Cookies and Sessions: A Study of what they are, how they can be Stolen and a Discussion on Security

Young B. Choi[1]
Department of Engineering & Science
College of Arts & Sciences
Regent University
Virginia Beach, USA

Yin L. Loo[2], Kenneth LaCroix[3]
College of Arts & Sciences
Regent University
Virginia Beach, USA

***Abstract*—Cookies and sessions are common and vital to a person's experience on the Internet. The use of cookies was originally used to overcome a memoryless protocol while using a tiny amount of the system's resources. Cookies make for a cohesive experience when shopping online, enjoying customized content, and even receiving personalized advertisements when casually surfing the Web. However, by design, cookies lack security. Our research begins by giving a background of cookies and sessions. It then introduces what session hijacking is, and a lab was constructed to test and show how a cookie can be stolen and replayed to gain authenticated access. Finally, the paper presents various countermeasures for common attacks and tools checking for authentication cookies vulnerabilities.**

Keywords*—AED; ARP spoofing; cookies; CSP; CSRF; HSTS; man-in-the-middle attack; newton; session hijack; web session; XSS

I. Introduction

Hypertext Transfer Protocol (HTTP) existed before cookie and led to the formation of cookies because of its design. On the Web, HTTP is the bedrock for data communication between the Web browser (also known as the client) and the Web server. Upon the click on a link (also known as the hyperlink or hypertext), the client makes a request to the Web server. After the client receives a response from the Web server, it disconnects. Every click on the link, even if it is the same link, sends a new and unrelated request. This process describes the "stateless" nature of the requests and the protocol because the Web server does not remember any of the earlier requests [5][8].

To overcome this occurrence, Web-based applications use cookies as a mean to establish state or "create a memory of where it left off' [6][8]. A *cookie*, therefore, is simply "a small piece of information that the server and client pass back and forth [5][6]. Montulli named the data file *cookie* because it functioned very much like the computer term *magic cookie* which is a data token that is passed back and forth between two parties [6][8]. The latest cookie replaces the existing cookie when there is new or updated information, and this is useful for the server to return to later [5]. There are other ways to achieve a stateful connection and using a cookie is one of the simpler ways. Many sites require a user to log in to experience customized contents. This is especially true for shopping cart applications [1]. Cookies are also used to trail users when they surf [1]. This is helpful for the site administrator to organize contents in a way that is more accessible to the users [1].

Information stored in cookies can be used to establish Web sessions; Web sessions are important because they facilitate a partially permanent information exchange between a browser and a server across multiple requests and replies [2]. Once a server authenticates a client, Web sessions are formed and bound. Subsequently, all requests from the client will include the cookie as part of an established session [2]. Web sessions also keep a user signed on to a website. However, due to the possibility of exposure, it is risky to store session information directly in a cookie. Instead, a session identifier is used to allow the web server to access state information when needed. But, this only improves the security somewhat, as the session could be transplanted and be used to freely communicate with the server as an authorized party [1], [6].

II. Introduction to Lab: Session Hijacking

In the following sections, a lab is used to test how a session can be hijacked by Address Resolution Proofing (ARP) spoofing [6]. Once the attacker has the session id, the attacker could have authorized access to the server by pretending to the legitimate user [6]. The lab may be reproduced for educational purposes because the first-hand experience of a "victim" can vastly increase the level of security awareness.

ARP translates Internet Protocol (IP) addresses to a physical machine address. The physical machine address (also known as the MAC address) is an alphanumeric string that uniquely identifies the Network Interface Card (NIC). The MAC address can be changed temporarily using tools that are available both on the Kali Linux and on Windows [13]. If the address can be changed, it can be spoofed [13]. Devices on the network maintain MAC addresses locally and MAC address of other devices may be learned via ARP requests [12]. Since no authentication or verification is needed from the requester, the ARP protocol can be exploited by flooding the network with false ARP requests.

The lab is an example of a Man-in-the-Middle attack (MiTM), where the attacker places himself or herself in between the victim and the router. See Fig. 1. Using the ARP spoofing technique, the attacker tricks the victim into thinking that the requests are coming from the router. Doing so causes the victim machine to think that the attacker's machine is the real router. As a result, all the victim's network traffic is being sent to the attacker. The attacker may choose to passively observe the packets or actively manipulate the packets before sending the traffic onto the real router.

While a Windows power-user may use commands such as arp –a to detect an irregularity, it is often hard for a normal user to realize that he or she is a victim of a MiTM attack. See Fig. 2 for a screenshot of an attack in action.

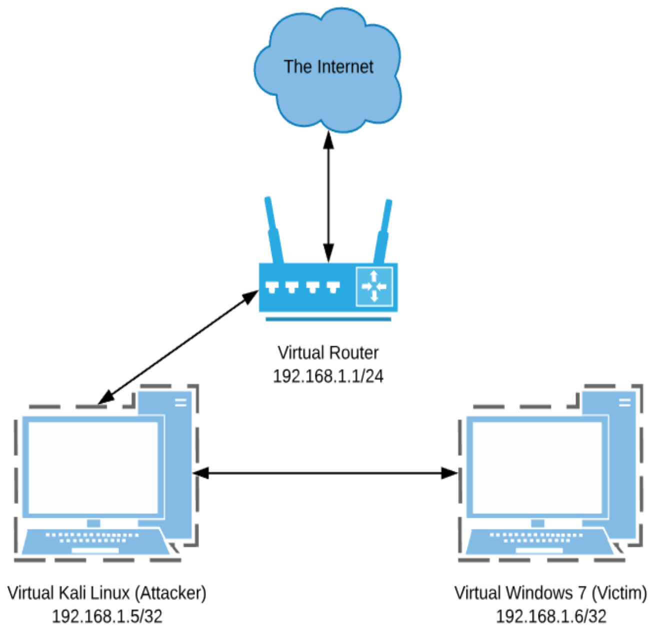


Fig. 1. A Diagram Depicting a Man-in-the-Middle Attack.

A. Test Environment

To understand how the cookie works and to test how a session can be hijacked, virtualization was used to create a lab to simulate an attack. The software used are Oracle VM VirtualBox, Microsoft Windows 7, and Kali Linux. Kali Linux was chosen because it is designed to support educational and ethical hacking, and it comes with the necessary software to simulate the attack. pfSsense is an operating system for routers, it is open source, and it performs Dynamic Host Control Protocol (DHCP) which made it suitable. A virtualized instance of pfSense was used for network communications and it acted as both the virtual router and firewall. The virtualized routing allowed the network traffic to be analyzed; thus, providing insight to each step of the attack. See Fig. 3 for a representation of the testing environment.

B. Environment Installation Notes

The installation of VirtualBox was comparable to the installation of programs on a computer. During the installation, VirtualBox installed a separate network adapter to facilitate communications between the host operating systems and the guest operating systems. Guest operating systems were installed next. Although the nature of Windows and Linux were varied, the process of installation was uncomplicated. Each virtual machine used the default settings and was set to allow only internal communication. Upon completion of the installation and configurations, the session hijack was ready to be tested.

C. Lab Proceedings

The lab assumed both the Windows 7 victim and the attacker were on the same network. The attacker shall exploit the vulnerability in ARP to pretend to be the router. This will cause the victim to believe the attacker and send all network traffic to the attacker's machine. Next, the lab simulated an attacker using a packet capturing and processing tool to isolate the session information from a cookie of a victim that was already authorized, i.e., signed on. Finally, the attacker shall use the cookie information to exploit an active session. The steps of intercepting the cookies were as follows: identifying the target, conducting ARP spoofing, analyzing the packets, and hijacking the session (or impersonating the victim) [6]. See Fig. 4.

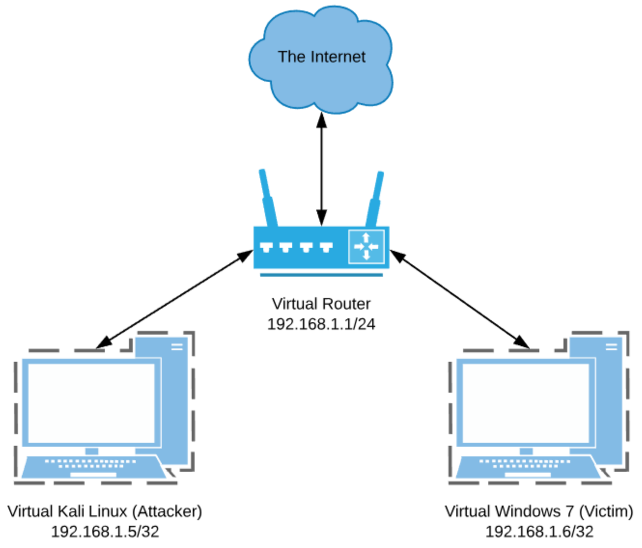
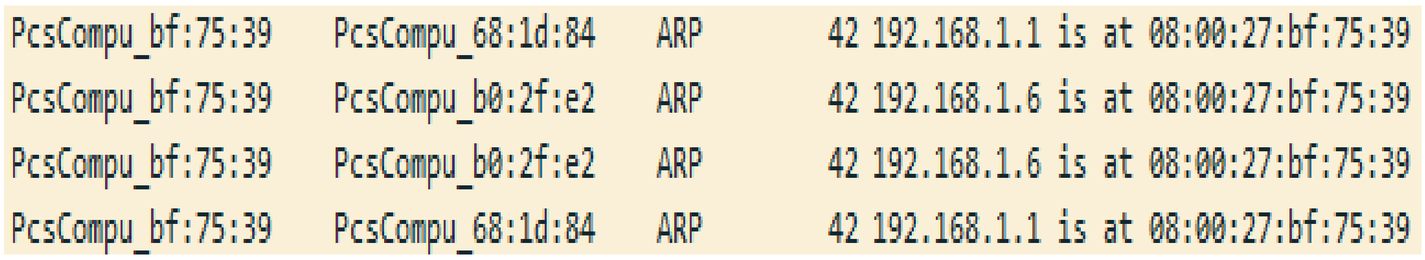


Fig. 2. Network Diagram of the Simulated Environment.



```
PcsCompu_bf:75:39    PcsCompu_68:1d:84    ARP        42 192.168.1.1 is at 08:00:27:bf:75:39
PcsCompu_bf:75:39    PcsCompu_b0:2f:e2    ARP        42 192.168.1.6 is at 08:00:27:bf:75:39
PcsCompu_bf:75:39    PcsCompu_b0:2f:e2    ARP        42 192.168.1.6 is at 08:00:27:bf:75:39
PcsCompu_bf:75:39    PcsCompu_68:1d:84    ARP        42 192.168.1.1 is at 08:00:27:bf:75:39
```

Fig. 3. ARP Spoofing in Progress (Captured from Wireshark).

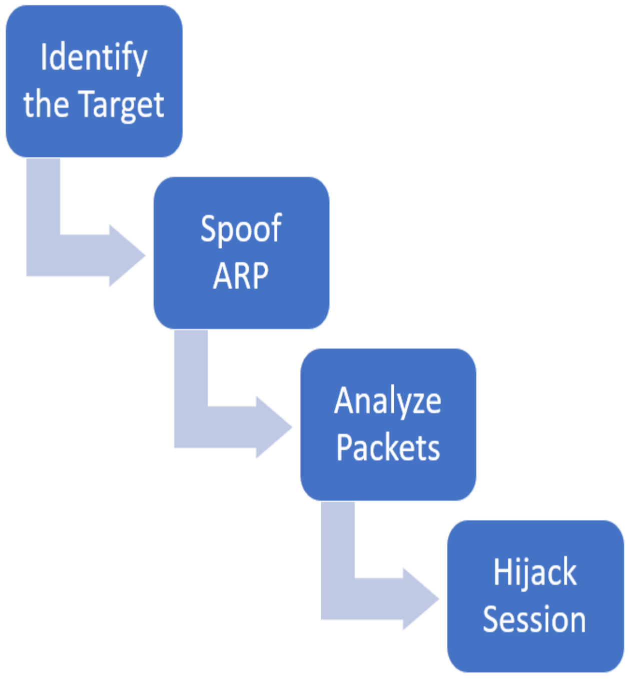


Fig. 4. Steps in an Attack Process.

III. Lab Methodology

The lab assumed both the Windows 7 victim and the attacker were on the same network. The attacker shall exploit the vulnerability in ARP to pretend to be the router and trick the victim into sending all the traffic to the attacker [6]. The attacker must then employ a packet capturing and processing tool to extract the session information from a cookie of a victim that was already authorized (or logged on). Finally, the attacker shall use the cookie information to exploit an active session. The steps of intercepting the cookies were as follows: identifying the target, conducting ARP spoofing, analyzing the packets, and hijacking the session (or impersonating the victim) [6]. See Fig. 4.

A. *Identify the Target*

In this step, the attacker identifies a target. The lab assumed that the network was flooded with false ARP requests by the attacker. The victim assumed the IP address of 192.168.1.6.

B. *Spoof ARP*

Kali Linux comes with two methods to perform ARP spoofing. One way is to use the arpspoof command from a terminal window of DNSUtils [6]. Before using this method, install the DNSUtils package by running the command sudo apt-get install dnsutils [6]. The next method uses a Graphical User Interface (GUI) application known as Ettercap. Ettercap is used for scanning for targets and for initiating the spoofing of ARP addresses [14]. See Fig. 5.

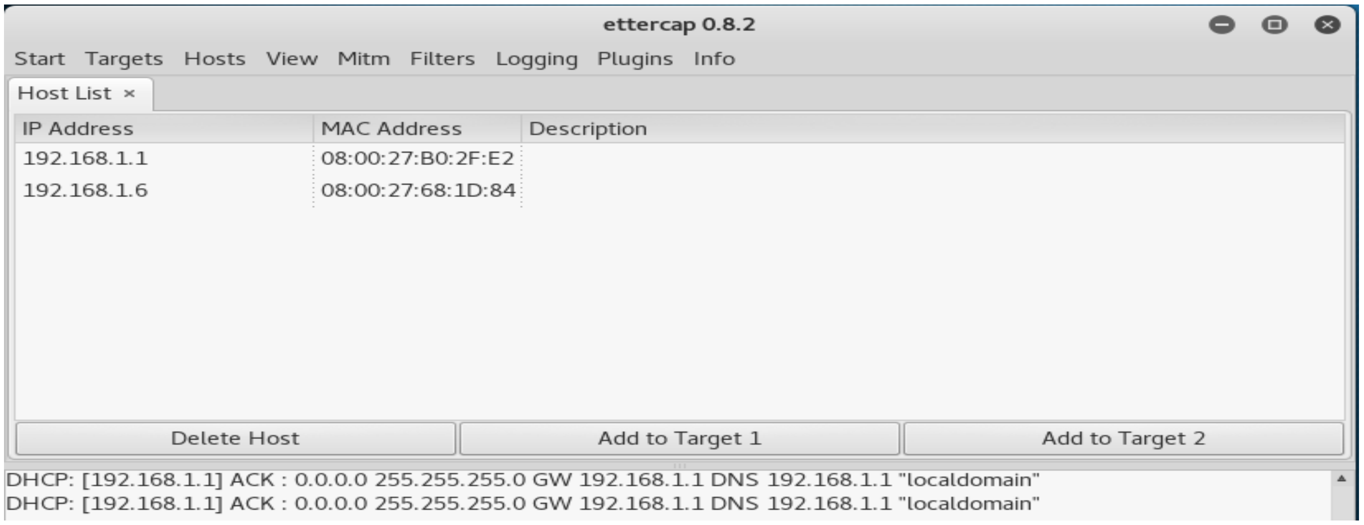
C. *Analyze Packets*

Wireshark and tcpdump are known as protocol analyzers and are readily available on the web. The tools are normally used for debugging network issues by capturing network traffic from a NIC and inspecting the packets. tcpdump is suitable in the absence of a GUI or Wireshark. Using the pcap file format, tcpdump files are readable by Wireshark. In the lab, Wireshark was used. The NIC was placed in promiscuous mode by Wireshark because the attacker wants to capture all traffic, including those not meant for that computer. Fig. 6 shows a screenshot of an intercepted cookie.

D. *Hijack Session*

In this step, the attacker had intercepted the right cookie and made it possible to replay the information in the cookie. Consequently, it is not necessary for the attacker to know the credentials to gain authorized access because the cookie already contains the information of an authenticated session.

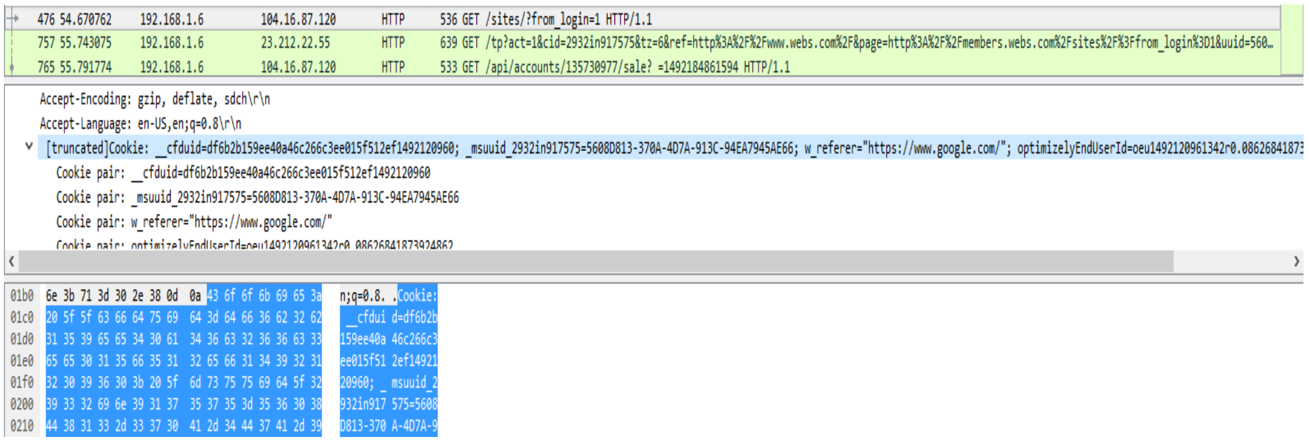
Once the target had been identified, the lab test was successful in showing a session can be hijacked by (1) tricking the target machine into thinking the attacker is the real router using ARP spoofing and then sending all the network traffic to the attacker's machine, (2) analyzing all the traffic packets to find a cookie with valid session information, and (3) replay the cookie to gain authorized access to server resources [6].



IP Address	MAC Address	Description
192.168.1.1	08:00:27:B0:2F:E2	
192.168.1.6	08:00:27:68:1D:84	

```
DHCP: [192.168.1.1] ACK : 0.0.0.0 255.255.255.0 GW 192.168.1.1 DNS 192.168.1.1 "localdomain"
DHCP: [192.168.1.1] ACK : 0.0.0.0 255.255.255.0 GW 192.168.1.1 DNS 192.168.1.1 "localdomain"
```

Fig. 5. Ettercap Scan Results.



```
476 54.670762   192.168.1.6   104.16.87.120   HTTP   536 GET /sites/?from_login=1 HTTP/1.1
757 55.743075   192.168.1.6   23.212.22.55    HTTP   639 GET /tp?act=1&cid=2932in917575&tz=6&ref=http%3A%2F%2Fwww.webs.com%2F&page=http%3A%2F%2Fmembers.webs.com%2Fsites%2F%3Ffrom_login%3D1&uuid=560…
765 55.791774   192.168.1.6   104.16.87.120   HTTP   533 GET /api/accounts/135730977/sale? =1492184861594 HTTP/1.1

    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8\r\n
  ˅ [truncated]Cookie: __cfduid=df6b2b159ee40a46c266c3ee015f512ef1492120960; _msuuid_2932in917575=5608D813-370A-4D7A-913C-94EA7945AE66; w_referer="https://www.google.com/"; optimizelyEndUserId=oeu1492120961342r0.08626841873
      Cookie pair: __cfduid=df6b2b159ee40a46c266c3ee015f512ef1492120960
      Cookie pair: _msuuid_2932in917575=5608D813-370A-4D7A-913C-94EA7945AE66
      Cookie pair: w_referer="https://www.google.com/"
      Cookie pair: optimizelyEndUserId=oeu1492120961342r0.08626841873924862

01b0  6e 3b 71 3d 30 2e 38 0d  0a 43 6f 6f 6b 69 65 3a   n;q=0.8. .Cookie:
01c0  20 5f 5f 63 66 64 75 69  64 3d 64 66 36 62 32 62    __cfdui d=df6b2b
01d0  31 35 39 65 65 34 30 61  34 36 63 32 36 36 63 33   159ee40a 46c266c3
01e0  65 65 30 31 35 66 35 31  32 65 66 31 34 39 32 31   ee015f51 2ef14921
01f0  32 30 39 36 30 3b 20 5f  6d 73 75 75 69 64 5f 32   20960; _ msuuid_2
0200  39 33 32 69 6e 39 31 37  35 37 35 3d 35 36 30 38   932in917 575=5608
0210  44 38 31 33 2d 33 37 30  41 2d 34 44 37 41 2d 39   D813-370 A-4D7A-9
```

Fig. 6. Observing an Intercepted Cookie in Wireshark.

IV. Security of Cookies and Sessions

By design, the cookie lacks security features. The cookie itself is not confidential and does not ensure integrity. Calzavara et al. correspondingly said that confidentiality and integrity are two standard security properties of web sessions and are typically targeted for attacks [2]. A cookie is not confidential because (1) it is accessible and modifiable by all the ports from the same server, (2) when a cookie is available to HTTP and HTTPS schemes, it is also accessible by FTP and Gopher schemes, and (3) when a resource can be retrieved from one path, it can also access stored cookies from another path [1][6]. Integrity cannot be maintained across subdomains because cookies that are set in one subdomain are indistinguishable from cookies set by another subdomain and this allows one subdomain to overwrite the cookies of another subdomain [1][6]. For example, subdomain A can use the cookies of subdomain B to initiate an attack against subdomain B [1][6]. Furthermore, as the lab demonstrated, the cookie could be stolen easily.

A. Improving the Security of the Cookie

The discussions surrounding the security of cookies is twofold. One thread centers on improving the security of the cookie and another focuses on preventing the cookies from being stolen.

In an effect to improve the security of cookies, the Secure and HttpOnly attributes were added. The use of the Secure attribute limits the cookie to secure channels only. Setting this attribute tells the client to attach the cookie only when the request is made over Secure Socket Layer/Transport Layer Security (SSL/TLS). However, a request could still go through if an attacker sends the request from a secured site.

The use of The HttpOnly attribute limits the cookie to HTTP request only, i.e., the client will attach the cookie only if the request is an HTTP request [1]. The attribute was introduced in 2002 to prevent the use of content injection attacks to steal authentication cookies [2]. Cross-site scripting (XSS) attack is the usual form of content injection attack which exploits a server-side vulnerability that allows an attacker to trick a user into disclosing sensitive information that is normally reserved for a legitimate website [7]. Setting the attribute limits the scope of cookies to Hypertext Transport Protocol Secure (HTTPS) requests, and it helps to prevent both the JavaScript from accessing the client-side cookie and the cookie from accessing non-HTTP APIs [1][3]. However, used on its own, it does not completely mitigate the dangers of an XSS attack and only protects against the theft of authentication cookies [7].

Another defense against XSS is Content Security Policy (CSP) which is a web security policy from the W3C to allow developers to specify the sources from which the browser is allowed to request for the resources embedded in a webpage [2]. Calzavara et al. say CSP is effective against XSS when properly configured; however, several challenges persist [2]. For one, it does not prevent general content injection attacks [2]. For another, a successful policy for legacy applications is time-consuming to deploy due to the manual whitelisting of inline scripts and styles and the careful identification of trusted origins [2]. Finally, the current implementation of CSP is neither significant nor effective [2].

When a cookie is used for authentication, the client will always attach the cookie when it requests for resources from the Web server. The lab demonstrated Cross-site Request Forgery (CSRF) attack can occur when a stolen authentication cookie is used to access authorized resources. A bad actor could use this exploit to make the server carry out malicious actions.

Instead of storing the authentication information in a cookie, Barth recommends utilizing the URL as part of the authorization process [1]. Goodwin and West go further to define a SameSite attribute as part of their proposal to the Internet Engineering Task Force (IETF) [4]. By limiting the scope of the cookie to the originating site, they say that it is possible to mitigate a CSRF attack [4]. At the point of submission, the attribute was implemented only in Google Chrome. As with the Secure and HttpOnly attributes, the rate of adoption from the different browser varies.

The ARP is a proven protocol in Local Area Networks (LANs). While there is no foreseeable plan to replace the protocol, some routers and operating systems allow for static ARPs to prevent a machine from learning a new MAC address other than that which was set [11]. Again, the rate of adoption depends on the proprietors of the routers and such a feature is likely not offered to consumer networking devices.

Recently, browser-based defenses have promoted as a helpful mechanism to protect web applications against session hijacking [3]. It does so by automatically detecting cookies that contain session information using client-side heuristics and then protecting the information against theft and unintended use [3]. The reality as Calzavara et al. found was that simple heuristics are limited in effectiveness because client authentication is based on complex and unpredictable usage of authentication cookies [3].

HTTP Strict Transport Security (HSTS) is a browser security policy that forces an upgrade of HTTP communications to protected-domains to HTTPS to prevent a MiTM from eavesdropping on unencrypted traffic [3]. Regarding HSTS, Calzavara et al. recommend the adoption of the Secure flag in addition to HSTS to prevent attackers from taking advantage of subdomain accesses to cause a leakage of the authentication cookies over HTTP [3]. However, there are several challenges with HSTS: (1) deploying HSTS requires careful analysis, and sometimes reorganization, of a site, (2) the adoption rate of HSTS is low; in 2014, only 3,406 sites out of about 150,000 popular sites had deployed HSTS, (3) HSTS may be vulnerable to SSL downgrading attacks, and (4) HSTS is often misconfigured [9].

B. Tools for Evaluating the Security of Cookies

Mundada et al. say that web developers need better handles to evaluate the security of the authentication cookies [9]. To this end, Mundada and team developed Newton: a tool and a Chrome extension for discovering all authentication cookies that allow a user to access the respective sub-services corresponding subordinate service of any site and identifying authentication cookies vulnerabilities [9]. In their analysis of 149 popular websites, 65 were found with security vulnerabilities [9]. Out of those, many have acknowledged and fixed the problem [9]. Mandada et al. believed that the tool could be widely accepted by developers, testers, administrators, to even savvy users [9].

Porat, Tikochinski, & Stulman is another team that developed a tool to check for authorization vulnerabilities on websites [10]. Authorization Enforcement Detection (AED) allows the administrator to surf the website normally, while the AED Proxy intercepts every request and forward it to the Web server, saving corresponding request/response pairs for analysis when a cookie is detected [10]. The result of the analysis is a categorization of authorization as safe, suspicious, or breach [10].

V. Conclusion

Essentially, the HTTP is stateless. Every request is independent, and the protocol does not remember any past request. The cookie was introduced as a mechanism that helps Web servers and applications remember where they left off. It is simply a small data file that servers and clients can pass back and forth, and it has no security features. Web sessions use cookies to establish semi-permanent exchanges between servers and clients involving multiple requests and responses. When a client is authenticated, a cookie that contains session information is passed back and forth. An ARP spoofing attack can hijack this cookie. A virtualized lab using pfSense, Kali Linux, and Windows 7 simulated the attack successfully. It demonstrated that the cookie could be stolen, and authenticated access is possible with the stolen cookie. The positive test result prompted the questions of how can the security of the cookie be improved and how can its theft be prevented? Cookie's attributes such as Secure, HttpOnly, and Samsite, were discussed. Also discussed were countermeasures such as CSP, HSTS, and client-side heuristics against common attacks like XSS and CSRF. Overall, there is not one solution that will solve all the issues. Although each solution improves the security, it comes with its own challenges. Therefore, researchers have recently developed tools such as Newton and AED to identify and evaluate cookies vulnerabilities. Following this research, the next logical step is to consolidate all the known countermeasures and establish a set of best practices for securing cookies.

References

[1] A. Barth, "HTTP State Management Mechanism," 2011.

[2] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta, "Surviving the Web: A Journey into Web Session Security," in *Companion of The Web Conference 2018 on The Web Conference 2018 - WWW '18*, Lyon, France, 2018, pp. 451–455.

[3] S. Calzavara, G. Tolomei, A. Casini, M. Bugliesi, and S. Orlando, "A Supervised Learning Approach to Protect Client Authentication on the Web," *ACM Transactions on the Web*, vol. 9, no. 3, pp. 1–30, Jun. 2015.

[4] M. Goodwin and M. West, "Same-Site Cookies," 20-Jun-2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site-00. [Accessed: 22-Apr-2017].

[5] D. M. Kristol, "HTTP Cookies: Standards, Privacy, and Politics," *arXiv:cs/0105018*, May 2001.

[6] Y. L. Loo and K. LaCroix, "Cookies and Sessions: A Study of What They Are, How They Work, and How They Can Be Stolen." 24-Apr-2017.

[7] Microsoft, "Mitigating Cross-site Scripting With HTTP-only Cookies." [Online]. Available: https://msdn.microsoft.com/en-us/library/ms533046.aspx. [Accessed: 11-Oct-2018].

[8] L. Montulli, "The reasoning behind Web Cookies," 14-May-2013.

[9] Y. Mundada, N. Feamster, and B. Krishnamurthy, "Half-Baked Cookies: Hardening Cookie-Based Authentication for the Modern Web," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security - ASIA CCS '16*, Xi'an, China, 2016, pp. 675–685.

[10] E. Porat, S. Tikochinski, and A. Stulman, "Authorization Enforcement Detection," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies - SACMAT '17 Abstracts*, Indianapolis, Indiana, USA, 2017, pp. 179–182.

[11] J. Singh, S. Dhariwal, and R. Kumar, "A Detailed Survey of ARP Poisoning Detection and Mitigation Techniques," *International Journal of Control Theory and Applications*, vol. 9, Feb. 2017.

[12] J. Singh and V. Grewal, "A Survey of Different Strategies to Pacify ARP Poisoning Attacks in Wireless Networks," *International Journal of Computer Applications*, vol. 116, pp. 25–28, Apr. 2015.

[13] M. Waliullah, A. B. M. Moniruzzaman, and M. Rahman, "An Experimental Study Analysis of Security Attacks at IEEE 802.11 Wireless Local Area Network," *International Journal of Future Generation Communication and Networking*, vol. 8, pp. 9–18, Feb. 2015.

[14] A. Yacchirena, D. Alulema, D. Aguilar, D. Morocho, F. Encalada, and E. Granizo, "Analysis of attack and protection systems in Wi-Fi wireless networks under the Linux operating system," in *2016 IEEE International Conference on Automatica (ICA-ACCA)*, 2016, pp. 1–7.