World Journal of Research and Review (WJRR)
ISSN: 2455-3956, Volume-14, Issue-3, March 2022 Pages 08-11

A Case Study on Cookies and Cyber Security

Naman Kumawat, Ramesh Chaudhary, Ankit Kumar Tiwari

***Abstract*—Cookies can be collected, edited, or embezzled since they are saved and transported in the form of text. This paper examines the working concept of Cookies and suggests rules for making Cookies that protect them from security concerns. Three options are offered to meet the security criteria.**

Cookies, web security, multiple web authentication, and digital signatures are some of the terms used in this article.

I. Introduction

Cookies are necessary for the modern Internet, but they also pose a risk to your privacy. Cookies are a necessary element of the web browsing experience. Cookies assist website developers in providing more tailored and easy website experiences.

Cookies allow websites to remember you, your logins, shopping carts, and other information. They can, however, be a gold mine of personal information for crooks to snoop on. It's easy to become overwhelmed when it comes to protecting your privacy online. Fortunately, even a simple awareness of cookies can assist you in keeping prying eyes away from your online activities. While the majority of cookies are harmless, some can be used to track you without your permission. Worse, if a criminal gains access to a computer, legal cookies can be spied on.

II. How cookies works

The diagram above shows how a cookie works. Figure 1 shows how to navigate a website without using a cookie. When a web server receives a URL request from a browser, it searches for cookies. If cookies aren't accessible, it creates a cookie for the user with a unique id, embeds the website's header in the cookie, and sends it to the user. After then, the cookie will be saved on the user's hard drive. The website database stores the user's preferences and settings, along with a link to the cookie's id value. The cookie is then passed with the URL every time the user visits the same site, and the web server, using the unique id in the cookie, pulls the user's personalized preferences from the database and passes them to them.

Naman Kumawat, Department of Computer Science, Vivekananda Institute of Technology, Jaipur, India
Ramesh Choudhary, Department of Computer Science, Vivekananda Institute of Technology, Jaipur, India
Ankit Kumar Tiwari, Assist. Prof. Department of Computer Science, Vivekananda Institute of Technology, Jaipur, India

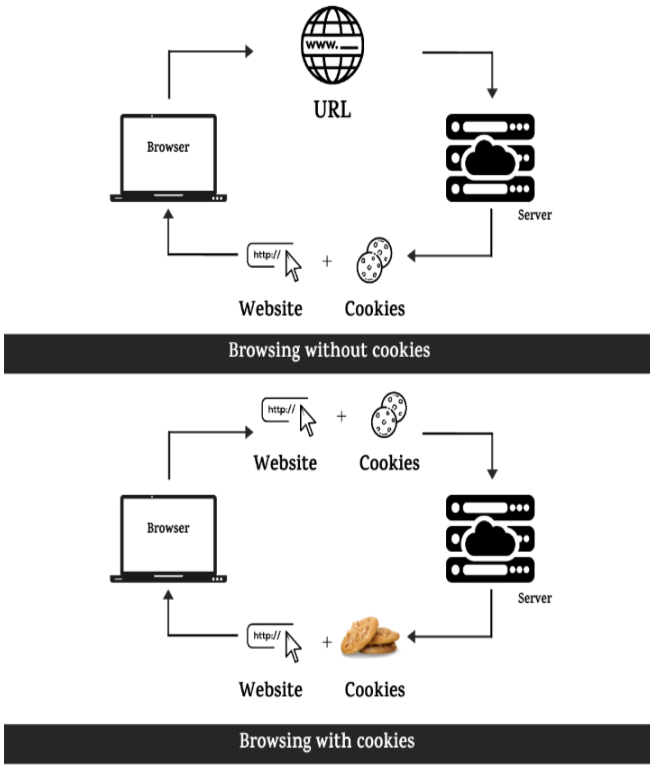

Fig. 1: URL request from a browser

III. Risk associated with cookies

A. Cross-Site request Forgery Attack

Cookies are vulnerable since they are delivered with every request, making it possible for attackers to use CSRF and make malicious requests. The impact of CSRF vulnerability is also determined by the victim's privilege, as well as the Cookie that is sent in response to the attacker's request. While data extraction isn't the primary goal of a CSRF attack, state changes will almost certainly have a negative impact on the web application under assault. As a result, it is recommended that you avoid implementing preventive approaches to protect your website from CSRF.

B. Session Fixation

Attacks on session fixation are based on the application level. An attacker forces the victim to use the attacker's or another's session ID in this type of attack. This is accomplished by exploiting the cookie's browser directive path, which allows the user to impersonate someone else. An attacker can use this strategy to persuade the user to log in as the attacker on several application levels.

C. Cross-site Scripting

An attacker must place the exploit in a cookie in order to carry out a cross-site scripting attack. The exploit vector will then retrieve the payload from the cookie and carry out the exploitation. If the cookie has already been set, this form of assault becomes more difficult.

D. Cookie Tossing Attack

Cookie tossing is one of the most common sorts of cookie attacks, and it works like this: Consider a user who goes to www.example.com and gets a domain cookie. The cookie is delivered to the web server the next time the user visits the same site. The issue now is that the cookie contains no path or website name. As a result, if an attacker creates a subdomain cookie and transmits it along with a real cookie, the web server will accept both. Because there is no rule requiring the browser to transmit the domain cookie first, it may opt to send the subdomain cookie first. If the malicious subdomain cookie is the first one received by the web server, it will be treated as valid, and the value of that cookie will be used to provide the user with a session. The web server is unable to determine which cookie is real since cookie properties such as domain path secure and HTTP only are not sent to it.

E. Cookies overflow attack

A parent domain cookie can be replaced with a subdomain cookie in this type of attack by employing a Jscript in the subdomain. Browsers have a restriction on how many cookies they can keep, and some, such as Chrome, don't check if the cookies are from a domain or a subdomain. It does nothing but save the cookies that are given to it. The subdomain cookies that will be updated will not be of the HTTP Only or secure types. After storing the subdomain cookie, an attacker can now modify the cookie's expiry date, rendering the cookie meaningless. Now the attacker has the ability to create a new malicious cookie and send it to the web server. Furthermore, there is no way for a web server to tell if a cookie is secure or HTTP Only. As a result, manufactured cookies can be used to carry out an attack.

IV. SECURITY REQUIREMENTS OF COOKIES

A. Confidentiality

Cookies typically contain information that is used to distinguish a user's status and individual material; thus, we must ensure that this information is kept private. Cookies' information can be revealed in two ways: first, it can be intercepted in transit, and second, it can be stolen when it is stored in the user's equipment.

B. Integrity

We must ensure cookie integrity in order to prevent an attacker from inserting harmful executable code into cookies using a specific mark. When cookies are used to verify a user's identity, if the content of the cookies is modified, the authentication will fail. An attacker might alter the content of a legitimate user's Cookies, preventing them from accessing a specific website. In cookies, the domain and path are extremely significant. Cookies will be transmitted to an attacker if these two are changed by an attacker.

C. Identifiability

Illegal tampering can be prevented by encrypting cookies. However, by uploading cookies obtained from genuine users to the Web site, the attacker can still impersonate the legitimate user. As a result, the criteria must be able to verify that the entity providing Cookies is the correctly owner of the Cookies

V. THE COOKIES SECURITY SOLUTIONS

A. Mark cookies as Secure

So, how do we ensure that that cookie is only accessible by our website? The first step is to ensure that the website is HTTPS-enabled. I'm not referring to using HTTPS instead of HTTP in this case. The only way to go is through HTTPS. No one can use a man-in-the-middle attack or something similar to inspect the traffic between the browser and the webserver if you solely use HTTPS. You'll need to notify it that cookies should only be available over HTTPS when you migrate to HTTPS. You can do so worldwide by adding the following to Web.config:

```
<system.web>
    ...
    <httpCookies requireSSL="true" />
</system.web>
```

B. *Mark cookies as HttpOnly*

There's a simple remedy because many cookies don't need to be accessible from JavaScript. HttpOnly cookies are marked. HTTP only cookies, as the name suggests, can only be accessible by the server during an HTTP (S!) request. The authentication cookie is a fantastic example of a cookie that should always be tagged as HttpOnly because it is only used to communicate between the client and the server.
Here's how to do it in Web.config:

```
<system.web>
    ...
    <httpCookies httpOnlyCookies="true" requireSSL="true" />
</system.web>
```

C. *Avoid TRACE requests*

It's not always enough to mark cookies as Secure and HttpOnly. Cross-Site Tracing (XST) is a hacking technique in which a hacker utilizes the request methods TRACE or TRACK to overcome cookies designated as HttpOnly. The TRACE method was created to aid debugging by allowing the client to view how a server interprets a request. This debugging information is printed to the response, allowing the client to read it.
If a hacker has successfully injected code onto your page, he/she could run the following script:

```
var xhr = new XMLHttpRequest();
xhr.open('TRACE', 'https://my.domain/', false);
xhr.send(null);
console.log(xhr.responseText);
```

The request, including server variables, cookies, and other data, is now written to the console if the recipient site supports TRACE requests. Even if the authentication cookie is designated as Secure and HttpOnly, this will reveal it.
Fortunately, TRACE queries from JavaScript are not supported by newer browsers. You can still rule out the option by making the following changes to your Web.config:

```
<system.webServer>
  <security>
    <requestFiltering>
      <verbs>
        <add verb="TRACE" allowed="false" />
        <add verb="TRACK" allowed="false" />
      </verbs>
    </requestFiltering>
  </security>
</system.webServer>
```

D. *Avoid CSRF*

We're nearly there. However, there is one issue that is lacking. All that effort to keep others from eavesdropping traffic between your client and server has resulted in yet another issue. You may have heard of a phenomenon known as Cross-Site Request Forgery (CSRF). The process of tricking a user into requesting a website where he or she is already signed in is known as cross-site request forgery (CSRF). This can be in the form of hidden forms, image elements, and more.
None of the aforementioned adjustments protect against CSRF. Both ASP.NET and ASP.NET Core have the ability to generate tokens that the server can use to validate each request. Allow your server to generate a one-of-a-kind token, then update all of your forms to incorporate it. ASP.NET (Core) checks the token before uploading data to the server and throws an error if it is invalid.
SameSite is a cookie characteristic that indicates whether or not your cookies are solely used for first-party requests. Let's have a look at an example to see what I mean. Cookies are transmitted between the client and server when a page on domain domain1.com requests a URL on domain1.com and the cookies are adorned with the SameSite attribute. Cookies are not transferred when domain2.com accesses domain1.com and the cookies on domain1.com are adorned with the SameSite attribute.

The SameSite attribute is supported by both.NET 4.7.2 and.NET Core 3.1. However, the simplest implementation (in my opinion) is to add a rewrite rule to Web.config:

```
<system.webServer>
  <rewrite>
    <outboundRules>
      <clear />
      <rule name="Add SameSite" preCondition="No SameSite">
        <match serverVariable="RESPONSE_Set_Cookie" pattern=".*" negate="false" />
        <action type="Rewrite" value="{R:0}; SameSite=lax" />
      </rule>
      <preConditions>
        <preCondition name="No SameSite">
          <add input="{RESPONSE_Set_Cookie}" pattern="." />
          <add input="{RESPONSE_Set_Cookie}" pattern="; SameSite=lax" negate="true" />
        </preCondition>
      </preConditions>
    </outboundRules>
  </rewrite>
  ...
</system.webServer>
```

SameSite=lax is automatically appended to all cookies by the rule. The term "lax" refers to when the cookie is sent in response to first-party requests or top-level navigation (URL in the browser changes). Another option is strict, which sends cookies only on first-party requests. A domain linked to your site will prevent IIS from sending the cookie in this scenario.

VI. Conclusion

The cookies system not only corrects HTTP's flaws, but it also adds a slew of additional features to the Web application. It is simple to implement and utilize at the same time. However, the widespread use of Cookies has resulted in a slew of security issues. During our analysis, we discovered that the following factors contribute to the security of cookies: First, cookies have design issues; second, the cookies application and configuration are improperly configured by developers and users, posing a safety risk. This study examines the two primary challenges in detail, including the operating principle and safety, as well as three strategies for improving the safety of Cookies.

Acknowledgment

This research was supported by Mr. Ankit Kumar Tiwari, Assistant Professor at Vivekananda Institute of Technology, Jaipur. In addition, the author would like to thank the reviewers for their valuable comments and suggestions.

REFERENCES

[1] Claudio Dodt. (2020 July 7). “Cookies: An overview of associated privacy and security risks”. Available: https://resources.infosecinstitute.com/topic/cookies-an-overview-of-associated-privacy-and-security-risks/.

[2] The Cookie Law Explained. Available: https://www.cookielaw.org/the-cookie-law/.

[3] Edwards, L. (2018). Data protection and e-privacy: From spam and cookies to big data, machine learning and profiling. Machine Learning and Profiling (2018 May 23). Forthcoming in L Edwards ed Law, Policy and Internet (Hart,2018).

[4] Thomas Ardal. (2019 December 19). “The ultimate guide secure cookies with web.config in .NET”. Available: https://blog.elmah.io/the-ultimate-guide-to-secure-cookies-with-web-config-in-net/.

[5] "What are cookies? What are the differences between them (session vs. persistent)?". Cisco. 17 July 2018. 117925.

[6] The Open Web Apllication Security Project (OWASP). 2018. Available: http://www.swascan.com/owasp/

[7] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In Proceedings of the 24th International Conference on World Wide Web, pages 289–299. International World Wide Web Conferences Steering Committee, 2015.

[8] Zachary Evans and Hossain Shahriar. Web session security: Attack and defense techniques. Case Studies in Secure Computing: Achievements and Trends, page 389, 2014

[9] The Cookie Collective. How We Classify Cookies, 2013. http://cookiepedia.co.uk/classify-cookies.