COSC 647 – Fall 2015 – Mary Snyder
Literature Review 5: Robust Defenses for Cross-Site Request Forgery"

1. What is the paper about? What is/are the vulnerability? What causes the vulnerability?

The paper discusses cross-site request forgery (CSRF). In this kind of attack, the user's session is used to make network requests (malicious or otherwise) through the user's browser. Browser security policies allow an attacker to render content that spoofs the user and give access to resources they normally would not have. The attacker could get network connectivity to other machines the user interacts with by getting the user's browser to send network requests for the attacker to those machines. This can occur because the requests would have the user's IP address, not the attackers, since the requests originated from the user's browser. The attacker could also read the browser state, such as cookies, certificates, or authentication headers and use them to gain access to sites that require such information for authentication. In addition, the attacker could write the browser state through the network requests it sends, which can lead to other more subtle attacks.

The paper also recognizes three different threat models for CSRF attacks/vulnerabilities: Network attacker, web attacker, and forum poster. Network attackers can use compromised DNS servers or even an "evil twin" wireless router to control the user's network connection. Web attackers have their own legitimate domain name along with valid HTTPS certificates. Their cross-site attacks involve having the user's browser issue requests using GET and POST methods. Forum posters can exploit submission of images or hyperlinks to issue requests from the honest site's origin using the HTTP*GET* method for HTTP headers.

2. What is/are the contributions of the paper? How was the vulnerability or insecurity discovered?

The paper three widely used techniques for defending against Cross-Site Request Forgery (CSRF): validating a secret request token, validating the HTTP Referer header, and validating custom headers attached to XMLHttpRequests. Each has their own downfalls that make them inadequate in defending against attacks. Using a secret token bounded to the user's session would force the attacker to guess the session's secret token; however when implementing the technique many overlook login requests because they lack a session to which the token would be bound. Validating the HTTP Referer header accepts only requests from trusted sources; however, this technique must also deal with requests that do not have a Referer header at all. Blocking any request without a Referer header may also reject a high number of valid requests, but allowing any without a Referer header makes it all too easy for an attacker to gain access. Validating custom headers in XMLHttpRequests is effective; however, it requires sites to conform to using an XMLHttpRequest for any/all state-modifying requests.

This paper contributes to defending against CSRF attacks in four ways: Explanation of the threat model, study of current browser behavior, proposal of new "Origin" header, and study of related session vulnerabilities. Evaluating the CSRF threat model brought to light several often-overlooked variations of attacks, which use network connectivity and logins. Attacks of this nature can cause very serious consequences if the vulnerabilities are exploited. By studying experimental measurement of Referer header suppression, the authors were able to develop an upgrade to Referer validation employing HTTPS and stricter Referer validation while ensuring the integrity of the Referer header. The "Origin" header would contain only information deemed necessary for CSRF defense, mainly the scheme, host, and port for the referring URL. By limiting the information contained in the header, privacy concerns over the content of the Referer header are addressed. The study of vulnerabilities and defenses for OpenID, PHP cookieless sessions, and HTTPS cookies was used to create a 202-line extension to Firefox for cookie defense.

3. The detailed techniques to solve the problem.

The paper claims to prevent CSRF attacks, browsers should be modified to send an Origin header with POST requests that identify the origin initiating the request (null if not able to determine). Since the Origin header only includes the information that is required to identify the initiator of the request, it does not have the privacy concerns of Referer headers, which contain the path or query portions of the URL. The Origin header is also only sent for POST requests, which would help prevent sensitive information from being linked if a user is following a hyperlink, etc. When using the Origin headers, all state-modifying requests will/must be sent using the POST method to be protected from attacks. The server rejects any requests containing an undesired Origin header (including null).

The Origin header approach takes after four other proposed defenses. XMLHttpRequest cross-site defense shares a similar header, but is only sent for XMLHttpRequests. The XDomainRequest API uses a Referer header, which omits the path and query; however, this causes the Referer header to be blocked by the network frequently, while the Origin header is hardly ever blocked. The JSONRequest API includes a Domain header that identifies the host name of the requestor like the Origin header; however, the Domain header does not include the requester's scheme and port, which gives the Origin header added protection against network attackers. The new browser API for authenticating client side communication between HTML documents in the HTML 5 specification uses the same process for validating as the Origin header; however, the validation is done on the client side for HTML 5 while it is done on the server side for the Origin header. The authors completed both the updates necessary for the browser and server components as proof of concept and for testing. The modifications for the Origin header on the browser side were contained to a few line patch for WebKit, an open source component for Safari, and an extension for Firefox, while the server side changes consisted of a few line change to ModSecurity web application firewall.

4. What are the strength/weaknesses of the paper?

The first weakness I could see is the bowser code has to be modified in many cases to send this new Origin header. This would mean until the method is accepted and adopted by all browsers, those wishing to use this protection would have to implement their own custom browser. While they seemed to have confined the changes necessary to it is still very impractical and in many cases undesirable to use a custom browser. Another weakness is this approach can be circumvented if an attacker embeds a frame to an honest site tricking the user into clicking a button inside the frame. This would cause the request to be initiated from the user's browser with the origin of the honest web site/user.

A strength of this approach is sites can detect if a non-supporting browser initiated the request simply by the lack of the Origin header. Since the design is such that the Origin header is null when suppressed by the browser, the Origin header will always be a part of the request. This prevents the attacker from spoofing a supporting browser to look as if they were a non-supporting browser.

5. What can you do better?

The main thing that could be improved is to find a common way to implement this for existing browsers without modification needed to code. Having different implementation of this approach for different browsers makes it harder to implement if a company uses different browsers. If this could be implemented as an extension for any of the existing browsers, this approach would be a lot more marketable and practical. It would also allow for easier implementation, as well as easy removal if deemed unnecessary in the future.