Proceedings of the Federated Conference on Computer Science and Information Systems pp. 391–395
DOI: 10.15439/2018F208
ISSN 2300-5963 ACSIS, Vol. 15

A new WAF-based architecture for protecting web applications against CSRF attacks in malicious environment

Michal Srokosz*, Damian Rusinek†, Bogdan Ksiezopolski†
*Polish-Japanese Academy of Information Technology
ul. Koszykowa 86, 02-008 Warszawa, Poland
msrokosz@pjwstk.edu.pl
†Maria Curie-Sklodowska University
pl. Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland
{damian.rusinek, bogdan.ksiezopolski}@umcs.lublin.pl

***Abstract*—Web application firewall is an application firewall for HTTP applications. Typical WAF uses static analysis of HTTP request, defined as a set of rules, to find potentially dangerous payloads in the requests. Generally, these rules cover common attacks such as cross-site scripting (XSS) and SQL injection which are server-related attacks. Cross-site scripting is client-side attack however the server is attacked and forced to return malicious response. Rule-based approach becomes useless when the attack is client-related, for example employing malware on the banking site. Malware allows to change the transfer data. This scenario is hard to detect because the browser displays valid transfer data and data is changed to the thieves' accounts number at the communication stage.**

In this paper we introduce a new web-based architecture for protecting web applications against CSRF attacks in malicious environemnt. In our approach we extend a classic, static WAF approach with historical and behavioral analysis, based on actions performed by the user in the past.

I. Introduction

ONE OF the ideas to increase Web applications security was Web Application Firewall, a proxy server used to defend web apps against attacks usually employed in the application layer in contrary to classic firewalls. WAFs are located between classic firewall and the application server. Such architecture allows the firewall to mitigate attacks on lower layers and WAF to detect and mitigate attacks on application layer. Both groups of mentioned attacks can have similiar consequences such as Denial of Service. DoS attacks, well known from lower layers[7], can also be performed on the application layer[5]. In [9] authors analyze 63 other articles about HTTP-GET flood attacks.

However, the application layer introduces a wide range of new threats to be detected and removed by WAF. OWASP Top Ten [10] is a powerful awareness document for web application security which represents a broad consensus about the most critical web application security flaws. The OWASP Top Ten list includes flaws, such as injections, cross-site scripting, cross-site request forgery, insecure session management, insecure direct object references, security misconfiguration, using components with known vulnerabilities and others. Most of web application firewalls focus on the technical attacks such as injections, cross-site scripting or cross-site request forgery while it is hard to detect other types such as insecure direct object references or business logic flaws because they are strictly application-dependent.

We are going to use request forgery attacks as an example of successful business attacks to present our new approach to detect and mitigate malicious requests. The most popular type of request forgery attacks are cross-site request forgery attack (CSRF) which makes a logged-on victim's browser send a forged HTTP request, together with the victim's session cookie and any other automatically attached authentication information to a vulnerable web application. In other words, the attacker forces the victim's browser to generate requests, which from the vulnerable application's perspective are legitimate. For this reason, on the server side we are not able to detect this only based on the technical attributes of the query. Although this is not a sophisticated attack, it indicates that the key players (Facebook, LinkedIn, etc.) had suffered from it.

Another type of request forgery attack is server-side request forgery (SSRF) and request forgery generated by malicious software. The SSRF differs from CSRF that the attacker forces a vulnerable application server to send a request. In the second type the attacker installs malicious software of victim's device which later sniffs the authentication data (eg. SMSes on the smartphone) and sends authenticated requests. According to reports by Symantec [11] and Kaspersky Lab [4] the malicious software is a significant problem with more than 30% of user computers subjected to at least one Malware-class attack and more than 170 mobile applications for credentials stealing in 2016. The financial Trojan threat landscape is dominated by three malware families: Ramnit, Bebloh (Trojan.Bebloh), and Zeus (Trojan.Zbot), responsible for 86 percent of all financial Trojan attack activity in 2016. Most anti-malware solutions is based on the detection of their presence.

Many commercial WAFs use signature-based techniques, attempting to find the malicious inputs appearing in the signature database. One can enumerate known WAFs, such

as F5, Juniper, Modsecurity and many others. In the scope of request forgery attacks the defense technique is to tokenize the requests. Such solutions are not only used by WAFs but many application frameworks provide such middleware as well.

Unfortunatelly, the use of tokens as the factor, which authenticate the requests is not sufficient in the malicious environemnt. In this paper, we are focused on a popular case of request forgery attack performed by malicious software installed on clients device (eg. mobile phone) and propose a mechanism to detect such attacks. The current Web Application Firewalls assume that the clients' devices is free from malicious software. This assumption in times of common malware can not take place.

The major contributions of the presented results can be summarized as follows:

- We present a successful request forgery attack on the application defended by classic WAF when client has malware installed.
- We propose new architecture for protecting web applications against request forgery attacks performed by malicious software.
- We extend our WAF proposal to include Two-Factor Authorization mechanism and user's history analysis.

The content of this paper is structured as follows. We discuss the related work in Section II. In section III we introduce the notation and describe the successful request forgery attacks leading to authorization bypass. Section IV describes our approach to detect and mitigate malicious business actions such as requests performed by malware. It includes the description of architecture, the detection algorithm. Finally, section V concludes this paper and describes the futher work.

II. Related Work

In the literature method of protecting web applications are not widely discussed. Researchers focus on non-standard attacks that can not be detected on classic firewalls and design new mechanisms for detecting these specific attacks.

In [3] authors propose an automatic method of HTTP attacks signature generation. Their approach relies on the use of a service-specific, semantic-aware anomaly detection scheme that combines stochastic learning with a model structure based on the HTTP protocol specification. The proposed soluction assume that the clienct is free from malicious software.

The article [8] proposes an approach that uses ontology models to detect web application attacks in HTTP protocol. Authors created three models of correct request, correct response and an attack. The HTTP requests are analyzed for compliance with the model and marked as a potential attack when forbidden values are found. This approach is similar to the whitelist approach, which is time-consuming and leads to many false positive alarms.

The authors in article [6] concentrate on SQL injection attack and propose the detection mechanism employing graphs and Support Vector Machine. The algorithm converts SQL query to the graph and uses previously trained SVM to detect SQL injection. The drawback of this algorithm is that it focuses only on the detection of tautology which is the first phase of the attack. When the mechanism blocks such query it can be considered as the presence of vulnerability.

In [1] authors conducted a review of the literature on popular web application attacks from OWASP Top 10 list, such as injections, access control or session management. Among the analyzed mechanisms were source code static analysis, dynamic detection of forbidden values and more complex such as comparison of responses which dropped responses outlying from the norm. The most popular solutions were based on the detection of forbidden values and authors stated that there does not exist a solution that is capable of detecting all injections, even in only one category such as cross-site scripting, because of many special cases of such flaw. All the discussed protection mechanisms assume that the client is free from malicious software.

III. Authorization bypasss with request forgery attacks

The aim of authorization bypass attacks is to perform an unauthorized action on behalf of authroized user. There exist many attack vectors and scenarios. In this section we describe four examples of such attacks, beginning with the simplest one employing social engineering techniques, to more complicated which uses malicious software.

A. Notation

We are going to use the following notation to describe the attack flows. The steps described correspond to the numbers in square brackets on matching figures.

- Actors:
  - $Client$ - the mobile or web client of the system,
  - WAF - web application firewall,
  - $Server$ - system endpoint server (reverse proxy),
  - $Attacker$ - an attacker (eg. malware).
- Messages:
  - $CREDENTIALS_{Client}$ - Client's credentials,
  - $SESS_{Client}$ - Client's session created by Server,
  - $EMAIL_{MAL}$ - malicious e-mail message with malicious link,
  - REQ_{BA} - a request to obtain form for business action BA,
  - $RESP_{FORM:BA}$ - a response that contains the form of business action BA,
  - $DATA_{FORM:BA}$ - form data to performa a business action BA,
  - $MODDATA_{FORM:BA}$ - modified (by malware) form data to performa a business action BA with malicious result,
  - $RESP_{BA}$ - a response that confirms the execution of business action BA,
  - $RESP_{MODBA}$ - a response that confirms the execution of modified business action BA,
  - $CSRF_{TAG}$ - a anti-CSRF tag,
  - $2FA_{OTP}$ - one time password from 2FA device, mandratory to authorize business action,

- Actions:
  - $ClickLink(EMAIL_{MAL})$ - user clicks the link from malicious e-mail,
  - $Verify(CREDENTIALS_{Client})$ - Server verifies Client's credentials and creates new session for him,
  - $WafVerify(Data)$ - WAF verifies the correctness of Data,
  - $CreateUser(DATA_{FORM:USER})$ - Server creates user record with form data,
  - $FormTag(RESP_{FORM:BA})$ - WAF creates a form tag to prevent CSRF attack,
  - $DoBA(DATA_{FORM:BA})$ - Server executes business action BA using form data,
  - $Intercept(Data)$ - Attacker intercepts data,
  - $ModifyBA(DATA_{FORM:BA})$ - Attacker modifies business action attributes (form data),
  - $AdditionalAuthRequired(Data)$ - WAF checks whether additional authorization is required for given action described by data,
  - $Send2FARequestForData(MODDATA_{FORM})$ - WAF sends challange to 2FA device for given business data for additional authorization,
  - $AbortForSecurityReason()$ - Client aborts operation (hacking attempt found).

B. The CSRF attack using a malware to bypass RSA Token and WAF

In this example we present an attack which bypasses the RSA Token and Web Application Firewall with the use of financial Trojan like ZEUS. RSA Token is a two-factor authentication device which generate a cryptographically-secure token to authorize the business action. ZEUS malware, on the other hand, allows to change the bank transfer data in online banking system. The attack is hard to detect by user because the browser displays valid transfer data and data is changed to the thieves' account number during the communication. Two-factor authorization, which does not user a device that displays the decription of operation to be authorized, is not effective for this type of attack.

The case background is the following. The Client has a bank account in the bank which requires that the transactions commissioned on the online service must be confirmed using one time password (OTP) generated by RSA Token. Client's device is infected with malware that is specialized in stealing money from the bank transaction system (eg. ZEUS).

The scenario of the attack is presented on figure 1.

(1) The Client logs in to the bank system.
(2) WAF performs static analysis if request is technically correct.
(3) Request was validated and pass to the banking system.
(4) The system validated the data entered.
(5) System created the session and returned it to the client.
(6) He wants to transfer money to his contractor.
(7) WAF verifies request.
(8) Pass it to the banking system.
(9) WAF receives form.
(10) WAF tags it with CSRF token.
(11) WAF sends it back to the user.
(12) The account number to which he wants to transfer the money is not added to any trusted transfer templates - the system will require authorization and one time password (OTP) code from the RSA Token.
(13) Malware detects an attempt to perform a transfer and, at the communication stage.
(14) Malware swaps the contractor account to the Attacker's account.
(15) A malicious request is sent to the system.
(16) WAF validates modified request.
(17) Pass it to the system.
(18) System performs transfer to Attacker's account.
(19) Malware, on the summary screen of the transfer, presents the Client with the account number of the contractor. The unaware Client gives the OTP code and authorizes the transfer to the Attacker's account.

IV. The new architecture for protecting web applications against request forgery attacks performed by malicious software

In this section we describe our approach to extend WAF security with behavioral analysis. The solution we want to propose increases the security and usability of the application that the WAF protects. It reduces the risk of a successful attack, even if your device is infected with malware.

We introduce behavioral analysis and user action history. The user request is analyzed by WAF before it reaches the target system. The WAF analyzes whether the user has performed similar actions in the past and whether they have been successfully commissioned. The similarity is calculated on the base of technical and business attributes describing actions. When the requested action is similar, the WAF does not require additional authorization. If not, the WAF asks for additional authorization to confirm the operation for the data entered. It would speed up the use of the target system and minimize the risk that the user confirms the operation with input altered by the Attacker.

A. History analyze and the similarity function

The added value of our solution to the classic Web Application Firewall is the Hisotry Analyzer module. With this module we are able to detect potencial abuse using cross-site request forgery. The key element of the proposed solution is the similarity function. We are going to use the following model to describe the proposed solution. In order to implement historical analysis, we need to introduce a concept of action. Actions, ie, business orders that a user has performed on a system that protects the WAF. The module checks to see whether similar operations have been performed by the user in the past. Similarity is calculated using an algorithm 1. If the action is similar to past operations, additional authorization is not required.

Definition 1. Technical attributes.

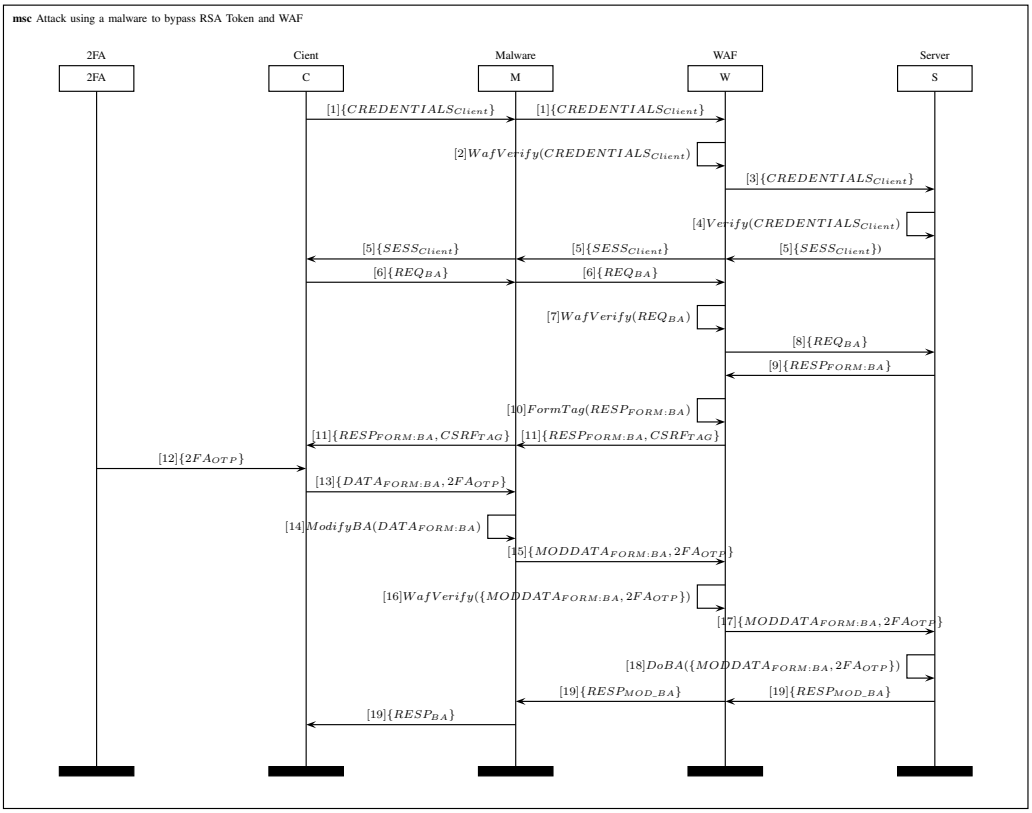

Fig. 1. Attack using a malware to bypass RSA Token and WAF.

$$T = \{t_1, t_2, ..., t_n\}$$
$$VT = \{vt_1, vt_2, ..., vt_n\}$$

The T set is a set of technical attributes and the VT set contains all of possible values of technical attributes. Technical attributes are not directly related to business data; they are rather a description of where and from what machine the action was initiated. This can be an ip address, browser fingerprint, country, time zone, and so on.

Definition 2. Business attributes.

$$B = \{b_1, b_2, ..., b_n\}$$
$$VB = \{vb_1, vb_2, ..., vb_n\}$$

The B set is a set of business attribute, the VB set represents all of possible values of business attributes. For example, the account number of the destination, the amount of the transfer for the service that executes the transfer.

Definition 3. Other variables.

$$Boolean = \{true, false\},$$
$$R,$$
$$Time$$

The $Boolean$ set is a set of boolean values, the R - set represents real numbers and $Time$ is a set of all possible timestamps

Definition 4. Actions and set of all possible actions.

$$A = \{a_1, a_2, ..., a_n\}$$
$$a_n = (T, B, VT, VB, Boolean, Boolean, Time)$$

The A set is a set of all possible actions and the a_n represents an action. This is a collection of all actions provided by a WAF-protected system with information about the actions taken by the user at a specific time along with the specific business effect. An action is defined by sets of technical and business attributes along with their values, two boolean values which states whether an action has been authorized with additional mechanisms and whether it has been allowed, and action's timestamp.

Definition 5. Function additionalAuthRequired

$$additionalAuthRequired(A \times 2^A) \rightarrow Boolean$$

Function $additionalAuthRequired$ returns whether additional authorization is required for given action.

Definition 6. Return elements.
The functions $attrs$, $values$, $time$ and $passed$ are return the elements of an action tuple a_i passed as an argument.

Definition 7. History.
The function $history$ is defined as follows: $history(a_i, A_m) \rightarrow A_n$, where $a_i \in A$, $A_m \subset A$ and

$A_n \subseteq A_m : a_j \in A_n \iff (passed(a_j) \wedge (time(a_j) <= time(a_i)))$

Definition 8. actionSimilarityThreshold function.

$$actionSimilarityThreshold(a_i) = max(similarityThreshold(vbt_i), \forall vbt_i \in 2^{VB \cup VT} : vbt_i \subseteq values(a_i))$$

Function $actionSimilarityThreshold$ returns similarity threshold for given action based on its attributes. It is calculated as the maximum similarityThreshold for all possible subsets of action's values of attributes

Definition 9. additionalAuthRequired function

$$additionalAuthRequired(a_i, A_m) = \begin{cases} true, & similarity(a_i, A_m) <= \\ & actionSimilarityThreshold(a_i) \\ false, & otherwise \end{cases}$$

The function $similarity$ depends on the method to be used to compare actions. The algorithm of calculating the similarity of action is presented in Alg. 1. In this state of study we are using simply algorithm based on weighted wage. The values of the similarity function are taken later in the verification ($additionalAuthRequired$) that the action is similar enough to those previously performed that no further verification is needed.

Algorithm 1: Function that returns the similarity between the action and the history

```
SIMILARITY (a_i, A_n)
    inputs: Action and Action set
    output: Similarity factor
    tmpSum ← 0
    tmpWage ← 0
    foreach a_j ∈ history(a_i, A_n) do
        foreach attr_i ∈ attrs(a_j) do
            if value(attr_i, a_i) ≈ value(attr_i, a_j) then
                tmpSum ← tmpSum + wage(attr_i)
            tmpWage ← tmpWage + wage(attr_i)
    if tmpWage = 0 then
        return 0;
    return tmpSum ÷ tmpWage;
```

V. Conclusions

In the article we presented the weaknesses of Web Application Firewalls which use signature-based and rule-based static analysis. We presented a successful request forgery attack on the application defended by classic WAF when client has malware installed.

To protect against such attacks we introduced an approach, based on historical and behavioral analysis of user requests, which reduces the need for additional forms of authorization. After sufficiently collecting and analyzing user's history, the additional authorization appears only in the situation that actually requires it. Such approach increases the responsiveness and general feel of the application.

In the future work, we plan to implement the proposed system and check the efficiency and accuracy of it.

References

[1] Deepa, G., Thilagam, P.S.: Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Information and Software Technology 74, 160 – 180 (2016), http://www.sciencedirect.com/science/article/pii/S0950584916300234

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Rfc 2616, hypertext transfer protocol – http/1.1 (1999), http://www.rfc.net/rfc2616.html

[3] Garcia-Teodoro, P., Diaz-Verdejo, J., Tapiador, J., Salazar-Hernandez, R.: Automatic generation of {HTTP} intrusion signatures by selective identification of anomalies. Computers & Security 55, 159 – 174 (2015), http://www.sciencedirect.com/science/article/pii/S0167404815001297

[4] Garnaeva, M., Sinitsyn, F., Namestnikov, Y., Makrushin, D., Liskin, A.: Overall statistics for 2016. Special report, Kaspersky Lab (December 2016), https://kasperskycontenthub.com/securelist/files/2016/12/Kaspersky_Security_Bulletin_2016_Statistics_ENG.pdf

[5] Jazi, H.H., Gonzalez, H., Stakhanova, N., A.Ghorbani, A.: Detecting http-based application layer dos attacks on web servers in the presence of sampling. Computer Networks 121, 25 – 36 (2017), http://www.sciencedirect.com/science/article/pii/S1389128617301172

[6] Kar, D., Panigrahi, S., Sundararajan, S.: Sqligot: Detecting {SQL} injection attacks using graph of tokens and {SVM}. Computers & Security 60, 206 – 225 (2016), http://www.sciencedirect.com/science/article/pii/S0167404816300451

[7] Mazur, K., Ksiezopolski, B., Nielek, R.: Multilevel modeling of distributed denial of service attacks in wireless sensor networks. Journal of Sensors 2016 (2016), https://www.hindawi.com/journals/js/2016/5017248/

[8] Razzaq, A., Anwar, Z., Ahmad, H.F., Latif, K., Munir, F.: Ontology for attack detection: An intelligent approach to web application security. Computers & Security 45, 124 – 146 (2014), http://www.sciencedirect.com/science/article/pii/S0167404814000868

[9] Singh, K., Singh, P., Kumar, K.: Application layer http-get flood {DDoS} attacks: Research landscape and challenges. Computers & Security 65, 344 – 372 (2017), http://www.sciencedirect.com/science/article/pii/S0167404816301365

[10] Wichers, D.: OWASP Top Ten Project. https://www.owasp.org/ (2013), [Online; accessed 12-March-2017]

[11] Wueest, C.: Istr financial threats review 2017. Special report, Symantec (May 2017), https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-financial-threats-review-2017-en.pdf