-)

Cookies are now baked into the European Union General Data Protection Regulation GDPR and UK GDPR as a form of **Personal Data** subject to strict law on the need for explicit consent to be given for processing.

Extract EU GDPR, Recital 30

Natural persons may be associated with online identifiers . . . such as internet protocol addresses, **cookie identifiers** or other identifiers . . . . This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them.

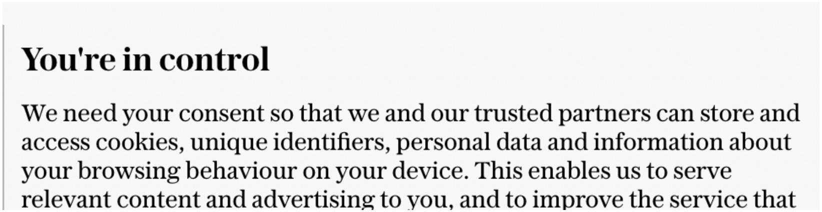
Messaging


We use cookies and similar technologies for the following purposes

Store and/or access information on a device

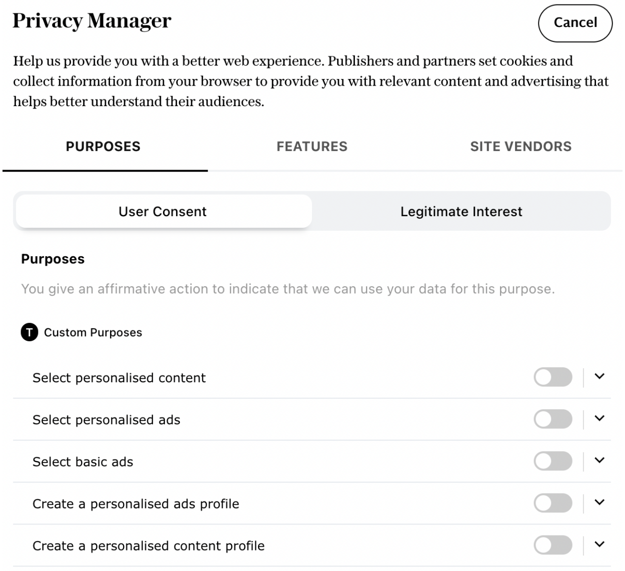
Personalised ads and content, ad and content measurement, audience insights and product development

Precise geolocation data, and identification through device scanning


You're in control

We need your consent so that we and our trusted partners can store and access cookies, unique identifiers, personal data and information about your browsing behaviour on your device. This enables us to serve relevant content and advertising to you, and to improve the service that

Privacy manager



Privacy Manager

Help us provide you with a better web experience. Publishers and partners set cookies and collect information from your browser to provide you with relevant content and advertising that helps better understand their audiences.

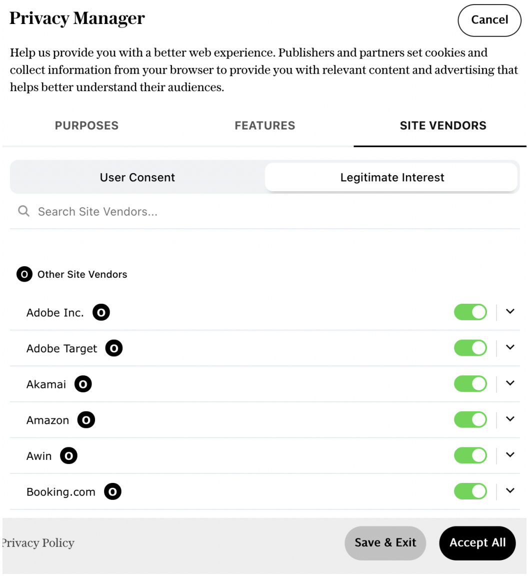
Purposes

You give an affirmative action to indicate that we can use your data for this purpose.

Custom Purposes

- Select personalised content
- Select personalised ads
- Select basic ads
- Create a personalised ads profile
- Create a personalised content profile

Privacy manager



Privacy Manager

Cancel

Help us provide you with a better web experience. Publishers and partners set cookies and collect information from your browser to provide you with relevant content and advertising that helps better understand their audiences.

PURPOSES | FEATURES | SITE VENDORS

User Consent | Legitimate Interest

Search Site Vendors...

O Other Site Vendors

Adobe Inc. O	on
Adobe Target O	on
Akamai O	on
Amazon O	on
Awin O	on
Booking.com O	on

Privacy Policy

Save & Exit | Accept All

Legitimate interests

(EU/UK) GDPR provides 6 lawful bases for processing personal data, the sixth one is

GDPR Article 6(1)(f)

Processing is necessary for the purposes of the legitimate interests pursued by the controller or by a third party, except where such interests are overridden by the interests or fundamental rights and freedoms of the data subject which require protection of personal data, in particular where the data subject is a child.

See guidance from UK ICO on legitimate interests.

Dozens of companies offer cookie consent management scripts, paid for or free, with varying formats (e.g., Quantcast, Osano, OneTrust).

Question. (Why) should you use a cookie management company?

Outline

Cookies in HTTP headers

Specified in RFC6265 (2011)

- Just ASCII plain text
  - Sent by server
  - **Stored in client** (database, filesystem, . . .)
  - Returned by client when visiting page again
- Cookies can be set by the server for a particular path/domain
  - then sent back for any page matching
- Multiple cookies may be set and returned
- Cookies may have a **limited lifetime**
  - set by `Expires` or `Max-Age`

Exercise. Discuss some of the security-related implications of cookies and their design.

Setting and retrieving cookies

We’ve already seen the `Set-Cookie` HTTP header line. It can be sent in responses (to set) and requests (to get).

Server -> User Agent (HTTP response)

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: mylanguage=en-GB; Path=/; Domain=example.com
```

User Agent -> Server (HTTP request)

```
Cookie: SID=31d4d96e407aad42; mylanguage=en-GB
```

Secure cookies?

RFC6265: *The Secure attribute limits the scope of the cookie to "secure" channels (****where "secure" is defined by the user agent****). When a cookie has the Secure attribute, the user agent will include the cookie in an HTTP request only if the request is transmitted over a secure channel (typically HTTP over Transport Layer Security (TLS) [RFC2818]).*

- . . . provided browser obeys this
- still, no harm in using (defence in depth)

the HttpOnly attribute is similar, and forbids the browser from allowing JavaScript access to the cookie, in principle at least.

Expiry dates

Server -> User Agent

```
Set-Cookie: mylanguage=en-US; Expires=Thu, 27 Oct 2028 10:18:14 GMT
```

User Agent -> Server

```
Cookie: SID=31d4d96e407aad42; mylanguage=en-US
```

- Of course, no guarantee cookie is kept for years. . .

Removing cookies

RFC6265: *To remove a cookie, the server returns a Set-Cookie header with an expiration date in the past. The server will be successful in removing the cookie only if the Path and the Domain attribute in the Set-Cookie header match the values used when the cookie was created.*

Server -> User Agent

```
Set-Cookie: lang=; Expires=Sun, 06 Nov 1994 08:49:37 GMT
```

User Agent -> Server

```
Cookie: SID=31d4d96e407aad42
```

- Again, no guarantee of what browser actually does
- . . . if indeed the same browser is being used

Where are my cookies stored?

Depends on the client program or browser used. . .

```
$ cd ~/Library/Application\ Support/Firefox/Profiles/∗.default/
$ sqlite3 cookies.sqlite
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> .tables
moz_cookies
sqlite> select count(∗) from moz_cookies
2536
sqlite> .schema moz_cookies
.schema moz_cookies
CREATE TABLE moz_cookies(id INTEGER PRIMARY KEY, originAttributes TEXT
sqlite> select count(host) from moz_cookies;
946
sqlite> select distinct host from moz_cookies where host like '%ac.uk'
blogs.ed.ac.uk
.birmingham.ac.uk
.ed.ac.uk
idp.brunel.ac.uk
je-s.rcuk.ac.uk
www.learn.ed.ac.uk
....
```

Outline

Sessions

Recall that a *session* is a way of linking a series of HTTP requests and responses over time, typically used to provide state tracking for an individual user on a device/browser.

Many web apps use session IDs (SIDs) as a credential.

- if an attacker steals a SID, she is logged in!

This is **session hijacking**.

Many possible theft mechanisms:

- XSS, sniffing, interception
- or: calculate, guess, brute-force
- also **session fixation**
  - using same SID from unauthenticated to logged in
  - attacker grabs/sets SID before user visits site

OWASP: Is the application vulnerable?

Poor Session ID (SIDs) management by:

- exposing SIDs in the URL (e.g., URL rewriting).
- SIDs are vulnerable to session fixation attacks.
- SIDs don’t timeout, or sessions/tokens aren’t invalidated in logout.
- SIDs are weak (small entropy, or predictable)
- Session IDs aren’t rotated after a new login.

Session hijacking defences

Web apps should implement defences, and discard SIDs if something suspicious happens.

- Link SID to IP address of client
  - but problems if behind NAT, transparent proxies
  - ISP proxy pools mean need to use subnet, not IP
  - subnet may be shared with attacker!
- Link SID to HTTP Headers, e.g. User-Agent
  - but can be trivially faked... and usually guessed
  - ...or captured (trick victim to visit recording site)

See the OWASP Session Management Cheat Sheet.

If in doubt, use a framework with buit-in protections.

General secure programming advice: reuse believed-to-be-secure solutions as far as possible. It may be tempting to use the latest fancy WhoJamig WebApp Framework but think twice unless you're sure it is well programmed for security, not just appearance.

Outlook for identity tracking

Some uses of cookies to track identity are being replaced:

- device IDs (still troublesome)
- persistent *advertising identifiers* obtained with consent

Question. What else?

However, session management still requires a token exchanged between the client and the server-side web application somehow. POST data can be used for this (though clunky).

Outline

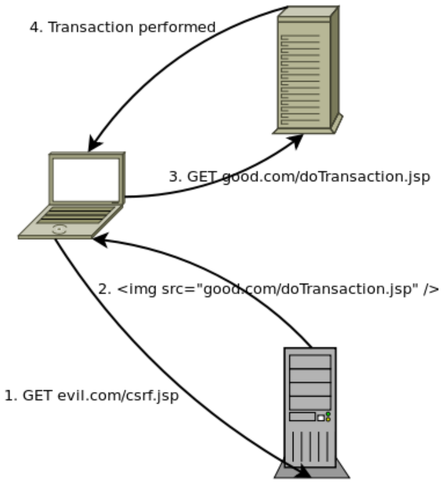
Cross Site Request Forgery (CSRF)

CSRF (CWE-352) is another technical attack which became common in the 2000s. Many CVEs still found.

- Attacker triggers malicious action
  - get user to open malicious link
  - browser undertakes action on target site
- Exploits *browser's* trust relationship with a web site
  - local intranet web site (home router admin, . . .)
  - banking or email site user is logged into
  - browser is authorized to connect here

Question. How does CSRF differ from XSS?

CSRF in pictures


4. Transaction performed
3. GET good.com/doTransaction.jsp
2. <img src="good.com/doTransaction.jsp" />
1. GET evil.com/csrf.jsp

CSRF in code

Alice **is logged in** to the (hypothetical) GeeMail web mail system.

She sends an email with this form:

```
<form
  action="http://geemail.com/send_email.htm"
  method="GET">
  Recipient’s Email address:  <input
    type="text" name="to">
  Subject:  <input type="text" name="subject">
  Message:  <textarea name="msg"></textarea>
  <input type="submit" value="Send Email">
</form>
```

Example GET request

Which sends a request like this:

```
http://geemail.com/send_email.htm?to=bob%40example.com
 &subject=hello&msg=What%27s+the+status+of+that+proposal%3F
```

Attacker’s cross-site request

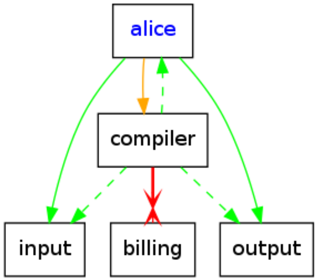
Now Mallory just needs Alice to visit a site which loads an image link, e.g., by putting a fake image on his own blog:

```
<img src="http://geemail.com/send_email.htm?
     to=charliegeemail.com&subject=Hi&msg=My+
     email+address+has+been+stolen">
```

and Alice’s browser will send an Email via GeeMail!

Confused Deputy problem

CSRF is a case of the Confused deputy problem.



The issue motivates the use of *capabilities*.

See *The Confused Deputy* by Norm Hardy, ACM SIGOPS Operating Systems Review, 1988. Picture credit: FP7 project Serscis

Avoiding CSRF problems

In general: tricky. Need some way to assure the server code that the request has come from intended place.

- `Referer` header not tamper proof, may be absent
- Session ID cookie sent based on *destination*
- The **Same Origin Policy** restricts client-side code

Best strategy (as a web developer): use a good framework that provides built-in protections.

But how do they work?

CSRF defence mechanisms

Some ideas:

- Don’t use GET for any (sensitive) state change
  - starting point
- Use a “double cookie” trick, repeated in POST
  - set a secure secret session ID in a cookie
  - submit it in cookie *and* hidden field on form
  - server-side, check fields identical
- Use a special CSRF token in POST
  - secure random number (challenge) for each login
  - send this with POST and check server-side
  - save state: generate using hmac from session ID

See *Robust defenses for cross-site request forgery*, Barth et al, ACM CCS 2008.

Browser **sandboxing** enhances this (e.g., in Chrome, separate tabs/frames run in separate processes).

Access control: Cross-Origin Resource Sharing

Modern browsers provide `XMLHttpRequest` JS API.

- script code sends messages back to the originating site, receives XML (or JSON, HTML, etc).

The same-origin policy became too restricted leading to hacks and workarounds and drafts for improvement including **Cross-Origin Resource Sharing (CORS)**.

CORS uses new HTTP headers to allow responses to indicate that they can be shared, for example:

```
Access-Control-Allow-Origin: http://www.example.com
```

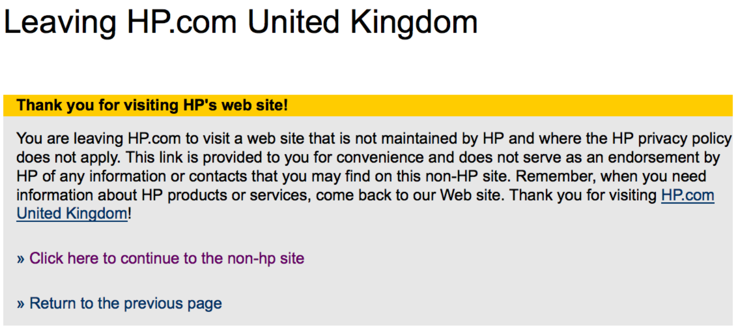
or

```
Access-Control-Allow-Origin: *
```

See the messaging standard part of HTTP for links to details.

Outline

Leaving HP



Leaving HP.com United Kingdom

Thank you for visiting HP's web site!

You are leaving HP.com to visit a web site that is not maintained by HP and where the HP privacy policy does not apply. This link is provided to you for convenience and does not serve as an endorsement by HP of any information or contacts that you may find on this non-HP site. Remember, when you need information about HP products or services, come back to our Web site. Thank you for visiting HP.com United Kingdom!

» Click here to continue to the non-hp site

» Return to the previous page

Unvalidated redirects and forwards

Web apps often allow *redirections* which

- send users off-site with a polite message
- or reroute them immediately

http://www.example.com/redirect.jsp?url=www.disney.com

Also webapps may use *forwards* which

- redirect internally to different parts of the same site

http://www.example.com/login.jsp?fwd=admin.jsp

Question. What’s the security concern here?

Giving attackers legitimacy

- Attackers can craft URLs that fool users:

www.example.com/redirect.jsp?url=www.evilhacker.com

These kind of **open redirect** links (CWE-601) are favourites for phishing attacks, especially as ultimate destinations can be concealed in URL encodings.

Notice this may not directly harm `www.example.com`.

So, preventing open reirects is a typical example of a *community wide* desirable security measure (like older cases in network security: open mail relays, ICMP broadcast, etc.): good practice of all provides security for others.

HTTP Redirect responses

URL redirection in HTTP allows resiliency for temporary outages or website reorganisations.

There are multiple variants with a response type with status codes begining with 3 (300-399).

```
HTTP/1.1 301 OK Moved Permanently
Location: http://www.disney.com
```

Validating redirects

The main risk is with *open* redirects from user supplied parameters as above.

Risks can be avoided by *validation*, to check the redirect is safe (e.g., to a recognised domain) before sending the response. But filtering can be tricky so better to:

1. Not use redirects at all
2. Use them but only with hard-wired URLS
3. If user-supplied parameters must be used, use indirection (indexes)

Another solution is to do the generation (& validation) of external links statically.

Question. Why is static generation of external links still not bullet-proof?

Outline

XML External Entities (XXE)

- Web applications process XML documents which can contain *external references* given as URIs.
- XML processors may obey these without restriction.
- Attacker may be able to upload/control files

Examples were known at least since 2002, exploits became common around 2014.

Defences: use more restrictive and specific formats for exchanging data, take care with deserialisation; configure DTD and XML processors to validate documents, enable security checks, prevent external entity processing.

This is CWE-611.

See the OWASP Cheat Sheet for some library specific advice.

DTD external entity

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
    <foo>&xxe;</foo>
```

Other references could be an internal server:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

or DoS with endless stream

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

XML bomb: A billion laughs

Nasty attacks are also possible without loading external entities:

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;
]>
<lolz>&lol9;</lolz>
```

See the Wikipedia entry.

Outline

Insecure deserialization

Concern: sending data from one application to another, especially when using programming-language specific **serialization** to encode (aka *marshalling*).

The **deserialization** of data received is the point where damage can occur.

PHP function of concern:

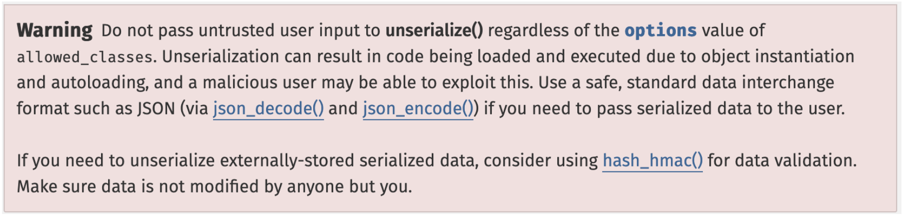
```
  unserialize(string $data, array $options = [])
```

The `options` argument can specify a maximum depth of deserialization and also a list of class names to accept. The default is to accept all all classes.

~~~~~

Question. What could go wrong in deserialization that makes it risky?
~~~~~

Warning from PHP manual



Warning Do not pass untrusted user input to **unserialize()** regardless of the **`options`** value of `allowed_classes`. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

If you need to unserialize externally-stored serialized data, consider using hash_hmac() for data validation. Make sure data is not modified by anyone but you.

Outline

Review questions

Cookies and sessions

- Explain two potential attacks on a web app user which exploit the app's usage of cookies.

Redirection

- What is an open redirect in a web application and why is it undesirable?

CSRF

- Draw a picture showing how a CSRF attack might work against an online banking user. What might an attacker be able to do? What does a CSRF defence mechanism need to be able to do?

XXE

- How might a browser's XML parser be exploited by an attacker and to what end?

References

This lecture contained material from:

- the OWASP Top 10
- *The Tangled Web: a Guide to Securing Modern Web Applications* by Michal Zalewski, No Starch Press, 2012.

as well as research papers and other sources cited.