UDC 004.056

Methods of counteraction of bypassing two-factor authentication using reverse proxy

A. V. Vlasenko[1, a], M. I. Ilin[1], I. V. Stopochkina[1, b]

[1] *National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Educational and Research Institute of Physics and Technology*

Abstract

The existing solutions for counteracting and preventing the interception of data and tokens of two-factor authentication are considered. Features that may indicate the presence of a silent reverse proxy server are chosen. It is proposed to analyze the information about additional time anomalies, which are usually created by the proxy server. The advantage of this approach is that the time characteristics information is generated on the client side, and the malicious proxy server cannot modify it. Machine learning methods were used to detect implicit signs of the presence of a proxy server. A new method of detecting a silent reverse proxy server that satisfies the following conditions is proposed: 1) the human factor is minimized, 2) use by an individual user is possible, 3) the method has acceptable impact on performance and can be used in real time.

Keywords: Two-factor authentication, silent reverse proxy servers, prevention

Introduction

Two-factor authentication is widely used as a means of protection against cyber attacks in cyberphysical systems, in mobile systems, in the organization of a reliable channel for communication of Internet of Things devices, and when working with web resources. Along with numerous works on the development of various methods and algorithms for two-factor authentication, a number of products are presented that allow to prevent the bypass of two-factor authentication (2FA). A significant portion of such attacks are phishing using silent reverse servers [1]. Appropriate products include Htrosbif and lbmap, which send a number of requests to the server and use the server's response signature database to identify proxy server software. But they cannot detect web server applications that have been redesigned for proxy servers [2],[3]. It is also used the http trace.nasl, which analyzes the HTTP response headers of the server, namely the Via header, and basing on this information concludes that there is a reverse proxy. However, silent reverse proxy servers modify response headers, which means that this tool is not effective in such cases. The Halberd tool detects web load balancers. The application sends the same request to the IP address of a potential malicious reverse proxy server and analyzes discrepancies in some headers of server responses [4]. The effectiveness of TLHS significantly depends on the type of software used for reverse proxy [5]. Using the WAFW00f tool, which aims to detect web server firewalls, results in any server firewall being defined as a reverse proxy that generates false-positive results; also the specifics of the tool often lead to false-negative results. The RevProbe tool [6] combines the approaches used in previous tools. However, it should be noted that only a few of these tools can be used to detect silent reverse proxies in real time, and the quality of their work depends on the technical component and requires skilled professionals to apply. Therefore, it is important to develop a method for detecting a silent reverse proxy at 2FA under the following conditions: 1) the human factor is minimized; 2) possible use by an individual user, and not only within the organization; 3) insignificant impact on speed and ability to work in real time.

1. Problem description

1.1. Authentication

The HTTP protocol is a stateless protocol, which means that each request to the server must contain some information with which it could identify an already authenticated client. The two most common approaches today are session-based authentication and token-based authentication [1].

In session-based authentication, after the user logs in, the server creates a session and stores the session data in the server's memory. The session ID is stored in cookies on the client side and sent to the server with each request. The received ID is compared with the session data on the server. At the end of the session, the data is deleted from the server.

Token-based authentication uses a different approach. During authentication, the server returns a JSON (JWT) web token to the client, which is stored on the client's side and sent with each request. The token contains all the necessary information for the correct identification of the client. No additional data is stored on the server.

[a] andvl@lll.kpi.ua
[b] irst-ipt@lll.kpi.ua

Modern web applications recommend the use of token-based authentication for several reasons:

- Easier support and bug fixes;
- Ability to create REST services;
- All necessary information is stored in the token itself and there is no need to store additional information on the server side.

These approaches also work with two-factor authentication, so if an attacker receives a token or cookie, they will be able to log in to the victim's account. In addition, there are other security issues. Cookies and tokens are stored either in the web repository or in cookies. JavaScript has access to the web repository in the same domain, which means that the token and cookies may be vulnerable to cross-site scripting (XSS) and cross-site request (CSRF).

Information can be intercepted using silent reverse proxy server. There are many developed applications that can be configured and used as a silent reverse proxy server. As an example, the following tools for penetration testing can be used: Modlishka, Muraena, Evilginx 2.

1.2. OAuth 2.0

During the development of web services, there was a problem of interaction between them, particularly how to grant an access to one web service to another using user rights. The first approach was to simply transfer the login and password, but this was quickly abandoned because there are risks of losing privacy. Therefore, a single standard has been developed that allows one service to securely use the data of another, OAuth. Then it was replaced by another standard, OAuth 2.0, and most web services use it. With this, applications exchange a sequence of data to obtain rights. This sequence of actions is often referred to as delegated authorization.

The sequence is as follows:

1) The Client wishes to provide service B with access to some information contained in service A. To do this, service B contains a forwarding form by which the Client enters the authorization server of service A.
2) The authorization server checks the Client and displays a form with a list of data and rights that will be granted to service B.
3) After confirmation, the authorization server of service A redirects the Client to service B.
4) After confirmation, the authorization server of service A redirects the Client to service B.
5) Service B directly contacts service A, send the data provided by the client.
6) Service A checks the data and responds to service B, providing it with an Access Token, with which you can obtain information without contacting the Client.

OAuth 2.0 was designed for authorization only. To implement authentication, there is OpenID Connect - a layer over OAuth 2.0, which adds information about the login and profile of the user logged in to the account. This allows to implement the ability to use a single login for multiple applications. This is a unified approach that simplifies the process for users.

But it has its drawbacks, in this work we highlight the most important: if an attacker takes possession of data to log in to an account that provides authentication services to other services, he will gain access to all these services. Currently, most authentication service providers automatically authenticate the user without requiring the user to enter a login or password and without verifying the second authentication factor.

1.3. Reverse proxies

The paper considers bypassing 2FA with the help of reverse proxy servers.

A proxy server is a server that divides a client-to-server connection into two TCP connections, one from client to proxy and the other from proxy to destination server. All traffic passes through it without changing. A proxy is explicit if you need to configure the IP address and proxy port on the client side (such as a browser). Otherwise, it is a transparent proxy server.

The reverse proxy server is purposefully installed by the web service owner to mediate HTTP communication from clients to web service servers. Thus, it is installed specifically for the web service, in contrast to the direct proxy server, which is often installed on the local network and mediates the connection of local clients.

To solve the problem of silent reverse proxy detection it is necessary to implement the following steps:

- Highlight the features that may indicate the presence of a silent reverse proxy;
- Develop a method for detecting a silent reverse proxy based on the machine learning method;
- Propose the concept of implementing user-oriented software to detect a reverse silent proxy;
- Develop appropriate software and perform a computational experiment.

2. Choice of features

Browsers automatically save and use queries and responses for further formatting into visual information. The data storage standard is HAR 1.2 (HTTP Archive).

The format is based on JSON, supports only UTF-8 encoding. In the browser, this information is contained in the Developer Tools, Network tab. There is an API through which this information can be obtained. In JavaScript, the interaction is implemented in the library devtools.network and window.perfomance.

For each sent request and received response, a HAR data structure is formed [7]. The list of received information includes a significant amount of information, including the time of interactions. The previous section discussed tools that use server responses to search for a proxy server. But this approach is not universal, proxy servers can modify responses.

One of the possible methods of information analysis may be the analysis of information about the time

of interactions. The proxy server creates additional time anomalies that can be tracked as all traffic passes through it. When interacting with a real target server, such anomalies should not be. The advantage of this approach is that the information about the time of interaction with the server is generated on the client side, the browser automatically generates it and the proxy server cannot modify it.

Consider the timings section:

- blocked [number, ms]; Time spent waiting in line for a network connection. Optional value.
- dns [number, ms]; The time required to determine the host name (DNS protocol). Optional value.
- connect [number, ms]; The time required to establish a TCP connection. Optional value.
- send [number, ms]; The time required to send an HTTP request to the server.
- wait [number, ms]; Response time from the server. Actually the time for getting the first byte.
- receive [number, ms]; The time required to read the entire response from the server. This is the time between receiving the first and last byte of information.
- ssl [number, ms]; The time required to validate the SSL / TLS certificate. Optional value.
- comment [line]. A comment that can be added by a user or application. Optional value.

All optional values will be set into -1 if not defined by the browser or if the information is taken from the cache. For a silent proxy detection application to work reliably, we have to use values that are always provided and will not be optional. Others can only be used as ancillary.

From these values theoretically useful for our purposes can be send, dns, connect, wait, receive and ssl. Additional measurements based on the information obtained will also be useful. These can be the values of Internet speed, the number of nodes to the target server and the response time of the target server to ICMP packets (hereinafter referred to as hop time).

An analysis of send distribution showed that all send values are zero. It means send parameters are not big enough to be written into HAR (all the numbers are rounded, and values less than 1 ms are rounded to 0 ms). The information on dns,connect, receive and ssl measurements is not complete for the final conclusions. They include some nonzero parameters, but other values are obtained from browser cache or are not big enough.

Values are calculated only for the first request to the server. In actual use, it is not possible to obtain these values for each request without clearing the cache. At this stage, it is impractical to use these parameters to build a classifier. The value of receive does not give a practical meaning because it depends on the capacity of the target server, how quickly it will form a final response. There may also be delays between receiving different data packets due to the quality of the communication channel.

The speed of the internet connection affects the response time. The number of nodes is measured independently of the query data based on the server address.

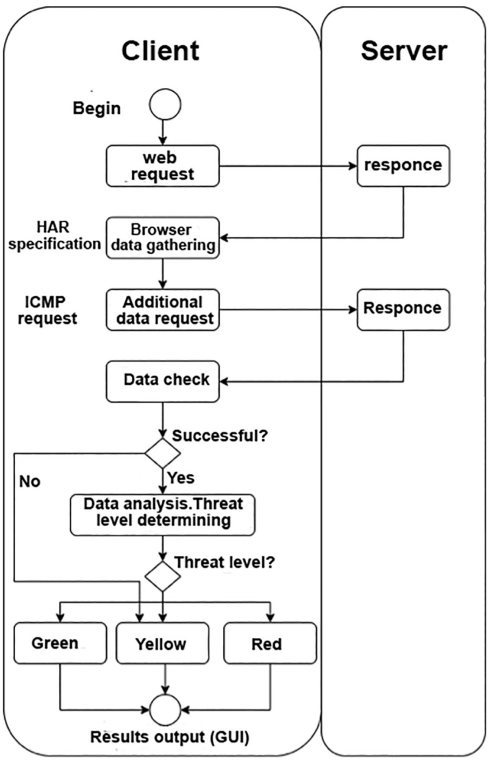

Fig. 1. Scheme of client-server interactions

This also applies to hop time. Considering the wait and hop options will allow you to find the proxy server in the middle.

The wait parameter was calculated. It based on the response of the destination server, even if there is a proxy in the middle.

The hop parameter is calculated for the end node of the connection. In the absence of a proxy server, the time will be calculated based on the response of the real server. And if there is a proxy server, then this parameter will be calculated basing on its response. This is where the mismatch between the wait and hop parameters occurs.

The measurements of the parameters can be implemented using libraries in Python.

Of all the files that the browser receives upon request, we are interested in the first. Its characteristic feature (this applies to all requests, regardless of the target server) is the type of Initiator. This parameter will correspond to the value of Other. This means that the initiator of this request is the client, and this request led to the appearance of other, additional.

The main disadvantage of finding a proxy server using time analysis is its dependence on its location. So if the malicious proxy is on the same subnet as the real proxy server, the hop time may not be different at all. But even in this case, there will be changes in the wait parameter. To solve this problem, we can use machine learning methods.

3. Silent reverse proxy detection method

The scheme of the method is presented in Fig. 1.

It is proposed to use the classic three-level security model, in which «green» will be responsible for the

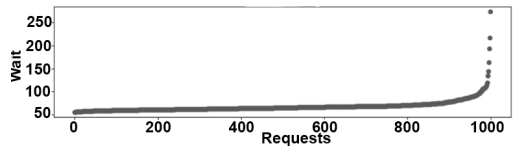

Fig. 2. Wait parameter changes

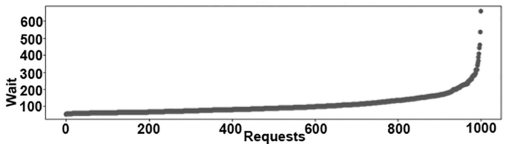

Fig. 3. Wait parameter changes while measuring internet speed

most secure state, and «red» for the presence of a proxy server, «yellow» - for a potentially dangerous situation. In this way, we will be able to reduce the level of errors of the second kind, due to the poor quality of Internet communication.

4. Setting up a computational experiment

For the experiment the Evilginx2 reverse proxy server was configured [8]. Google was chosen as the target server, namely the accounts.google.com authorization service. To automate the data collection process, a Python program was developed using Selenium Webdriver to work with the browser, a *s*peedtest-cli library to measure Internet speed, and an icmplib library to trace the path to the destination node. To test the feasibility of using the parameters, three test measurements were made: one control using only the information provided by the browser (Fig. 2), the second with the addition of Internet speed measurements (Fig. 3) and one with the addition of path trace measurements (Fig. 4). This was done to determine the effect of additional measurements on performance.

As a result of elimination of unsuitable values the received data set contains values of parameters id, speed, hop, wait, quantity, proxy, where id - request number, speed - Internet speed, hop - time of response of the final node to ICMP request, wait - time to receive the first byte, quantity - the number of nodes to the final. The proxy parameter corresponds to the presence of a proxy server. The obtained parameter distributions with and without a proxy server for several parameters are shown in Fig. 5, 6, 7, 8.

It can be seen that the hop parameters are not very different, but the wait parameter has increased significantly. The correlation matrix (Table 1) shows interdependence of parameters.

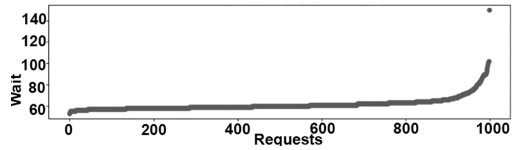

Fig. 4. Wait parameter changes while tracing the path

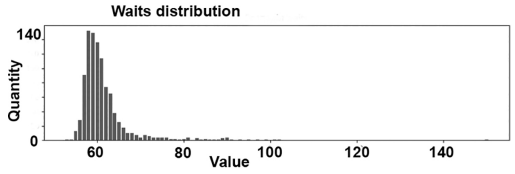

Fig. 5. The wait parameter distribution for queries without a proxy

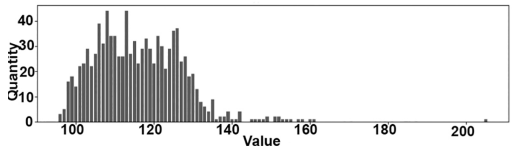

Fig. 6. The wait parameter distribution for proxy requests

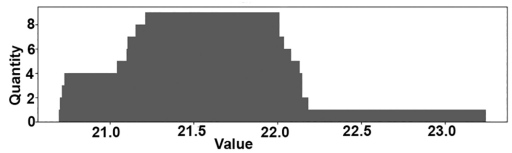

Fig. 7. The hops time distribution for queries without a proxy

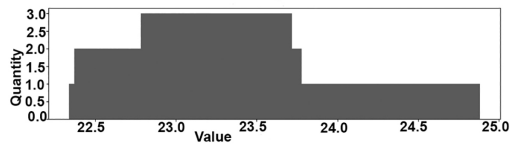

Fig. 8. The hops time distribution for proxy requests

Table 1. Correllation matrix

	Wait	Hop	Proxy
Wait	1	0,93	0,95
Hop	0,93	1	0,98
Proxy	0,95	0,98	1

Table 2. Accuracy of results

	SVM	RF	LR
Accuracy	93,2	97,5	94,6

Table 3. Characteristics of work in real time

	Mode, ms	Average, ms	Deadline, ms
Measurements	22	24	30
Classification	18	20	50
In general	40	44	80

5. Using machine learning for the problem

Classifiers were trained on the obtained data for detecting the presence of a proxy server. The results with the accuracy are shown in Table 2. Support vector machine, Random Forest, Logistic Regression algorithms were applied.

The false positives may occur due to failure in connection channel. There may be also false negative results due to location of the proxy server in the subnet of the destination server. The difference between wait parameters will be smaller than usual.

6. Realization

The implementation of the method is proposed as an extension for a web browser. The advantage of this solution is the ease of use by the average user. General scheme of work:

1) Obtaining data.
2) Data analysis.
3) The final decision.

Software components are:

1) Data collection component.
2) Data analysis component.
3) Graphical interface.

The HAR browser specification with the window.perfomance library of the JavaScript programming language and the icmplib library in the Python programming language were used for data collection. Logistic regression method, binary classification was used for data analysis. This is because although the Random Forest method gave the best result in terms of accuracy, logistic regression features can be used to improve the application and determine different levels of danger, as the output gives the probability of belonging to a class. In particular, the qualitative score «green» corresponds to the probability of having a proxy server in the range (0; 0.05], «yellow» - (0.05; 0.15], «red» - (0.15; 1]). Yellow level is also assigned in case of failure in data collection. The characteristics of the method in real time (Table 3) indicate the efficiency of the proposed approach.

7. Conclusions

The proposed method of silent reverse proxy server detection, in contrast to existing tools, can be used to protect data in real time, and does not require special user training. However, the method is not focused on identifying the exact structure of malicious infrastructures, which may be the subject of further research. The method slightly depends on the technical component of the reverse server. However, the training sample should be expanded to include technically different proxy servers.

8. References

References

[1] A. V. Vlasenko and I. V. Stopochkina, "Penetration testing in sphere of social engineering with 2fa bypass and prevention means.," in *Materials of All-Ukrainian scientific-practical conference «Theoretical and applied problems of physics, mathematics and computer science»*, pp. 256–260, Institute of Physics and Technology, 2020.

[2] "Htrosbif.- [electronic resource]." `http://freshmeat.sourceforge.net/projects/htrosbif`.

[3] "HTTP TRACE / TRACK methods allowed.- [electronic resource]." `https://vulners.com/nessus/XST_HTTP_TRACE.NASL`.

[4] "Traceroute-like HTTP scanner.- [electronic resource]." `https://www.agarri.fr/blog/archives/2011/11/12/traceroute-like_http_scanner/index.html`.

[5] "Halberd.- [electronic resource]." `https://github.com/jmbr/halberd`.

[6] A. Nappa, R. F. Munir, I. K. Tanoli, and et al., "Revprobe: Detecting silent reverse proxies in malicious server infrastructures.," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, vol. 4 of *5*, pp. 101–112, ACSAC 16, 2016.

[7] "HAR 1.2 spec.- [electronic resource]." `http://www.softwareishard.com/blog/har-12-spec/`.

[8] "Evilginx2.- [electronic resource]." `https://github.com/hash3liZer/evilginx2`.