

An Automated Vulnerability Assessment Approach for WebAPI that Considers Requests and Responses

Toru Taya*, Masaki Hanada†, Yoichi Murakami†, Atsushi Waseda†,
Yuki Ishida‡, Takao Mimura‡, Moo Wan Kim† and Eiji Nunohiro†

*Graduate School of Informatics, Tokyo University of Information Sciences

† Department of Informatics, Tokyo University of Information Sciences

‡ SecureBrain Corporation

Email: g20004tt@edu.tuis.ac.jp, mhanada@rsch.tuis.ac.jp

Abstract—In recent years, WebAPIs are being published to allow external users to use Web services. On the other hand, the number of attacks that exploit WebAPI vulnerabilities is increasing. In order to prevent damage caused by abusing WebAPI vulnerabilities, the OWASP (Open Web Application Security Project) has published a guideline (OWASP API Security Top 10) describing the 10 vulnerabilities with the highest security risk in WebAPIs. However, the guideline does not describe the methods of attack and detailed countermeasures. Therefore, there are some vulnerabilities that are difficult to detect with existing WebAPI vulnerability assessment tools. In this paper, we propose a vulnerability assessment method that automatically obtains the WebAPI reference and repeatedly analyzes the requests through WebAPI and their responses. The proposed vulnerability assessment method will enable vulnerability assessment for vulnerability items that are difficult to be detected by existing vulnerability assessment tools among vulnerability items described in the guideline (OWASP API Security Top 10).

Index Terms—WebAPI, Vulnerability Assessment

I. INTRODUCTION

WebAPIs (Web Application Programming Interface) are published in various places and purposes, such as online banks, e-commerce sites, and weather information sites. Therefore, WebAPIs have become an indispensable technology for modern Web applications. WebAPIs are used to exchange data between Web applications. WebAPIs use HTTP/S protocols and data formats such as JSON and XML are often used instead of HTML to make it easier for applications to understand. In this study, we conduct vulnerability assessment for REST WebAPIs. While WebAPIs are becoming more and more popular, an article tampering vulnerability [1] was used to exploit the Wordpress WebAPI [2] in 2017. In order to confirm that WebAPIs do not contain vulnerabilities, many tools for assessing WebAPI vulnerabilities have been released. However, existing vulnerability assessment tools do not take into account the logic and characteristics of WebAPIs and do not issue the detection queries necessary for the targeted vulnerability assessment. In this paper, we propose a new method to detect such vulnerabilities. The proposed WebAPI vulnerability assessment method considers the requests through WebAPI and their response.

This paper is organized as follows: Section II describes the 10 vulnerability items with the highest security risks in WebAPIs. Section III describes related research. Section

IV describes existing vulnerability assessment tools. Section V describes our proposed method for WebAPI vulnerability assessment considering the requests through WebAPI and their responses. Section VI shows that the proposed method can detect the vulnerability of article tampering using the Wordpress WebAPI, which is difficult to detect by existing vulnerability assessment tools. Finally, Section VII summarizes this paper.

II. OWASP API SECURITY TOP 10

The Open Web Application Security Project (OWASP), which conducts research and develops guidelines in the field of Web application security, has published a guideline [3] (OWASP API Security Top 10) on the 10 most risky vulnerabilities related to WebAPIs. The guideline describes the countermeasures that should be implemented and the scenarios in which vulnerabilities can occur. The 10 vulnerability items reported in the guideline are shown in the table I. There are four major categories of vulnerabilities. The first category consists of vulnerabilities related to access control, such as the use of WebAPIs for administrators by ordinary users due to a lack of appropriate permissions. The second category is vulnerabilities related to data allocation, such as the WebAPI server sending data to the client with the assumption that filtering will be done on the web application side, or adding undefined data to the request body. The third category is vulnerabilities that occur in the development and operation phases, such as vulnerabilities that expose problematic APIs used in the development phase and mistakes related to the security settings of the server that provides the WebAPI. The last category is the same as vulnerabilities that exist as traditional Web threats, such as injection and incorrect logging settings.

In this study, we focus on vulnerability items No.1 and No.5 in the OWASP API Security Top 10. The details of the vulnerability items are described below.

A. No.1 Broken Object Level Authorization

The vulnerability “No.1 Broken Object Level Authorization” is a vulnerability in which an unpublished WebAPI endpoint is used.

The scenario described in the OWASP API TOP 10 is that if the endpoint `/shop/{shopName}/revenue_data.json` is

TABLE I
LIST OF VULNERABILITY(OWASP API SECURITY TOP 10)

No.#	Name of Vulnerability items
No.1	Broken Object Level Authorization
No.2	Broken User Authentication
No.3	Excessive Data Exposure
No.4	Lack of Resources & Rate Limiting
No.5	Broken Function Level Authorization
No.6	Mass Assignment
No.7	Security Misconfiguration
No.8	Injection
No.9	Improper Assets Management
No.10	Insufficient Logging & Monitoring

published, the string “{shopName}” will be rewritten using a list of shop names obtained in advance from other endpoints. After that, the sales data for each shop name is accessed and this vulnerability (No.1) is exploited. Countermeasures include preparing an authentication process for the endpoint and providing appropriate privileges to the user, as well as using random and unpredictable values for strings such as {shopName} in the scenario.

B. No.5 Broken Function Level Authorization

The vulnerability “No.5 Broken Function Level Authorization” is a vulnerability that allows a general user to use an endpoint that requires authorization.

The scenario described in the OWASP API TOP 10 is the use of an endpoint for administrators that is not publicized to general users, or changing the HTTP method of an endpoint that is publicized to general users to a HTTP method that is not publicized.

Countermeasures are to implement an authentication mechanism and to check if permissions are properly set.

As mentioned above, the OWASP API Security Top 10 guideline provides an overview of each vulnerability item, examples of attack scenarios and simple countermeasure methods. However, it does not describe the details of the attack query, specific countermeasures, and their detection methods. Therefore, there are some vulnerability items that are difficult to detect by existing vulnerability assessment tools.

III. RELATED RESEARCH

Takamatsu et al. have proposed a vulnerability assessment method that relies on the logic of Web applications [5]. This related research proposes a vulnerability assessment method for web applications that takes into account the logic of the web application. Here, the logic of the web application is to utilize the information used for authentication and the pattern of the response when the request is successfully completed. When conducting vulnerability assessment, authentication information and the content of the response are very important because it is necessary to assess a wide range of aspects of a target.

Authentication information and the content of the response are also important in WebAPIs. Therefore, in this paper, we focus on the logic of WebAPIs.

IV. VULNERABILITY ASSESSMENT TOOL

There are many tools available for detecting WebAPI vulnerabilities.

However, these vulnerability assessment tools cannot detect all of the 10 vulnerabilities described in the OWASP API Security Top 10. The reason for the failure of these tools to detect all the vulnerabilities in the OWASP API Top 10 is considered to be that they attempt to detect the vulnerabilities by sending predetermined requests, and that they cannot specify the authentication information for each target WebAPI.

In addition, various request body data (hereinafter called “parameters”) used for vulnerability assessment are set by a user who uses these tools, and if the user cannot set appropriate parameters, the assessment becomes difficult.

Features of two representative vulnerability assessment tools and vulnerability items of the OWASP API TOP 10 that can be assessed are described below.

A. Automatic API Attack Tool

“Automatic API Attack tool [6]” can flexibly generate vulnerability detection test cases according to the WebAPIs to be vulnerability-tested by loading the WebAPI reference in JSON or YAML format.

For example, for an endpoint that uses a parameter of “INT type” as “id”, Automatic API Attack Tool will change the value of “id” to a numeric value of another type, such as long or double, and send a vulnerability detection request.

This tool is able to detect vulnerabilities in No.1, No.5, and No.9 of the OWASP API Security Top 10.

However, the WebAPI reference in JSON or YAML format describing the WebAPI information is required, and the vulnerability assessment may require a lot of work and time.

B. Vooki, Rest API Scanner

“Vooki, Rest API Scanner [7]” is a vulnerability assessment tool that sends a request to detect vulnerability items No.3, No.7, and No.8 of the OWASP API Security Top 10. However, all the endpoints, headers, and parameters used in the WebAPI must be configured by a user who uses this tool. If the necessary information for vulnerability assessment cannot be set, the vulnerability may not be detected even if it exists.

V. PROPOSED METHOD

A. Outline

The proposed method automatically obtains a reference and dynamically generates a vulnerability detection query according to the content. This method solves the problem of existing vulnerability assessment tools that only send a predefined vulnerability detection query.

To solve the problem that the user needs to set a huge number of parameters, the proposed method automatically obtains information on the HTTP request method (hereinafter called “method”), the content of the endpoint, the data (i.e., parameters) to be entered in the request body, the type of parameters, and the existence of authentication from the reference. Then the proposed method automatically sets the parameters and other information.

TABLE II
EXAMPLE OF WORDPRESS WEBAPI REFERENCE

Method	Endpoint	Parameters and Type	Auth
GET	/posts	context(str), author(str)	False
POST	/posts	title(str), content(str)	True
GET	/posts/{id}	id(int), context(str)	False
POST	/posts/{id}	id(int), title(str)	True
...	...	...	...

B. Detection Procedure

The flowchart of the detection procedure of the proposed method is shown in Figure 1.

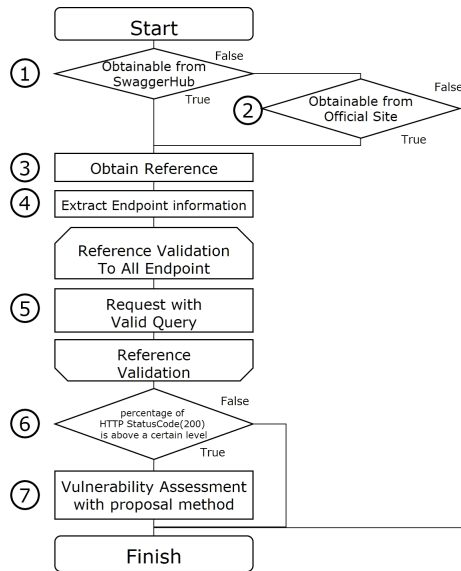


Fig. 1. Flow Chart of the Proposal Method

In this section, prerequisites for the proposed method are described.

1) *Obtaining references*: In the proposed method, the first step in conducting a vulnerability assessment of a WebAPI is to obtain the reference of the target WebAPI and extract the information to be used for the vulnerability assessment (① – ④ in Figure 1).

An example of the extracted information (③ in Figure 1) is shown in Table II.

In the proposed method, the reference is obtained by the following two methods.

(1) SwaggerHub (① in Figure 1)

When WebAPI references are provided in HTML format, it is difficult to efficiently obtain the above-mentioned information (Table II). Therefore, the proposed method uses SwaggerHub [8] to obtain the reference of the WebAPI for vulnerability assessment. SwaggerHub is a website that provides a variety of information for WebAPI developers. SwaggerHub has many references of WebAPIs written in JSON or YAML format.

(2) Reference on the official website (② in Figure 1)

POST /wp-json/wp/v2/posts HTTP/1.1
Content-Type: application/json

```
{“title”:“ArticleName”,“content”:“ArticleContent”}
```

Fig. 2. Example query for Verify

If the reference does not exist on SwaggerHub, the proposed method obtains the reference from the official website of the WebAPI.

2) *Validation check of the reference*: To confirm the validity of the reference, a proper query is generated based on the information extracted from the reference (Table II) and a request is actually sent (⑤ in Figure 1).

If there is a variable in the endpoint that takes an optional value, a parameter identical to the variable in the endpoint is retrieved from the endpoint one level up and the first value of the variable is used as the variable in the endpoint.

For example, if there is an endpoint of /wp/v2/posts/{id} (where “id” is a variable that can take any value of numeric type), the proposed method sends a request to /wp/v2/posts, which is an endpoint one level up, searches for a parameter (e.g., {“id” : 5}) with the variable “id” in the response, and uses the first value (e.g., 5 in {“id” : 5}) of the variable “id” in the endpoint. After that, the proposed method rewrites the variable “id” to the first value 5 and sends a request to /wp/v2/posts/5. If there is no endpoint one level up or if there is no parameter with the same name as the variable that takes an arbitrary value, the proposed method rewrites the variable “id” to 1 (assuming 1 as an arbitrary value) and sends a request to /wp/v2/posts/1. If “id” is a variable that can take any value of string type, the proposed method rewrites the variable “id” to a random string (assuming “abcd” as a random string) and sends a request to /wp/v2/posts/abcd.

As for parameters (data to be entered into the request body) of POST method, the key and its value are set in the same way as above. Figure 2 shows an example of a request sent in the validity check for the second row endpoint in Table II.

Finally, the proposed method checks the HTTP status code of the response to a request (e.g., request to /wp/v2/posts/1). If the percentage of errors in the 400 range (e.g., HTTP 404 Not Found) is below a certain level (⑤ in Figure 1), the proposed method determines that the obtained reference is valid (⑤ in Figure 1).

If this percentage is above a certain level, the vulnerability assessment using the proposed method is considered to be impossible because the reference is not valid.

Therefore, the proposed method is applied on the assumption that a valid WebAPI reference has been obtained.

C. Detection of the No.1 Broken Object Level Authorization

The detection of No.1 is conducted using the following three detection methods. The first method is to detect endpoints that are not described in the reference, using the names of keys in the JSON data (hereinafter called *candidate keys*) (Figure 4). The second method is to detect endpoints that are not

described in the reference by substituting or merging endpoint values based on information using the request and its response (Figure 6). The third method is to detect endpoints that are not described in the reference by substituting or merging parameter values based on the parameter types defined in the reference and information obtained from the request and its response (Figure 7).

Specific detection methods are shown below.

(1) Detecting vulnerable endpoints by combining path

Endpoints that are not described in the reference are detected by using the *candidate keys* included in the response of the validation check of the reference (④ in Figure 1). The flowchart of this detection process is shown in Figure 4.

The proposed method extracts all the *candidate keys* from the validation response (① in Figure 4) and enters them after the common part of the WebAPI endpoint to be vulnerability-tested (hereinafter called “the base path”) to form a new endpoint (Figure 3). For example, as shown in Figure 3, if /wp/v2 is given as the base path and “id”, “title”, and “status” are given as the *candidate keys* extracted from the response of the validation check, the proposed method constructs three new endpoints, /wp/v2/id, /wp/v2/title, and /wp/v2/status.

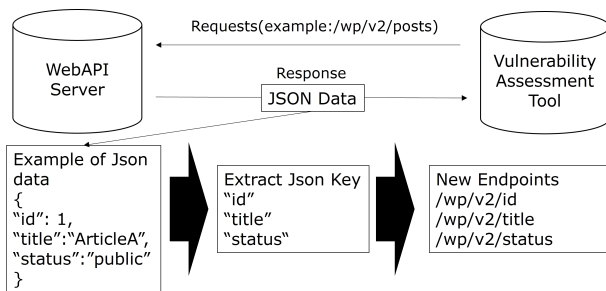


Fig. 3. Reconfigure endpoints

The reason for constructing the endpoints by this method is that, in the case of CMS WebAPIs, CMS-specific features (e.g., “title”, “comment”, “category”) are included in the parameters of the response to a request (e.g., request to /wp/v2/posts), and thus the *candidate keys* are likely to be valid endpoints. For example, in the case of Wordpress, 8 of the 87 *candidate keys* obtained from the validation check of the Wordpress reference are the same as the endpoints described in the reference.

The proposed method sends a GET request to the above-mentioned endpoint and checks the HTTP status code of the response (③ in Figure 4). If the HTTP status code can be confirmed to exist such as 200 or 401, the proposed method checks if the endpoint is described in the reference (④ in Figure 4). For example, if a request is sent to /wp/v2/status and the HTTP status code is 200, the proposed method checks if the endpoint is described in the reference. If it is described in the reference, the proposed method determines that it is a valid endpoint and that it does not contain any vulnerabilities. If not, it is determined that it is not a valid endpoint and the No.1 vulnerability is included.

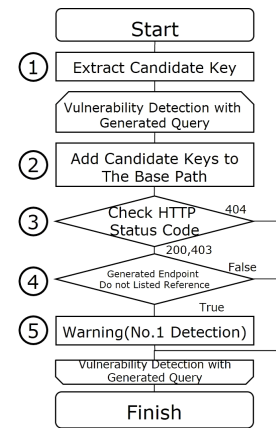


Fig. 4. Flowchart of endpoint detection not described in the reference

```
{ "id": 38,
  "title": "ArticleName" },
{ "id": 1,
  "title": "Hello world!" }
```

Fig. 5. Response to requests for /wp/v2/posts

(2) Detecting vulnerable endpoints by replacing variables

The purpose of this section is to detect unpublished endpoints. For numeric values, vulnerability verification is performed on boundary value analysis using the response, and for strings, vulnerability verification is performed on *candidate keys*, which are the names of JSON keys included in the response.

By replacing variables included in endpoints, endpoints that are out of range or endpoints that are not described in the reference can be searched.

For example, if there is an endpoint of /wp/v2/posts/{id} (where “id” is a variable that can take any value of numeric or string type), the proposed method detects endpoints that are out of range or endpoints that are not described in the reference by replacing the “id” using the *candidate keys* (④ in Figure 1) included in the response of the validation check of the reference (① in Figure 6).

The detection is done through the following process.

1) Detecting valid endpoints

To obtain the boundary value, if there is an endpoint of /wp/v2/posts/{id}, the proposed method checks whether the endpoint /wp/v2/posts, which is an endpoint one level up, is described in the reference (② in Figure 6). If the endpoint /wp/v2/posts is listed in the reference, the proposed method checks the response to the request to /wp/v2/posts and retrieves all the JSON data (keys and values) from the response body.

Figure 5 shows an example of the response to the request to /wp/v2/posts. In Figure 5, the values of 38 and 1 are used for “id”.

2) Request generation and vulnerability detection by responses

For those endpoints that take optional values of numeric or string type, the keys and values are retrieved from the responses to the requests to the endpoints one level up and endpoints for new queries are generated using the bounds of the values (④ and ⑤ in Figure 6).

In Figure 6, 38 and 1 are set as values of the key “id”, so the boundary value of the maximum value is 39 (maximum value (38) + 1), the boundary value of the minimum value is 0 (minimum value (1) - 1), and the mean value is 19.5. Fractions of the mean value after the decimal point are rounded to an even number. In this case, 20 is used as the mean value. The proposed method constructs a new endpoint (i.e., /wp/v2/posts/39, /wp/v2/posts/1 and /wp/v2/posts/20) with the boundary values set in the variables included in the endpoint (not including parameters and authentication), and sends the request to the new endpoint.

For those endpoints that take optional values of string type, the values of *candidate key* are set in the variables included in the endpoint (⑥ in Figure 6) and the proposed method sends the request to the new constructed endpoint.

After sending the request, the proposed method checks the HTTP status code from the response (⑦ in Figure 6). If the HTTP status code is 200 or 401, the proposed method determines that there is the No.1 vulnerability (⑧ in Figure 6).

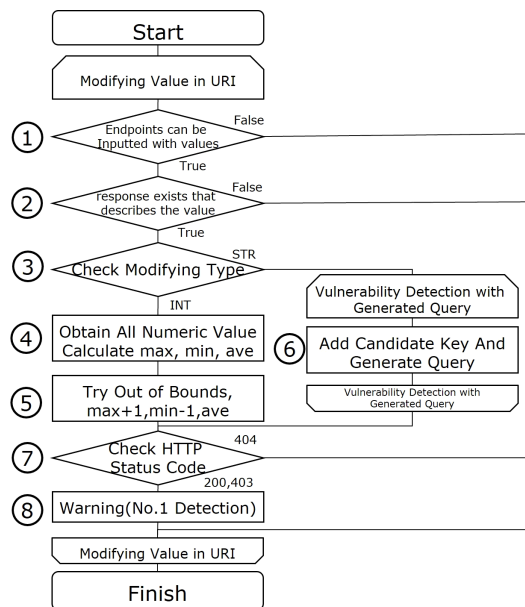


Fig. 6. Flowchart of changing numbers and strings in an endpoint

(3) Detecting vulnerable endpoints by replacing parameters

The purpose of this section is to perform vulnerability verification using values of a different type from the type defined in the reference or values combined with values of a different type as variable values, since there may be problems when type-casting is used in WebAPI.

By replacing parameters (data input to the request body), endpoints that are not described in the reference are detected. The replacing process is divided into cases for parameters of numeric or string type (② in Figure 7).

In the case of parameters with values of numeric type (③ in Figure 7), the value of numeric type is replaced with the value of string type. In the proposed method, when the value of string type is constructed, the values “a” of string type are used. For example, if the value of the key “id” is 5 (i.e., {“id” : 5}), the proposed method replaces 5 with “a” and generates {“id” : “a”}. In addition, the proposed method also replaces 5 with “5a” and generates {“id” : “5a”}. Here, the value “5a” means that the numeric value 5 is converted to the string value “5” and the string values “5” and “a” are concatenated.

In the case of parameters with values of string type (④ in Figure 7), the value of string type is replaced with the string value to which the numeric value is converted. In the proposed method, when the value of string type is constructed, the values “1” of string type are used. For example, if the value of the key “status” is “publish” (i.e., {“status” : “publish”}), the proposed method replaces “publish” with “1” and generates {“status” : “1”}. In addition, the proposed method also replaces “publish” with “publish1” and generates {“id” : “publish1”}. Here, the value “publish1” means that the numeric value 1 is converted to the string value “1” and the string values “publish” and “1” are concatenated.

After sending the request, the proposed method checks the HTTP status code of the response to the sent request (⑤ in Figure 7). If the HTTP status code (200 or 401) indicates the existence of the vulnerability, the proposed method determines that there is the No.1 vulnerability (⑥ in Figure 7).

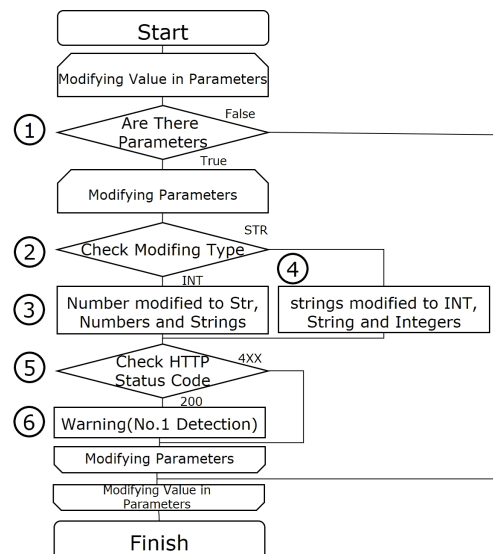


Fig. 7. Flowchart for changing numbers and strings in parameters

D. Detection of No.5 Broken Function Level Authorization

The purpose of this section is to detect vulnerabilities by sending a request to a valid endpoint that requires authen-

tication. Therefore, the process of detecting valid endpoints that require authentication is added before the vulnerability detection in the No.1 vulnerability. After the process of detecting endpoints that require authentication, the process is the same as the No.1 vulnerability detection.

Figure 8 shows the flowchart of the detection of No.5.

(1) Extracting the endpoints that require authentication from the WebAPI reference (① in Figure 8). Sending a request to the extracted endpoint that requires authentication using the valid endpoint and parameters used in the validation check. The authentication information is not included at this time (② in Figure 8).

(2) Checking the HTTP status code of the response. If the response is returned with an HTTP status code such as 400 or 401, it is indicated that the parameter format is incorrect or that authentication is required. In this case, it is determined that the No.5 vulnerability is not included (③ in Figure 8).

After that, among the endpoints that need to be authenticated, the proposed method checks if the endpoint needs to be parameterized (⑤ in Figure 8). If parameters need to be set, the replacing process for the parameters is performed.

The parameter replacing process (⑥ and ⑦ in Figure 8) is described in “(3). Detecting vulnerable endpoints by replacing parameters” of Section V-C.

After replacing the parameters, the proposed method sends a request that does not include authentication information. After that, the proposed method checks the HTTP status code of the response, and if the HTTP status code contains a 200 indicating that the request was successfully processed (⑧ in Figure 8), the replacing process is done and it is determined that the request contains the No. 1 vulnerability (⑨ in Figure 8).

VI. EXPERIMENT

A. Experimental Environment

In this experiment, we use the WebAPI of the Wordpress to detect the vulnerability. Wordpress version 4.7.0 or 4.7.1 contains a vulnerability of article tampering by ignoring authentication, which is classified as No.5 in the OWASP API Security Top 10. In this experiment, we build Wordpress 4.7.0 and detect the article tampering vulnerability by using the Wordpress WebAPI.

B. Outline of the Wordpress Vulnerability of Article Tampering [1]

(1) Sending a GET request to the WebAPI endpoint /wp/v2/posts to retrieve the list of articles. Figure 9 shows a part of the response. The value of “id” in Figure 9 is the list of posts published in Wordpress. In the response of Figure 9, there are articles with IDs of 38, 15 and 1.

(2) Because the endpoint /wp/v2/posts/{id} has a vulnerability to ignore authentication, a request with parameters as shown in Figure 10 is sent. In Figure 10, an example of article tampering is shown for an article with an ID of 38.

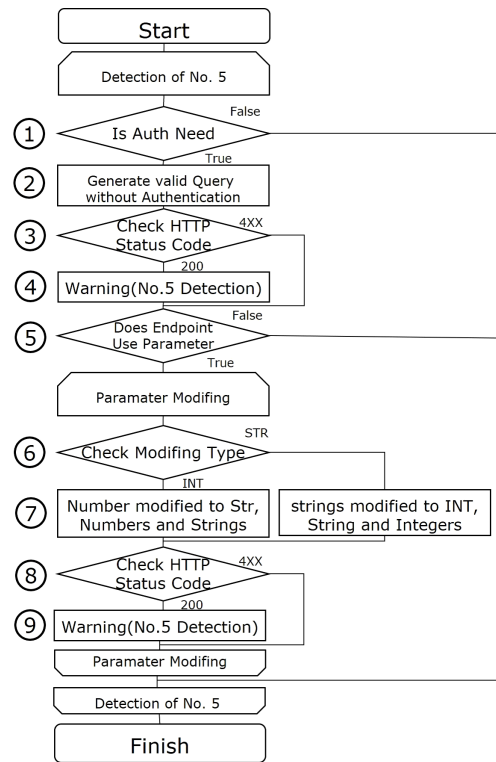


Fig. 8. Flowchart of No.5 detection

```

{
  "id": 38,
  "title": "ArticleName" },
  {
    "id": 15,
    "title": "ArticleNameB" },
  {
    "id": 1,
    "title": "Hello world!" }

```

Fig. 9. Part of Response of /wp/v2/posts

(3) The request in (2) is successfully processed, and the article is tampered with successfully. Figure 11 shows a part of the response when the tampering is successful.

```

POST /wordpress/wp-json/wp/v2/posts/38 HTTP/1.1
Host: localhost:50080
Content-Type: application/json

{"id": "38a", "title": "Title Changed"}

```

Fig. 10. Example query for tampering

```

{
  "id": 38,
  "title": "Title Changed" }

```

Fig. 11. Example Response for tampering

TABLE III
FIRST FOUR CASES EXTRACTED FROM THE REFERENCE.

Method	Endpoint	Parameters and Type	Auth
GET	/posts	context(str), author(str)	False
POST	/posts	title(str), content(str)	True
GET	/posts/{id}	id(int), context(str)	False
POST	/posts/{id}	id(int), title(str)	True

TABLE IV
EXECUTION RESULTS OF THE No.5 DETECTION IN THE PROPOSED
METHOD IN /wp/v2/posts/{id}

Value of "id"	Value of "title"	HTTP status code	note
38	"Title Changed"	—	①
"a"	"Title Changed"	401	②
"38a"	"Title Changed"	200	③
38	"1"	401	④
38	"Title Changed1"	401	⑤

C. Experimental Results and Discussion of the Proposed Method for Vulnerability Assessment

The vulnerability assessment using the proposed method was performed on the experimental environment. First, we describe the results of obtaining the references and the validation check (Figure 1). The Wordpress reference was obtained from SwaggerHub (① – ③ in Figure 1).

Next, the information used for vulnerability assessment is extracted from the reference (④ in Figure 1). The first four extracted results are shown in Table III.

The validation of the extracted endpoint is checked (⑤ in Figure 1). A request is sent to the endpoint obtained from Table III, and the HTTP status code in the response confirms that the reference is valid.

After the validation check was completed, we conducted the detection method for No.1 and No.5. No.1 was not detected by the proposed method, because the vulnerability of No.1 does not exist in the environment of Wordpress 4.7.0.

Next, we describe the detection of No.5 (Figure 8) and its result. The first and second columns of Table IV show the change pattern of the parameters (the values of the keys "id" and "title") used in the endpoint /wp/v2/posts/{id} in the fourth line of Table III. The third column of Table IV shows the HTTP status code of the response for the pattern of the parameters (i.e., the first and second columns). In /wp/v2/posts/{id}, parameters such as "id", "title" and "content" are used, however only "id" and "title" are used in this experiment for simplicity. We use the value 38 as the value of the key "id", which is the first appearance of the key "id" in the response of the endpoint /wp/v2/post one level up.

The value of key "title" can be retrieved from the response of /wp/v2/posts. However, to make the success of the tampering obvious from the change of the title name, the valid parameter to use for "title" is the string "Title Changed" instead of the string "title" that first appeared. The valid parameters are shown in note column ① of Table IV.

The first thing to replace in the detection of No.5 is the key "id" of the first parameter. Since the key "id" is numeric type,

the value of string type is used. As a result of this replacing, the value of the key "id" becomes "a" (the note column ② of Table IV). The HTTP status code of the response was 401, so the proposal method concludes that the vulnerability No. 5 does not exist under this condition.

Next, the numeric value 38 is converted to the string value "38" and the string values "38" and "a" are concatenated. As a result of the concatenation, the value of the key "id" becomes "38a" (the note column ③ of Table IV). A request with the replaced parameters was sent and the response returned the HTTP status code 200 indicating that the request was successfully processed. Figure 11 shows a portion of the response body for successful tampering. In Figure 9, the value of the key "title" with the value 38 of the key "id" is "ArticleName". However, in Figure 11, the value of the key "title" with the value 38 of the key "id" is changed to "Title Changed". This result means that /wp/v2/posts/{id} (the note column ③ of Table IV) has the No.5 vulnerability.

After the proposal method has completed all the patterns for the value of the key "id", next replace the value of the key "title". The value of the key "id" is reverted to 38, which is the valid parameter pattern shown in note column ① of table IV.

Since the value of the key "title" is string type, this value is replaced with the value of string type including the string value which the numeric value is converted to string type. As a result of the replacing, the value of the key "title" becomes "1" (the note column ④ of Table IV). In this parameter pattern, the HTTP status code of the response is 401, so the proposal method concludes that there is no vulnerability in terms of No.5 under this condition.

Next, the string values "Title Changed" and "1" are concatenated and the concatenated value is used. As a result of the replacing process, the value of the key "title" becomes "Title Changed1" (the note column ⑤ of Table IV). In this parameter pattern, the HTTP status code of the response is 401, so the proposal method concludes that there is no vulnerability in terms of No.5 under this condition. The proposed method can detect the vulnerability of article tampering using the Wordpress WebAPI (which is classified as No. 5 in the OWASP API Security Top 10), which has been difficult to be detected by the existing vulnerability assessment tools.

VII. CONCLUSION

In this paper, we have proposed a method to repeatedly analyze requests and their responses for two vulnerabilities in the OWASP API Security Top 10. In the experiments, we showed that the proposed method can detect the vulnerability of article tampering using the Wordpress WebAPI. In the future, we plan to develop a method that can detect vulnerabilities other than those listed in No.1 and No.5 of the OWASP API Security Top 10. In addition, we need to conduct an evaluation experiment using Wordpress, which is actually in operation.

REFERENCES

- [1] Marc-Alexandre Montpas, "Content Injection Vulnerability in WordPress," <https://blog.sucuri.net/2017/02/content-injection-vulnerability-wordpress-rest-api.html>, 2021/8/21.
- [2] Wordpress, "WP-API," <https://developer.wordpress.org/rest-api/>, 2021/8/21.
- [3] OWASP, "OWASP API Security TOP 10 2019," <https://raw.githubusercontent.com/OWASP/OWASP-API-Security/master/2019/en/dist/owasp-api-security-top-10.pdf>, 2021/8/21.
- [4] OWASP, "OWASP TOP 10 - 2017," https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf, 2021/8/20.
- [5] Yusuke Takamatsu, Yuji Kosuga, Kenji Kono: "Automated Detection of Session Management Vulnerabilities in Web Applications," In Proceedings of IEEE Annual Conference on Privacy, Security and Trust(PST'12), pp.112-119, Jul. 2012.
- [6] imperva, "automatic-api-attack-tool," <https://github.com/imperva/automatic-api-attack-tool>, 2021/8/20.
- [7] VegaBird TECHNOLOGIES, "Vooki - Web Application and API Vulnerability Scanner (DAST TOOL)," <https://www.vegabird.com/vooki/>, 2021/8/20.
- [8] SwaggerHUB, "SwaggerHUB," <https://app.swaggerhub.com/search>, 2021/8/20.



Toru Taya received Bachelor of Information Sciences from Tokyo University of Information Sciences, Japan, in 2020. Entered Graduate School of Informatics, Tokyo University of Information Sciences in the same year.



Masaki Hanada received the B.E. degree in resources engineering from Waseda University in 1996, the M.S. degree in information science from Japan Advanced Institute of Science and Technology (JAIST) in 1999, and the M.S. and D.S. degrees in global information and telecommunication studies from Waseda University in 2003 and 2007, respectively. He worked at Waseda University and Tokyo University of Science. After joining Tokyo University of Information Sciences as an Assistant Professor in 2011, he has been a Professor in the

Department of Information Systems, Tokyo University of Information Sciences, since 2019. His research interests include network QoS control, network resource control and management, and network security. He is a member of the IEEE, IEICE and IPSJ.



Yoichi Murakami received his Bachelor of Business Administration and Information Sciences from Tokyo University of Information Sciences, Japan, in 2000, his M.Sc. in Bioinformatics from the School of Life Sciences, University of Sussex, United Kingdom, in 2006, and his Ph.D. in Science from the Graduate School of Frontier Bioscience, Osaka University, Japan, in 2012. In 2017, he joined the Department of Informatics, Tokyo University of Information Sciences, as an assistant professor, and in 2019 he became an associate professor. His current

research interests are in the areas of bioinformatics, machine learning, data mining and knowledge discovery.



privacy protection. He is a member of the IEICE and IPSJ.

Atsushi Waseda received the B.E. degree in communication engineering from the University of Electro-Communications in 2000. He received his M.S. and Ph.D. in information science from Japan Advanced Institute of Science and Technology (JAIST) in 2002, and 2007, respectively. He worked at the National Institute of Information and Communications Technology and KDDI research inc. After joining Tokyo University of Information Sciences as an Assistant Professor in 2019. His research interests include information security, quantum security and



Yuki Ishida received the Bachelor's and Master's degrees in Informatics from Tokyo University of Information Sciences, Japan, in 2014 and 2016, respectively. He joined SecureBrain Corporation in Tokyo in 2019, after participating in security product development at a software company. He is a software engineer and has been engaged in research and development for cyber security. He is a member of the IEICE and IPSJ.

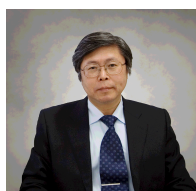


Takao Mimura received the Bachelor of Business Administration and Information Science from Hokkaido Information University in Hokkaido, Japan in 2007. He joined SecureBrain Corporation in Tokyo, Japan in 2014 after participating in product development at software companies. He is a senior software engineer and has been engaged in research and development for cyber security.



IMS and Parlay. In 2005 he joined Tokyo University of Information Sciences and has been engaged in research on Ubiquitous Network.

Moo Wan Kim received B.E., M.E. and Ph.D degree in electronic engineering from Osaka University, Osaka, Japan in 1974, 1977 and 1980, respectively. He joined Fujitsu Lab. in 1980 and had been engaged in research and development on multimedia communication systems, Intelligent Network, ATM switching system and operating system. In 1998 he joined Motorola Japan and had been engaged in research and development on CDMA2000 system. In 2000 he joined Lucent Japan and had been engaged in research and development on W-CDMA system,



Eiji Nunohiro received the Doctor of Engineering from Nihon University. He worked at Hitachi, Ltd in 1985. After joining Tokyo University of Information Sciences as an Associate Professor in 2002, he has been a Professor in the Department of Information Systems, Tokyo University of Information Sciences, since 2007. His research interests include compiler and parallel algorithm in high performance computing, cyber security, programming training support system. He is a member of the IPSJ.