A NOVEL APPROACH FOR DETECTING AND PREVENTING CROSS SITE SCRIPTING AND HTTP PARAMETER POLLUTION

Ponnamanda Bhavani, A., Rajani Devi, S.B and K. Syamasundararao

Dept of Information Technology, *Nalanda Institute of Technology*

ABSTRACT

Present situations, the most critical attacks are those that combine Cross site scripting techniques to access systems and Hypertext Transfer protocol parameter pollution techniques to access the information by polluting the HTTP parameters. The potential damage associated with this kind of threats, the total absence of background and the fact that the solution to mitigate these vulnerabilities must be worked together with programmers, systems administrators and database vendors justifies an in-depth analysis to estimate all the possible ways of implementing this technique. It is a quite simple but effective hacking technique. HPP attacks can be defined as the feasibility to override or add HTTP GET/POST parameters by injecting query string delimiters. It affects a building block of all web technologies. We have to investigate business logic flaws triggered by HPP. As we know, it is tricky and time consuming since manual testing is required. In this paper we are proposing a novel approach to prevent http parameter pollution using reverse proxy. This approach provides a mechanism of records HTML response in order to test the application behavior as well as unexpected exploits.

Keywords: *HPP, HTTP, Reverse proxy and XSS.*

INTRODUCTION

Unlike early Web sites, which were merely meant to deliver text in a practical fashion, nowadays sites are not only capable of hosting rich content, such as images, videos, and audio material, but also provide platforms for users to contribute such data and share it with the rest of the world. The goal of the XSS attack is to steal the client cookies, or any other sensitive information, which can identify the client with the web site. For example, in one audit conducted for a large company it was possible to peek at the user's credit card number and private information using a XSS attack (Hallaraker and Vigna, 2005). This was achieved by running malicious JavaScript code at the victim (client) browser, with the "access privileges" of the web site. These are the very limited JavaScript privileges which generally do not let the script access anything but site related information. It should be stressed that although the vulnerability exists at the web site, at no time is the web site directly harmed. Yet this is enough for the script to collect the cookies and send them to the attacker. The result, the attacker gains the cookies and impersonates the victim. As long as the input provided by users is benign and the Web applications are used as intended, the challenges are easily met by developers and service providers. However, for various reasons, such as simple curiosity, destructive intentions, or hope for financial profit, there will always be people who aim to exploit Web sites and their users to their advantage. Therefore, even though users expect modern Web services to integrate their content seamlessly and effortlessly into the provided applications, protection of their local computer systems is required, when viewing Web content created and submitted by potentially malicious entities. There is no significance or any type of improvements related to HTTP parameter pollution (Endler, 2002). But it seems to be very simple but very effective.

Http parameter pollution continuously leads the most wide spread application vulnerabilities, Open Web Application Security Project (Gundy and Chen, 2009). Recently, Yahoo Mail affected by Http parameter pollution (Kirda *et al.* 2006). Many of the banking applications are vulnerable to http parameter pollution. PayPal faces the problem of tampered refund transactions due to http parameter pollution. Http parameter pollution client side is about injecting additional parameters to links and other src attributes. The precedence of GET/POST/Cookie may influence the application behaviors and it can also be used to override parameters

Apache Tomcat/6.0.18
POST /foo?*par1=val1&par1=val2*HTTP/1.1
Host: 127.0.0.1
FIRST occurrence, GET parameter first
Apache Tomcat/5.0.8
POST /foo***?****par1=val1&par1=val2*HTTP/1.1
Host: 127.0.0.1
Last occurrence, GET parameter first
According to OWASP Strawman Classification (Jovanovic *et al.,* 2006). HPP will divide into following categories:

Client-side

First order HPP: Attacker can override the action parameter value to edit then get the information from user. This seems very simple and also does not contain any XSS attacks also but it should reflect in application behavior. It is usually done by masquerading the visual link with actual link.

Second order HPP: This is completely related to the functionalities of link or form. But it is almost related to Anti-Cross site scripting forgery and functional UI readdressing. It was stored in persistent manner.

Third order HPP: This is completely related to parsing unexpected parameters and it generates the HPP using JavaScript regular expressions. It is about to use of HPP on polluted parameters. It is usually called as content pollution.

Standard HPP: The frontend will build the back-end request based on application manages the parameter occurrences. This attack was completely depending on the application behavior and business logic implementation. The same attack was injected on Yahoo mail also. Attacker sends payload containing additional parameters injected in recipient parameters. Depending on Front end server and back end server last occurrence parameter action will be performed. It is very effective attack on bank transactions.

Second order HPP: Uniform resource locator could be affected as well as if regular expressions are too permissive. This HPP attack was completely related to regular expressions. Even though regular expressions are very useful for validation but they are very effective and easy to use for finding the URL parameters.

The main contributions of this paper summarized as follows:

- We introduce this approach, a solution for detecting and preventing http parameter pollution using reverse proxy (Peter Wurzinger *et al.*, 2009)
- In contrast to previously proposed methods, it does not require any manual testing
- It is not only restricted to HPP also it can also detect XSS and other attacks.

OUR APPROACH

HPP tester operates on a reverse proxy, which relays all traffic between the Web server that should be protected and its visitors (as depicted in Figure 1). The proxy forwards each Web response, before sending it back to the client browser, to a JavaScript/HPP detection component, in order to identify embedded HPP content. In the JavaScript detection component, HPP tester puts to work a fully functional, modified Web browser, that notifies the proxy of whether any scripts and polluted parameters are contained in the inspected content and it will record the response in a log file. In order to detecting the malicious content and benign input JavaScript and polluted parameters we are modifying the web application. In this paper, we are proposing a new novel approach to detecting and preventing HPP as well as XSS using reverse proxy (Cache). It was very simple and easy to deploy(as depicted in Figure 2).

METHODOLOGY

- First we request a page of the original web application.
- The requested URL is received by the Proxy (cURL) and writes into the HPP tester (cache).
- Then Input filtering is performed in the requested URL by checking the query String.
- If found any parameter pollution (or) cross site scripts the request is transferred back to the client (Browser).
- If the requested URL is correct, it is sent to the back-end server.
- To summarize, the main components of HPP tester are:
- A HPP detection component, which, given the Web server's response, is capable of determining whether script content/polluted parameter is present or not.
- A reverse proxy installed in front of the Web server, which intercepts all HTML responses from the server and subjects them to analysis by the HPP detection component.
- A set of scripts/parameters to automatically encode/decode scripts.

IMPLEMENTATION

Web Application Implmentation: Generally, in order to locate legitimate scripts and additional parameters in the original Web application, it is advisable to utilize a similar mechanism as the HPP detection component later used to identify malicious parameters (as described in section B). Therefore, the first step for deploying HPP tester is to identify all legitimate script calls and polluted parameters in the original Web application, and to replace each one by a unique identifier for scripts and for parameters a log file.

There are three requirements for a unique identifier: First, it must not contain any valid HTML tags, Second, it must not contain what would be interpreted as JavaScript by a browser, so that when rendering a page it is safe to conclude that all script executions stem from illegitimately injected scripts. Third, the mapping must be reversible, so that after probing a page for scripts, the original condition with functional JavaScript code can be reestablished.For Additional parameters first we have to find the query string delimiter and split the url into strings based on delimiter and decode the parameters when those are in encoded format.So it is very is to find the number of parameters that are to be processed excess. For our prototype implementation, we defined a set of strings that directly indicate the presence of JavaScript code, such as the script tag and also it will detect the parameters that are to be added additionally to url.

HPP tester component: Even though there are exact specifications on how an HTML parser is supposed to identify and interpret JavaScript code, browsers often attempt to compensate for Web developers' mistakes and also process and execute scripts that do not match the specification. Not only does this lead to incompatibilities between different browsers and their according parser implementations, but it also opens unforeseeable possibilities for a Web developer to initiate a script execution. For this reason, crafting a custom parser and basing the decision on whether it contains a script or not on its output and url additional parameters and content pollution, is likely to produce unsatisfactory results. More precisely, a parser that strictly follows the specifications would miss certain malformed scripts and last occurance parameters that a browser would execute. For this reason, we chose to put a new version of an actual Web browser to work in the JavaScript/HPP detection component, in order to render the page and decide whether there is script code included or paramters are polluted. This new version web browser will works for any type of browser.

Reverse Proxy: We adapted the cURL (Scott and Sharp, 2002) is a computer software project providing a library and command-line tool for transferring data using various protocols. The cURL project produces two products, libcurl and cURL. In this paper we use libcurl. This is a library created by Daniel Stenberg, that allows you to connect and communicate to many different types of servers with many different types of protocols. libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols. libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading (this can also be done with PHP's ftp extension), HTTP form based upload, proxies, cookies, and user+password authentication. libcurl is free, thread-safe, IPv6 compatible, feature rich, supported and fast. Software programmers incorporate libcurl into their programs. All (inbound) HTTP requests are forwarded unchanged to the Web server. Only the (outbound) HTTP replies are inspected more closely. Each reply is first checked on whether it consists of HTML code (as opposed to, e.g., a file download), and can therefore contain scripts. All HTML pages are forwarded to the HPP tester detection component, which determines whether the page contains any JavaScript code/polluted parameters, and reports its findings back to the proxy. If, expectedly, no code is found, the page is deemed clean and the proxy delivers it to the client, after decoding all unique identifiers and restoring the original legitimate script content. If, however, JavaScript content is identified in the page the proxy obtained from the server, it is most likely injected, and the proxy returns a warning message to the client, instead of the actual content. If server side HPP attack identified in the page the proxy obtained from server and returns alert message to client and also record all parameters.

EVALUATION

Detection of Server side HPP: The ability of HPP tester to correctly detect polluted parameters at server side strongly depends on how precisely the HPP detection component works in locating additional parameters within any url. In order to verify that our implementation works satisfactorily also in non-traditional ways of embedding script code and also polluted parameters and business logic flaws, we evaluated it on the OWASP Strawmann HPP server side attacks (Kals *et al.,* 2006), a collection of various HPP attack code snippets, that cover a broad range of nuances regarding filter evasion. All tested examples that work in an unmodified browser have been successfully detected by our HPP tester component.

Detecting Server side HPP attacks: In order to evaluate the quality of our prototype implementation, first, we aimed to assess its ability to correctly identify injected scripts. For that purpose, we deployed three well-known Web applications in a test environment, all of which are vulnerable to XSS, and applied HPP tester as their protection. That is, we encoded all JavaScript code into unique identifiers in the applications' source code, and installed the reverse proxy and HPP detection component in front of the application server. It can record the HTML response and request parameters so it is very easy to find the server side HPP attacks and also it is very easy to find the client side HPP also, because this approach was using a new browser. So in order to detect the polluted parameter each request will processed by this browser only. It can handle by CURL. It is very effective than all other techniques that are discussed in earlier. I tested the excite search engine (Kals *et al.,* 2006) and normally it facing the problem of content parameter pollution. But my novel approach solves this problem and also prompts the user with alert message. It is very effective because even though parameters are in encoded format also it decode it. My test results are completely detect the server side attack which was Yahoo Mail pay load[6].For this We created a fake mail and we are sending a pay load (Scott and Sharp, 2002) to user. HPP tester detects the malicious payload and sends the alert message to user. And also it records a complete response message in a file (Link 1 & 2).

***Detection of XSS*:** The ability of HPP tester to correctly detect XSS attacks strongly depends on how precisely the JavaScript detection component works in locating JavaScript content within HTML code. In order to verify, our implementation works satisfactorily also in non-traditional ways of embedding script code and very effective than previous approach (Peter Wurzinger *et al.,* 2009).

RESULTS

HPP tester adds to the latency two-fold: First, by putting an additional stepping stone between client and server, namely the reverse proxy, all traffic is relayed instead of a direct transmission, and thus, takes longer to arrive at its target. Second, and more importantly, the HPP/JavaScript tester detection component effectively has to render each page before it can be delivered to the client. Due to the additional requirements for processing power introduced by HPP tester, clearly, performance degradation is introduced, meaning that the client will experience higher latency when requesting content from a HPP tester protected Web server, as compared to a server that does not feature this protection. Our implementation was completely based on the cURL. But earlier approach was python-based reverse proxy it needs two systems to deploy apache web server and browser. Our implementation was relatively give more speed than earlier approach (Peter Wurzinger *et al.,* 2009). We have conducted experiments to measure the magnitude of the performance penalty inflicted by our prototype implementation. Our experiments are degrading the performance when file size is decreasing. The factor by which a deployment without HPP tester outperforms a HPP tester protected setup decreases steadily with increasing file size. This can be attributed to the constant effort for proxy relaying as well as initializing the browser.

CONCLUSION

We are presenting a server-side solution for protecting users of a web application from cross site scripting and Http parameter pollution using reverse proxy by intercepting all HTML responses and forwarding them to HPP tester consisting of full fledged browser. In previously there is no implementation on HPP to test. HPP tester also records all HTML responses and keeps track of number of polluted parameters effectively. This approach was minimal proposed solution to mitigate server side HPP attacks.

REFERENCES

Endler. D. 2002. “The Evolution of Cross Site Scripting Attacks”. *technical report*, iDEFENSE Labs.
Gundy, M.V and H. Chen. 2009. Noncespaces: “Using randomization to enforce information flow tracking and thwart cross site scripting attacks”. in Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS). February 8-11, San Diego, CA.
Hallaraker, O and G. Vigna. 2005. ”Detecting Malicious JavaScript Code in Mozilla”, in Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). Shanghai, China
Jovanovic, N., Kruegel, C and E. Kirda. 2006. ”Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)”,in IEEE Symposium on Security and Privacy, 44: 6.
Kals, S., Kirda, E., Kruegel, C and N. Jovanovic. 2006. ”SecuBat: A Web Vulnerability Scanner” in World Wide Web Conference, 2006. Edinburgh, Scotland.
Kirda, E., Kruegel, C., Vigna, G and N. Jovanovic. 2006. ”Noxes: A client-side solution for mitigating cross-site scripting attacks.” in 21st ACM Symposium on Applied Computing (SAC). Dijon, France.
Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, and Christopher Kruegelk. 2009. “SWAP: Mitigating XSS Attacks using a Reverse Proxy“in SESS’09, IEEE, May 19, 2009, Vancouver.
Scott, D and R. Sharp. 2002. ”Abstracting Application-level Web Security.” in 11th World Wide Web Conference. May 7-11, Honolulu, Hawaii, USA.

Figure 1: Scheme of HPP tester

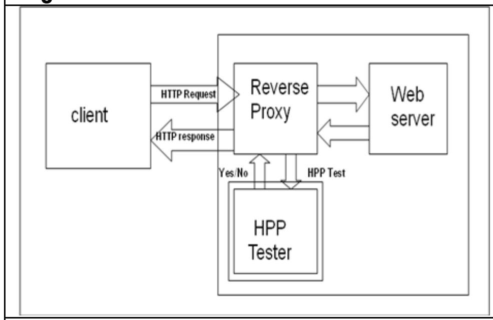
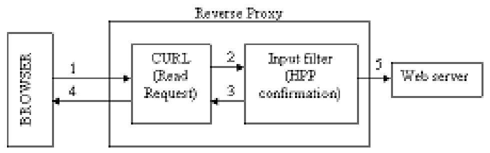


Figure 2: HPP detector procedure



Link 1. Example of Content parameter pollution:
http://search.excite.it/image/?q=dog&page=1%26%71%3d%66%75%63%6b%6f%66%66%20%66%69%6e%67%65%72%26%69%74%65%6d%3d%30

Link 2. Yahoo mail payload:
http://localhost:8080/mail/showFolder?fid=Inbox%2526cmd=fmgt.emptytrash%26DEL=1%26DelFID=Inbox%26cmd=fmgt.delete

****