Why Aren't HTTP-only Cookies More Widely Deployed?

Yuchen Zhou
University of Virginia
yz8ra@virginia.edu

David Evans
University of Virginia
evans@virginia.edu

***Abstract*—HTTP-only cookies were introduced eight years ago as a simple way to prevent cookie-stealing through cross-site scripting attacks. Adopting HTTP-only cookies seems to be an easy task with no significant costs or drawbacks, but many major websites still do not use HTTP-only cookies. This paper reports on a survey of HTTP-only cookie use in popular websites, and considers reasons why HTTP-only cookies are not yet more widely deployed.**

***Index Terms*—HTTP-only, cookies, web application security, adoption of security technologies**

I. Introduction

HTTP cookies are used as authentication tokens by nearly all websites that require user credentials. Cookies provide a way for websites to manage persistent state within the stateless HTTP protocol since browsers send cookies back to the server as part of the HTTP request header.

An attacker who acquires a copy of an authentication cookie may be able to use it to impersonate the user and conduct a session on their behalf. The most common way to steal a cookie is through a cross-site scripting (XSS) attack. Since the injected script runs in the security context of the host site, it can access the cookie through the DOM (by reading document.cookie) and can transmit its value to a server controlled by the attacker (for example, by embedding the cookie value in the URL of an image). XSS attacks remain one of the largest security problems on the web [7] and major sites continue to be vulnerable to them. A recent serious attack against apache.org used a XSS attack to steal authentication cookies [2]. XSS attacks also emerged at the iPhone version of Facebook in January 2010 [12]).

The idea of HTTP-only cookies is to denote cookies that are not visible through the DOM. HTTP-only cookies were introduced by Microsoft in 2002 and first implemented in Internet Explorer 6 [13]. To mark a cookie as HTTP-only, the *HTTP-only* attribute [15] is added to the cookie field. A browser that supports HTTP-only cookies sends the cookies back in HTTP request headers, but will not permit a script to access an HTTP-only cookie. However, as we discuss in Section 3, HTTP-only cookies do not prevent all XSS exploits and using them is not a substitute for eliminating XSS vulnerabilities, but judicious use of HTTP-only cookies does make certain exploits impossible.

Despite their apparent simplicity, HTTP-only cookies require both clients and sites to change like many proposed web security enhancements. This makes adoption a difficult challenge, and it has taken many years for HTTP-only to reach critical mass. Some key events of the deployment of HTTP-only are shown in Figure 1. It took more than five years for Firefox to add HTTP-only support, and some popular web frameworks still do not support it today (see Section III).

In this paper, we evaluate the effectiveness of HTTP-only cookies (Section II) and report on a survey of the use of HTTP-only cookies by popular sites and web application frameworks (Section III). We discuss the possible reasons why HTTP-cookies are still not universally adopted more than eight years since they were introduced (Section IV). We are not able to reach a definitive answer why HTTP-only cookies are not more widely deployed, but we hope the experience with HTTP-only cookies can provide insight into what make proposed security technologies succeed or fail, and how to overcome some of the hurdles to widespread adoption.

II. Effectiveness of HTTP-only cookies

HTTP-only cookies are a defense-in-depth strategy. They do not eliminate XSS vulnerabilities, but make it more difficult to exploit them by preventing cookie-stealing. Here, we consider how valuable this mitigation is in practice, and how well the HTTP-only defense satisfies its goals.

A. Other ways to exploit cross-site scripting

There are many ways to exploit cross-site scripting attacks other than stealing cookies. An attacker may be able to inject a script to capture and transmit the user's keystrokes, or use an injected script to issue requests directly using the user's credentials. These attacks do not necessarily need to steal cookies, but instead just take advantage of the fact that authenticated users have cookies stored in their browser that are automatically appended to HTTP request headers. Nevertheless, cookie stealing enables certain exploits that benefit from persistent access to the user's credentials. In addition, cookies may contain private information that could not be exposed otherwise. Various defenses such as tying cookies to IP addresses, setting the expiration time for a cookie, and encrypting private content in cookies may mitigate some of these threats. However, studies indicate that as many as 3% of users change their IP address during a session [8], which makes IP address checking inappropriate.

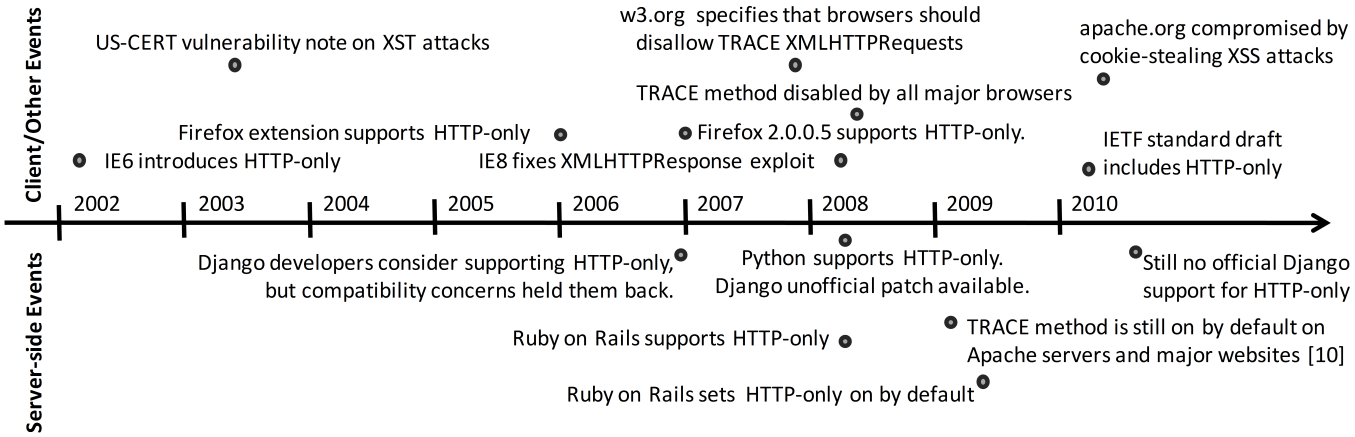

Fig. 1. HTTP-Only Deployment Timeline

B. Other Ways to Steal Cookies

HTTP-only cookies only prevent cookies from being accessed through the DOM in scripting-related attacks, but provide no defense against other ways of stealing cookies. Another way to steal cookies is to eavesdrop on network traffic. Eavesdropping is mitigated by cryptographic measures, most notably TLS/SSL. Cookies that have a *secure* attribute should only be sent over HTTPS [9]. This policy is enforced by all current major browsers. An attacker who has more access to a victim's machine may also be able to steal cookies by accessing local files. As long as the browser stores cookies in a sensible way, however, an attacker who is able to do this probably has enough access to the victim's machine to also install a keylogger and acquire the login credentials directly.

C. Vulnerabilities with HTTP-only Cookies

Even if HTTP-only cookies are used and an attacker needs to use an injected script to exfiltrate the cookies, other techniques such as cross-site tracing and using XMLHttpRequests may enable an attacker to steal HTTP-only cookies.

Cross-site tracing. The HTTP TRACE request is a method designed for debugging problems such as network connection errors between servers. When a client sends a TRACE request to a compliant server, the server responds by echoing back the header sent by the client. An attacker can exploit this to circumvent HTTP-only cookies by injecting code that sends an asynchronous XMLHttpRequest with the TRACE method and receiving the HTTP-only cookie in the message echoed-back by the server [6]. This vulnerability was reported in a US-CERT vulnerability note [11], recommending that web servers reject TRACE requests. The latest version of Apache (Version 2.2) still enables the TRACE method by default. However, recent browsers, including Firefox 3.6 and IE 8, no longer allow XMLHttpRequests that use the TRACE method. Section III reports results from our survey showing that many servers still respond to TRACE requests.

XMLHttpRequest related attacks. Another way to circumvent HTTP-only restrictions is to exploit the fact that HTTP-only cookies may still be included in responses to non-TRACE XMLHttpRequests. Some web applications set an authentication cookie on every response even if the previous request already included a cookie. For such sites, simply sending a XMLHttpRequest to the site and collecting the response (which contains an authentication cookie) works just as well as cross-site tracing attacks [19].

Browsers can thwart this attack by blocking set-cookie headers in the responses. This could also be mitigated at the server side by not resending authentication cookies to requests that already include cookies. All major recent browsers (including IE 8[1], Firefox 3.6, Opera 10.10, Google Chrome 4.1 and Safari 4.0.5) strip cookies from responses to XMLHttpRequests, so are not vulnerable to this attack. Users with older browsers, however, may still be vulnerable to these attacks if servers return authentication cookies in response to incoming HTTP requests that already contain authentication cookies.

III. Survey of HTTP-Only Cookie Use

We conducted a survey of popular web sites to determine how widely HTTP-only cookies are deployed and whether or not sites using HTTP-only cookies are vulnerable to the attacks described in the previous section.

Sample. Our survey sample is based on the global top 100 sites according to Alexa.com [1]. We excluded 29 sites that do not use authentication cookies (e.g., www.bbc.co.uk), 6 adult sites, and 15 sites that substantially duplicate previous sites (e.g., www.google.de was excluded in favor of www.google.com). This left the set of 50 sites shown in Table I.

Methodology. To conduct the survey, we wrote a simple Firefox extension that searches for the HTTP-only string in set-cookie header fields. We used an automated script to load each test URL and determine if the site issues HTTP-only cookies. We wrote an extension instead of using network packet sniffer such as Wireshark because many of the sites use SSL, which

[1]Unlike other browsers, IE 8 prevents accesses to HTTP-only cookies, but not to non-HTTP-only cookies. Firefox and all other browsers prevent accesses to all cookies, regardless of HTTP-only field.

would prevent us for observing the plaintext headers. By using a Firefox extension, we can see the decrypted headers.

We could automate testing for HTTP-only cookies before login, but to see if sites used HTTP-only for authentication cookies we needed to perform sessions as authenticated users. We did not find a satisfactory way to automate this, so we manually registered for all sites, logged in, and tested if they are using HTTP-only cookies.[2]

Results. The results of our survey are summarized in Table I. Among the 50 major sites that we tested, 8 of them use HTTP-only cookies before the user logs in, and 16 more sites issue HTTP-only cookies after the user logs in. A slight majority (26) of the sites do not use HTTP-only cookies at all. This includes many very popular and commercial sites such as amazon.com, twitter.com, and mail.yahoo.com. It is also surprising that some sites (e.g., ebay.com, yahoo.com) do use HTTP-only cookies, but not for authentication cookies.

Nearly eight years after its introduction, this simple security option is still not deployed by the majority of websites. We speculate on reasons for this in Section IV.

Vulnerabilities. As we discussed earlier, there are potential vulnerabilities in HTTP-only cookies which we examine here. As mentioned previously, both threats are mitigated by most modern browsers but users using older browsers may still be vulnerable.

First, we checked if these sites correctly reject TRACE requests. Our expected return value should be 403 Forbidden or 405 Method not Allowed. As we can see from Table I, 7 out of the 50 sites respond with either 200 OK or 302 Found to TRACE requests. These sites would be vulnerable to cross-site tracing attacks for users whose browsers do not block these requests.

We also checked all the sample sites to see if these sites return a set-cookie header in the response to an XMLHttpRequest, and what is the browser behavior to this. None of the 24 sites that use HTTP-only authentication cookies returned cookies in responses to requests that already contained authentication cookies, so they were not vulnerable to this problem. Of the other 26 sites, there are five sites that return authentication tokens in response to these requests. Since these sites are not currently using HTTP-only, they are already vulnerable to cookie-stealing XSS attacks. However, even if they mark their cookies at HTTP-only these sites would still be vulnerable to the XMLHttpRequest attack.

Web Frameworks. Given the need to manually create and account and login to a site to test if its authentication cookies use HTTP-only, it was not feasible to perform a large scale survey of less popular websites. Instead, we observe that many websites today are built using web frameworks, and it is likely that most sites built with frameworks will use their default cookie and authentication options. The adoption path for web frameworks also illustrates some of the challenges in getting new security technologies widely adopted.

[2]We attempted to use bugmenot [5] to automate logins, but found that most sites blocked accounts known to be from bugmenot.

On by Default	vBulletin 4.0, 22 Feb 2010 Ruby on Rails 2.3.2 (Mar 2009)
Supported but off by default	Symfony 1.4 (Feb 2010) Ruby on Rails 2.2.2 (Nov 2008)
Not Supported	Django 1.1.1 (July 2009) Spring Framework 3.0 (Feb 2010) Pylons 0.9.7 (2008) Ruby on Rails 2.1.2 (Oct 2008)

TABLE II
WEB FRAMEWORK HTTP-ONLY SUPPORT AND DEPLOYMENT

Table II summarizes support for HTTP-only cookies in several popular web frameworks. The results are collected by either using documentation or replies from the web framework's development team, or by directly examining the cookie implementation source code in cases where a definitive answer was not found. The latest version of Ruby on Rails is the only open source framework we found that set HTTP-only flag on by default. The previous version of Rails and the current version of Symfony) both support HTTP-only but it needs to be manually configured by users Another framework that appears to set HTTP-only on by default is vBulletin, a close-source commercial product that is used by many Internet forums.[3] The other frameworks we examined do not support HTTP-only. Pylons does not provide support for authenticating users but instead relies on third party authentication add-ons like Authkit and Repoze.who (neither of which support HTTP-only). Django includes support for authentication which does not support HTTP-only, but there is an unofficial patch available for it which we discuss further in Section IV.

IV. DISCUSSION

Despite their apparent security benefits, simplicity, and lack of obvious drawbacks, our survey shows that HTTP-only cookies are, after eight years, still only deployed on about half of major websites and not used by default on most web frameworks. We sent email requests to all the sites that were not using HTTP-only cookies, but have not received any illuminating replies. Here, we consider several possible reasons why HTTP-only cookies may not be used.

Protection effectiveness. As discussed in Section II, HTTP-only cookies are a defense-in-depth mechanism intended to thwart a particular way of exploiting a XSS vulnerability. However, XSS attacks are not targeted merely for cookie stealing. It is also possible that a site considers the other ways of exploiting a XSS vulnerability to be just as severe as the cookie-stealing prevented by HTTP-only cookies. This scenario makes it plausible for developers to focus more on XSS defenses than merely HTTP-only cookie mechanism adoption.

In addition to this, sites may elect not to employ HTTP-only cookies because they believe their site is not vulnerable to XSS exploits. However, we find this reason hard to support

[3]Our results for vBulletin are based on testing a sample of websites built using it. Sites built with vBulletin 4.0 consistently use HTTP-only cookies

Type	No.	Sites
No HTTP-Only Authentication Cookies	26	4shared.com[1] adobe.com[2a] amazon.com[2] baidu.com, cnet.com[2a] cnn.com[2] craigslist.org, dailymotion.com, digg.com[2] ebay.com[2] espn.go.com[2] flickr.com[2a] hotfile.com, imageshack.us[2] imdb.com[1] kaixin001.com[1a] linkedin.com[2] livejournal.com, mail.yahoo.com, mail.qq.com, mediafire.com[2] megaupload.com[a] megavideo.com[a] msn.com[2] nytimes.com[1] orkut.com, renren.com[2] soso.com, thepriatebay.org[2] tudou.com, twitter.com[1] yahoo.com[2] youku.com[2]
Use HTTP-only	24	aol.com[2] blogger.com, conduit.com[2] doubleclick.com, google.com, facebook.com, hi5.com[2] live.com, mail.163.com, mail.sina.com.cn, myspace.com[2] photobucket.com[2] sohu.com[2] taobao.com[a] wikipedia.org[a] wordpress.com[2] youtube.com[2]

TABLE I
SURVEY RESULTS

[1] Sites that respond with authentication cookies to every HTTP request (potentially vulnerable to XMLHttpRequest attack).
[2] Sites that respond with non-authentication cookies to every HTTP request.
[a] Sites that respond with 200/302 to HTTP TRACE requests.

as several of the sites in our survey that do not use HTTP-only cookies have been vulnerable to XSS attacks (e.g., twitter.com [18] and nytimes.com [14]).

Page functionality. Making cookies HTTP-only can disrupt web page functionality by preventing scripts from reading the value of cookies. There is little, if any, reason why a well designed website would include scripts that access state from authentication cookies directly, but it may be difficult to remove all uses from legacy code.

We did a experiment to test whether making cookies HTTP-only would break any of the survey sites. Since we are not able to modify the servers, we developed a Firefox extension to append the HTTP-only attribute at the client side.

Even though to do a complete dynamic test on any one website is impossible, we distributed our extension to a small group of users for experiments and no users reported encountering any problems. We also explicitly tested the 26 sample sites that did not use HTTP-only. Only one site, the Chinese social networking site renren.com, experienced any problems. This site includes an online chatting system that rejects all user logins when our extension is running (other functions on the site continue to work normally). We examined their script [17] and found that is does retrieve values from authentication cookies using code like `uin=this.getCookie("userid");`.

Although our extension was written primarily to conduct this experiment, the results indicate that it could be used to provided added protection for clients who visit websites that do not use HTTP-only cookies.

Software stack compatibility. As a result of our survey results, we asked the Django development team why they had not added support for HTTP-only cookies. They have pointed out a thread that offered some explanation [3]. Python version (2.5) does not support HTTP-only cookies, although support was added to Python v2.6 (released October 2008). Since the Django framework is based on Python, it uses Python web authentication module directly instead of its own cookie handling mechanism. As Django does not want to lose backward compatibility (servers running 2.5 or earlier version of Python would encounter error when `cookie.HTTP-only = true` is called in the server side script), they are unwilling to support HTTP-only.

Standards compliance. Another reason frameworks and sites may not use HTTP-only cookies is HTTP-only is not part of the cookie specification [9]. Using HTTP-only cookies might break existing web clients or other web infrastructure. This problem could have been avoided if instead of introducing a new attribute, HTTP-only was implemented by just defining a new property (e.g., `HTTPonly = True;`). This would be consistent with the cookie specification. For browsers that do not support HTTP-only this would not introduce errors, but browsers that support this property could interpret the HTTPonly property as the HTTP-only attribute is interpreted. Django developers also explicitly pointed out that this is one minor reason that they hesitated to implement HTTP-only, and it may also explain why other frameworks do not turn it on by default. As far as we know HTTP-only has finally made it to the newest IETF cookie protocol draft [4] by Adam Barth at Mar of 2010. By now we could only hope that this draft get accepted at this moment, but still cannot be too optimistic about the deployment of the protocol.

Difficulty to deploy client-server solutions. Finally, we observe that the challenge of deploying any mechanism that involves changes to both clients and servers is not unique to HTTP-only, but a longstanding issue in network evolution. Any mechanism that provides benefits only with both endpoints in a network adopt it faces major hurdles in deployment. A similar (but even less successful) deployment experience applies to the Set-Cookie2 header which first appeared in RFC2965 [10]. It was introduced ten years ago with the goal of adding more security features to the previous Set-Cookie header (e.g., each cookie specifies a port and browsers should return cookies only if they match the corresponding port). Browsers are also required to include information on how the cookie was set with the returned cookie. Unlike HTTP-only, Set-Cookie2 has not been adopted by any major browsers, and very few servers use it.

Many network protocols such as TLS involve a handshake protocol where the endpoints negotiate the highest level protocol they have in common. This allows new versions to be deployed while maintaining backwards compatibility, but reduces the incentive for endpoints to update their implementations. Various efforts have been made to try to support more dynamic deployment of network protocols including STP [16]. This

requires mechanisms for running untrusted code on network hosts to automatically implement a new protocol, so dynamic protocol updating raises additional security concerns.

V. Conclusion

HTTP-only is a mechanism that was introduced eight years ago and appears to be very simple to implement and deploy, but has not yet been widely deployed. Some of this may be due to questions about the real value of HTTP-only in thwarting XSS exploits, but it is also clear that HTTP-only does thwart some exploits and it has very little cost. This example illustrates the difficulty in deploying even simple security mechanisms that require cooperation between servers and clients, as well as considering interactions with other web functionalities (especially XMLHttpRequest). The value of the mechanisms is only realized when both servers and clients implement it correctly, so it takes a long time to achieve enough critical mass to achieve widespread deployment.

Acknowledgements

We would like to thank anonymous reviewers for their helpful comments. We thank Yan Huang for help developing the Firefox extension. This work was funded in part by the National Science Foundation (awards 0627527 and 0541123) and AFOSR (MURI).

References

[1] Alexa. Top Sites Provided by Alexa the Web Information Company. http://www.alexa.com/.

[2] Apache Infrastructure Team. Cookie Stealing XSS Attacks on Apache Server. https://blogs.apache.org/infra/entry/apache_org_04_09_2010, 2010.

[3] Arvin. Discussion Thread of Django HTTP-only Support. http://code.djangoproject.com/ticket/3304, 2007.

[4] Adam Barth. HTTP State Management Mechanism Draft. http://tools.ietf.org/html/draft-ietf-httpstate-cookie-05, 2010.

[5] Bugmenot development team. An Automated Login Script to Bypass Compulsory Registration. http://www.bugmenot.com/.

[6] Jeremiah Grossman. Cross-Site Tracing (XST): The New Techinques and Emerging Threats to Bypass Current Web Security Measures using TRACE and XSS. Technical report, WhiteHat Security, 2003.

[7] Jeremiah Grossman. Seventh Website Security Statistics Report. Technical report, WhiteHat Security, 2009.

[8] Paul Johnston. Authentication and Session Management on Web. Technical report, Westpoint White Papers, 2004.

[9] D. Kristol. Request For Comments 2109: HTTP State Management Mechanism. http://www.ietf.org/rfc/rfc2109.txt, 1997.

[10] D. Kristol. Request For Comments 2965: HTTP State Management Mechanism. http://www.ietf.org/rfc/rfc2965.txt, 2000.

[11] Art Manion. Vulnerability Note VU#867593: Web Servers Enable HTTP TRACE Method by Default. http://www.kb.cert.org/vuls/id/867593, 2003.

[12] Marco Jetson. An iphone XSS exploit. http://www.exploit-db.com/exploits/10947, 2010.

[13] Microsoft Corporation. Mitigating Cross-Site Scripting with HTTP-only Cookies. http://msdn.microsoft.com/en-us/library/ms533046(VS.85).aspx, 2002.

[14] Mox. An Example of XSS Exploit in nytimes.com. http://www.xssed.com/mirror/34125/, 2008.

[15] OWASP, Open Web Application Security Project. Introduction of HTTP-only Cookies. http://www.owasp.org/index.php/HTTPOnly, 2010.

[16] Parveen Patel, Andrew Whitaker, David Wetherall, Jay Lepreau, and Tim Stack. Upgrading Transport Protocols using Untrusted Mobile Code. *ACM SOSP*, 2003.

[17] Qian Xiang Company. A Sample Script of using document.cookie at Client-side Javascript. http://s.xnimg.cn/a7989/jspro/webpager.js, 2010.

[18] James Slater. Massive Twitter Cross-Site Scripting Vulnerability. http://www.davidnaylor.co.uk/ massive-twitter-cross-site-scripting-vulnerability.html, 2009.

[19] Wladimir Palant. XMLHttpRequest Allows Reading HTTP-only Cookies. https://bugzilla.mozilla.org/show_bug.cgi?id=380418, 2007.