



Machine learning approach to vulnerability detection in OAuth 2.0 authentication and authorization flow

Kindson Munonye¹ · Martinek Péter¹

Accepted: 21 April 2021 / Published online: 13 May 2021
© The Author(s) 2021

Abstract

Technologies for integrating enterprise web applications have improved rapidly over the years. The OAuth framework provides authentication and authorization using the users' profile and credentials in an existing identity provider. This makes it possible for attackers to exploit any vulnerability arising from exchange of data with the provider. Vulnerability in OAuth authorization flow allows an attacker to alter the normal flow sequence of the OAuth protocol. In this paper, a machine learning-based approach was applied in the detection of potential vulnerability in the OAuth authentication and authorization flow by analyzing the relationship between changes in the OAuth parameters and the final output. This research models the OAuth protocol as a supervised learning problem where seven classification models were developed, tuned and evaluated. Exploratory Data Analytics (EDA) techniques were applied in the extraction and analysis of specific OAuth features so that each output class could be evaluated to determine the effect of the identified OAuth features. The models developed in this research were trained, tuned and tested. A performance accuracy above 90% was attained for detection of vulnerabilities in the OAuth authentication and authorization flow. Comparison with known vulnerability resulted in a 54% match.

Keywords Machine learning · Authentication and authorization · OAUTH 2.0 · OpenID Connect · Vulnerability detection · Classification · Exploratory data analysis

Abbreviations

AG	Authorization Grant
AS	Authorization Server
AT	Authentication Token
AUROC	Area Under the ROC
CNN	Convolution Neural Network
EDA	Exploratory Data Analysis
FA	Factor Analysis
FSM	Finite State Machine
GBC	Gradient Boosting Classifier
HTTP	HyperText Transfer Protocol
IdP	Identity Provider
MiM	Man in the Middle
OAUTH	Open Authorization
OIDC	Open ID Connect
OWIN	Open Web Interface

PCA	Principal Components Analysis
RNN	Revolutional Neural Network
RO	Resource Owner
RP	Relying Party
SSO	Single Sign On
SVM	Support Vector Machine
URL	Universal Resource Locator

1 Introduction

The OAuth protocol provides a framework for a client application to grant access to its resources (data and services) for other web applications based on the user's consent [1].

Recent authentication and authorization(AA) approaches include passwords, graphical passwords, third-party federation, 3D passwords and biometrics [2]. Each method has its own benefits and drawbacks. The OAuth method of authentication which is relatively new provides a framework for granting third parties access to the user's secure resources without having to disclose their credentials to the third party services [3].

✉ Kindson Munonye
kindson.munonye@edu.bme.hu

Martinek Péter
martinek@ett.bme.hu

¹ Budapest University of Technology and Economics,
Budapest, Hungary

OAuth has already been widely adopted by large industries and is gradually becoming a standard for authorization not only in the Web but also in mobile applications. OAuth strives to resolve the drawbacks of proprietary authorization protocols by creating a universal and interoperable authorization mechanism between applications and services. OAuth can also be applied in protecting user information in Single Sign On (SSO) approach [4]. SSO allows a user to login into a new application using a profile data you already have stored in another service. OAuth services are provided by major brands such as Facebook, Google, Yahoo and Microsoft. The benefit of this approach is that users have a centralized location for their credentials and do not need to create user profiles across different applications and services.

OpenID Connect builds an identity layer on top of the OAuth 2.0 protocol [5]. This added functionality makes it possible for the Relying Party (RP) or third-party services to verify the end user's identity by relying on an authentication process performed by an OpenID provider (OP). This service is oftentimes provided by the same identity provider and therefore the two services neatly integrate and complement each other: OpenID provides authentication, while the OAuth protocol is responsible for authorization. OAuth (combined with OpenID) is one of the most widely used authentication and authorization protocol on the Internet [2].

However, the growth and development of OAuth and OpenID have also brought new security challenges. This is because attackers now have additional platform for phishing and other forms of attacks on vulnerable systems. Additionally, achieving robust and practical security in OAuth is a non-trivial process since the system's operation relies on proprietary specification and implementation guidance [4]. This allows the implementation details to be figured out by developers without a standardized best practice.

Therefore, this research provides an approach for modeling the OAuth protocol, mining-related data, extracting the OAuth features, building a machine learning-based model for vulnerability detection and using an exploratory data analytics approach to examine the OAuth features. Identifying vulnerabilities would help mitigate OAuth-related attack such as Cross-Site Scripting (XSS) [6].

The rest of this research is arranged as follows: Chapter 2 examines the architectural framework of the OAuth protocol and the derived OIDC protocol. The threat model for the OAuth protocol is also presented. In chapter 3, existing research efforts in this area are presented. Chapter 4 presents details of the methodology, tools and techniques employed in this research. Test results and discussion are presented in chapter 5. Finally, in chapter 6, this research is summarized, and achievements are outlined. Challenges encountered and possibilities of further research in this area are also presented.

2 OAuth 2.0 and oidc architecture

The OAuth protocol specification allows for the users to authenticate via an existing Identity Provider thereby eliminating the need to maintain user account for different web applications. In this way, users can use the same credentials across different applications without having to provide their login credentials multiple times. This process allows for the Relying Parties (RP) to receive an access token which is presented along with each request [5]. In the context of the OAuth framework, the access token is automatically attached to each request originating from the client application along with other request parameters. This request is processed by the server if it has valid request parameters.

The OAuth framework documentation, RFC 6749 [7] defines the standards for the framework but leaves implementation details to Identity Providers (IdP). This makes organizations to adopt non-standard development processes resulting in a non-optimal performance and possible vulnerabilities.

2.1 The OAuth protocol

In this section, architecture of the OAuth protocol and a walkthrough of the authentication and the authorization flow are presented.

As shown in Fig. 1, the flow is initiated when a client application requests the login using an Authorization Server. The user enters his credentials and is provided with a consent screen. Then, the user confirms to allow login using the specific authorization server and access is granted. After that, an authorization token can be obtained which can be used to access secure resources or to perform other actions.

The OAuth is made up of six basic components through which the data flows and value of the parameter continue to be modified until the last state is reached. These components are outlined in Table 1 along with their functions.

Figure 2 outlines the various steps of the OAuth protocol:

User \rightarrow RP: A user visits a web application (1) which is the Relying Party (RP). RP \rightarrow User: The RP asks for user authentication (2). This could be done in the form of a page displaying various authentication options. User \rightarrow IdP: Then, the user selects to be authenticated by an external IdP and provides the credentials to the IdP (3). IdP \rightarrow User: The IdP sends the authorization grant back to the user (4). User \rightarrow RP: Then, the user is redirected back to the specified redirection end point defined by the *callback_url* (5). RP \rightarrow IdP: The user then uses the

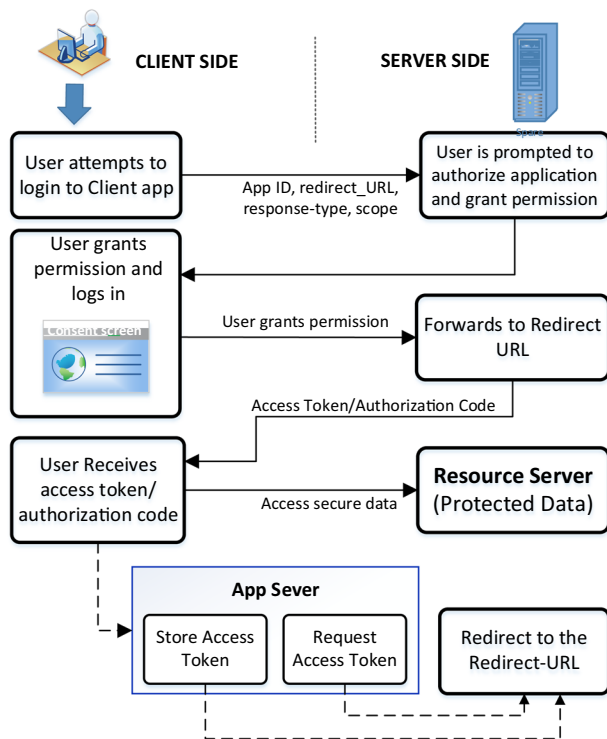


Fig. 1 The OAuth Architecture

authorization grant received in step 4 to request for access token (6). This request is made by also providing the scope and the original *redirect_uri*. *IdP* \rightarrow *RP*: The IdP checks whether the data provided are correct and if so, sends back the access token to the RP (7). *RP* \rightarrow *RS*: With the access token, the RP requests for the restricted resources (8). *RS* \rightarrow *RP*: If the access token is verified as correct by the resource server, then access is granted.

From Fig. 2, the identity provider(right) contains existing user credentials. However, it's also possible for a malicious IdP to impersonate the real IdP alter the flow sequence of operation or gain unauthorized access to data transferred between the end points. When this happens, the system

should be able to detect this alteration in a timely manner based on the metadata attached to the requests.

The system specified in Fig. 2 could be described as vulnerable if an external system is able to inject one or more sequences of operations that could affect the final output. This is covered in Chapter 4 under the vulnerability model.

3 Related works

A number of research works have been carried out on the general area of detecting and mitigating vulnerabilities in the software. Use of machine learning techniques and related approaches has also been applied for vulnerability discovery in web applications. For clarity and comprehensiveness, each related research is examined in a separate subsection.

3.1 Automatic vulnerability classification

This is a two-step process: First, identify the possible vulnerabilities, then perform classification using neural network or other approach like Naïve Bayes [8]. The author recommended feature extraction from the application overall workflow and then using these features as inputs to an (Neural Network and have the vulnerability classifications as output. This is quite impressive but focuses only on classification.

3.2 Buffer overflow vulnerability prediction

In this research, the authors presents an argument for the use of static analysis and machine learning to identify vulnerable statements in the application code [9]. The steps are highlighted in Fig. 3. So basically, the assumption is that the binary of the source program is provided as a whitebox.

The buffer overflow prediction works well for non-proprietary systems but also includes the added complexity to build a code-agnostic dissembler.

Table 1 The OAuth components

Components of OAuth	Description
Resource Owner(RO)	This is normally a user or a web server which owns the resources the client wants to access
Client	This is the third party service or application that needs to be authorized by using the OAuth protocol
Authorization Server (AS)	The service that authenticates the resource owner and performs the authorization check. This may be the same as RS
Authorization Grant (AG)	This is a piece of code received by the Client after authentication. It is used to access restricted resources on behalf of the RO
Redirect URL	This is the URL that the user is redirected to after a successful authentication with the authorization server
Access Token (AT)	This is a piece of data issued by the AS that lets the client request access from the RS

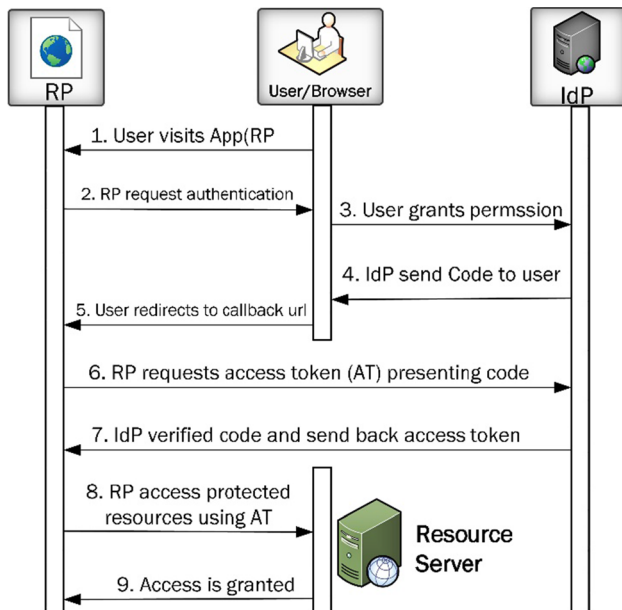


Fig. 2 Steps of the OAuth protocol

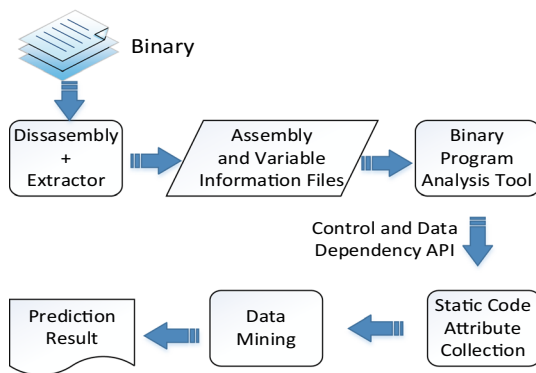


Fig. 3 BO approach workflow

3.3 Use of DFA and machine learning

The first step of the use of Data Flow Analysis(DFA) and machine learning is feature extraction [10]. This was achieved using a series of analysis including reaching definitions analysis (RDA), taint analysis and reaching constants analysis (RCA). This method was successfully applied in the detection of SQL injection (SQLi) and Cross-Site Scripting (XSS) vulnerabilities in PHP applications.

3.4 Optimized attack vector repertory

This is a method proposed in 2015 and follows a process of generating an attack vector repertory automatically, optimizing the vector and then uses an optimization model. A machine learning classification algorithm was proposed to

build an optimization model. The optimization model is trained with the historical testing data and the model obtains knowledge about the attack vector features from the training data. Then, based on the knowledge, the optimization model optimizes the attack vector repertory and selects those used to detect XSS(Cross-Site Scripting) vulnerabilities in web applications [11]. This method however is specific in the detection of XSS vulnerabilities and not generic to a wider range of other possible attacks.

3.5 Vulnerability detection in source code

The authors applied neural network modeling using open source C and C++ dataset to detect potential vulnerabilities in the source codes [12]. The authors recommended a combination of natural feature representation of lexed function source code with a powerful ensemble classifier, the random forest (RF). Feature extraction approach, similar to those used for sentence sentiment classification with convolutional neural network (CNN) and recurrent neural network (RNN), was leveraged for function-level source-code vulnerability detection. This is based on an assumption of the functional structure of the source code.

3.6 Use of data mining in FP prediction

This research explores the use of hybrid methods including static code analysis and taint analysis [13]. A trade-off is reached between human coding knowledge about vulnerabilities for taint analysis and automatically obtaining the knowledge using machine learning techniques. The method was tested using a tool for performing automatic code correction for PHP codes. The focus of this research as outlined by the authors is “enforcing information-flow security at language level.” Taint analysis, data mining and code correction were applied.

3.7 Backdoored network identification

This research attempts to provide a proof that outsourced model training could be a source of potential vulnerability. An adversary can create a maliciously trained network, *BadNet*, that has a state-of-the-art performance on the user’s training and validation samples but behaves badly on specific attacker-chosen inputs [14]. More of motivation for further research was provided than actual methods of solution.

3.8 Static analysis tool (DEKANT)

The authors of this work developed an intelligent web analysis tool that intelligently learns to detect vulnerability in web applications. Then Hidden Markov Model(HMM) was used to characterize vulnerabilities based on a set of source code

slices with their code elements annotated as tainted or not taking into consideration the code that validates, sanitizes or modifies the inputs [15]. The focus of this research was on the detection of surface vulnerabilities which relates specifically to user input pattern.

3.9 Static analysis and data mining

This is a research that really motivated the current work. Various machine learning approaches were applied together with static analysis and taint analysis for detecting vulnerability. Specifically, various classification models were developed and evaluated [16]. The automatic code correction was recommended for correction of code bugs and errors. It is also to be noted that this research was applied mainly to codes written in unsafe languages like PHP while the current research is code-language-agnostic.

3.10 Text as a method of prediction

In this case, the focus is the analysis of text files. Similar to the DEKANT mentioned earlier, this approach has its main artifacts, the source codes of the application under analysis [17]. The lexical and syntactic analysis of the resulting feature vector where every word, called a monogram, was further preprocessed and engineered using data analytics algorithms. The drawback is that consideration was not given to live application workflows like the OAuth protocol being considered in this research.

3.11 Vulnerability detection in mobile

This is similar to other previous works but applied to mobile applications. Here, the authors proposed applying malware detection algorithm, the N-gram analysis [18]. The features of the application components were extracted and a machine learning tool was used to create a model for malware classification.

4 Methodology

In this chapter, the methodology used in this research is comprehensively presented and discussed. An argument is also presented on the basis for a machine learning-based approach, as well as evaluation metrics.

4.1 The approach

The approach has three phases: (1) dataset generation (2) feature extraction (3) model development. A model-based approach is adopted similar to [19] where a web application workflow is represented as a Finite State Machine(FSM)

derived from a property graph. The goals of such a model are to address the six challenges identified as follows.

- *C1—determine state transitions*
- *C2—determine the system features and outputs*
- *C3—relationship between the features and state transitions*
- *C4—transition in non-trivial workflows*
- *C5—security-relevant state changes*
- *C6—comprehensive reusable representation of application functionality*

In this research, the OAuth protocol is modeled as a system where the parameters change between the initial and the final phase of the cycle. In this way, the system parameters representing the OAuth feature set were extracted after series of test runs.

The system provides a method of modeling the relationship between the parameters(features) that combine to produce an output. Therefore, if applied to the analysis of the OAuth protocol workflow, then the combination of features that produces a particular output is established. Hence, any alteration to this combination would represent a potential vulnerability injected from an external source.

The first task for this research is to extract the features of the OAuth workflow and then extract the features that have the most effect on the variability of the output. This was done using exploratory data analysis technique including Principal Components Analysis(PCA) and/or Factor Analysis(FA) [20].

This approach to vulnerability detection is a novel way that aims to analyze how the parameters of OAuth-enabled system affect the final output of the system based on the result of the analysis of requests originating (or appearing to originate) from the client application. The parameters that make up the feature set are metadata included in the header and body of the *HTTP* request/response cycle. At the end of the cycle, a final output is produced classified as success, failure or undefined.

Since the output of the OAuth workflow is expected to be successful with valid combination of parameters, the proposed machine learning approach is based on the assumption that each failure or undefined output class of the OAuth workflow could represent a potential vulnerability.

The approach to vulnerability detection is based on the following heuristics:

H_1 : The final output of the OAuth authorization flow depends on the feature set associated with the request/response cycle.

H_2 : Any HTTP request contains certain properties that can be analyzed before the request is processed by the server and the server can decide to either process the

request or discard the request based on the result of the analysis.

H_3 : The OAuth protocol provides both optional and required parameters

H_4 : The optional parameters defined by the OAuth protocol standard can be adjusted to optimize the output of the implementation.

Additional HTTP parameters can be included by the particular client application. This includes *referrer header*, *origin header* and *cookie property value* which could either be lax or strict.

4.2 Modeling

The main goal of modeling is to analyze the OAuth workflow of Fig. 4 and create a representation that allows a close examination of the workflow and the changes that occur and thereby address challenges C_1 , C_2 , C_3 and C_6 .

To address the challenge C_1 , we first model the system using an FSM which maps to a property graph as shown in Fig. 5. The property graph is a directed graph where the nodes and edges can have a key-values set of properties. The example of Fig. 4 (representing the first four steps of the OAuth workflow) shows four nodes: n_1 , n_2 , n_3 and n_4 . Each node has one key-value pair property. For instance, $k_1 = v_1$ for n_1 , $k_2 = v_2$ for n_2 , $k_3 = v_3$ for n_3 and $k_4 = v_4$ for n_4 . The nodes have labels indicated as L' and L'' . Similarly, the edges are labeled e_1 , e_2 , e_3 and e_4 .

Once the overall workflow of the OAuth protocol is represented in this way, then an FSM is derived. To minimize the complexity of presentation, the complete property graph and FSM for the OAuth workflow are not provided here.

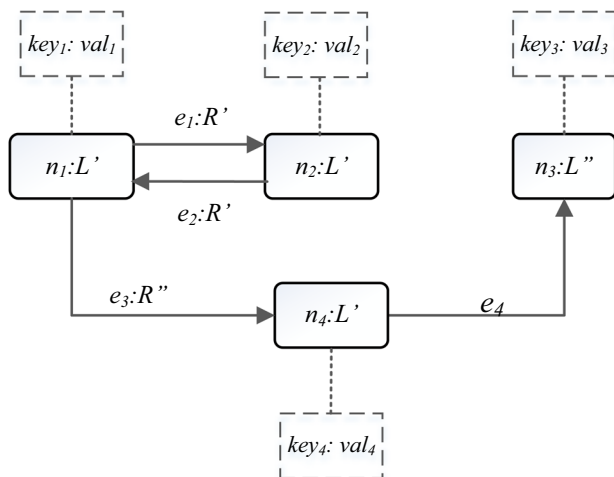


Fig. 4 Labeled Property Graph

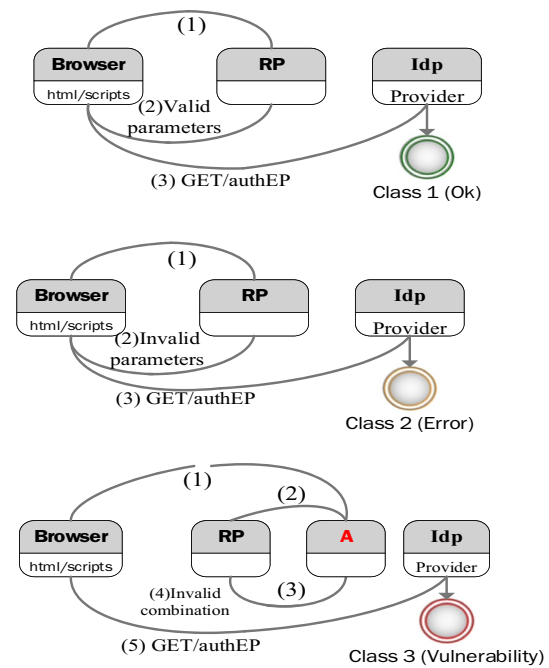


Fig. 5 Basic FSM flow for Classes 1, 2, 3

The key-value property graph of Fig. 5 corresponds to the FSM (finite state machine) in Fig. 6. This illustrates the first four steps of the OAuth workflows. The model being developed would be able to detect an invalid request or response which would result in the termination of the workflow with a failure (either an error or an undefined state). In this case, the path e_1 - e_2 - e_3 - e_4 results in a failed output and effectively terminates the flow.

Challenge C_2 is addressed by determining which state transitions are security relevant while C_3 consists of

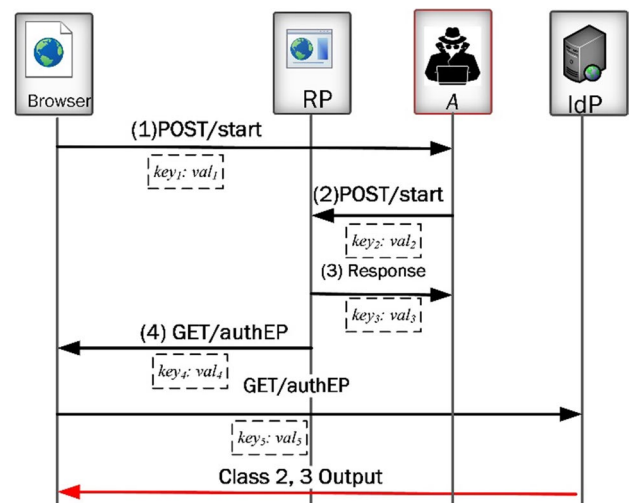


Fig. 6 Instance of OAuth Vulnerability Model

determining the relationships between identified request parameters and the state transitions. Challenge C_5 and C_6 are handled by further EDA which in this research was the application of Factor Analysis with results presented in the next chapter. Figure 5 shows a basic FSM for the first four steps which ensures the model is able to detect a potential vulnerability. In the first part which represents a successful output (Class 1), the flow terminates with a *200 Ok* status response. In the second part, the flow terminates with a deterministic error code (a specified in the OAuth framework documentation [21]) which indicates that an invalid input was provided by the user. However, the last part represents a potential vulnerability because the request from the browser to the IdP contains unknown parameters which was injected by an attacker (A). In this case, a HTTP response code is returned which is not predetermined.

If the complete OAuth protocol application workflow can be represented as an FSM, then the challenges identified could then be reduced to examining the state changes that result in what output. The next subsection provides a threat model for the OAuth workflow with a subset of Fig. 6.

4.3 The threat/vulnerability model

It could be understood that an invalid output (Class 2 or 3) of Table 3 is a result of either a combination of some invalid parameters (resulting in error) or a deliberate disruption of the flow by an external threat (resulting in undefined output). In this research, we focus on the latter. Figure 6 indicates an instance of a typically occurring threat. Here, a possible attacker inserts between two authentic OAuth component: the RP and the IdP. This could be an example for the Man-in-the-Middle (*MiM*) attack but in this case, it's an impersonation of the real IdP. However, based on the combination of the values exchanged, and the metadata of the attacker's origin, the IdP could detect an invalid flow, sends a Class 3 output and terminates the workflow.

4.4 Dataset generation

As of the time of this research, no suitable OAuth-specific dataset was found in existing repositories and therefore fuzzing techniques were applied for generating the initial test suite. Fuzzing has been applied in various researches as a vulnerability discovery technique [22,23,24]. The first step involved obtaining a list of OAuth Identity providers which are currently active between 2015 and 2020 and have a high number of clients available in Github under the *omniauth* project [25]. The list was further narrowed down based on the providers that have been listed in the *OwinAuthProvider* project.

OAuth configuration files for the selected providers were obtained from the Open Web Interface for .Net

Table 2 Identified OAuth protocol feature set

x	Description	x	Description
x_1	response_type	x_{18}	refresh_token
x_2	client_id (RP)	x_{18}	Cache-Control
x_3	redirect_uri (callback url)	x_{19}	Accept header
x_4	Request scope	x_{20}	Accept-Language
x_5	State/Profile	x_{21}	Connection
x_6	Protocol version	x_{22}	Host header
x_7	Code-Request-state	x_{23}	Content-Type
x_8	grant_type	x_{24}	Cookie
x_9	Token-Request-code	x_{25}	Content-Length
x_{10}	Token-Request-redirect_uri	x_{26}	Origin header
x_{11}	Token-Request-client_id	x_{27}	Referer header
x_{12}	Username(Idp Registration)	x_{28}	User-Agent
x_{13}	Password (Idp Registration)	x_{29}	Accept-Encoding
x_{14}	access_token	x_{30}	Request Method
x_{15}	token_type	x_{31}	Referrer Policy
x_{16}	expires_in	x_{32}	X-Requested-With

Table 3 Summary of original dataset

Original Dataset	Outputs		
	Success(s) Class 1	Failed(e) Class 2	Failed (u) Class 3
Total			
213, 840	145, 411	51, 322	17,107

(OWIN)-based application. Modules are provided for back-channel interfacing with the IdP using classes available in the OWIN library.

To generate the dataset, the Microsoft's Project Katana was implemented on a local test environment by creating a client for each of the IdPs provided in the configuration. Katana provides a flexible set of components for developing OWIN OAuth-enabled application [26]. This was a very challenging task since the implementation of the OAuth workflow for each of the providers could not be automated due to security-related concerns. Therefore, each procedure as specified for each provider was followed to create the 140 unique RP-IdP pairs. For each of the test cases, the output falls under one of three categories: success, failure and undefined. These three outputs have been mapped to terms Class 1, Class 2 and Class 3, so the problems can be handled using supervised learning methods.

4.5 Extracting the OAuth/OIDC feature set

A manual approach was used since each of the OAuth workflow needs to be examined and individual features are tabulated. Additional parameters were identified by examining the http request headers from http requests made between

the RP and the IdP. For each of the test cases, the outputs are groups under one of the three categories:

- *Success*: In this category, the protocol flow is completed successfully with a http success code.
- *Failure*: This class of output indicates a failure which returns a http status code indicating an error traceable to one of the input parameters.
- *Undefined*: The flow terminates with a status code not traceable to any of the inputs.

The 32 parameters that affect the output of the OAuth protocol AA flow are presented in Table 2.

We now represent the data in Table 2 in terms of the feature set X where each feature is identified by the corresponding subscript (1–32)

$$X = \{x_1, x_2, \dots, x_{32}\} \quad (1)$$

The output of each text case is represented as class, and in our test cases, each one resulted in one of the three classes given by

$$Y = \{\text{success, error, undefined}\} \quad (2)$$

Let the original number of variables (features) be p where:

4.6 $p = 32$

The number of variables (components) would subsequently be reduced to M where $M < p$ using Exploratory Data Analysis (EDA) techniques.

Table 3 shows the summary of the dataset used for the study indicating the proportion of each class of output. The probability values have been included to indicate that the data are being estimated based on the entire distribution.

4.7 OAuth workflow automation

The objective of the workflow automation is to execute several test cases of the OAuth workflow and log the changes in the OAuth parameters in a file. This file would later be used in building the classification model. The workflow is based on stochastic model checking property graph as specified in [27] based on Fig. 4. In this case, observable actions are distinguished from unobservable(internal) action. The observable appears on the external interface of the modeled component. Therefore, the values could be altered to examine the effect on the output. Unobservable, in this context are implicit to the OAuth component and, therefore, cannot be changed by this research process. This includes the internal logic of the IdP. To automate the model checking, the Selenium automation tool and a custom code are used to log the

changes in parameter values. These are captured from the request/response metadata available in the header of each request/response. Table 3 shows the summary of the classes of outputs obtained.

Detailed classification model is provided in Fig. 7. The model as shown is implemented between the client and the server side. The model intercepts request originating from the client to the server end points. The request parameters are evaluated by the model based on a knowledgebase of previously authenticated requests. Based on the evaluation, a vulnerability score is assigned which is compared with a predetermined threshold with range representing the classes (C1, C2 and C3). The request is executed if it is within the non-vulnerable threshold, else it is discarded, and the authentication flow is re-initialized.

4.8 A classification problem

From the model, feature set and dataset available, it could be seen that this is a typical classification problem. According to the OAuth 2.0 documentation [3], the outputs could be either a failure with error message or success. However, from the automation results obtained, it is observed that a set of outputs (8%) does not provide any of the specific errors defined in the documentation. It could therefore be concluded that this ‘undefined’ output which is based on an invalid combination of inputs represents a possible vulnerability and hence could be handled by creating a model based on large dataset derived from the results of the workflow

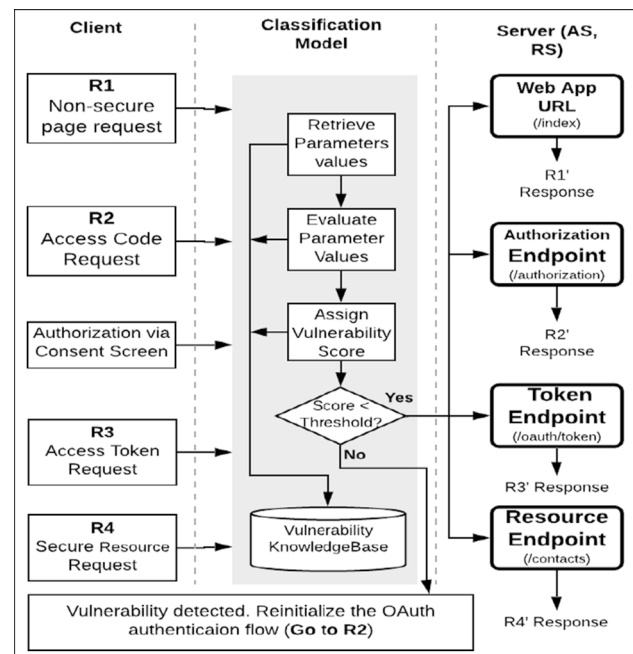


Fig. 7 Classification model architecture

automation. Therefore, the dataset was split into three parts: the training set, test set and the tuning set. The training set was used to train the model based on seven different classification models. The tuning set was used to determine the most efficient hyperparameters [10]. These are the parameters of the model training algorithm that is set prior to the learning process. To optimize these parameters, a grid search was applied. This implementation was done using the Python *scikit-learn* library which contains a number of probabilistic classifiers and tools for feature extraction, model selection, normalization and validation [28].

4.9 Existing vulnerability dataset

A search was made on several publicly available repositories for OAuth-related vulnerability dataset including National Vulnerability Database (NVD) [29], Open Source Vulnerability Database(OSVDB) [30] and WhiteSource Vulnerability Database [31] but unfortunately existing vulnerability dataset specific to the OAuth protocol and covers the 32 request/response features identified in this research could not be found. However, a related dataset, the PHP Security Vulnerability (PSV) dataset was obtained as gathered and analyzed by Walden et al. [32]. Of the 233 vulnerabilities, the OAuth-related vulnerabilities include 19 Code Injection, 12 Cross-Site Request Forgery (CSRS), 86 Cross-Site Scripting (XSS) and 73 classified as Authorization Issues. The dataset was obtained from three PHP-based web applications: Drupal, Moodle and PHPMyAdmin. Of the 233 vulnerabilities, 190 are OAuth related and are specified as actual lines of codes in the application source files. This research was able to determine that only 123 actually related to the OAuth protocol. The goal is to attempt to pass the AA flow through these codes and match the output with the Class 3 outputs obtained through the model developed. A match of 54% was obtained as presented in Table 9.

4.10 Metrics for classifier evaluation

To really evaluate the performance of the classifiers used, 10 different metrics were used which are calculated based on four parameters. These parameters are indicated in Table 2 with each of the parameters in each quadrant of a confusion matrix (CM). The CM is a cross-reference table where the rows are predicted results (by the classifier) while the columns are observed instance.

In this research, the terms *false positive* (FP) and *true positive* (not FP) have been used to describe the situation where the classifier indicated a situation where the analyzer is incorrect (not an actual vulnerability) or correct (actual vulnerability). In this section, the terms, *false positive*(*fp*) and *true positive*(*tp*), refer to the same terms: the output of the FP classifier. The same is true of *true*

Table 4 Generic confusion matrix

Predicted	Observed	
	Yes(FP)	No(not FP)
Yes (FP)	<i>True Positive(tp)</i>	<i>False positive(tp)</i>
No (not FP)	<i>False negative(fn)</i>	<i>True negative(m)</i>

negative(*tn*) and false negative. As adopted by [16] for clarity, the upper-case FP and lower-case *fp*, *tp*, *fn* and *tn* are used consistently (Table 4).

True positive rate(*tpr*): This is also called *recall* and is a measure of the number of true positives compared to the sum of true positive and false negatives and is given by:

$$tpr = \frac{tp}{(tp + fn)}$$

False positive rate(*fpr*): This is a measure of how the classifier deviates from the correct classification and is given by:

$$fpr = \frac{fp}{(fp + tn)}$$

Accuracy(*acc*): This is a measure of the total number of observations correctly classified. It is given by:

$$acc = \frac{(tp + tn)}{(tp + tn + fp + fn)}$$

Kappa Statistic (*kappa*): This is a measure of how the predicted classes agree with the observed class. It is divided into six categories: worst, bad, reasonable, good, very good and excellent. It is given by:

$$kappa = \frac{(po - pe)}{(1 - pe)}$$

where

$$po = acc$$

and

$$pe = \frac{(P * P' + N * N')}{(P + N)^2}$$

and

$$P = (tp + fn), P' = (tp + fp)$$

$$N = (fp + tn), N' = (fn + tn)$$

It is also important to mention that some of the metrics listed above are statistical such as the *fpr*, *tpr* and *kappa* while others are probabilistic such as the *acc* and *pr*.

F1 Score: This is a weighted average of the classes. It is given by:

$$F_1 = 2 * \frac{\text{Prec} * \text{Recal}}{\text{Prec} + \text{Recall}}$$

Since there are a wide range of classification algorithms, it is necessary to evaluate these various classifiers based on the above metrics together with the uniqueness of the dataset. As it can be seen in Table 5, the dataset is made of 17,107 of undefined outputs (Class 3) out of a total of 213,840. This means that the class of interest represents only 8% of the data. So seven classifiers models which have been applied in similar studies [33], 34, 35 were developed and evaluated including: *Random Forest*, *Naïve Bayes*, *Decision Tree*, *K-Nearest Neighbors*, *Logistic Regression* and *Support Vector Machine (SVM)*. The metric of interest is accuracy(acc) specifically for classification of the Class 3 outputs which are presented in Chapter 5.

5 Results and discussion

In this section, the dataset used for the evaluation is first briefly described. The first part of this chapter presents the performance of different probabilistic and statistical classifiers used for this research with focus on the *Gradient Boosting Classifier (GBC)*. This is because, this classifier produced the best performance metric (*accuracy of 83%*) compared to others for detecting vulnerable classes.

5.1 Generating the dataset

The method and techniques for generating the dataset were already presented in the previous chapter, so the summary of the dataset used for this research is given here in Table 5. The total number observation (each of which corresponds to a single OAuth workflow) is 213,840. This is split into 70% training set, 20% test set and 10% tuning set.

Table 5 Dataset summary

Input dataset	Outputs			
		Success(s) Class 1	Failed(e) Class 2	Failed (u) Class 3
Training set	149,688	101,788	35,925	11,975
Tuning set	42,768	29,082	10,264	3,421
Test set	21,384	14,541	5,132	1,711
Total	213,840			

5.2 Detecting succesful workflows (Class 1)

The goal is to perform classification based on individual classes so that the performance metrics for each for the classifier on each class could be obtained. So, for the Class 1, the classifier prediction performance was obtained the same with Class 2 and 3. In this way, we could use either classification- or regression-based models.

The first classifier considered in the *Gradient Boosting classifier (GBC)* is shown in Fig. 8. After training, the classifier with the training set (149,688 instances), the performance on the test set (21,384 instances) is presented based on the metrics: $acc = 0.8297(83\%)$ and *Area Under the ROC Curve(AUROC)* = 0.710 which provides a summary of the *fpr* and *tpr* metrics.

Performance of this GB classifier is indicated in Table 6 which gives the confusion matrix, as well as additional metrics provided by the classification summary of Table 6

From the classification performance for the GBC, a precision of 0.96 gives a very high ration of correctly predicted positive instances to the total predicted positive instances. This gives a better performance than [16] with a precision of 92.5%, an F1 score (weighted average of precision and recall) of 0.92, which is also a very good performance output. Detailed results can be observed in Table 7.

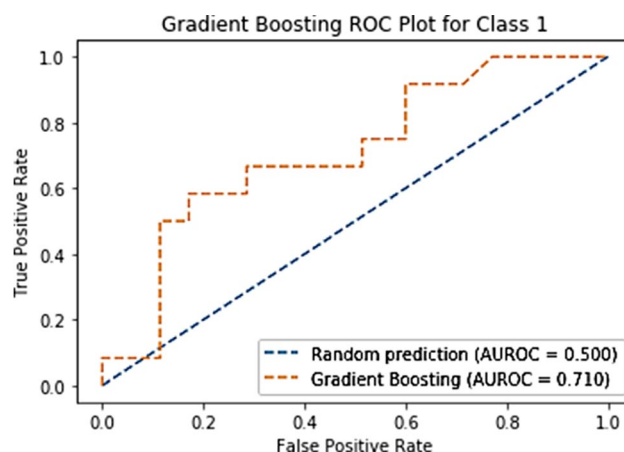


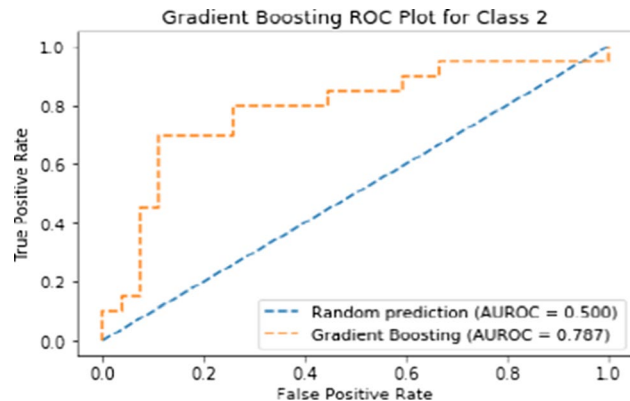
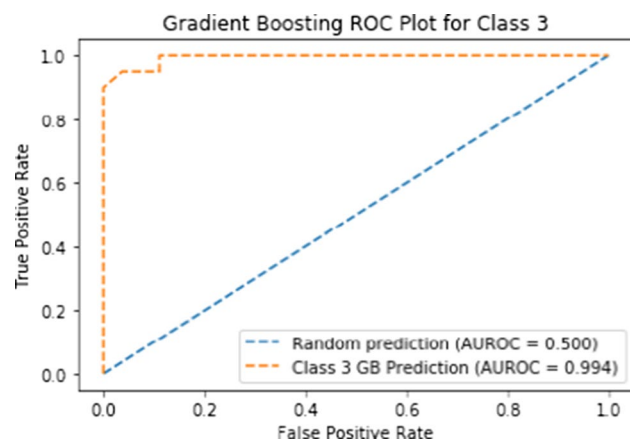
Fig. 8 ROC Performance for GBC

Table 6 Confusion Matrix Output of GBC (*scikit-learn*)

		Observed	
Predicted		Yes (FP)	No (not FP)
	Yes (FP)	18,874	2394
	No (not FP)	881	20,619

Table 7 Classification Report Output(*scikit-learn*)

	Precision	Recall	F1-score	Support
0	0.962	0.894	0.921	21,268
1	0.91	0.962	0.93	21,500
Accuracy			0.921	42,768
Macro avg	0.934	0.921	0.921	42,768
Weighted avg	0.934	0.921	0.921	42,768

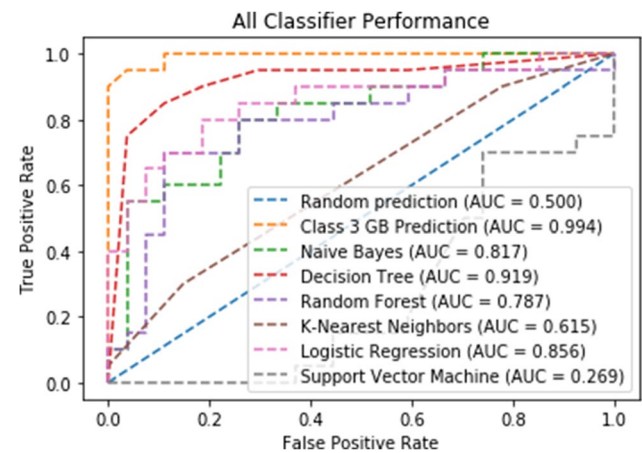
**Fig. 9** GBC Performance on Failed Outputs**Fig. 10** GBC Performance for Undefined Outputs

5.3 Detecting failed workflows (Class 2)

Figure 9 shows the performance of the GB classifier for Class 2. An improvement in the performance over the Class 1 outputs was observed with an AUROC of 0.787 and an accuracy score of 0.767(76%). It is also necessary to note that slightly, but different values were obtained after performing training and tuning but the values of the acc remain between 74 and 78%.

Table 8 Classifier Performance Summary

	Prec(%)	Recall (%)	F1 (%)	AUC
GBC	80.34	74.70	71.71	0.994
RFC	78.15	67.69	81.37	0.787
NBC	69.44	69.95	67.44	0.817
DTC	73.54	71.78	73.31	0.919
KNN	75.13	68.95	70.02	0.615
LRC	78.59	79.49	80.15	0.856
SVC	71.40	68.87	68.04	0.269

**Fig. 11** ROC Curve for all seven classifiers

5.4 Detecting failed undefined workflows(Class 3)

Figure 10 shows the performance of the GB classifier for the Class 3, undefined outputs which represents potential vulnerabilities (Table 8).

From Fig. 10, it is very clear that the Gradient Boosting classifier performs best in prediction potentially vulnerable outputs (Class 3). It is to be noted that Fig. 10 indicates an AUROC of 0.994 which are the highest values obtained after a series of training and tuning.

5.5 Performance of different classifiers

This subsection provides the results of the performance of different classification algorithms in classifying an OAuth-based workflow. Models were also built based on the following seven classifiers: *Random Forest*, *Naïve Bayes*, *Decision Tree*, *K-Nearest Neighbors*, *Logistic Regression* and *Support Vector Machine (SVM)*. Additional models were considered during the experimentation with the seven selected for this research given the best results.

The SVM classifier provides the highest accuracy score in detecting the successful (*Class 1*) and failed (*Class 2*)

outputs but scored lower than the GBC on predicting undefined (*Class 3*) outputs.

The combined output of the seven classifiers evaluated is given in Fig. 11. It can be seen that the GBC provides the most accurate prediction ($AUC = 0.994$) while the SVM Classifier performed worst ($AUC = 0.269$).

5.6 Comparison with proven vulnerabilities

The combination of parameters resulting in the class 2 and class 3 outputs was pattern-matched against the HTTP parameters obtained from the PSV dataset. The result is given in Table 9. For each of the 11 Code Injection vulnerabilities, 15 AA parameters were logged. Of the nine parameters, 13 values provided match with parameter set that resulted in the potentially vulnerable outputs of the classification model, which is 73% match. Similar results were obtained for CQRF, XSS and Auth with the overall average percentage match 33%, 56% and 53%, respectively, giving an average of 54%. Therefore, 54% of potential vulnerability detected by the classification models in this research were actual vulnerability based on existing vulnerability dataset.

5.7 Principal component analysis output

Of the 32 variables making up the feature set of the OAuth protocol workflow, it's generally obvious that just a few of them form the marker variables which have the most effect on the output. Exploratory Data Analysis (EDA) was used to determine this by applying Principal Component Analysis (PCA) and Factor Analysis (FA).

The functions *prcomp()* and *pca()* which are available in the *pcaMethods()* library of R were used to carry out the analysis of the original dataset to determine the principal components and calculate the proportion of variance PVE

Table 9 Comparison with existing vulnerabilities

Known Vulnerability Class	Cases	OAuth Features	Identical	% Similarity
Code Injection	11	15	8	73%
CSRF	9	13	3	33%
XSS	54	11	30	56%
Auth. Issues	49	16	26	53%
Average Matching Metric (%)				54%

Table 10 Summary of First Six Principal Components

PC-1	PC-2	PC-3	PC-4	PC-5	PC-6	
S.DEV	1.989	1.605	1.451	1.172	1.131	1.032
PVE	0.219	0.143	0.117	0.076	0.071	0.059
CVE	0.219	0.363	0.480	0.5559	0.627	0.686

for each of the components for all the 32 features. The PVE gives the explanatory value of the value of variables used in a study [36] and, therefore, the higher the percentage of variance of a proposed model, the more valid the model seems to be [37].

The result of the Principal Components Analysis carried out on the original data set is shown in Table 10.

Summing the proportion of variance for the first five PCs gives us a value of 0.627 which means that PC-1 to PC-5 can be used to explain more than 50% of the variation in the dataset. Adding one more component gives us a PVE of 68.59%. At PC-6, there is a cumulative proportion equivalent to 68.59%. Therefore, six key features were safely extracted which significantly affected the output. These features from the original feature set were further examined to determine sensitivity to the model output.

While it's possible to add more components, the methods of this research could still be established with six components.

The screen plot shown in Fig. 12 indicates that after about the 6th and the 7th component, the curve begins to flatten, and additional component does not contribute to the overall variance.

5.8 Determining the marker variables

Table 10 shows the output of the factor analysis indicating the correlation between all the 32 variables and the factors. The number listed in the six columns is the item

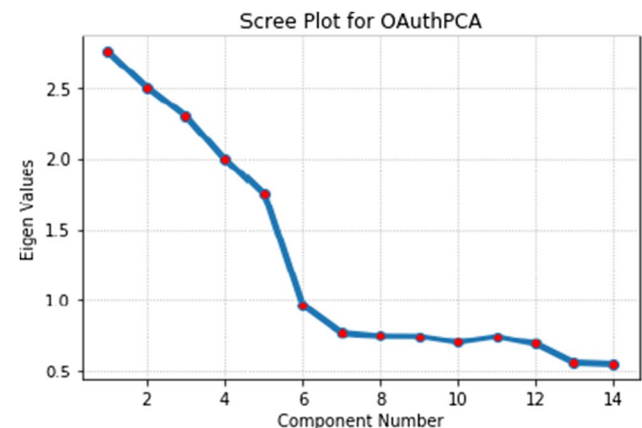


Fig. 12 Screen Plot for the OAuth Dataset

Table 11 Six-Factor summary for variable/factors correlation

Variables		Factors					
Var	Description	Redirect data	Origin/Refer- rer Details	Scope Speci- fication	Authentica- tion Type	Protocol Details	Browser Data
X ₁	Redirect_uri	0.936	0.252	0.018	0.381	0.008	0.019
X ₂	Token- Request- Redirect_uri	0.649	0.280	0.010	0.366	0.052	0.054
X ₃	Token- Request- code	0.598	0.255	-0.114	0.242	0.092	0.073
X ₄	Scope	0.213	0.178	0.751	0.199	0.287	0.195
X ₅	State	0.288	0.423	0.720	0.075	0.191	0.281
X ₆	Code- Request- state	0.240	0.345	0.574	0.152	0.305	0.186
X ₇	Protocol ver- sion	0.025	-0.360	0.302	0.046	0.895	0.269
X ₈	Referer Policy	0.220	0.412	0.154	-0.129	0.673	0.026
X ₉	Grant_type	0.223	0.436	0.430	0.865	0.015	0.196
X ₁₀	Access_token	0.458	0.389	0.280	0.824	0.458	0.148
X ₁₁	Token_type	0.220	0.300	0.356	0.705	0.271	0.012
X ₁₂	Response_ type	0.380	0.023	0.117	0.681	0.208	0.434
X ₁₃	Refresh_ token	0.292	0.095	0.299	0.626	0.392	0.183
X ₁₄	User-Agent	0.109	0.337	0.005	0.253	0.199	0.758
X ₁₅	Cookie	0.415	0.132	0.165	0.298	0.250	0.533
X ₁₆	Origin	0.099	0.682	0.297	0.083	0.384	0.346
X ₁₇	Referrer	0.066	0.648	0.298	0.061	0.101	0.234
X ₁₈	Request Method	0.091	0.725	0.327	0.288	0.399	0.262

loadings. For each variable, a coefficient shows the correlation between the variable and the factor.

Based on the *PCA*, a six-factor solution was chosen for ease of presentation and is presented in Table 11. These data are obtained after the solution has been optimized by eliminating variables that have *crossloadings* (high correlation with more than one factor) and variables with low loading (> 0.3). Of the 32 variables, 12 were deleted leaving 18 for analysis.

The table was reorganized to ease readability such that items from the same factor have been grouped together and ordered in the descending order of their loadings.

The names of the six factors were chosen based on the marker variables which are variables with the highest loadings in the factor. These features provide the best indication of the chosen factor. In Table 10, the marker variables include *redirect_uri*, *scope*, *Protocol version*, *grant type*, *User-Agent* and *origin* with loadings of 0.936, 0.751, 0.895, 0.865, 0.758 and 0.682, respectively. These variables are shown in Table 11.

Following each of the marker variables are the next set of variables that provided further indication of the chosen factor. For example, in case of the *scope* specification in column 5, the next two features are the state with a loading of 0.720 and the *Code-Request-State* with a loading of 0.574. This research however focuses on examining the effect of the six identified marker variables.

6 Summary and conclusion

This research adopts a model-based approach similar to (Pellegrino, Johns, Koch, & Backes, 2017). But in our approach, the OAuth protocol implementation is modeled as a system where the parameters change between the initial and the final phase of the cycle. In this way, the system parameters representing the OAuth feature set were extracted after series of test runs.

6.1 Summary of achievements

This research has created a learning-based model for classification OAuth-based workflows for web application to detect potential vulnerabilities. Also, a framework has been created and tested for extraction of the feature set of a request/response cycle of the OAuth implementation. A bit of

data mining techniques has also been applied to generate necessary training dataset (213,840 instances) required for the supervised learning technique applied in this research. These models clearly lead to improved security for web application with the single-sign-on feature enables.

6.2 Challenges encountered

While there has been success in reaching the aims and objectives of this work, there were also a few challenges as outlined below:

Lack of open-source OAuth datasets: As of the time of this research, there's no open-source dataset available specific to the OAuth protocol and its implementations. Therefore, this research would refine the data collection method and possible build a repository of dataset specific to the OAuth protocol. This could be a possible area of further research.

It is also quite tasking to build workflows based on different Identity Providers (IdPs) as 140 of them were identified in this research.

Lack on homogeneity on IdPs: The 140 IdP analyzed during this research do not implement a homogenous process on a granular level on registration of third-party application (RP) and its authorization.

6.3 Possibility of future works

A possible area of further research is the application more advanced machine learning approaches to the detection of vulnerability in OAuth-enabled web application and in the actual removal of detected vulnerabilities detected. This could also be extended to mobile application and IoT-enabled applications as well. An ongoing research on removal of vulnerabilities continues as an extension of this one.

Another interesting area currently being studied is the development of formal model for verification and validation of the OAuth protocol and its implementations. This is because, although formal models have been applied to enterprise application, attention has not been given to the specific processing logic for the OAuth 2.0. Finally, it is recognized that some vulnerability results from user actions rather than the application workflow. Therefore, it's possible to have a machine learning methodology to possibly track user actions and provide real-time recommendations to user on the possible security implication of actions taken.

Author's Contribution: MP played the most important role in this research. The author(s) worked together to produce the complete manuscript.

Availability of Data and Material: Because of the size and propriety of the dataset used for this research, it could not be included in this manuscript.

Declaration

The authors hereby declare that there was no conflict of interest as a result of this research and the resulting manuscript.

Acknowledgement The guidance Martinek Péter with the Department of Electronics and Informatics at the Budapest University of Technology and Economics was very important to the success of this research.

Funding Open access funding provided by Budapest University of Technology and Economics.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Fett, D., Küsters, R., Schmitz, G.: A comprehensive formal security analysis of OAuth 2.0. *Proc. ACM Conf. Comput. Commun. Secur.* **24–28**, 1204–1215 (2016). <https://doi.org/10.1145/2976749.2978385>
2. Ferry, E., Raw, J.O., Curran, K.: Security evaluation of the OAuth 2.0 framework. *Inf. Comput. Secur.* **23**(1), 73–101 (2015). <https://doi.org/10.1108/ICS-12-2013-0089>
3. "RFC 6749 - The OAuth 2.0 Authorization Framework. Accessed Jun. 07, 2020 <https://tools.ietf.org/html/rfc6749>
4. Li, W., and Mitchell, C. J.: Security issues in OAUTH 2.0 SSO implementations. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8783, no. August, pp. 529–541, (2014). doi: <https://doi.org/10.1007/978-3-319-13257-0>
5. Li, W., Mitchell, C. J., and Chen, T.: Mitigating CSRF attacks on OAuth 2.0 and OpenID Connect. pp. 1–18, 2018, [Online]. Available: <http://arxiv.org/abs/1801.07983>
6. Laurie, B., Langley, A., and Kasper, E.: RFC 6962: certificate transparency. *RFC*, pp. 1–27, (2013).
7. Harer, J. A. et al.: Automated software vulnerability detection with machine learning. (2018), [Online]. Available: <http://arxiv.org/abs/1803.04497>
8. Padmanabhuni, B.M., Tan, H.B.K.: Buffer overflow vulnerability prediction from x86 executables using static analysis and machine learning. *Proc. Int. Comput. Softw. Appl. Conf.* **2**, 450–459 (2015). <https://doi.org/10.1109/COMPSAC.2015.78>

9. Kronjee, J., Hommersom, A., Vranken, H.: Discovering software vulnerabilities using data-flow analysis and machine learning. *ACM Int. Conf. Proc. Ser.* (2018). <https://doi.org/10.1145/3230833.3230856>
10. Guo, X., Jin, S., Zhang, Y.: XSS vulnerability detection using optimized attack vector repertory. In: *Proceedings of 2015 International Conference Cyber-Enabled Distributed Computer Knowledge Discover CyberC 2015*, pp. 29–36, (2015). doi: <https://doi.org/10.1109/CyberC.2015.50>
11. Russell, R. et al.: Automated vulnerability detection in source code using deep representation learning. In: *Proceedings of 17th IEEE International Conference Machine Learning Application ICMLA 2018*, pp. 757–762, (2019). doi: <https://doi.org/10.1109/ICMLA.2018.00120>
12. Medeiros, I., Neves, N. F., and Correia, M.: Automatic detection and correction of Web application vulnerabilities using data mining to predict false positives. In: *WWW 2014—Proceedings of 23rd International Conference World Wide Web*, pp. 63–73, (2014). doi: <https://doi.org/10.1145/2566486.2568024>
13. Gu, T., Dolan-Gavitt, B., and Garg, S.: BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. (2017). [Online]. Available: <http://arxiv.org/abs/1708.06733>
14. Medeiros, I., Neves, N., and Correia, M.: DEKANT: a static analysis tool that learns to detect web application vulnerabilities. In: *ISSTA 2016—Proceedings of 25th International Symposium Software Testing Analysis*, pp. 1–11, (2016). doi: <https://doi.org/10.1145/2931037.2931041>
15. Medeiros, I., Neves, N., and Correia, M.: Detecting and removing web application vulnerabilities with static analysis and data mining k-nearest neighbor LFI local file inclusion LR logistic regression MLP multi-layer perceptron Nb Naive Bayes OSCI OS command injection PHPCI PHP command injection. pp. 1–16, (2015). doi: <https://doi.org/10.1109/TR.2015.2457411>
16. Hovsepian, A., Scandariato, R., Joosen, W., and Walden, J.: Software vulnerability prediction using text analysis techniques. In: *MetriSec'12—Proceedings of 4th International Work Secure Measure Metrics*, pp. 7–9, (2012). doi: <https://doi.org/10.1145/2372225.2372230>
17. Dhaya, R., and Poongodi, M.: Detecting software vulnerabilities in android using static analysis. In: *Proceedings of 2014 IEEE International Conference Advance Communication Control Computing Technology ICACCCT 2014*, no. 978, pp. 915–918, (2015). doi: <https://doi.org/10.1109/ICACCCT.2014.7019227>
18. Pellegrino, G., Johns, M., Koch, S., Backes, M., and Rossow, C.: Deemon: Detecting CSRF with dynamic analysis and property graphs. In: *Proceedings of ACM Conference Computing Communication Secure*, no. October 2017, pp. 1757–1771, (2017). doi: <https://doi.org/10.1145/3133956.3133959>
19. Meyer-Baese, A., Wismueller, A., Lange, O.: Comparison of two exploratory data analysis methods for fMRI: Unsupervised clustering versus independent component analysis. *IEEE Trans. Inf. Technol. Biomed.* **8**(3), 387–398 (2004). <https://doi.org/10.1109/TITB.2004.834406>
20. “RFC 6749 - The OAuth 2.0 Authorization Framework. Accessed Jun. 12, 2020 <https://tools.ietf.org/html/rfc6749>
21. Wang, Y., Jia, P., Liu, L., and Liu, J.: A systematic review of fuzzing based on machine learning techniques
22. Grieco, G., Grinblat, G. L., Uzal, L., Rawat, S., Feist, J., and Mounier, L.: Toward large-scale vulnerability discovery using machine learning. In: *CODASPY 2016—Proceedings of 6th ACM Conference Data Application Security and Privacy*, pp. 85–96, (2016). doi: <https://doi.org/10.1145/2857705.2857720>
23. Cheng, L. et al.: Optimizing seed inputs in fuzzing with machine learning. Accessed: Jan. 17, 2021. [Online]. Available: <https://github.com/karpathy/char-rnn>
24. List of Strategies omniauth/omniauth Wiki. Accessed Jun. 10, 2020 <https://github.com/omniauth/omniauth/wiki/List-of-Strategies>
25. OWIN and Katana | Microsoft Docs. Accessed Jun. 10, 2020 <https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/>
26. Katoen, J. P.: Stochastic model checking. In: *Stochastic Hybrid Systems*, CRC Press, pp. 79–106, (2006)
27. Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., Mueller, A.: Scikit-learn. *GetMobile Mob. Comput. Commun.* **19**(1), 29–33 (2015). <https://doi.org/10.1145/2786984.2786995>
28. “NVD - Vulnerabilities.” <https://nvd.nist.gov/vuln> (accessed Jan. 16, 2021).
29. Vulnerability Database. Accessed Jan. 16, 2021 <https://vuldb.com/>
30. Open Source Vulnerability Database | WhiteSource. Accessed Jan. 16, 2021 <https://www.whitesourcesoftware.com/vulnerability-database/>
31. Walden, J., Stuckman, J., and Scandariato, R.: Predicting vulnerable components: software metrics vs text mining
32. Narudin, F.A., Feizollah, A., Anuar, N.B., Gani, A.: Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput.* **20**(1), 343–357 (2016). <https://doi.org/10.1007/s00500-014-1511-6>
33. Singh, J., Singh, G., Singh, R.: Optimization of sentiment analysis using machine learning classifiers. *Human-centric Comput. Inf. Sci.* (2017). <https://doi.org/10.1186/s13673-017-0116-3>
34. Antonelli, M., et al.: Machine learning classifiers can predict Gleason pattern 4 prostate cancer with greater accuracy than experienced radiologists. *Eur. Radiol.* **29**(9), 4754–4764 (2019). <https://doi.org/10.1007/s00330-019-06244-2>
35. Fichman, M.: Variance explained: Why Size Does Not (Always) Matter. (1999). Accessed: Jun. 11, 2020. [Online]. Available: <http://repository.cmu.edu/tepper>
36. Lorenzo-Seva, U.: How to report the percentage of explained common variance in exploratory factor analysis. Accessed: Jun. 11, (2020). [Online]. Available: <http://psico.fcep.urv.cat/utilitats/factor/>