Cross-Site Request Forgery

https://campus.barracuda.com/doc/42048710/

Description

Cross-Site Request Forgery (CSRF) is a type of "confused deputy" attack in which a website acts on a request that an authenticated client has unknowingly initiated, often as a result of accessing a malicious website or image from a different tab in the same browser session. The malicious site assumes that the user is already authenticated to the target server and embeds hidden links in the targeted pages. When the victim's browser tries to fetch these links from the target server, it automatically adds authentication tokens like cookies to the requests. Such attacks typically aim at state-changing requests like financial transactions, password changes, or email address changes. The attacker does not receive confirmation of success, since the website would not send any confirmation to the victim. But, by successfully making these state changes without the victim's even suspecting anything, the attacker can gain ongoing access to the victim's account.

Effects

Successful attacks forge a predictable request and embed it in an image tag or another hidden object so an unsuspecting authenticated user invokes the request, unaware that anything has happened. This can be accomplished by tricking users into visiting a malicious website, or just by having them load or view a malicious image.

To succeed, the victim must currently be logged into the targeted website. CSRF attacks can execute any action the website allows the victim to perform. The attacker must correctly guess all required fields in the request. When the request is properly formed to a website the victim is currently authenticated on, the action appears to come from an authenticated "trusted" client, and the server complies with the request. As mentioned earlier, CSRF attacks typically focus on state-changing requests, since confirmation goes to the authenticated user rather than to the attacker.

Methods (Input Vectors)

Any predictable request can be forged by an attacker and embedded in seemingly safe objects that an authenticated user might access. Because of the low likelihood that a currently authenticated user would happen to access the malicious object, the best methods embed the request where an authenticated user is most likely to encounter it, such as on a related website. But CSRF attacks are also achieved through misleading emails with malicious links or invisible images with unexpected

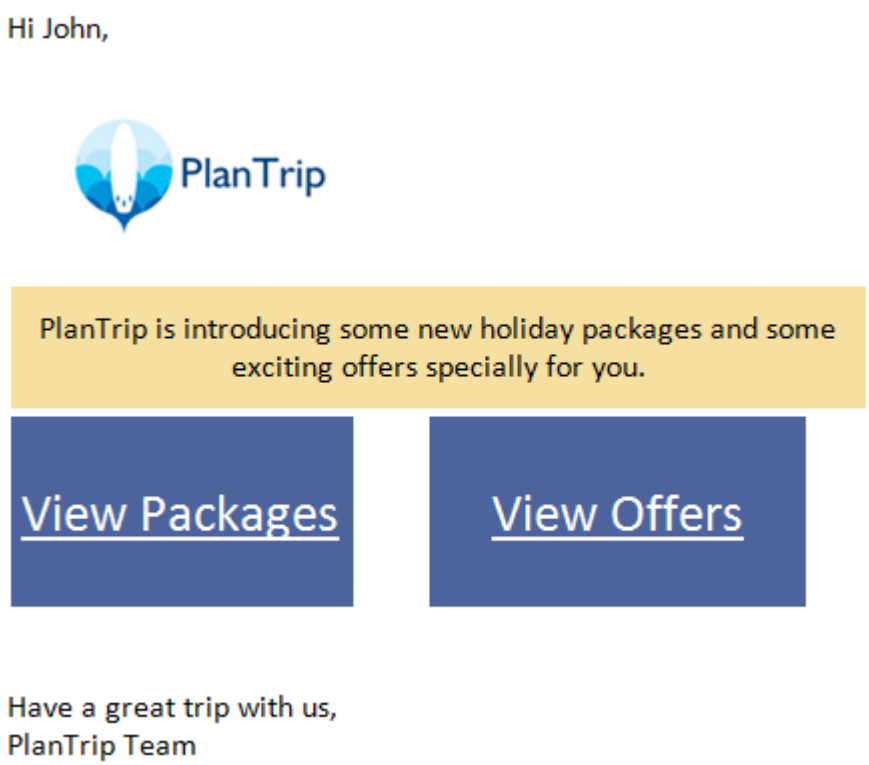
embedded URL requests on websites vulnerable to Cross-Site Scripting (XSS) attacks.

Example

An attack is considered a CSRF attack when a client, John, is already authenticated to a web application and if is forced in some way to perform an operation without his knowledge. The client might be fooled into performing these operations from different mediums (i.e., a malicious website, fake email, instant message, or malicious blog). The example below shows how John might be tricked into transferring money from his account to the attackers' account when John is already authenticated to the bank website.

Step 1

John receives the fake email below from the PlanTrip team (an attacker's identity)



Step 2

The attacker sends the above email to John by injecting the HTML code below shown here:

```
<table width="35%" cellpadding="10">

  <tr>

    <td style="text-align:center;background:#f5dfa1;display:block"><font size="3">PlanTrip is introducing some new holiday packages and some exciting offers specially for you.</font></td>

  </tr>

</table>

<table width="20%">

  <tr>

    <td style="text-align:center;background:#4c649b"><a href="http://www.bank.com/accounts/?transfer.php&ToAccountNumber=5555544444&Amount=100000" style="color:#3b5998;"><font size="5"><span style="color:white">View Packages</span></font></a></td>

    <td width="100"></td>

    <td style="text-align:center;background:#4c649b"><a href="http://www.bank.com/accounts/?transfer.php&ToAccountNumber=5555544444&Amount=100000" style="color:#3b5998;"><font size="5"><span style="color:white" >View Offers</span></font></a></td>

  </tr>

</table>
```

Step 3

The code is written in such a way that if either the `View Packages` or `View Offers` link is clicked, an HTTP request is sent to the URL: `/accounts/?transfer.php&ToAccountNumber=5555544444&Amount=100000` of the www.bank.com website.

The attacker has taken the URL which would perform the money transfer and embedded it in the HTML code.

John would read the above email and find it exciting to know more about the packages and offers. Here, John is unaware that if he clicks the `View Packages` or `View Offers` link, he would transfer $100,000 to the attacker's account.

Protecting against CSRF

OWASP recommends the Synchronizer Token Pattern for general protection against CSRF. This involves embedding or adding a random token to forms and URLs that make their location unpredictable. These tokens should be tied to single-user sessions, be time sensitive and should be generated using random number cryptography. This forms a "challenge token" mechanism that is expected by the server with each subsequent client request. If the token is absent or incorrect, the server denies the request. Effectively, this renders the URLs so dynamic that attackers are not able to predict their locations, thus mitigating CSRF. This is the primary mechanism used by the Barracuda Web Application Firewall to protect against CSRF.

Another mechanism to mitigate CSRF is to check the referrer header, which is hard to spoof in the context of CSRF attacks. If the referrer header points to a suspicious domain, the request is denied. However, this requires either a whitelist of the "good" referrers or a block-list of "bad" referrers to be maintained, which can be tricky. Commonly, requests originating from the same domain or sub-domain as the protected one can be considered secure. However, if there are other XSS or redirect vulnerabilities, such checks could be easily bypassed. Also, if the requests are genuinely referred from other domains, such mechanisms could generate false positives.

Other mechanisms to mitigate CSRF include challenges such as CAPTCHA and URL encryption.

Tags

OWASP Top 10, PCI-DSS

See Also

CWE-352, OWASP, WASC

Figures