Cookies and Session Vulnerabilities

Derrick Lee
dlee3@scu.edu

Another person
email

Abstract—Cookies provide the fundamental backbone of modern web services. There are a number of security practices that are important to keep in mind to prevent attacks relating to cookies.

I. What Are Cookies

Cookies are part of the basis of what we consider the internet. From websites keeping you logged in to ad services tracking you across them, cookies allow states to be preserved across an otherwise stateless HTTP protocol. This is because the internet was built stateless, so each HTTP request makes a new connection and doesn't remember previous ones.

With the advancement of more complex websites and services requiring the need to maintain a state, cookies are used to send information in between the client and server. Utilizing this additional information that can be stored in the browser and sent back to the server in the next request opens up opportunities to add extra features that wouldn't have been possible without cookies. Such examples are keeping users logged in, saving personalized settings, and recording user behavior [1].

Cookies take the form as a key and value. Various different kinds of data can be stored as the value, encoded as a string. Cookies are set by the server via the Set−Cookie HTTP response header and are sent back from the browser for subsequent requests, also as a cookie header.

```
HTTP/2.0 200 OK
Content-type: text/html
Set-Cookie: a_cookie=my_cookie_value
Set-Cookie: another_cookie=yum

[page content]
```

Listing 1. Server response to set a cookie

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: a_cookie=my_cookie_value;
    another_cookie=yum
```

Listing 2. Client request including cookie

II. How Are Cookies Used?

In what ways do cookies benefit, make web browsing smoother, or more convenient for the user? At least for the case of preserving user logins, personalizing the browser, and providing added security, cookies are currently the easiest-to-implement solution that provides all these services at once.

A. Session Handling

One of the biggest use cases, and one of the most sensitive, is using cookies to preserve the state for user logins. This is done with a session cookie which saves an identifier on the user's browser and has the corresponding session data on the server. When requests are made from a user's browser, the session cookie is also sent which allows the server to identify the user and match them to the correct session data.

While this is convenient for the user, this function can pose some additional security vulnerabilities, and a more detailed explanation on how cookies handle the protection of user information can be found in the following section about "Cookie Vulnerability."

B. Auto-Form Completion

Similar to session handling, cookies can be used to save user input of personal, yet repetitive information like full names, email addresses, and date of births. This information is packaged into a cookie and sent to the web browser, which will help users automatically fill out this data the next time the same information is requested. While it might seem like an invasion of privacy, users only

have to fill out cumbersome forms– like shipping or billing information, medical or insurance forms, and the FAFSA – once.

C. Personalization

Various user settings can be saved in cookies as well. While a lot of settings are saved on the server or in other more modern and efficient storage APIs like localStorage and sessionStorage, cookies can be used to store things like a user's preferred language, location, or theme. For example, if a website has an optional dark mode, a cookie could be used to save such an option and provide the user with the website in a dark theme. In this specific example, the same functionality can be achieved in modern websites when rendering is done on the client [2].

III. Cookie Security

While cookies allowed a more featureful internet, it also increased the additional attack surface. Security was not in the original feature set, containing neither confidentiality protection nor integrity for the data sent between the client and server. With these vulnerabilities, there are several ways to prevent attacks relating to cookies.

IV. Cookie Vulnerabilities and Server Mitigations

Many of the security vulnerabilities related to cookies can be mitigated by properly configuring and implementing systems that interact with cookies on the server side. Having a secure system relies on the server to be aware of the following concepts and take action to prevent such attacks. A system's resilience is only as strong as its weakest link, so is it important to consider all the different security practices. Failure to do so could result in unauthorized access to user accounts, which depending on the service could mean breaches in private data, administrative privileges, and possibly large sums of money.

A. Confidentiality

Cookies, like regular HTTP requests, are sent over an unencrypted channel where anyone can view the content. While it is not good practice to store sensitive data in cookies, it still has critical information like a session identifier.

To prevent people from snooping on cookies they're not intended to view, the secure attribute on the cookie can be set to limit the cookie to requests that are only transmitted over HTTPS with TLS (Transport Layer Security). This ensures that the cookies are only sent when they're used on an encrypted site and hides the content from malicious users. However, this does not provide integrity of cookies — attackers can modify cookies from an insecure channel. The secure flag should be used in combination with other safety mechanisms like HSTS, described in detail in the next section.

```
HTTP/2.0 200 OK
Content-type: text/html
Set-Cookie: my_secure_cookie=no_snooping_here;
    Secure

[page content]
```

Listing 3. An example of server response header to set a secure cookie

Thus, communicating over encrypted HTTPS connections prevents the "middle men" to view cookies and associated data, effective against network MITM attacks [3].

B. Integrity and HSTS

Defined as keeping data from being tampered with or modified, integrity is a very important aspect of security. HTTP Strict Transport Security (HSTS) mitigates some of the associated issues related to MITM, cookie sniffing, and SSL stripping [4].

There is one aspect of HSTS that is important to keep in mind — it uses a Trust on First Use model [5]. If the first connection to the server is already compromised, HSTS may not prevent any attacks relating to using unencrypted channels as an attack vector.

HSTS ensures all connections after the initial HSTS header are made over HTTPS until it expires. The recommended duration for such a policy is over 120 days, and ideally 1 year [6]. This means that no unencrypted connections are allowed to be made for the entire duration, and thus the attack window is very small.

Another security aspect of HSTS is subdomains, the additional part of main domain names that are useful for navigating and organizing different parts of a site. An example of a subdomain for example.com is subdomain.example.com. The "includeSubDomains" directive, should be set so that the HSTS policy will also cover subdomains. Otherwise if it isn't set, subdomains will be accessible over HTTP, opening up the attack surface.

```
Strict-Transport-Security: max-age=31536000;
    includeSubDomains
```

Listing 4. Example of the HTTP Strict-Transport-Security response header

The above example enforces HTTPS on all current and future subdomains for 1 year.

C. Session Hijacking and XSS

Session hijacking attacks are very harmful because they not only give attackers access to private data, but also bypass authentication to use a victim's session. For example, hijacking cookies could allow full access to a victim's Facebook account, email, or even bank accounts. Not only will the attacker be able to view your private details associated with such accounts, but they can also do anything as victim, such as sending messages, emails, money, etc.

One example of session hijacking is through Cross-Site Scripting (XSS) attacks. When malicious scripts are injected into websites, possibly by exploited 3rd party scripts or unsanitized inputs, an attacker's script (written in JavaScript) can read and receive a victim's cookies.

With a malicious script, it is extremely easy to both read and modify cookies.

```
<script>
document.cookie = "a_cookie=its_modified_now";
document.cookie = "another_cookie=attacc";

console.log(document.cookie);
// logs "a_cookie=its_modified_now;
    another_cookie=attacc"
</script>
```

Listing 5. Example of reading and modifying cookies

From the example above, it can be seen that reading and changing cookies is trivial, and can be extended to send the victim's cookies to an attacker which can be used to hijack the victim's session.

In order to prevent XSS attacks, the HttpOnly cookie flag provides very good protection. Since XSS attacks in most, if not all, use injected JavaScript code, the HttpOnly is effective as it prevents the access to cookies via JavaScript. This means attackers who inject malicious scripts cannot access cookies, and thus user data is protected.

```
Set-Cookie: my_extra_secure_cookie=
    no_snooping_here; Secure; HttpOnly
```

Listing 6. An example of server response header to set a Secure and HttpOnly cookie

Using HttpOnly cookies is the same as Secure cookies, just an addition to the Set-Cookie response header.

D. Cross-Site Request Forgery (CSRF)

Since cookies, including ones that include session data, are sent with HTTP requests automatically, it makes cross-site request forgery (CSRF) attacks easier. In short, CSRF impersonates a user and sends requests as the victim with possibly malicious actions. These actions include inserting a url that performs an action like sending money into an image element on a malicious website. Doing this would cause the victim's browser send a request for the image url, which in reality is a url set by the attacker, that would receive the victim's session cookies and thus do some requests as the victim [7].

Some examples of CSRF attacks are shown below [8].

```
<img src="https://bank.example.com/withdraw?
    account=bob&amount=1000000&for=mallory">
```

Listing 7. CSRF implementation with a GET request

```
<form action="https://bank.example.com/
    withdraw" method="POST">
  <input type="hidden" name="account" value="
      bob">
  <input type="hidden" name="amount" value
      ="1000000">
  <input type="hidden" name="for" value="
      mallory">
</form>
<script>window.addEventListener('
    DOMContentLoaded', (e) => { document.
    querySelector('form').submit(); }</script>
```

Listing 8. CSRF implementation with a POST request and invisible form

There are a few ways to mitigate CSRF, including idempotent GET endpoints, CSRF tokens, SameSite attribute, and HTTP headers.

One of the most important aspects to prevent such an attack is maintaining idempotent GET endpoints. In listing 1 above, the image element performs a GET request with the malicious parameters, withdrawing money from a victim and sending it to the attacker. In this case, it is obvious that the request does modifications to the victim's account, which should not be done. Idempotent GET endpoints are supposed to be "safe," meaning that GET only retrieves data, does not modify anything, and repeated calls do not change the outcomes [9].

Another way to prevent CSRF is to use CSRF tokens. A CSRF token is a randomly generated value and inserted into the HTTP request header or as a form parameter. This is sent with the request and is required to be correct so that the request is accepted. This prevents malicious requests like the second example above, as the attacker cannot guess the token and cannot get the token from the user's browser [10].

In addition to CSRF tokens, cookies can have the SameSite attribute set. This only allows cookies to be sent to websites that the user is currently on as a first party cookie. For example, if cookies made by example-bank.com have the SameSite attribute, the cookie will only be sent when a user is on example-bank.com and will not be sent if example.com sends a request to example-bank.com. More information about first and third party cookies are described in the next section.

```
https://bank.example.com/withdraw?account=bob&
    amount=1000000&for=mallory&token=1990
    AAAFF39F34E0D36C1D4380657AD5
```

Listing 9. CSRF token with a GET request

With the CSRF token sent with the request, the attacker will not be able to perform the same attack above with the HTML image tag. Similarly, the token can be embedded and sent as a form field to prevent attacks with POST requests.

Finally, certain HTTP headers can further prevent CSRF attacks. The referer header tells the server the previous page or the page that sent a request [11]. This means the server can check which page sent the request, and if the request is sent from a 3rd party site, it can be rejected. If a user visits malicious.example.com which sends a request to bank.example.com, the request to bank.example.com will have the Referer HTTP header set to malicious.example.com. In this case, bank.example.com will be able to check the header and handle the illegitimate request properly.

V. Tracking and Privacy

Cookies may seem to provide a lot of benefits to website users. However, they can be used – at the cost of the user's privacy – to track users for analytics or advertising purposes. Thus, awareness of third party cookies and their possible risks are essential to maintaining user privacy.

A. Third party cookies

When visiting a website, a third party cookie can be set by a different website from sources like embedded JavaScript. For example, if you are visiting example.com and a cookie is set for example-ads.com, it is considered a third party cookie. First party cookies, on the other hand, are set by the website you are visiting. An example would be example.com setting cookies for example.com. This means any website making requests to example-ads.com would use the same third party cookie, and thus show your browsing habits to example-ads.com.

Analytics services can track users on websites that aren't directly associated with them, and are extremely widespread on the internet. For example, Google Analytics is used on a large amount of websites, currently used in 67% of the top 1 million websites [12].

Facebook is also known for tracking users, even if you're not using Facebook. If you've seen a Facebook like or button on a website, external resources are fetched from Facebook, and thus passing data to the 3rd party. This also could happen if you don't see any Facebook related media, such as with Facebook Pixel which embeds invisible tracking scripts similar to Google Analytics [13]. If you are logged into your Facebook

account, your browsing habits can be directly linked to who you are.

While tracking and analytic services extend further than cookies, it is important to keep in mind, and one of the methods of tracking is to utilize cookies.

In order to prevent tracking, or rather reduce the amount of tracking done, users can disable third party cookies in browsers. You can also install an adblocker like uBlock Origin, many of which also block common tracking and analytics scripts. However, it is important to know that doing these things does not prevent all tracking. There are many other ways to identify users across browsing sessions even with cookies disabled such as browser fingerprinting, but are out of the scope of this paper [14].

VI. Client Mitigation and Prevention

What is a definitive way of protection from security risks? That is, to take preventive measures of avoiding situations that compromise your security. A system with an absolute zero attack surface is impossible, but there are ways to decrease risks and mitigate threats. While a big part of security related to cookie relies on the servers and service providers to correctly implement their systems keeping the vulnerabilities in mind, there are a few ways users can reduce the attack surface.

A. Keeping Up To Date

Using a modern, up to date browser is vital to having HSTS and HTTPS support. Many sites now use HSTS and many more use HTTPS. There are even certain top-level domains (TLDs) such as .dev that are on the HSTS preload list, making HTTPS required on all connections without additional configuration.

The HSTS preload list is also a valuable tool browsers use to enforce the use of HTTPS and further reduce the attack surface of HSTS's Trust on First Use model by having websites hardcoded into browsers to use HTTPS only. Most modern browsers like Chrome, Firefox, and Safari have HSTS preload lists.

Modern browsers also are more strict against unsecured connections, providing the user with big warnings and possibly preventing the user from connecting when there is an invalid security certificates (expired, wrong host, self signed, or untrusted root certificates) or even when there is unencrypted content.

Additionally, having an up to date browser is key to having modern cryptography so that the browser's connection to the server uses the strongest possible security. Having legacy cryptography could mean the encryption being used to transmit data over HTTPS is possibly breakable, which causes the Secure cookie flag to be less effective.

B. Using a VPN

A Virtual Private Network (VPN) allows users to route their traffic through an encrypted tunnel on a remote server. Contrary to popular belief, a VPN does not provide anonymity or extra security. While the controversies of popular VPN providers and misbeliefs of VPNs will not be discussed in this paper, it is important to note what a VPN can or cannot provide. In short, a VPN will not keep your browsing habits anonymous, won't add extra security to unencrypted traffic, and is most definitely not a replacement for the aforementioned security practices.

However, one thing a VPN may provide is more privacy from your Internet Service Provider (ISP) and more importantly other users on a public Wi-Fi network. This reduces the threat from MITM attackers on a local network sniffing packets or attempting to tamper with them. Also, this only applies to attacks on the local network, provides little to no protection against remote attacks such as XSS and CSRF.

Conclusion

Cookies provide a rich world of internet services. There may be quite a few different ways for a malicious actor to eavesdrop or attack users and services, but there are also many ways to prevent such attacks from happening.

Since there are so many security aspects to keep in mind when designing a system that utilizes cookies, opportunities to attack may likely result from improper implementations and practices from both engineers and users. Being aware of

what attack vectors exist are key to preventing cookie attacks.

References

[1] K. LaCroix, Y. L. Loo and Y. B. Choi, "Cookies and Sessions: A Study of What They Are, How They Work and How They Can Be Stolen," 2017 International Conference on Software Security and Assurance (ICSSA), Altoona, PA, 2017, pp. 20−24.

[2] "HTTP Cookies". MDN Web Docs, developer.mozilla. org/en−US/docs/Web/HTTP/Cookies.

[3] S. Sivakorn, I. Polakis and A. D. Keromytis, "The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information," 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA , 2016, pp. 724−742.

[4] Xiaofeng Zheng, X. Zheng, J. Jiang, J. Liang, H. Duan , S. Chen, T. Wan, and N. Weaver, "Cookies Lack Integrity: Real−World Implications", Proceedings of the24th USENIX Security Symposium, Washington, D. C. 2015

[5] "Secure Your Website with HSTS". CA Security Council, 8 Oct. 2014, casecurity.org/2014/10/08/secure −your−website−with−hsts/.

[6] Yang, Dingjie, et al. "The Importance of a Proper HTTP Strict Transport Security Implementation on Your Web Server". Qualys Blog, 28 Mar. 2016, blog. qualys.com/securitylabs/2016/03/28/the−importance −of−a−proper−http−strict−transport−security− implementation−on−your−web−server.

[7] "CSRF". MDN Web Docs, developer.mozilla.org/en−US /docs/Glossary/CSRF.

[8] "HTTP Cookies". MDN Web Docs, developer.mozilla. org/en−US/docs/Web/HTTP/Cookies.

[9] Ju, et al. "Idempotent REST "APIs –REST API Tutorial, restfulapi.net/idempotent−rest−apis/.

[10] "Cross−Site Request Forgery Prevention". Cross−Site Request Forgery Prevention · OWASP Cheat Sheet Series, cheatsheetseries.owasp.org/cheatsheets/Cross− Site_Request_Forgery_Prevention_Cheat_Sheet.html.

[11] "Referer". MDN Web Docs, developer.mozilla.org/en− US/docs/Web/HTTP/Headers/Referer.

[12] "Analytics Usage Distribution in the Top 1 Million Sites". Analytics Technologies Web Usage Distribution, trends.builtwith.com/analytics.

[13] "Facebook Pixel". Facebook for Business, www.facebook .com/business/learn/facebook−ads−pixel.

[14] "Browser Fingerprinting: What Is It and What Should You Do About It"? Pixel Privacy, pixelprivacy.com/ resources/browser−fingerprinting/.