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.

1. 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 have 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.

1. 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 this cannot be determined). Since the Origin header only include 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 would also only be 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 be/must be sent using the POST method to be protected from attacks. The server rejects any requests containing an undesired Origin header (including null).

ffectively takes over and controls the parsing decisions instead of the browser.

Security Analysis. Although the Origin header has a simple design, the use of the header as a CSRF defense has

a number of subtleties.

• Rollback and Suppression. Because a supporting browser will always include the Origin header when making POST requests, sites can detect that a request was initiated by a supporting browser by observing the presence of the header. This design prevents an attacker from making a supporting browser appear to be a non-supporting browser. Unlike the Referer header, which is absent when suppressed by the browser, the Origin header takes on the value null when suppressed by the browser.

• DNS Rebinding. In existing browsers, The Origin header can be spoofed for same-site XMLHttpRequests. Sites that rely only on network connectivity for authentication should use one of the DNS rebinding defenses in Section 2, such as validating the Host header. This requirement is complementary to CSRF protection and also applies to all the other existing CSRF defenses described in Section 4.

• Plug-ins. If a site opts into cross-site HTTP requests via crossdomain.xml, an attacker can use Flash Player to set the Origin header in cross-site requests. Opting into cross-site HTTP requests also defeats secret token validation CSRF defenses because the tokens leak during cross-site HTTP requests. To prevent these (and other) attacks, sites should not opt into crosssite HTTP requests from untrusted origins.

Adoption. The Origin header is similar to four other proposals that identify the initiator of a request. The Origin header improves and unifies these proposals and has been adopted by several working groups.

• Cross-Site XMLHttpRequest. The proposed standard for cross-site XMLHttpRequest [50] included a Access-Control-Origin header to identify the origin issuing the request. This header is sent for all HTTP methods, but it is sent only for XMLHttpRequests. Our specification for the Origin header is modeled off this header. The working group accepted our proposal to rename the header to Origin.

• XDomainRequest. The XDomainRequest API [39] in Internet Explorer 8 Beta 1 sends cross-site HTTP requests that omit the path and query from the Referer header. This truncated Referer header identifies the origin of the request. Our experimental results suggest that the Referer header is frequently blocked by the network, whereas the Origin header is rarely blocked. Microsoft has announced that it will adopt our suggestion and rename XDomainRequest’s truncated Referer header to Origin.

• JSONRequest. The JSONRequest API for crosssite HTTP requests [7] included a Domain header that identifies the host name of the requester. The Origin improves on the Domain header by including the requester’s scheme and port. The JSONRequest specification editor accepted our proposal to replace the Domain header with the Origin header in order to defend against active network attackers.

• Cross-Document Messaging. The HTML 5 specification proposes a new browser API for authenticated client-side communication between HTML documents [20]. Each message is accompanied by an origin property that cannot be overwritten. The process for validating this property is the same as the process for validating the Origin header, except that the validation occurs on the client rather than on the server.

The unreliable behavior.

Both the updates necessary for the browser and the server components were completed for testing. The Origin header for the browser side was 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 ModSecurity web application firewall.

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

The first weakness I could see is the bowsers all have to be modified to send this new Origin header.

for any application not written in PHP, an alternate version of Blueprint needs to be run, which has a separate process to communicate with the web application over TCP. The program already consists of a server-side component as well as a client side script library; therefore, adding the separate process makes things even more complicated and increases the possible points of failure. A second weakness that I saw is they ran their tests on popular web application using the most popular web browsers. While that will give the public at large an idea how well it works overall, many company’s application will be unique in the types of data and content that they deal with and may in some cases be using old/outdated browsers. The type of attacks that would work in those cases may never be exercised in the testing on with the most popular applications/browsers. A third weakness is their reported overheads. They report a range of overhead for memory consumption to be from 0% to 13.6% with the average being 5%. While this may seem “modest” to them a 13.6% overhead could impact overall performance. They also measured the overhead for page size and for WordPress it was a staggering 52.4%. This does not seem like a result that would be acceptable to any company even if it does offer the protection they advertise. Again for WordPress the average increase in processing time was a high 55%. This again would never be acceptable to companies that want their information to the user as quickly as possible.

The first strength of the program is that tests proved it is resilient against even subtle XSS attacks. The platforms used in testing embedded attack strings in a variety of different contexts for a template web page. The program was able to defend against all 94 different XSS attacks tested and there were no successful attacks due to exploits of browser parsing quirks. Since it was tested on eight of the most popular web browsers, which make up over 96% of the browser market, this goes a long way to show how effective it is against XSS attacks for the vast majority of users. Another strength is Blueprint preserves the rendering order of the page content by interpreting the models synchronously as the page is rendered. The content model is accepted as input to a model interpreter. The model interpreter constructs the parse tree and when complete, everything including the invocation script is removed. This means all the processing is done without leaving a trace that it was even performed.

1. What can you do better?

The main this that I think needs to be address to make this product better are the resource utilization. Overheads as high as 52.4% for page size, 13.6% for memory consumption overhead, and increase in processing times as much as 55% should be reduced. While the paper claims the overall latency is imperceptible to a user, there is a caveat that they read the web page in a continuous manner from top to bottom. They also suggest mitigations such as serving fewer comments per page. These “suggestions” seem to be hacks, rather than real solutions to problems incurred by using the Blueprint product. It is not a very good business model to ask the company using your product to change how they render web pages to overcome the shortcomings in your product. Address those resource utilization issues and I think the overall product would be greatly improved.

Certificate Errors:

Certificate Errors. If the user is willing to click

through HTTPS certificate errors, much of the protection

afforded by HTTPS against network attackers

evaporates. A number of researchers [38, 31, 24] have

addressed this threat model, but, in this paper, we

assume users do not click through certificate errors.

User Tracking:

User Tracking. Cross-site requests can be used by

cooperating web sites to build a combined profile of

a user’s browsing activities. Most browsers include

third-party cookie blocking features that are designed

to discourage such tracking, but these features can be

circumvented [26].