Web Attacks 102

Dave Kukfa

Last time: Offense

This time: Defense

Topics

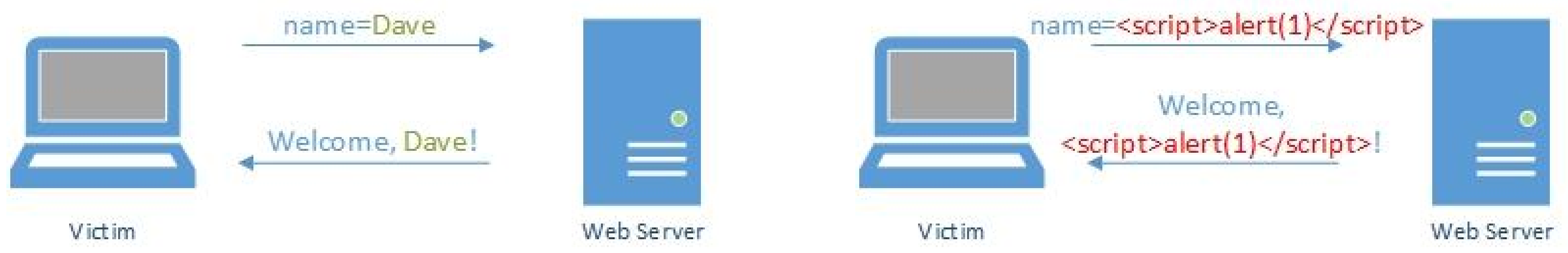
- XSS Recap
- XSS Defense Methodology
- SQLi Recap
- SQLi Defense Methodology
- ModSecurity

Me

- Dave Kukfa
- 3rd year CSEC
- Web
- Pentesting
- Reversing
- http://kukfa.co

XSS recap

Problem: user input being returned without being checked



name=Dave
Welcome, Dave!
Victim
Web Server
name=<script>alert(1)</script>
Welcome, <script>alert(1)</script>!
Victim
Web Server

- If user input contains HTML code, this code will run in victims’ browsers
- Malicious input can do bad things
    - Steal session cookies
    - Alter information on the page
    - Attempt to exploit the user’s OS

XSS defense methodology

Series of server-side checks:

1. Validate input
2. Validate output
3. Eliminate dangerous input points

Validate input

Incoming user input should be checked to make sure you're getting what you're expecting

- Limit input length
  - e.g. age parameter limited to 3 characters
- Only allow a certain subset of characters (whitelisting)
  - e.g. age parameter limited to 0-9 only
- Regex matching

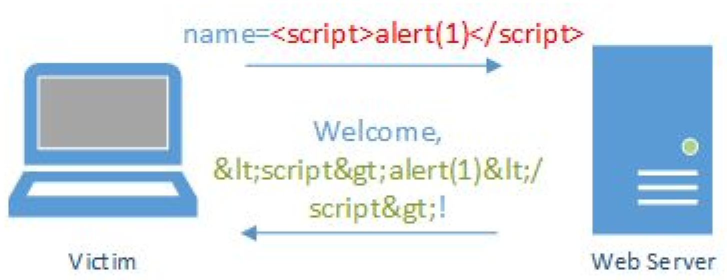
If any part of the input fails a check, do not attempt to process it any further

Validate output

User input being returned in a response from the server should be properly encoded to ensure it’s not interpreted as code

- HTML-encode problem characters
  - < > ‘ “ &
    - These characters can be interpreted as HTML tags or event handlers
      - Event handler: <button onclick="alert(1)">
    - AKA interpreted as code, or having to do with code
  - e.g. < ⟶ &lt;

New and improved



- Still appears as <script>alert(1)</script> on the screen
- But is &lt;script&gt;alert(1)&lt;/script&gt; in the source code

Eliminate dangerous input points

User input should never be inserted in an existing script tag or event handler

- Very difficult to secure
- Attack path is extremely wide at that point
- Filters would be easily avoided

SQLi recap

Problem: user input can break out of the structure of a SQL query and execute custom SQL code

SELECT * FROM users WHERE username='dave' and password='pass123';

SELECT * FROM users WHERE username='' or 1=1; #~~' and password='pass123';~~

Can do other things like:

- Extract critical data
- Add/remove/modify data
- Shut down DBMS

SQLi defense methodology

- Parameterized queries
  - Define the structure of the query, then pass the input in
  - Best way to prevent SQLi
- Defense-in-depth
  - Other measures to limit the impact of SQLi if it does happen

Parameterized queries

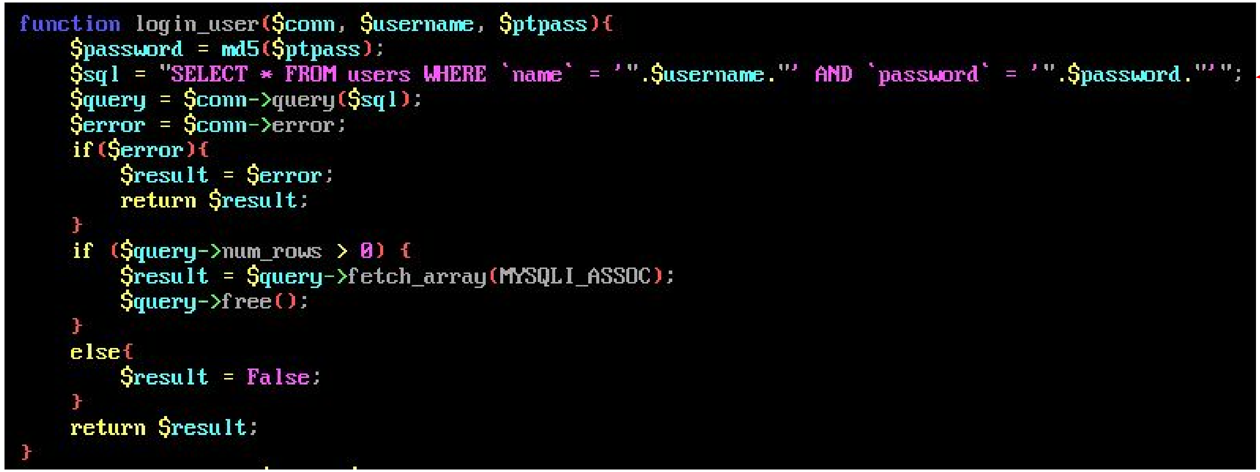
Call a series of functions to form a query:

1. Define the structure of the query, leaving placeholders for user input
    - User input can’t break out of the structure
2. Fill in placeholders with input
    - Inputs are passed in as parameters
    - Thus, *parameterized* queries

This way, user input will always be interpreted as data and never as code

Vulnerable login code

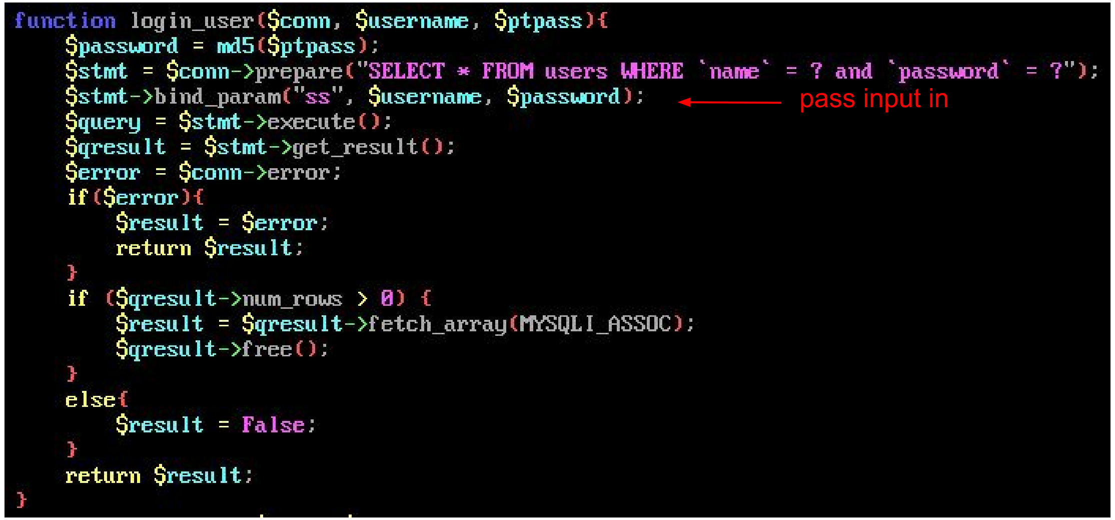
From RC3’s web challenge:



```
function login_user($conn, $username, $ptpass){
    $password = md5($ptpass);
    $sql = "SELECT * FROM users WHERE `name` = '".$username."' AND `password` = '".$password."'";
    $query = $conn->query($sql);
    $error = $conn->error;
    if($error){
        $result = $error;
        return $result;
    }
    if ($query->num_rows > 0) {
        $result = $query->fetch_array(MYSQLI_ASSOC);
        $query->free();
    }
    else{
        $result = False;
    }
    return $result;
}
```

user input inserted directly into query

Using parameterized queries



```
function login_user($conn, $username, $ptpass){
    $password = md5($ptpass);
    $stmt = $conn->prepare("SELECT * FROM users WHERE `name` = ? and `password` = ?");
    $stmt->bind_param("ss", $username, $password);
    $query = $stmt->execute();
    $qresult = $stmt->get_result();
    $error = $conn->error;
    if($error){
        $result = $error;
        return $result;
    }
    if ($qresult->num_rows > 0) {
        $result = $qresult->fetch_array(MYSQLI_ASSOC);
        $qresult->free();
    }
    else{
        $result = False;
    }
    return $result;
}
```

define query structure, using placeholders for input

pass input in

Defense-in-depth

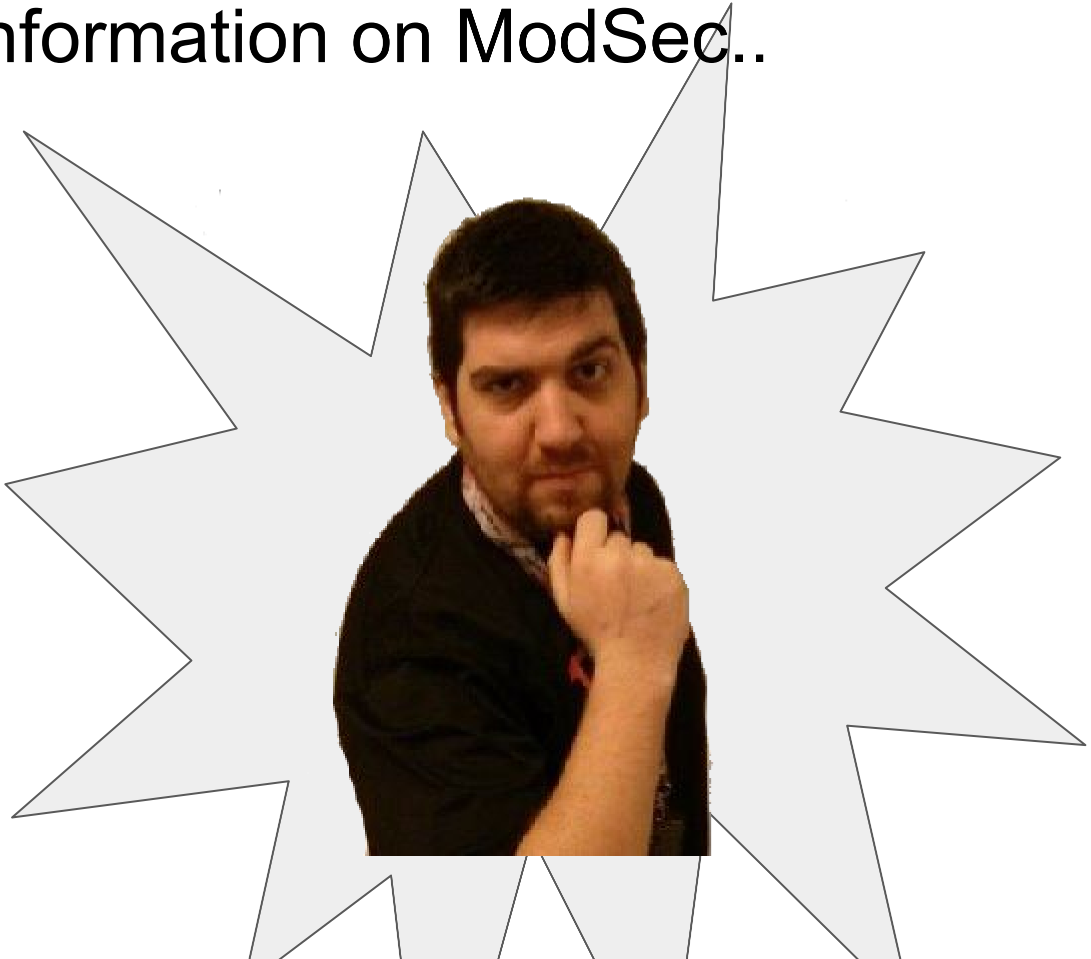
Additional security controls to reduce the impact of a SQL injection attack

- Use DBMS account with as few permissions as possible
  - If elevated permissions are required, switch to a separate account with higher permissions
  - Switch back to the lower-privileged account for regular use
- Remove or disable unnecessary DB functions
  - xp_cmdshell in MSSQL
    - Runs shell commands
- Segregate data between different systems

ModSecurity

- Web Application Firewall (WAF)
  - Compatible with Apache, Nginx, IIS
- Can block attacks like XSS and SQL injection
  - Wide range of other uses (e.g. logging, detection only)
- Flaws in application should still be fixed, but running ModSec is a good preventative measure

For more information on ModSec..



Questions?

- dxk2652@rit.edu