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

The paper discusses a new defense strategy against cross-site scripting (XSS) attacks called Blueprint. According to the paper, the number one security threats for the Internet at large are XSS attacks. The attacks rely upon unauthorized script code injected into a web page to extract confidential and sensitive user data.

In a cross-site request forgery (CSRF) attack, the attacker disrupts the integrity of the user’s session with a web site by injecting network requests via the user’s browser. The browser’s security policy allows web sites to send HTTP requests to any network address. This policy allows an attacker that controls content rendered by the browser to use resources not otherwise under his or her control:

1. Network Connectivity. For example, if the user is behind a firewall, the attacker is able to leverage the user’s browser to send network requests to other machines behind the firewall that might not be directly reachable from the attacker’s machine. Even if the user is not behind a firewall, the requests carry the user’s IP address and might confuse services relying on IP address authentication [36].

2. Read Browser State. Requests sent via the browser’s network stack typically include browser state, such as cookies, client certificates, or basic authentication headers. Sites that rely on this authentication state might be confused by these requests.

3. Write Browser State. When the attacker causes the browser to issue a network request, the browser also parses and acts on the response. For example, if the response contains a Set-Cookie header, the browser will modify its cookie store. These modifications can lead to subtle attacks, which we describe in Section 3.

In-Scope Threats. We consider three different threat models, varying by attacker capability:

• Forum Poster. Many web sites, such as forums, let users to supply limited kinds of content. For example, sites often permit users to submit passive content, such as images or hyperlinks. If an attacker chooses the “image’s”URL maliciously, the network request might lead to a CSRF attack. The forum poster can issue requests from the honest site’s origin, but these requests cannot have custom HTTP headers and must use the HTTP“GET”method. Although the HTTP specification [6] requires GET requests to be free of side effects, some sites do not comply with this requirement.

• Web Attacker. A web attacker is a malicious principal who owns a domain name, e.g. attacker.com, has a valid HTTPS certificate for attacker.com, and operates a web server. These capabilities can all be obtained for $10. If the user visits attacker.com, the attacker can mount a CSRF attack by instructing the user’s browser to issue cross-site requests using both the GET and POST methods.

• Network Attacker. A active network attacker is a malicious principal who controls the user’s network connection. For example, an“evil twin” wireless router or a compromised DNS server can be exploited by an attacker to control the user’s network connection. These attacks require more resources than web attacks, but we consider this threat in-scope for HTTPS sites because HTTPS is designed to protect against active network attacks

The strategy of the defense outlined in this paper and the goal for Blueprint is to minimize the burden on the browser in determining untrusted content.

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.

Blueprint claims to satisfy the three main objectives of preventing XSS attacks: 1) it is robust, protecting even with browser quirks, 2) it supports structure, benign HTML derived from untrusted user input, and 3) it is compatible with existing browsers current in use. Since the parsing behavior of browser can be unreliable, Blueprint effectively takes over and controls the parsing decisions instead of the browser. The Blueprint application uses information about the flow of untrusted HTML in a browser to create a “blueprint” or structural representation of untrusted web content, with XSS attacks removed. The generated tree is fed to the browser’s document generator bypassing the browser parser and avoiding its analogous and/or unreliable behavior. The implementation does not require knowledge of how the browser parses the data and instead enforces the application’s understanding of the web content on the browser. In doing this, the effect of the browser parser’s analogous behavior does not come into play.

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

The first weakness I could see is 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.