WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sruthi Sridhar Bopparthi

Email: bopparsr@mail.uc.edu

Figure 1: Sruthi's Headshot

Repository Information

Repository's URL: https://github.com/SruthiAelay/waph-bopparsr.git

This is a private repository which is used to store all the codes related to course Topics in Computer Systems. The structure of this repository is as mentioned below.

Hackathon 4 - Cross-site Request Forgery (CSRF) Attack and Protection

Lab's overview

In this hackathon, I'll engage in a hands-on exploration of Cross-Site Request Forgery (CSRF) vulnerabilities within a vulnerable web application. Taking on the roles of both attacker and victim, I'll gain firsthand experience with the mechanics and consequences of CSRF attacks. As the attacker, I'll simulate crafting deceptive requests aimed at exploiting the trust of authenticated users to execute unauthorized actions. Meanwhile, in the victim role, I'll witness

how innocuous-seeming actions on the web application can unwittingly trigger malicious operations on my behalf. This immersive exercise provides me with invaluable insights into the intricacies of CSRF vulnerabilities and equips me with the knowledge to better defend against such attacks in real-world scenarios.

Link to Hackathon : https://github.com/SruthiAelay/waph-bopparsr/tree/main/Hackathon/Hackathon4

#PART 1 : The Attack

https://waph-hackathon.eastus.cloudapp.azure.com/csrf

The attacker tricks the victim into clicking on a link that takes them to their server in this CSRF attack scenario. In this instance, the attacker tricks the target into changing their password. The attacker requests a password modification from the main server by pretending to be the victim. The victim first logs onto the attacker's website. The victim is sent to changepassword.php when they choose to change the password. Using browser tools to examine the page reveals a form that is susceptible. There are insufficient authentication mechanisms on this form, and it just has one input box called "newpassword." It then sends a POST request without doing any more authentication checks. The attacker uses JavaScript to send a fake "POST" request to changepassword.php in order to take advantage of this vulnerability. The altered value that the attacker entered into the "newpassword" input field is included in this request. The CSRF attack is started when the victim clicks on the given link and visits the attacker's website.

HTML Code:

```
<!DOCTYPE html>
<html>
<title>WAPH-Hackathon 4 </title>
<body>
    <h1>WAPH Hackathon 4: Cross Site Request Forgery Attack</h1>
    <h2> By Hacker Sruthi Sridhar Bopparthi</h2>
    <script>
        function CSRF()
        {
            var form = document.createElement('form');
            form.action="https://waph-hackathon.eastus.cloudapp.azure.com/csrf/changepasswor
            form.method='POST';
            form.target='_self';
            form.innerHTML="<input type='hidden' name='newpassword' value='bopparsrhacker'/>
            document.body.appendChild(form);
            alert('CRSF attack for Hackathon-4 is about to happen by - Sruthi sridhar Boppar
            form.submit();
        }
        CSRF();
    </script>
```

```
</body>
</html>
```

To lure the victim into clicking on the link, the attacker strategically places a comment with the link to the malicious site, coupled with a convincing message. This comment is meticulously crafted to evoke the victim's curiosity or manipulate their emotions, compelling them to follow the link and unwittingly succumb to the attacker's scheme. For instance, the attacker may construct a comment that tantalizes the victim's curiosity with promises of exclusive content or sensational information, or they may exploit the victim's emotions by presenting a seemingly urgent or emotionally resonant plea. By employing such tactics, the attacker aims to exploit human psychology and induce the victim to click on the link, thereby falling prey to the attacker's malicious intentions.

```
Hello Everyone! <a onclick="window.location='http://bopparsr.waph.io/bopparsr-csrf.html'">Cl
here</a> for 3% extra credit!! limited offer!!
```

Upon logging into the website, the victim stumbles upon a comment left by the attacker on a public blog. Intrigued, the victim clicks on the link embedded in the comment, unknowingly redirecting themselves to the attacker's site. Once on the attacker's site, an alert message is triggered, signaling the initiation of the attack. Subsequently, a POST request is generated and sent to 'changepassword.php,' with the value manipulated by the attacker. Since the request originates from the victim's browser while they are logged in, the server perceives it as authentic and proceeds to execute the password change. As a result, subsequent login attempts by the victim prove futile, as their password has been altered by the attacker's nefarious actions.

Video Link: Video Link

Part II: Understanding the CSRF vulnerability and protection mechanism

Vulnerabilities exploited in Part I

1) The CSRF attack capitalizes on a deficiency in the web server's authentication mechanism, which overlooks the distinction between the request referrer and the user. Instead, it relies solely on session authentication and browser origin checks. The absence of supplementary authentication measures renders the server susceptible to CSRF attacks. In essence, the server fails to verify the legitimacy of the request beyond basic session authentication, allowing malicious actors to exploit the trust established by the user's active session. This oversight underscores the importance of implementing robust authentication protocols that encompass thorough validation of request origins and user identities to fortify against CSRF vulnerabilities.

2) Websites that are open to the public should take precautions to stop hackers from inserting HTML links into comments. Users are prone to clicking on links without question since they typically have faith in the content posted on these platforms. Permitting attackers to incorporate HTML links, however, presents a serious security risk because these links might direct users to dangerous websites or help carry out phishing and cross-site scripting attacks. Public blogging platforms must implement strict content moderation guidelines and tools to detect and remove harmful links from comments in order to reduce these hazards. By doing this, these platforms can successfully defend their users against possible dangers, promoting a more secure and safe online environment for everybody.

Protection Mechanisms

1) By authenticating the source of requests, CSRF protection tokens—unique values sent along with every page or request—play a critical part in thwarting CSRF assaults. These tokens make guarantee that the server accepts only valid requests by keeping them secret from both users and attackers. Attackers are unable to get around this security mechanism when CSRF tokens are included in requests because the server verifies the token's existence and validity before handling the request.

2) By telling the browser to deliver cookies only if the request comes from the same domain as the target website, same-site cookies provide an efficient defense against cross-site request forgeries (CSRF) attacks. Same-Site cookies dramatically lower the danger of cross-site scripting request (CSRF) attacks by limiting the automatic inclusion of cookies in cross-origin requests. By limiting cookie transmission to direct user interaction with the website, this technique guards against fraudulent usage of session credentials.

3) Requests using custom headers can help servers authenticate requests by verifying the origin of the request. Nevertheless, the server neglects to verify the request referrer's header in the attack outlined, making it open to abuse. Attackers can alter requests to appear legitimate and circumvent the server's authentication procedures by performing illegal actions if request headers are not properly validated.

4) By making it more difficult for attackers to carry out unauthorized actions—even if they falsify the request—additional authentication procedures, such as requiring users to re-enter their password for crucial activities, increase defenses against cross-site scripting script reconnaissance (CSRF) attacks.