Exploits using (reflected) XSS

Exploits user's trust in server
Exploits using (stored) XSS

![Diagram of XSS attack]

Exploits user's trust in server

Cross-site request forgery (CSRF)

1. Alice's browser loads page from bad.com
2. Script runs causing evilform to be submitted with a password-change request by loading www.good.com/update_pwd with attacker-specified field

```html
<form method="POST" name="evilform" target="hiddenframe" action="https://www.good.com/update_pwd">
<input type="hidden" id="password" value="badpwd">
</form>
<iframe name="hiddenframe" style="display: none"></iframe>
<script>document.evilform.submit();</script>
```

3. Browser sends authentication cookies to good server. Honest user's password is changed to badpwd!
Notes

- XSS attacks exploit the trust a client browser has in data sent from the legitimate website
  - But attacker controls what the website sends to the client browser
  - (attackers inject client-side scripts into web pages)

- CSRF attacks exploit the trust the legitimate website has in data sent from the client browser
  - But attacker controls what the client browser sends to the website

- XSS vulnerabilities are “more general”
  - Simply inject a script that, when viewed, submits a form on behalf of the user with parameters chosen by the attacker…

Defenses
Preventing XSS

- Escaping/encoding input
- Validation/sanitization
  - Suppress/escape <, >, " etc, … at time they are input by a user
- Can apply these techniques at the time data is read, or at the time the resulting page is displayed to the client

Preventing XSS

- Drawbacks
  - Sometimes these characters may be legitimate
  - Unclear when all malicious text is filtered out
- Very difficult (impossible?) to get sanitization right
- Several sanitizers exist…
  - …and several exploits of them are known
- Better to err on the conservative side
Preventing XSS

- Some work done on preventing XSS attacks at the browser level
  - Browser plug-ins (e.g., NoScript)
  - Browser itself (e.g., Google chrome)

- Mitigate XSS attacks for session hijacking
  - “HTTP-only” cookies sent only to the issuing server
  - Bind cookies to user’s IP address

Preventing CSRF attacks

- Inspect referrer headers
  - HTTP protocol specifies a header indicating the URL of the document from which current request originated

- So good.com can try to prevent CSRF attacks by ignoring POST requests if the referrer is not good.com

- However…
  - Referrer fields can be absent for legitimate reasons (e.g., new window; stripped by proxies)
**Complete mediation**

- Prevent CSRF attacks by requiring user re-authentication
- Not practical to do this all the time
  - User will be come frustrated!
- Can require for 'high-value' transactions

**Other**

- **Client-side protection**
  - (Assumes servers do not use GET requests for modifying data)
  - Browser plug-in that filters out POST requests unless requesting site and target site satisfy same-origin policy
    - Might still filter out some legitimate requests

- **Server-side protection**
  - Prevent CSRF attacks by allowing the legitimate server to remember or distinguish links in 'fresh' pages it serves, from links embedded in attacker pages
Other, cont

- Prevent CSRF
  - by having server give visitors a random number in a cookie, and
  - Requiring it to be echoed in a hidden field of form when submitted

- Details
  - When a CSRF launched, Trudy crafts form, but
    - She doesn't know the random value because
      - It's in a cookie, and she cannot touch that
      - It's not in the form when it comes from the server, just needs to be in the form when submitted to server
  - Note that the cookie is auto-appended to the attack form by the browser per usual procedure
    - Means that server will see the number, and will expect it in the form

Privacy and Anonymity
Privacy and anonymity

- Database privacy
- [Privacy in social networks]
- [Privacy on the web]
- [Anonymous communication]
- [Pseudonymity]

None of these are addressed by any of the techniques we have discussed so far!

What is different here?

- Privacy/pseudonymity
  - Different trust relationships – interactions between entities that partially trust each other
  - Inherently “fuzzy” – ok if a few people learn some information; not ok if everyone learns all information

- Anonymity
  - Classical crypto hides the contents of what is being communicated, but not the fact that communication is taking place
+ Databases...

- Several possibilities
  - Medical data
  - Scientific research (on human subjects)
  - US census data
  - Employment data
  - ...

- Data about oneself (e.g., on smartphone)

+ Two models

- Non-interactive data disclosure
  - Users given access to “all data” (after the data is anonymized/sanitized/processed in some way)
  - Note: it does not suffice to just delete the names!

- Interactive mechanisms
  - Users given the ability to query the database
Database privacy

- Want to be able to discern statistical trends without violating (individual) privacy
  - An inherent tension!

- Questions:
  - [How to obtain the raw data in the first place?]
  - How to allow effective data mining while still maintaining (some level of) user privacy?

- Serious real-world problem
  - Federal laws regarding medical privacy
  - Data mining on credit card transactions, web browsing, movie recommendations, ...

Database privacy

- A user (or group of users) has authorized access to certain data in a database, but not to all data
  - E.g., user is allowed to learn certain entries only
  - E.g., user is allowed to learn aggregate data but not individual data (e.g., allowed to learn the average salary but not individual salaries)
  - E.g., allowed to learn trends (i.e., data mining) but not individual data

- How to enforce?

- Note: we are assuming that authentication/access control is already taken care of...
Database privacy

- The problem is compounded by the fact that 'allowing effective data mining' and 'privacy' are (usually) left vague
  - If so, solutions are inherently heuristic and ad-hoc
- Recent work toward formally pinning down what these notions mean

The problem

- A user may be able to learn unauthorized information via inference
  - Combining multiple pieces of authorized data
  - Combining authorized data with "external" knowledge
    - 87% of people identified by ZIP code + gender + date of birth
    - Someone with breast cancer is likely a female
- This is a (potentially) serious real-world problem
  - See the article by Sweeney for many examples
### Example

- Say not allowed to learn any individual's salary

<table>
<thead>
<tr>
<th>Name</th>
<th>UID</th>
<th>Years of service</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>001</td>
<td>12</td>
<td>$65,000</td>
</tr>
<tr>
<td>Bob</td>
<td>010</td>
<td>1</td>
<td>$40,000</td>
</tr>
<tr>
<td>Charlie</td>
<td>011</td>
<td>20</td>
<td>$70,000</td>
</tr>
<tr>
<td>Debbie</td>
<td>100</td>
<td>30</td>
<td>$80,000</td>
</tr>
<tr>
<td>Evan</td>
<td>101</td>
<td>4</td>
<td>$50,000</td>
</tr>
<tr>
<td>Frank</td>
<td>110</td>
<td>8</td>
<td>$58,000</td>
</tr>
</tbody>
</table>

**Request denied!**

### Example

- Give me Alice's salary

**Request denied!**

- Give me the list of all names

Alice, Bob, Charlie, Debbie, Evan, Frank

- Give me the list of all salaries

$40,000, $50,000, $58,000, $65,000, $70,000, $80,000

Solution: return data in order that is independent of the table (e.g., random; sorted)
Some solutions

- In general, an unsolved problem
- Some techniques to mitigate the problem
  - Inference during database design
    - E.g., recognize dependencies between columns
    - Split data across several databases (next slide)
  - Inference detection at query time
    - Store the set of all queries asked by a particular user, and look for disallowed inferences before answering any query
    - Note: will not prevent collusion among multiple users
    - Can also store the set of all queries asked by anyone, and look for disallowed inference there
- As always, tradeoff security and usability

Using several databases

- DB1 stores (name, address), accessible to all
- DB2 stores (UID, salary), accessible to all
- DB3 stores (name, UID), accessible to admin

- What if I want to add data for “start-date” (and make it accessible to all)?
  - Adding to DB2 can be problematic (why?)
  - Adding to DB1 seems ok (can we prove this?)
**Statistical databases**

- Database that only provides data of a statistical nature (average, standard deviation, etc.)
  - Pure statistical database: only stores statistical data
  - Statistical access to ordinary database: stores all data but only answers statistical queries
  - Focus on the second type

- Aim is to prevent *inference* about any particular piece of information
  - One might expect that by limiting to aggregate information, individual privacy can be preserved

**Database privacy**

- Two general methods to deal with database privacy
  - **Query restriction**: Limit what queries are allowed. Allowed queried are answered correctly, while disallowed queries are simply not answered
  - **Perturbation**: Queries answered “noisily”. Also includes “scrubbing” (or suppressing) some of the data

- (Could also be combined)
Query restriction

- Basic form of query restriction: only allow queries that involve more than some threshold $t$ of users

- Example: only allow sum/average queries about a set $S$ of people, where $|S| \geq 5$ (say)

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Years of service</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>F</td>
<td>12</td>
<td>$65,000</td>
</tr>
<tr>
<td>Bob</td>
<td>M</td>
<td>1</td>
<td>$40,000</td>
</tr>
<tr>
<td>Charlie</td>
<td>M</td>
<td>20</td>
<td>$70,000</td>
</tr>
<tr>
<td>Dan</td>
<td>M</td>
<td>30</td>
<td>$80,000</td>
</tr>
<tr>
<td>Evan</td>
<td>M</td>
<td>4</td>
<td>$50,000</td>
</tr>
<tr>
<td>Frank</td>
<td>M</td>
<td>8</td>
<td>$58,000</td>
</tr>
</tbody>
</table>

Give me SUM Salary WHERE Gender='F'

Request denied!
Query restriction

- Basic query restriction doesn't work…

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Years of service</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>F</td>
<td>12</td>
<td>$65,000</td>
</tr>
<tr>
<td>Bob</td>
<td>M</td>
<td>1</td>
<td>$40,000</td>
</tr>
<tr>
<td>Charlie</td>
<td>M</td>
<td>20</td>
<td>$70,000</td>
</tr>
<tr>
<td>Dan</td>
<td>M</td>
<td>30</td>
<td>$80,000</td>
</tr>
<tr>
<td>Evan</td>
<td>M</td>
<td>4</td>
<td>$50,000</td>
</tr>
<tr>
<td>Frank</td>
<td>M</td>
<td>8</td>
<td>$58,000</td>
</tr>
</tbody>
</table>

```
Give me SUM Salary WHERE Gender='M'
```

$363,000  ↔  Alice's salary: $65,000  ↔  $298,000
Note

- Each query *on its own* is allowed
- But inference becomes possible once both queries are made

Basic query restriction

- Can try to prevent this by allowing queries about a set $S$ only if $|S|$ and $|S^c|$ are both large
- Does this help?

- Let $S$ be an arbitrary subset, containing roughly half the records in the database
- Request $\text{SUM}(\text{Salary}, S \oplus \{i\})$ and $\text{SUM}(\text{Salary}, S)$
- Determine salary of user $i$...