Improving Web Security:
Finding and fixing vulnerabilities in web security mechanisms

Devdatta Akhawe, Adam Barth,
Peifung E. Lam, John C. Mitchell and Dawn Song
Introduction

- The Web is complex and fast evolving.
- New browser features, protocols, and standards added at a rapid pace.
- Vulnerabilities and security invariants assumed by web applications.
- We believe that abstract yet informed models of the Web will be amenable to automation, reveal practical attacks, and support useful evaluation of alternate designs.
Introduction (cont.)

- The Web mechanisms we have studied include:
  - HTML5 Forms
  - Referer validation
  - WebAuth protocol

- Our analysis reveals previously unknown attacks
- Countermeasures proposed for each attack
These web mechanisms were analyzed using a common approach we have developed which involves:

- A formal model of the web
- Implementation of the formal model in Alloy
- Modeling of the web mechanisms under study in Alloy
Outline of the talk

• Attacks and countermeasures for
  • HTML5 Forms
  • Referer validation
  • WebAuth protocol
Modeling the Web

- A formal model of the Web
- Implementation of the model in Alloy
- Statistics of Alloy implementation
Attacks and countermeasures

- HTML5 Forms
- Referer validation
- WebAuth protocol
• HTML5 is the next major revision of HTML
• FormElement API in HTML5 can generate HTTP requests with PUT and DELETE methods
• Same origin policy applies to such requests
HTML5 Forms Spec

Same Origin

Browser

1. HTML5 Form

2. PUT/DELETE

Web Site

Cross-Origin

Browser

1. HTML5 Form

2. PUT/DELETE

Attacker Site

Web Site 2
HTML5 Forms - Attack

1. HTML5 Form

2. PUT/DELETE

3. HTTP Redirect

4. PUT/DELETE

1. HTML5 Form

2. PUT/DELETE

3. HTTP Redirect

4. PUT/DELETE

HTML5 Forms Spec

Browser

Attacker Site

Web Site 2

Attack!
Exploitation

- Attacker could illegitimately modify/delete resources on a RESTful website

Countermeasure

- Refuse to follow redirects of PUT/DELETE requests generated from HTML Forms
- Verified the fix up to a finite size in our model
- Recommendation accepted by the HTML5 working group
Referer Validation

- A proposed defense against Cross-Site Request Forgery (CSRF) and Cross-Site Scripting (XSS) [F. Kerschbaum, 2007]

- Websites would reject a request unless
  1. the referer header is from the same site, or
  2. the request is directed at an “entry” page vetted for CSRF and XSS vulnerabilities
Referer Validation - proposal

Figure adapted from F. Kerschbaum, “Simple cross-site attack prevention,” 2007, with attack (in red) added.

Web Site 2 → Protected Site

Entry Page → Allowed link

Internal Page → Forbidden and potentially malicious link
Referer Validation - Attack

Figure adapted from F. Kerschbaum, “Simple cross-site attack prevention,” 2007, with attack (in red) added.

Attacker Site

Protected Site

Entry Page

Allowed link

Internal Page

Forbidden and potentially malicious link
Exploitation

- CSRF and XSS can be carried out on websites protected with Referer Validation

Countermeasure

- This vulnerability is difficult to correct as Referer header has been widely deployed
- Websites can try to suppress all outgoing Referer headers using, for example, the noreferrer relation attribute on hyperlinks.
WebAuth

- Web-based Single Sign-On protocol
- WebAuth and a similar protocol, Central Authentication Service (CAS), are deployed at over 80 universities worldwide
- Although we analyze WebAuth specifically, we have verified the same vulnerability exists in CAS
WebAuth Protocol

1. request webkdc-service token
2. return webkdc-service token and session key

3. request resource
4. redirect to WebKDC w/ request token
5. redirect to WebKDC w/ request token
6. return login form w/ request token in a hidden form field
7. post login form w/ user credentials
8. set cookie w/ webkdc-proxy token; return a URL w/ id token pointing to WAS
9. access the URL link w/ id token
10. set cookie w/ app token; return requested resource

Figure adapted from http://webauth.stanford.edu/protocol.html
WebAuth Protocol - Attack

1. request webkdc-service token

2. return webkdc-service token and session key

3. request resource

4. redirect to WebKDC w/ request token

5. redirect to WebKDC w/ request token

6. return login form w/ request token in a hidden form field

7. post login form w/ user credentials

8. set cookie w/ webkdc-proxy token; return a URL w/ id token pointing to WAS

Attacker completes steps 1-8 and induces the user's browser to send message 9

9. access the URL link w/ id token

10. set cookie w/ app token; return requested resource
Exploitations

- An insider can share privileged web resources with unprivileged users without sharing login credentials
- Attacker can steal sensitive user information by logging users into attacker’s account
Countermeasure

• Store a nonce in a host cookie to bind messages 3 and 9, and splice in messages in between by including the nonce in the request and id tokens.

• Verified the fix up to a finite size in our model
Modeling the Web

- A formal model of the Web
- Implementation of the model in Alloy
- Statistics of Alloy implementation
A formal model of the web

• We model web entities including browser, servers, and network

• Our threat models include attackers with various capabilities, such as:
  • web attacker with no special network privilege, and
  • network attacker that can eavesdrop and/or modify unencrypted traffic at will
Main security goals we have identified include:

- **Security invariants**
  - Assumptions about how today’s Web works
  - Example: no DELETE in cross-origin HTTP requests
- **Session integrity**
  - Attacker does not participate in the HTTP transaction
Alloy

- A declarative language based on first-order logic
- Facts and predicates about a model are declared
- The Alloy code is translated into a SAT instance
- SAT solver searches for counterexamples using bounded exhaustive search
MetaModel in Alloy
Example code for session integrity

fun involvedServers[t:HTTPTransaction]:set NetworkEndpoint{
  (t.*cause & HTTPTransaction).resp.from
  + getTransactionOwner[t].servers
}

pred webAttackerInCausalChain[t:HTTPTransaction]{
  some (WEBATTACKER.servers & involvedServers[t])
}
## Statistics for the case studies

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Lines of new code</th>
<th>No. of CNF clauses</th>
<th>CNF gen. time (sec)</th>
<th>CNF solve time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML5 Form</td>
<td>20</td>
<td>976,174</td>
<td>27.67</td>
<td>73.54</td>
</tr>
<tr>
<td>Referer Validation</td>
<td>35</td>
<td>974,924</td>
<td>30.75</td>
<td>9.06</td>
</tr>
<tr>
<td>WebAuth</td>
<td>214</td>
<td>355,093</td>
<td>602.4</td>
<td>35.44</td>
</tr>
</tbody>
</table>

- The base model contains some 2,000 lines of code
- Tests were performed on an Intel Core 2 Duo 3.16GHz CPU with 3.2 GB memory
Conclusion

- We identified previously unknown attacks in HTML5 Forms, Referer validation, and WebAuth.
- Proposed countermeasures to the attacks.
- These attacks are identified based on a formal model of the Web that we have developed, which is then implemented in the Alloy language.
- This modeling approach not only enables us to discover practical new attacks, but also serves to verify the security of alternate designs, up to a certain size of the model.
References


Questions?

Thank you!