OVERSCAN: OAuth 2.0 Scanner for Missing Parameters

Karin Sumongkayothin(✉), Pakpoom Rachtrachoo, Arnuphap Yupuech, and Kasidit Siriporn

Faculty of Information and Communication Technology, Mahidol University, Salaya, Thailand
karin.sum@mahidol.ac.th,
{pakpoom.rac,arnuphap.yup,kasidit.sir}@student.mahidol.ac.th

Abstract. The websites are developed rapidly and wildly used by people around the world. The main reason is the increase of the immense number of internet users, which results in the security control of accessing sensitive information is necessary. The authorization server as the one security aspect which controls the access permission to the system. Many authentication protocols were proposed to meet these functional requirements. The open-standard authorization (OAuth) protocol is one of the well-known solutions widely used. However, many developers still misuse this protocol, which can cause security breaches. This paper proposes a tool named *OVERSCAN*, which is an OAuth2.0 scanner for misused or missing parameters. The experiments of using OVERSCAN have been conducted over 45 samples supporting OAuth2.0 protocol. The results show that 84.4% of samples lack significant parameters which can cause security problems.

Keywords: OAuth · Vulnerability scanner · Network protocol security

1 Introduction

The growth of internet user has been dramatically increasing since it was introduced in the last few decades. To achieve the information security preservation, the access grant over the particular information is necessary. The authorization is the process to determine the user access levels. Many mechanisms can serve the authorization flow, and one of them is Open Authorization (OAuth). OAuth is an authorization framework which is wildly used to grant access permission through a trusted third-party service. Even though the framework is officially provided under RFC [7], the incorrect OAuth implementation still exists until the present. The inaccurate implementation such as missing some particular OAuth parameters or HTTP headers may prompt security concerns such as stealing sensitive information, gaining the illegal access, and identity theft [4,13]. The OAuth2.0 framework defines the role of components as:

– Resource Owner: the user who delegate access to his protected information.

J. K. Liu and X. Huang (Eds.): NSS 2019, LNCS 11928, pp. 221–233, 2019.
https://doi.org/10.1007/978-3-030-36938-5_13

- Client (aka. Relying Party): the service API or application which asks for permission to access the *resource owner*'s protected information.
- User agent: the intermediary that is used by the *resource owner* to interact with the *client*.
- Authorization server (aka. Identity Provider): the server which grants the scoped access to the *client* to access the protected information on behalf of *resource owner*.
- Protected resource server: the server that holds the protected information which can be accessed by using provided grant from the *authentication server*.

The framework supports many *grant types* which each type has a different flow to able access the protected resource. However, the most commonly used are *implicit* and *authorization code* grant type. The difference between the two mentioned grant types are as illustrated in Figs. 1 and 2.

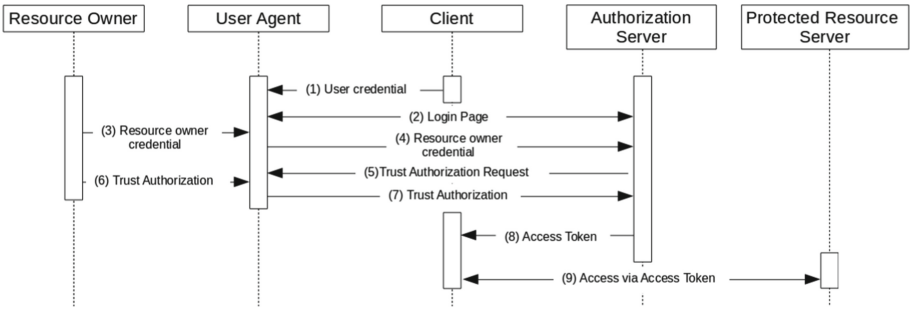

Fig. 1. Implicit grant flow

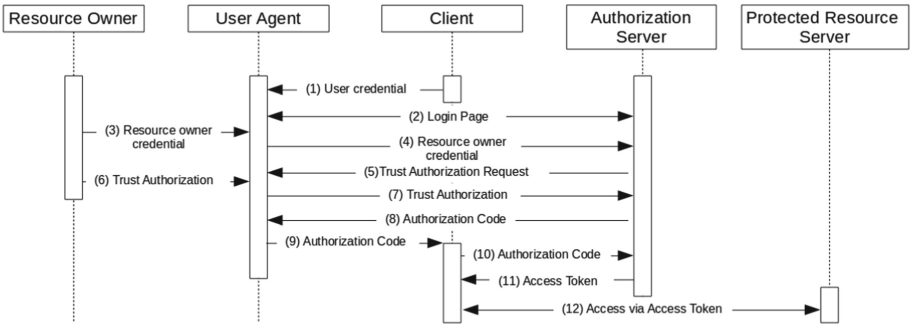

Fig. 2. Authorization code grant flow

The clear difference between implicit and authorization code grant is how the client acquis the access token. In implicit grant type, the client directly receives

access token after the resource owner confirming the trust authorization (step (7) and (8) of Fig. 1). In authorization code grant type, the *authorization code*(aka. code) is sent back to the user agent before passing on to the client (step (8) of Fig. 2). The client then exchanges the code with the access token that was created by the authorization server (step (9) of Fig. 2). Although the implicit grant is clearly better in the term of performance, authentication code grant is recommended when requiring higher security.

In this paper, we propose the tool named OVERSCAN, which is the scanner for the OAuth2.0 vulnerabilities focusing on the web application. It was implemented with the JAVA language as the Burpsuit[1] extension which covers the vulnerabilities listed in the RFC6819 [10].

1.1 Related Work

The OAuth2.0 framework [7] was introduced as the guideline for implementing the flow to delegate access to the unauthorized client. It was classified as a double-redirection protocol [5] since it redirects the request back and forth between client and authorization server through the user-agent application. The first redirection is when the client sends the redirects URI to make the user agent forwards its request to the authorization server. The second redirection takes place after confirming the trust authorization message. The response message is redirected to the client or authorization server depending on the grant type is being used. Since the flow consists of redirections, the security flaws may exist if it is not well-implemented. OAuth 2.0 Threat Model and Security Considerations [10] was published to provide the additional security considerations in OAuth 2.0. It shows the possible security flaws in OAuth flow under incorrect parameter configurations.

Pai et al. [12] proposed the formal method based on knowledge flow analysis [14] to verify the security of OAuth2.0 protocol. As a result, it emphasizes the existence of a security flaw in OAuth client credentials flow. Chari et al. [3] presented another security analysis method by using Universal Composability Security Framework [2]. The universal composability paradigm can guarantee the strong security properties of the protocol, although it was used as a component of an arbitrary system. Their analysis was focused on OAuth2.0 authorization code flow. It shows the necessity of using the SSL like functionality to protect the communication channel between the client and the authorization server. They also show that the session identifier does not need to be decided in advance, which can mitigate the possibility of the attacks such as the session hijacking or session swapping. Feng and Sathiamoorthy [15] introduced the method to analyze the vulnerability of the OAuth 2.0 framework using an attacker model. The attacker model consists of four modules representing the type of attacks which are monitoring attack, replay attack, phishing attack, and impersonation attack. Their experiment was conducted focusing on the data transmission of the user-agent & authorization server, and user-agent & client. The results show

[1] https://portswigger.net/burp.

that the OAuth2.0 framework is susceptible to such simple attacks as cross-site request forgery attack, replaying attack, and network traffic interception. The root cause is that the framework did not explicitly define which component affects the security of the protocol. For example, there were no recommendations of using TLS to protect the callback endpoints, enforcing process to ensure the security of the client, nor limiting of multiple uses of authorization code.

In analytical terms of practical usage, Argyriou et al. [1] studied the possible mechanism causing the flaws in the OAuth 2.0 framework. Because the framework does not clearly define the formal standard of communication primitive between the user-agent and the client; the authentication method, therefore, depends on the decision of the developer. The analysis was done by investigating the communication between the resource owner and the authorization server based on the OAuth 2.0 framework. It shows that the misconfiguration in implicit grant flow and authorization code flow may lead to many security flaws such as cross-site request forgery, session wrapping, and mixing redirect endpoint. The [6,8] reiterated that the security issues of the OAuth 2.0 flow were caused by the implementation, not from the framework. They also stated that missing or incorrect using some significant parameters such as a state-parameter or X-FRAME-OPTIONS in HTTP header, will cause the security risk.

Zhou et al. [16] implemented the tool named SSOScan for verifying the communication characteristics of the application using Facebook Single Sign-On APIs. It automatically identifies the risks that occur during the authentication process. SSOScan can detect four vulnerabilities which are access token misuse, signed requests misuse, app_secret parameter leakage, and user OAuth credential leakage. The vulnerability analysis proceeds in two ways consisting of the simulation attack for checking the access token and signed request misused, and passive monitoring to identify the credential leakage. As a result, 20.3% of websites using Facebook SSO are vulnerable, where users are unable to login due to 2.3% implementation error. Another OAuth2.0 testing tool, OAuthGuard, was proposed by Li et al. [9]. This tool focuses on analyzing the vulnerabilities of web application utilizing Google Sign-in. By scanning 137 sample sites using OAuthGuard, it shows that 40.9% of samples have at least one serious vulnerability, where 9.5% have an insecure implementation.

Unlike the existing tools, OVERSCAN is designed to compatible with any web application supporting OAuth 2.0. It serves as the free extension of Burp Suite Community Edition[2] to identify the possible threats caused by insecure implementation.

1.2 Contribution and Paper Organization

The most vulnerabilities in OAuth services caused by a faulty client design as has been stated in [6]. Since the client is an application, it is somewhat unmanageable and hard to identify the configuration. Fortunately, the information sent back

[2] https://portswigger.net/.

and forth between the client and authorization server must be through the user-agent. It allows to intercept and investigate the information at this point.

In this paper, we propose the OAuth 2.0 vulnerability scanner named OVERSCAN. It works as a proxy which will intercept the incoming and outgoing packets of the internet browser. It focuses on identifying the missing significant parameter and HTTP header, which lead to the security issue. The main contributions of this paper are:

- Propose new scanning tool, OVERSCAN which works nicely to any clients and authorization servers.
- Describe the conceptual design and implementation of OVERSCAN.
- Conduct the experiments over 25 websites on vary authorization servers (45 scannings in total) to identify the possible missing parameters that cause of weak security.

The rest of the paper is organized as follows. In Sect. 2, we provide an overview of OVERSCAN design and construction. Then the misused parameter and related vulnerabilities, that can be identified by OVERSCAN are described in Sect. 3. We deliver the analysis and results of the experiment conducted on the sample web applications in Sect. 4. We give a discussion of OVERSCAN limitation in Sect. 5. We conclude the paper in Sect. 6.

2 Design and Construction Overview of OVERSCAN

OVERSCAN is designed in the purpose of analyzing data transmitted through the web browser during the access-token request process. Therefore, capturing the information that is sent in and out across the browser is necessary for the operation. This section we describe the details of OVERSCAN design and implementation.

From the objective to analyze the OAuth traffic to discover the vulnerability, we implement OVERSCAN as the free extension of a Burp Suite Community Edition. Burp Suite Community Edition (in short Burp) is the free graphical interception proxy which can capture all requests and responses between the browser and target applications. It allows the user to extend the Burp's functionality by adding the additional code called the *extension*. The traffic pass through the browser will be intercepted by Burp then checked by OVERSCAN for the possible security issue. OVERSCAN operation consists of four phases as illustrated in Fig. 3.

2.1 Traffic Classification

OVERSCAN will retrieve all traffic captured by Burp and then proceed the classification as of Fig. 2. It first checks whether the traffic is request or response and then classifies whether it is under OAuth protocol or not. Only OAuth related message will be highlighted and sent to scan for vulnerabilities. OVERSCAN supports investigating the vulnerability that may occur during implicit or

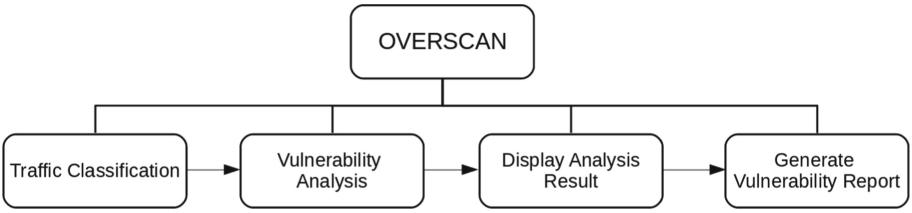

Fig. 3. OVERSCAN operation

authorization grant request. To distinct these two requests from the other messages, the parameter *response_type* must be observed. It will contain the value *"token"* when it is implicit grant request while *"code"* for the authorization code grant request (Fig. 4).

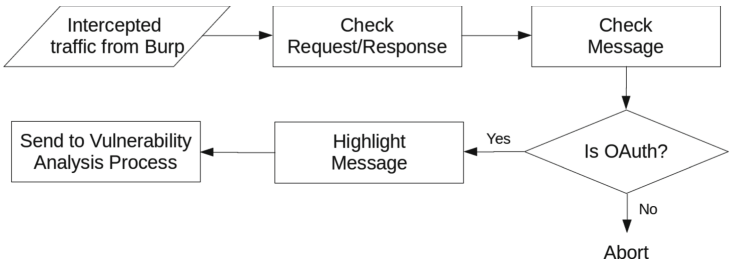

Fig. 4. OVERSCAN traffic classification

2.2 Vulnerability Analysis

What is used to verify for vulnerabilities consists of two parts: OAuth missing parameter and missing secure parameter in HTTP headers. The target message is checked based on the countermeasure methods mentioned in RFC6819. Then the scanning results will be passed to display by Burp as Fig. 5. The details of missing parameters and supported vulnerabilities can be found in Sect. 3.

2.3 Display Analysis Result

OVERSCAN highlights and adds the comment to the vulnerable OAuth traffic displayed in Burp Proxy HTTP History. The highlight color is according to the severity levels defined by the Common Vulnerability Scoring System (CVSS)[3]. The severity can be categorized into three levels: High (Orange), Medium (Yellow), and None (Green). The example of the display result is as shown in the Fig. 6.

[3] https://www.first.org/cvss/v3.0/specification-document.

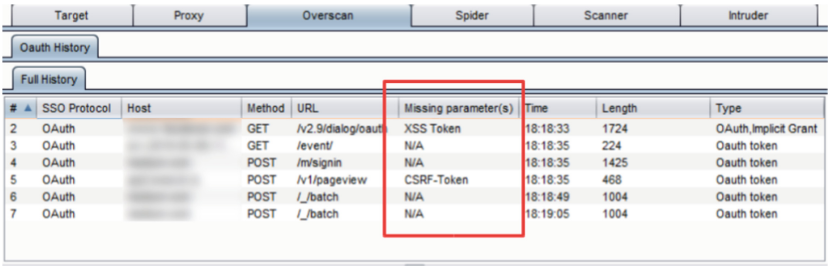

Fig. 5. Vulnerability analysis results

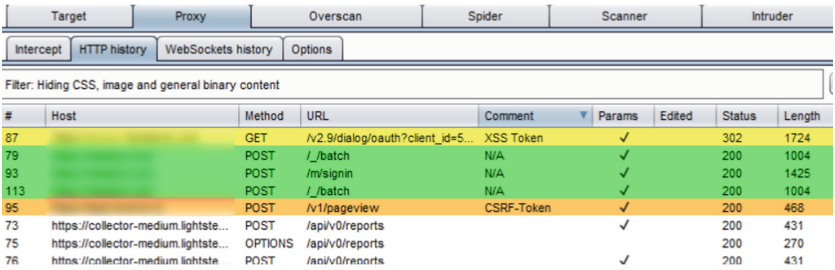

Fig. 6. Display analysis results (Color figure online)

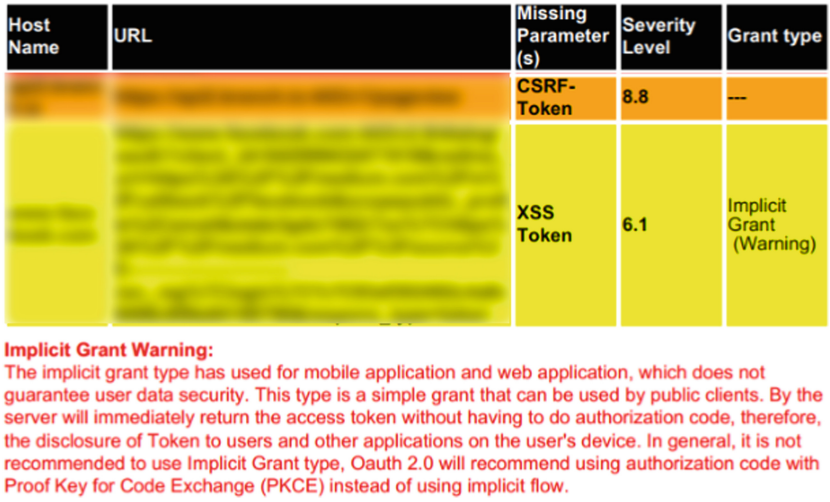

Fig. 7. Vulnerability report

2.4 Generate Vulnerability Report

To arrange the information in an orderly manner and more understandable, OVERSCAN summarizes the found vulnerabilities in the form of document report. The report will give the details of the vulnerability and the suggestion to migrate the issues. The report can be generated in either HTML or PDF format. Figure 7 shows the example of the report generated from the vulnerabilities found by OVERSCAN.

3 Supported Vulnerabilities

Parametric usage methods are particularly relevant to OAuth protocol security. Missing some parameters may result in allowing the adversary to obtain the credential information. The list of parameters supported by OVERSCAN for security checking is as according to Table 2.

Table 1. List of parameters for security checking

Missing parameter	Possible threat	Severity level
X-CSRFToken	– CSRF attack against redirect URL (Authentication code & Implicit grant)	8.8 High (CVSS 3.0)
State-Parameter	– CSRF attack against redirect URI (Authorization code & Implicit grant) – DoS using manufactured authorization "codes"	7.8 High (CVSS 3.0)
Redirect_URI (HTTPS for redirect URI)	– Authorization code phishing – User session impersonation	6.1 Medium (CVSS 3.0)
X-XSS-Protection	– Authorization codes can be stolen through vulnerable client	6.1 Medium (CVSS 3.0)
X-Content-Type-Options	– Authorization codes can be stolen through vulnerable Client	6.1 Medium (CVSS 3.0)
X-Frame Options	– Clickjacking attack against authorization	4.7 Medium (CVSS 3.0)

3.1 X-CSRFToken

X-CSRFToken is an HTTP token to against Cross-Site Request Forgery (CSRF) attack. It is also known as CSRF-Token. This token is a large random unique number which is unpredictable for each authentication request. It is associated with the HTTP header to ensure the validity of the source of information.

3.2 State Parameter

State Parameter is the parameter preserving the state information of authentication procedures which allows the user to restore the previous state of the application. It is also useful for CSRF attack mitigation on the redirection endpoint. The value of this parameter is unique and non-guessable, which will be generated during the initial request. To validate the response, the recipient must confirm that the state-parameter of request and response message must be the same value. Since OAuth was classified as double-redirection protocol, it is susceptible to the CSRF attack without state parameter.

Another benefit of using state parameter is Denial of Service (DoS) mitigation. The attack scenario is when the attacker floods the valid URIs with a random authorization code to the client. Generally, the client will forward all the received messages to the authorization server. Due to a large number of HTTPS connections, it can cause the server out of service. However, when the state parameter is implemented; the client will drop the message containing the invalid state parameter. The attacker needs the right state parameter in order to successfully attack, which results in decreased attack effectiveness.

3.3 Redirect_URI

Redirect_URI is a parameter to change the direction of the traffic to the next endpoint of the flow. The URI of all endpoints should be integrated the SSL/TLS protection (HTTPS) to prevent authorization code phishing and user session impersonation.

3.4 X-XSS-Protection

They also stated that missing or X-XSS-Protection is a parameter in the HTTP response header that stops pages from loading when cross-site scripting (XSS) attacks are detected. This vulnerability affects when the attacker discovers the XSS flaw in the client. The attacker can inject the script (e.g. Javascript) then lures the user to send the request containing the malicious redirect URI. Since the redirect URI includes a malicious script, the attacker can steal the authorization code or access token from the user. By the fact that using pre-configured redirect URI instead of a dynamic one may solve this issue, it cannot guarantee that every server will follow this configuration. By using X-XSS-Protection parameter, we can make sure that the malicious script will not be loaded. There are four ways to configure X-XSS-Protection parameter, which are:

- **X-XSS-Protectoin: 0:** disable XSS protection.
- **X-XSS-Protectoin: 1:** enable XSS protection and the browser sanitizes the page if cross-site scipt was detected.
- **X-XSS-Protectoin: 1; mode = block:** enable XSS protection and the browser prevents the page form rendering if cross-site scipt was detected.
- **X-XSS-Protectoin: 1; report =** ⟨**report-uri**⟩**:** enable XSS protection and the browser sanitizes the page and sends the report to defined URI.

3.5 X-Content-Type-Options

Another solution to mitigate the impact of XSS attacks rather than X-XSS-Protection is to use X-Content-Type-Options parameter. X-Content-Type-Options is the element in HTTP header which can prevent the MIME sniffing for sending XSS attack by the attacker. MIME Sniffing is the feature that the web browser uses to examine the downloaded asset content to determine the file format. However, MIME Sniffing can cause a security issue when the attacker disguises an HTML file as a valid file type. It allows the attacker to successfully bypassing the protection to upload the malicious code to the server. It can cause an XSS attack when the web browser renders the malicious HTML file. To mitigate the attack, providing X-Content-Type-Options with *nosniff* option will disable MINE Sniffing functionality. By disabling MINE Sniffing functionality, the web browser will no longer analyze the received content.

3.6 X-Frame Options

As mentioned in [10], Clickjacking is one of the malicious methods to let the attacker steals the user's authentication credentials. The malicious site may construct transparent iFrame with an invisible button wrapping around the significant locations (e.g Authorize button). Once the user clicks on that location, he did click the hidden button then sends the user's credentials to the attacker. X-Frame Options is the security element in HTTP header, which provides the feature of Clickjacking prevention. X-Frame Options can contain three values, which are:

- **DENY:** disable the loading of the page in a frame.
- **SAMEORIGIN:** allows the page to be loaded in a frame on the same origin as the page itself.
- **ALLOW-FROM ⟨uri⟩:** allows the page to be loaded only in a frame on the specific URI.

4 Experimental Analysis

The experiment was conducted over 45 samples of web application supporting an OAuth 2.0 protocol. The sample group consists of the local and international websites that use the service from the different OAuth authorization servers: Facebook, Google, and other servers. In this experiment, we focus on examining two aspects: the grant type used and the missing parameters that may cause a security weakness. We found that 42% out of the samples still use Implicit grant type to delegate the user access right. The numbers are high even though using the implicit grant type was reported as causing the security susceptible [11]. In the aspect of using the security-related parameters, the experimental yield the result as Fig. 8. The most parameter that was missing from the sample group is X-XSS-Protection, accounting for 48.9% of the total samples. The second

highest of missing parameters obtained from the experiment is X-Content-Type-Options, representing 42.2%. Next is X-Frame Options, which accounts for 37.8% of the total. The remaining three are X-CSRFTOKEN, State Parameter and Redirect_URI which represent 15.6%, 11.1% and 2.2%, respectively. Interestingly, the top three missing parameters are the secure parameter used in the HTTP header, which X-Frame Options is one of them. It was specified by [7] that is necessary for the security of the OAuth application. Nonetheless, there are over 37% of websites from the experiment that does not use this parameter. Furthermore, one of the experimental results shows that the redirection endpoint does not support SSL/TLS, which may lead to serious security flaws such as the session hijacking.

By the fact that some samples are free of the missing parameter, while some have more than one. Figure 9 demonstrates the number of samples containing a different number of missing parameters. More than 50% of samples contain one to two missing parameters necessary for security purposes, and only 13.3% possess all the parameters that we have considered.

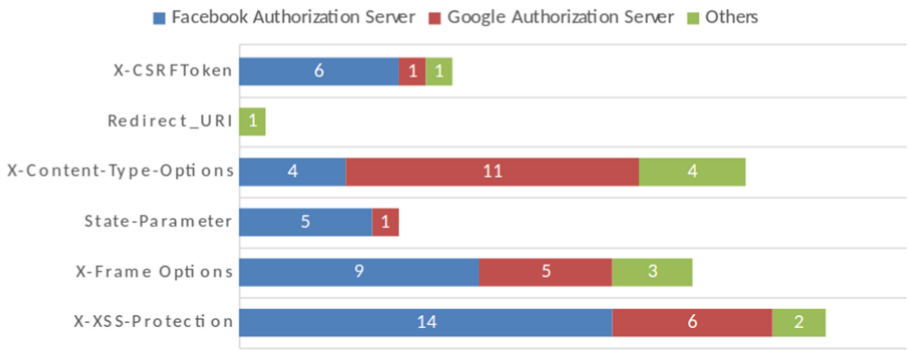

Fig. 8. Missing parameter

5 Features and Limitations

OVERSCAN was designed to compatible with a variety of authorization servers that support OAuth 2.0 protocol. It provides the abilities to determine the vulnerability, which possibly occurs during OAuth authentication flow via the parameters missing from the transmitted message. Table 2 shows the comparison of the supporting features with the other two OAuth vulnerability scanners: SSOSCAN and OAuthGuard.

Although OVERSCAN supports most of the vulnerabilities detected by the other two scanners, it cannot analyze the message directly sent between the client and authorization server. Since it uses Burp as the host for the operations, it can only intercept the message sent through the web browser. Therefore, it impossible for OVERSCAN to verify the security flaws happening after the beginning of

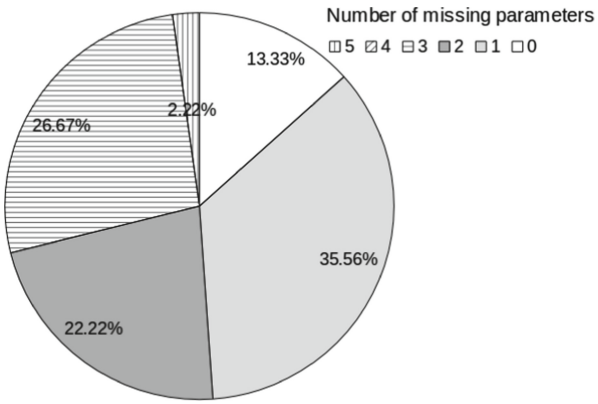

Fig. 9. Number of samples which have 0 to 5 missing parameters

Table 2. Comparison of supported features

Supported feature	OAuthGuard	SSOScan	OVERSCAN
Support a wide range of identity providers			✓
Stand alone application	✓	✓	
Identify OAuth grant type			✓
Protection	✓		
Instant warning message	✓	✓	
Generate technical report	✓		✓
CSRF vulnerability detection	✓		✓
Impersonation attack detection	✓		✓
ClickJacking attack detection			✓
Unsafe redirect URI detection	✓	✓	✓
Client secret leakage detection		✓	
Authorization code leakage detection	✓	✓	✓

the Access Token exchange process. Aside from that, OVERSCAN is unable to support protection and instant warning inline with the above reasons. Instead of giving the instant warning message when anomalies were found, OVERSCAN will summarize them in the readable technical report.

6 Conclusion

OVERSCAN is the anomaly detector which aims for identifying the missing significant security parameters used during the OAuth 2.0 grant request process. It can determine missing parameters which result in threats such as CSRF attack, confidential data leakage, and session stealing. As the experimental results of using OVERSCAN over 45 samples, only 15.56% are free of missing parameter.

The parameter which most frequently missing is XSS protection while insecure Redirect URI is the least.

References

1. Argyriou, M., Dragoni, N., Spognardi, A.: Security flows in OAuth 2.0 framework: a case study. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10489, pp. 396–406. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66284-8_33
2. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols, vol. 2000, pp. 136–145, November 2001. https://doi.org/10.1109/SFCS.2001.959888
3. Chari, S., Jutla, C.S., Roy, A.: Universally composable security analysis of OAuth v2.0. IACR Cryptology ePrint Archive, vol. 2011, p. 526, January 2011
4. Chen, E.Y., Pei, Y., Chen, S., Tian, Y., Kotcher, R., Tague, P.: OAuth demystified for mobile application developers. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014, pp. 892–903. ACM, New York (2014). https://doi.org/10.1145/2660267.2660323http://doi.acm.org/10.1145/2660267.2660323
5. Corella, F., Lewison, K.P.: Security analysis of double redirection protocols (2011)
6. Ferry, E., O'Raw, J., Curran, K.: Security evaluation of the OAuth 2.0 framework. Inf. Comput. Secur. **23**, 73–101 (2015). https://doi.org/10.1108/ICS-12-2013-0089
7. Hardt, D.: The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/rfc6749.txt
8. Li, W., Mitchell, C.J.: Security issues in OAuth 2.0 SSO implementations. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 529–541. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_34
9. Li, W., Mitchell, C.J., Chen, T.: OAuthguard: protecting user security and privacy with OAuth 2.0 and OpenID connect. arXiv abs/1901.08960 (2019)
10. Lodderstedt, T., McGloin, M., Hunt, P.: OAuth 2.0 threat model and security considerations. RFC 6819, RFC Editor, January 2013
11. Lodderstedt, T., Bradley, J., Labunets, A., Fett, D.: OAuth 2.0 security best current practice. Internet-Draft draft-ietf-oauth-security-topics-09, IETF Secretariat, November 2018. http://www.ietf.org/internet-drafts/draft-ietf-oauth-security-topics-09.txt
12. Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S.: Formal verification of OAuth 2.0 using alloy framework. In: 2011 International Conference on Communication Systems and Network Technologies, pp. 655–659, June 2011. https://doi.org/10.1109/CSNT.2011.141
13. Richer, J., Sanso, A.: OAuth 2 in Action. Manning Publications, New York (2017)
14. Torlak, E., van Dijk, M., Gassend, B., Jackson, D., Devadas, S.: Knowledge flow analysis for security protocols. CoRR abs/cs/0605109 (2006). http://arxiv.org/abs/cs/0605109
15. Yang, F., Manoharan, S.: A security analysis of the OAuth protocol. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), pp. 271–276, August 2013. https://doi.org/10.1109/PACRIM.2013.6625487
16. Zhou, Y., Evans, D.: SSOScan: automated testing of web applications for single sign-on vulnerabilities, August 2014