リクエストとそのレスポンスを考慮したWebAPI脆弱性診断手法

田谷 透[1,a)]　花田 真樹[2]　村上 洋一[2]　早稲田 篤志[2]　石田 裕貴[3]　三村 隆夫[3]　布広 永示[2]

概要：近年，Web サービスを外部から利用するために WebAPI を公開するケースが増えている．一方，WebAPI の脆弱性を悪用した被害も増加傾向にある．WebAPI の脆弱性を悪用した被害を防止するため，OWASP(Open Web Application Security Project) は，報告書 (OWASP API Secyrity Top 10) に，WebAPI のセキュリティリスクの高い 10 項目の脆弱性の内容を記述している．しかし，この報告書には，攻撃の具体的手法と詳細な対策手法が記述されていない．そのため，既存の WebAPI の脆弱性診断ツールにおいて，検出困難な脆弱性の項目が存在する．本研究では，WebAPI のリファレンスを自動的に取得し，WebAPI のリクエストとそのレスポンスの解析を繰り返し行う脆弱性診断方式を提案する．これにより，報告書 (OWASP API Secyrity Top 10) に記述されている脆弱性を標的とした攻撃のうち，既存の脆弱性診断ツールで検出困難な脆弱性の項目に対する脆弱性診断が可能となる．

キーワード：WebAPI, 脆弱性診断

A Method for WebAPI Vulnerability Assessment Considering Requests and Responses

Toru Taya[1,a)]　Masaki Hanada[2]　Yoichi Murakami[2]　Atsushi Waseda[2]　Yuki Ishida[3]
Takao Mimura[3]　Eiji Nunohiro[2]

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 WebAPI vulnerabilities, the OWASP (Open Web Application Security Project) has published a guideline (OWASP API Security Top 10) describing the ten vulnerabilities that pose the highest security risk to WebAPIs. However, the guideline does not describe the specific 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 request and its response. The proposed vulnerability assessment method will enable vulnerability assessment for vulnerability items that are difficult to be detected by existing vulnerability assessment tools among the attacks targeting the vulnerabilities described in the guideline.

Keywords: WebAPI, Vulnerability Assessment

1. はじめに

WebAPI はネット銀行，EC サイト，気象情報サイトなど様々な場所や使用目的で提供されており，現代の Web ア

1 東京情報大学大学院 総合情報学研究科
Tokyo University of Information Sciences, Graduate School of Infomatics
2 東京情報大学 総合情報学部
Tokyo University of Information Sciences, Department of Infomatics
3 株式会社セキュアブレイン
SecureBrain Corporation
a) g20004tt@edu.tuis.ac.jp

プリケーションにとって必要不可欠な技術となっている．WebAPI（Web Application Programming Interface）は，Web を通してプログラム間でデータをやり取りする際に使用される．WebAPI では，Web 閲覧同様に HTTP/S を使用して通信を行うが，プログラムが理解しやすいように，HTML ではなく JSON や XML などのデータ形式を用いることが多い．本研究では，REST WebAPI を対象とした脆弱性診断を行う．WebAPI が普及する一方で，2017 年には Wordpress の公開する WebAPI[1] を悪用した記事改ざんの脆弱性 [2] が攻撃に利用され，大きな被害をもたらし，WebAPI のセキュリティ問題がより注目されるようになった．WebAPI に脆弱性が含まれていないことを確認するため，WebAPI の脆弱性を診断するツールが多く公開されている．しかし，既存の脆弱性診断ツールは，WebAPI のロジックや特性を考慮しておらず，対象とする脆弱性診断に必要な検出クエリを発行しないため，Wordpress の記事改ざんの脆弱性 [2] のような広く知られている脆弱性であっても検出を行うことができないという問題がある．そこで，本研究では，WebAPI へのリクエストとそのレスポンスを考慮した WebAPI 脆弱性診断手法を提案する．

本論文の構成は次のとおりである．2 章では，WebAPI の最もセキュリティリスクの高い 10 の脆弱性項目について述べる．3 章では，本研究に関連する研究について述べる．4 章では，既存の脆弱性診断ツールについて述べる．5 章で本研究の提案手法である，リクエストとそのレスポンスを考慮した WebAPI 脆弱性診断手法について述べ，6 章では，既存の脆弱性診断ツールで検出困難である Wordpress の WebAPI を使用した記事改ざんの脆弱性を提案手法で検出可能であることを示す．最後に 7 章で，本論文のまとめを述べる．

2. OWASP API Secyrity Top 10[3]

Web アプリケーションのセキュリティ分野の研究やガイドライン作成などの活動を行なっている OWASP（Open Web Application Security Project）は，WebAPI に関する最もセキュリティリスクの高い 10 の脆弱性項目と取るべき対策，脆弱性の発生シナリオについて，ガイドライン (OWASP API Secyrity Top 10) をまとめている．ガイドラインでは，表 1 に示す 10 の脆弱性項目が報告されている．脆弱性項目には，大きく 4 つの分類がある．1 つ目の分類は，適切な権限が与えられていないことにより，管理者用の WebAPI が一般ユーザに使用されてしまうなど，アクセスコントロールに関わる脆弱性である．2 つ目の分類は，WebAPI のサーバーが，Web アプリケーション側での処理を行うことを前提にデータをフィルタリングせずにクライアントに送信してしまうことや，リクエストボディに定義されていないデータを追加することで不具合を発生させるような，データの割り当てに関連する脆弱性である．3 つ目の分類は，開発段階の問題がある API が公開されてしまう脆弱性や WebAPI を提供するサーバのセキュリティ設定に関する誤りなど，開発運用段階で発生する脆弱性である．最後の分類は，インジェクションやログ取得設定の誤りなど，従来の Web の脅威として存在する脆弱性 [4] と同一のものである．

表 1 OWASP API Secyrity Top 10 の脆弱性一覧
Table 1 List of Vulnerability(OWASP API Secyrity Top 10)

No.#	脆弱性名称
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

本研究では, OWASP API Secyrity Top 10 のうち, No.1, No.5 の脆弱性項目を診断対象とする．以下で脆弱性項目の詳細について述べる．

2.1 No.1 Broken Object Level Authorization

“No.1 Broken Object Level Authorization” の脆弱性は，未公開の WebAPI のエンドポイントが使用される脆弱性である．発生シナリオとしては，/shop/{shopName}/revenue_data.json というエンドポイントが公開されていた場合，{shopName} という文字列に対してあらかじめ他のエンドポイントなどから入手した shopName のリストを用いて書き換える処理を行い，shopName ごとの販売データにアクセスされることによって脆弱性が使用されるというシナリオが記述されている．対策手法としては，エンドポイントに対して認証プロセスを用意し，ユーザに適切な権限を用意することや，シナリオの {shopName} のような文字列をランダムで予測できない値にすることなどがある．

2.2 No.5 Broken Function Level Authorization

“No.5 Broken Function Level Authorization” の脆弱性は，一般ユーザによって管理者用の WebAPI などの権限が必要なエンドポイントを使用される脆弱性である．発生シナリオとしては，一般ユーザには公開していない管理者用のエンドポイントが使用されてしまうことや一般ユーザ向けに公開されているエンドポイントの HTTP メソッドを公開されていないメソッドに変更することが挙げられている．対策手法としては，一般ユーザに公開していないエンドポイントにも認証のメカニズムを実装すること，権限が

適切に設定されているかを確認することなどがある．

OWASP API Secyrity Top 10 の報告書には，上述のように，各脆弱性項目に関する概要，攻撃の発生シナリオの例，簡単な対策手法が記述されているが，それらの攻撃クエリの詳細や具体的な対策やその検出方法について記述されていない．このため，既存の脆弱性診断ツールでは検出困難な脆弱性の項目が存在する．

3. 関連研究

Web アプリケーションのセキュリティに関する多くの研究が行われている．Web アプリケーションのロジックに依存した脆弱性診断手法を提案した研究として，高松らの研究 [5] がある．本関連研究では Web アプリケーションのロジック，つまり，認証に使用する情報やリクエストが正常に終了したときの応答のパターンなどを考慮した，Web アプリケーションの脆弱性診断手法を提案している．脆弱性診断を行う際には，診断対象の幅広い面から診断を行う必要があるため，認証の情報やそのレスポンスの内容は非常に重要となる．WebAPI でも同様に重要であると考えられるが，本関連研究では Web アプリケーションのロジックにのみ着目し，WebAPI のロジックや特徴を扱っていない．

4. 既存の脆弱性診断ツール

WebAPI の脆弱性を検出することを目的としたツールは多く公開されている．しかし，これらの脆弱性診断ツールは，OWASP API Secyrity Top 10 に記述されている 10 の脆弱性項目の全てを網羅する検出は実施できない．網羅する検出ができていない原因として，あらかじめ決められたリクエストを送信することで検出を試みていることや，診断対象とする WebAPI ごとの認証情報を指定できないことが考えられる．加えて，診断に用いる様々なリクエストボディのデータ (以下，パラメータ) をユーザ側が指定することになっており，ユーザ側が適切なパラメータを設定できない場合に診断が困難となる．以下に 3 つの脆弱性診断ツールの特徴と診断可能な脆弱性項目について述べる．

4.1 Automatic API Attack Tool[6]

“Automatic API Attack tool” は，特徴として，JSON または YAML 形式の WebAPI リファレンスを読み込ませることで，WebAPI に合わせて柔軟に脆弱性検出のテストケースを作成することができる．例えば，“id”:“INT 型” のデータを用いるエンドポイントに対し，Automatic API Attack Tool は，“id” の値に long 型や，double 型など，別の型の数値に変更し，脆弱性検出のリクエストを送信する．このツールでは，OWASP API Secyrity Top 10 の No.1，No.5，No.9 の脆弱性項目の検出を行うことができる．しかし，WebAPI の情報を記述した JSON または YAML 形式の WebAPI リファレンスが必要であり，脆弱性の診断に多くの作業や時間を要する場合がある．

4.2 Vooki, Rest API Scanner[7]

“Vooki, Rest API Scanner” は，OWASP API Secyrity Top 10 の脆弱性項目の No.3，No.7，No.8 の脆弱性項目を検出するリクエストを送信する脆弱性診断ツールである．しかし，WebAPI の診断に用いるエンドポイント，使用されるヘッダ，パラメータ全てを診断するユーザ側で設定する必要がある．脆弱性診断に必要な情報を設定できない場合，脆弱性が存在しても検出できない可能性がある．

5. 提案方式

5.1 概要

提案方式では，既存脆弱性診断ツールの，決められた脆弱性検出クエリしか送信しないという問題点に対して，リファレンスを自動的に取得し，内容に応じて脆弱性の検出クエリを動的に生成する．膨大な全てのパラメータをユーザ側で設定する必要があるという問題点に対しては，HTTP リクエストメソッド（以下，メソッド），エンドポイントの内容，リクエストボディに入力するデータ（以下，パラメータ），パラメータの型，認証の有無に関する情報をリファレンスから自動的に取得し，パラメータ等の情報を自動的に設定することで解決を図る．

5.2 検出手順

提案方式の検出手順のフローチャートを図 1 に示す．以下では，図 1 に記載するフローチャートに従って手順を説明する．

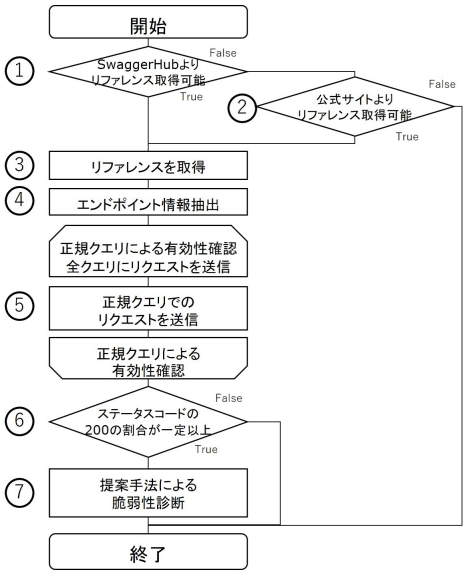

図 1 提案方式のフローチャート
Fig. 1 Flow Chart of the Proposal Method

表 2　Wordpress の WebAPI におけるリファレンスの内容抽出例
Table 2　Example of Wordpress WebAPI Reference

メソッド	エンドポイント	パラメータと型 (抜粋)	認証
GET	/posts	context(str), author(str)	不要
POST	/posts	title(str), content(str)	必要
GET	/posts/{id}	id(int), context(str)	不要
POST	/posts/{id}	id(int), title(str)	必要
…	…	…	…

5.2.1　提案手法の前提条件

5.2.1.1　リファレンスの取得

本提案方式では，WebAPI に対する脆弱性診断を行う最初の段階として，診断対象とする WebAPI のリファレンスを取得し，脆弱性診断に用いる情報を抽出する（図 1 の①から④）．Wordpress の WebAPI におけるリファレンスの内容を取得し，抽出した情報（図 1 の③）の例を表 2 に示す．

本提案方式では，以下の 2 つの手法でリファレンスを取得する．

（1） SwaggerHub（図 1 の①）

WebAPI のリファレンスは，HTML 形式で提供されている場合，上述した情報（表 2）を効率的に取得することが困難である．そのため，本提案手法では，SwaggerHub[8] を利用し，診断対象とする WebAPI のリファレンスを取得する．SwaggerHub とは，WebAPI の開発者のための様々な情報やツールを提供しているサイトである．SwaggerHub には，JSON または YAML 形式で記述された WebAPI のリファレンスが数多く存在している．

（2） 公式サイトのリファレンス（図 1 の②）

リファレンスが SwaggerHub に存在しない場合，診断対象とする WebAPI の公式サイトからリファレンスを取得する．

5.2.1.2　リファレンスの有効性確認

リファレンスの有効性確認は，リファレンスから抽出した情報（表 2）を基に正規のクエリを生成し，リクエストを送信する（図 1 の⑤）．

このとき，エンドポイント内に任意の値を取る変数があった場合，一階層上のエンドポイントから，エンドポイント内の変数の名称と同一のパラメータを取得し，最初に出現した値をエンドポイント内の変数として使用する．例えば，/wp/v2/posts/{id} のエンドポイント（“id” が任意の値を取る変数）があれば，一階層上のエンドポイントである，/wp/v2/posts にリクエストを送信し，レスポンスから “id” という名称のパラメータを探し，最初に出現した “id” の値に変数を書き換える．一階層上のエンドポイントが存在しないか，任意の値を取る変数の名称と同一のパラメータが存在しない場合，変数を “1”（適当な値として 1 を使用する）に書き換える．

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

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

図 2　有効性確認のリクエスト例
Fig. 2　Example query for Verify

エンドポイント内に任意の文字列を取る変数も数値と同様に，一階層上のエンドポイントから，任意の文字列を取る変数の名称と同一のパラメータを探し，最初に出現した文字列を取得し，その文字列にエンドポイント内の変数を書き換える．一階層上のエンドポイントが存在しないか，任意の文字列を取る変数の名称と同一のパラメータが存在していなかった場合，ランダムな文字列に置き換える．

パラメータ（リクエストボディに入力するデータ）に関しても，一階層上のエンドポイントからパラメータに使用する名称と同じキーの値から，数値・文字列を決定する．表 2 の 2 行目エンドポイントに対する有効性確認で送信するリクエストの例を，図 2 に示す．

リクエストに対するレスポンスの HTTP ステータスコードを確認し，400 番台のエラー（HTTP 404 Not Found など）の割合が一定以下であった場合（図 1 の⑤），取得したリファレンスは有効であると判断をする．レスポンスの HTTP ステータスコードの 400 番台のエラーの割合が一定以上なら，リファレンスの有効性が確認できず，本提案方式による脆弱性診断は実施不可能と判断する．そのため，有効な WebAPI のリファレンスが取得できていることを前提に本提案手法を適用する．

5.2.2　No.1 Broken Object Level Authorization の検出

No.1 の検出は，次の 3 つの検出方法によって行う．1 つ目は，JSON データのキーの名称（以降，候補キー）を利用した，リファレンスに記述されていないエンドポイントを検出する手法（図 4）である．2 つ目は，エンドポイントの値に対して，リクエストとそのレスポンスを利用した情報を基に置換や結合を行い，リファレンスに記述のないエンドポイントを検出する手法（図 6）である．3 つ目は，パラメータの値を，リファレンスに定義されていたパラメータの型やリクエストとそのレスポンスを利用した情報を基に置換や連結を行い，リファレンスに記述のないエンドポイントを検出する手法である（図 7）．

具体的な検出方法を次に示す．

（1） パス結合によるエンドポイントの検出

リファレンスの有効性確認のレスポンス（図 1 の④）に含まれている候補キーを利用し，リファレンスに記述がないエンドポイントの検出を行う．この検出処理のフローチャートを図 4 に示す．候補キーを，有効性確認のレスポンスから全て抽出し（図 4 の①），診断対象とする WebAPI のエンドポイントの共通箇所（以降，ベースパス）の後に

入力し，新たなエンドポイントを構成する（図 4 の②）．この例として，図 3 に示すように，ベースパスとして/wp/v2 が与えられ，有効性確認で抽出した候補キーとして “id”, “title”, “status” が与えられた場合，新たに “/wp/v2/id”, “/wp/v2/title”, “/wp/v2/status” の 3 つのエンドポイントを構成する．

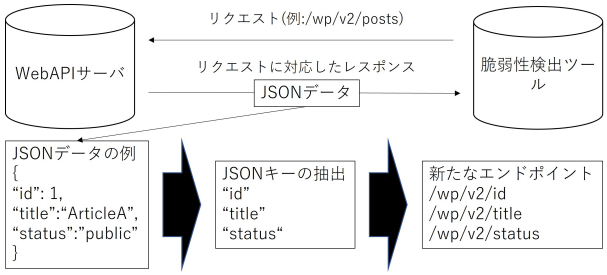

図 3 エンドポイントの再構成
Fig. 3 Reconfigure endpoints

この手法によりエンドポイントを構成する理由は，CMS の WebAPI であれば，CMS 固有の特徴（例えば，“title”, “comment”, “category” など）がレスポンスのパラメータに含まれるために，候補キーが有効なエンドポイントである可能性が高いと考えられるためである．例えば Wordpress の場合，Wordpress のリファレンスの有効性確認で取得した候補キー 87 種類のうち，8 種類がリファレンスに記述されているエンドポイントと同一になる．

構成したエンドポイントに対して GET リクエストを送り，レスポンスの HTTP ステータスコードを確認する（図 4 の③）．HTTP ステータスコードが，200 や 401 等の存在していることが確認できる HTTP ステータスコードであった場合，リファレンスに記述されているかを確認する（図 4 の④）．例えば，“/wp/v2/status” でリクエストを送信し，HTTP ステータスコードが 200 の場合，リファレンスに記述されているエンドポイントかを確認する．リファレンスに記述されていた場合，正規のエンドポイントと判定し，脆弱性が含まれていないと判定する．HTTP ステータスコードが 200 等であり，リファレンスにも記述されていないエンドポイントであった場合，正規のエンドポイントではないと判定し，No.1 の脆弱性が含まれていると判定する．

（2） 変数の置換・変更によるエンドポイントの検出

エンドポイントを変更することによって，範囲外の有効なリクエストや，リファレンスに記述されていないエンドポイントを探索する．例えば Wordpress の場合，特定の記事の情報を取得するエンドポイントとして，“/wp/v2/posts/{id}” が存在する．エンドポイント内に任意の値を取るものを，有効性確認で抽出した候補キー（図 1 の③及び④）から抽出する（図 6 の①）．

エンドポイント内の {id} を変更することによって，範囲外の有効なリクエストやリファレンスに記述されていないエンドポイントを検出する．

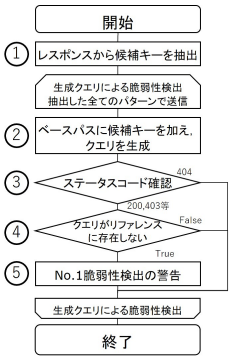

図 4 リファレンスに記述がないエンドポイント検出のフローチャート
Fig. 4 Flowchart of endpoint detection not described in the reference

{ “id”: 38,
“title”: “ArticleName” },
{ “id”: 1,
“title”: “Hello world!” }

図 5 /wp/v2/posts へのリクエストに対するレスポンスボディ (抜粋)
Fig. 5 Response to requests for /wp/v2/posts

INT 型の任意の値を受け取るエンドポイントがある場合，レスポンスを利用して，{id} を使用されている境界の値に変更し，範囲外の有効なリクエストやリファレンスに記述されていないエンドポイントを検出する．検出は以下の処理によって行う．

(1) 正規のエンドポイントの検出

境界の値を取得するため，“/wp/v2/posts/{id}” に対する，一階層上のリクエストに対するエンドポイント（“/wp/v2/posts”）がリファレンスに記述されているか確認する（図 6 の②）．“/wp/v2/posts” のエンドポイントがリファレンスに存在した場合，該当エンドポイントへのリクエストに対するレスポンスを確認し，{id} に使用されているレスポンスボディ内の JSON データの値をすべて取得する．

図 5 に/wp/v2/posts へのリクエストに対するレスポンスの例を示す．図 5 では，“id” として，38 と 1 の値が使用されていることが分かる．

(2) レスポンスによるリクエスト生成と脆弱性の検出

エンドポイント内に INT 型の任意の値，または文字列を取るものについて，一階層上のエンドポイントへのリクエストに対するレスポンスから，使用されている値を取得し，値の境界を用いた新たなクエリを生成する（図 6 の④及び⑤）．上記の例では，id として 38, 1 が使用されているため，最大値（38）の境界値を 39（最大値（38）+1），最小値（1）の境界値を 0（最小値

(1) -1)，平均値を 19.5 と求める．平均値の小数点以下の端数については，偶数への丸めを行う．この場合では，平均値として 20 を使用する．設定した境界値をエンドポイント内の変数に設定した新たなエンドポイントを構築し（パラメータ，認証については含めない），リクエストを送信する．

エンドポイント内に STR 型の任意の文字列を取るものについては候補キーを使用し，エンドポイントを変更する（図 6 の⑥）．設定した候補キーをエンドポイント内の変数に設定した新たなクエリを生成し，リクエストを送信する．

それぞれの手法でリクエストを送信した後，レスポンスから HTTP ステータスコードを確認する（図 6 の⑦）．このとき，HTTP ステータスコードが 200 または 401 であった場合，範囲外の有効なリクエストが存在するとして，No.1 の脆弱性が含まれていると判定する（図 6 の⑧）．

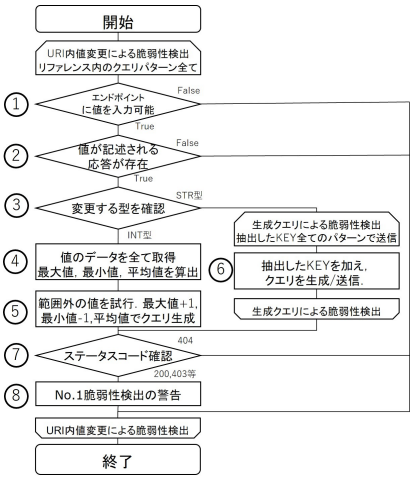

図 6 エンドポイント内の数値・文字列の変更のフローチャート
Fig. 6 Flowchart of changing numbers and strings in an endpoint

(3) パラメータの置換・変更によるエンドポイントの検出

パラメータ（リクエストボディに入力するデータ）を変更することにより，リファレンスに記述されていないエンドポイントを検出する．上述の “レスポンスによるリクエスト生成と脆弱性の検出” と同様に，パラメータ内に任意の値または文字列を取るものを有効性確認で抽出した候補キー（図 1 の③及び④）から抽出する (図 7 の①)．

書き換え処理は，INT 型および STR 型のパラメータで場合分けを行い，実行する（図 7 の②）．

INT 型の数値のパラメータの場合（図 7 の③），数値を文字列型に変更したパラメータ，数値と文字列を結合したパラメータに変更する．本研究では，ここで使用する文字列を “a” に，使用する数値を “1” に設定する．例えば，“id” として数値のパラメータなら，文字列型への変更では “id”:“a” に変更する．数値と文字列型の結合では，“id”:“1a” に変更する．変更する処理を行う前の，基のパラメータに関しては，リファレンスの有効性確認で使用した正規のパラメータを使用する．

STR 型の文字列のパラメータの場合 (図 7 の④)，文字列を数値に変更したパラメータ，文字列と数値を結合したパラメータに変更する．例えば，“status” として文字列のパラメータであった場合，数値への変更では，“status”:“1” に変更する．文字列と数値の結合では，“status”:“publish1” に変更する．このとき，使用するエンドポイントは，リファレンスの有効性確認で使用した正規のエンドポイントを使用する．

送信したリクエスト対するレスポンスの HTTP ステータスコードを確認し（図 7 の⑤），存在していることがわかる HTTP ステータスコード（200 または 401）であった場合，範囲外の有効なリクエストが存在するとして，No.1 の脆弱性が含まれていると判定する（図 7 の⑥）．

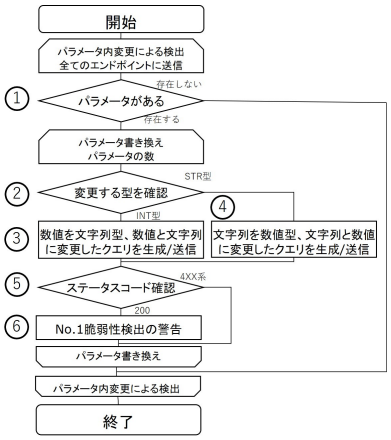

図 7 パラメータ内の数値・文字列の変更のフローチャート
Fig. 7 Flowchart for changing numbers and strings in parameters

5.2.3 No.5 Broken Function Level Authorization の検出

No.5 の検出のフローチャートを図 8 に示す．

(1) WebAPI リファレンスから，認証が必要となるエンドポイントについて抽出を行う（図 8 の①）．抽出した認証が必要となるエンドポイントに対して，有効性確認で使用した正規のエンドポイントとパラメータを用いてリクエストを送信する．このとき，認証情報は含めない（図 8 の②）．

(2) レスポンスの HTTP ステータスコードを確認し，HTTP ステータスコードが 400 や 401 のように，パラメータの形式が間違っていることや認証が必要となることなどを表すレスポンスが返ってきた場合，No.5 の脆弱性が含まれていないと判定する（図 8 の③）．

次に，認証が必要となるエンドポイントのうち，パラメータを設定する必要があるエンドポイントか確認を行う（図 8 の⑤）．パラメータを設定する必要がある場合，パラメータに関して変更処理を行う．

パラメータの変更処理（図 8 の⑥から⑦）については，5.2.2 節（No.1 の脆弱性検出）で述べた “（3）パラメータの置換・変更によるエンドポイントの検出” と同様である．パラメータの変更処理の後，認証情報を含めないリクエストを送信する．送信したリクエスト対する HTTP ステータスコードを確認し，正常に処理が行われたことを示す 200 がレスポンスの HTTP ステータスコードに記述されていた場合（図 8 の⑧），変更処理を行ったパラメータの条件でリクエストを処理できるため，No.1 の脆弱性が含まれていると判定する（図 8 の⑨）．

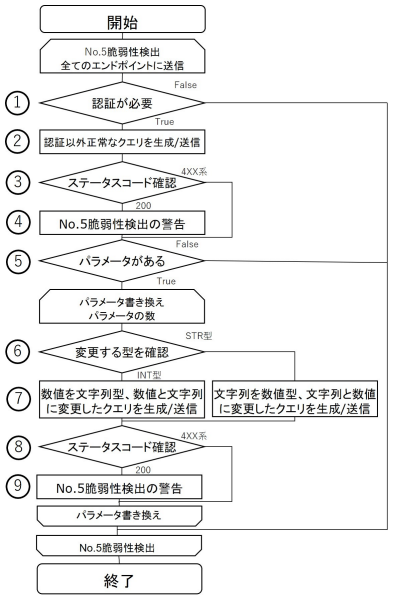

図 8 No.5 の検出のフローチャート
Fig. 8 Flowchart of No.5 detection

6. 評価実験

6.1 実験環境

評価実験では，Wordpress の WebAPI を利用し脆弱性の検出を行う．Wordpress のバージョン 4.7.0 または 4.7.1 には，OWASP API Secyrity Top 10 の No.5 に該当する，認証を無視した記事改ざんの脆弱性が存在する．本実験では，Wordpress4.7.0 を構築し，Wordpress の WebAPI を利用した記事改ざんの脆弱性に関する検出を行う．

6.2 Wordpress 記事改ざんの脆弱性 [2] の概要

（1） 記事一覧を取得する WebAPI のエンドポイント “/wp/v2/posts” に GET リクエストを送る．図 9 にレスポンスの一部を示す．図 9 の “id” の値が，Wordpress で公開されている記事の一覧である．図 9 のレスポンスでは

```
{ “id”: 38,
 “title”: “ArticleName” },
{ “id”: 15,
 “title”: “ArticleNameB” },
{ “id”: 1,
 “title”: “Hello world!” }
```

図 9 /wp/v2/posts のレスポンスボディ (一部のみ)
Fig. 9 Part of Responce of /wp/v2/posts

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

{“id”:“38a”, “title”:“Title Changed”}
```

図 10 改ざんのためのリクエストの一部
Fig. 10 Example query for tampering

```
{ “id”: 38,
“title”: “Title Changed”}
```

図 11 改ざん成功時のレスポンスボディ (一部のみ抜粋)
Fig. 11 Example Responce for tampering

表 3 リファレンスより抽出した初めの 4 件
Table 3 First four cases extracted from the reference.

メソッド	エンドポイント	パラメータと型 (抜粋)	認証
GET	/posts	context(str), author(str)	不要
POST	/posts	title(str), content(str)	必要
GET	/posts/{id}	id(int), context(str)	不要
POST	/posts/{id}	id(int), title(str)	必要

ID が 38，15，1 の記事が存在することがわかる．

（2） エンドポイント “/wp/v2/posts/{id}” には認証を無視する脆弱性があるため，リクエストにパラメータとして，図 10 のようなパラメータのリクエストを送信する．図 10 では，記事の改ざんは，ID が 38 の記事に対して行われる例である．

（3） （2）のリクエストに対して正常に処理され，記事の改ざんが行われる．図 11 に改ざん成功時のレスポンスの一部を示す．

6.3 提案方式の脆弱性診断の適用結果と考察

実験環境に対して提案方式による脆弱性診断を行った．まず，リファレンスの取得結果と，有効性確認について述べる（図 1 の①から③）．Wordpress のリファレンスは SwaggerHub から取得した．（図 1 の①から③）

次にリファレンスより，脆弱性診断に用いる情報を抽出する (図 1 の④)．抽出した結果の初めの 4 件を表 3 に示す．

抽出したエンドポイントに対し，有効性の確認処理を行う（図 1 の⑤）．表 3 より取得したエンドポイントに対しリクエストを送信，レスポンスの HTTP ステータスコードよりリファレンスが有効であることを確認した．

表 4　/wp/v2/posts/{id} の提案方式における No.5 検出の実行結果
Table 4　Execution results of the No.5 detection in the proposed method in /wp/v2/posts/{id}

“id” の値	“title” の文字列	HTTP ステータスコード	備考
38	Title Changed	—	①
a	Title Changed	401	②
38a	Title Changed	200	③
38	1	401	④
38	Title Changed1	401	⑤

有効性確認の終了後，No.1，No.5 の検出方式を実施した．No.1 については，Wordpress 4.7.0 の環境には No.1 の脆弱性が存在しないため，提案方式では検出されなかった．

次に，No.5 の検出 (図 8) と，その結果について述べる．No.5 では，表 4 のリストを読み込み，リストの上から順に処理を行う．表 4 に，表 3 の 4 行目のエンドポイント “/wp/v2/posts/{id}” で使用されるパラメータの No.5 検出処理による変化パターンとレスポンスの HTTP ステータスコードを示す．“/wp/v2/posts/{id}” では，“id”，“title”，“content” などのパラメータを使用するが，実験では簡略化のため “id”，“title” のみを使用する．“id” の値は一階層上のエンドポイント “/wp/v2/post” のレスポンスから，最初に出現した “id” の値である “38” を使用する．

“title” は “/wp/v2/posts” のレスポンスから取得可能だが，改ざんの成功をタイトル名の変更から明確にするため，“title” に使用する正規のパラメータとして，最初に出現した “title” の文字列ではなく，“Title Changed” の文字列を設定する．正規パラメータを表 4 の①に示す．

No.5 の検出での書き換えは初めに，1 番目のパラメータの “id’ に対して行う．“id”は，INT 型なので，STR 型の文字列に変更する．変更した結果，“id”は “a”となる (表 4 の②)．表 4 の②のパラメータを使用したレスポンスの HTTP ステータスコードは 401 であったため，この条件では No.5 の脆弱性は存在しないと判定する．

次に，“id”の数値を，文字列型と結合する．結合した結果，“id”は “38a”となる (表 4 の③)．書き換えたパラメータによりリクエストを送信し，そのレスポンスから，正常にリクエストが処理されたことを表す HTTP ステータスコード 200 が返ってきた．改ざん成功時のレスポンスボディの一部を図 11 に示す．図 9 では，“id”が 38 の “title” は，“ArticleName”だが，図 11 では，“id”が 38 の “title” が，設定した ‘Title Changed” に変化している．これにより，“/wp/v2/posts/{id}” は，表 4 の③のパラメータにおいて，No.5 の脆弱性があると判定する．

“id” に関して，書き換えのパターンを全て完了したため，次に，“title” について書き換え処理を行う．“id” は，表 4 の①に示された，正規のパラメータパターンである “38” に戻す．

文字列型である “title” について，最初に INT 型の数値にパラメータを書き換える．書き換えた結果，“title”は “1” となる (表 4 の④)．このパラメータパターンにおける，レスポンスの HTTP ステータスコードは 401 であったため，この条件では No.5 の脆弱性はないと判定する．

次に，“title”の文字列型を INT 型の数値と結合する書き換え処理を行う．書き換え処理をした結果，“title”は “Title Changed1”となる (表 4 の⑤)．パラメータパターンにおける，レスポンスの HTTP ステータスコードは 401 であったため，この条件では No.5 の脆弱性はないと判定する．本提案方式によって，既存の 3 つの脆弱性診断ツールで脆弱性診断が困難となっていた，（OWASP API Secyrity Top 10 の No.5 に該当する）Wordpress の WebAPI を利用した記事改ざんの脆弱性の検出が可能であることが確認できた．

7. おわりに

本論文では OWASP API Secyrity Top 10 の 2 つの脆弱性項目に対し，リクエストとそのレスポンスを繰り返し解析する診断手法について提案した．評価実験では，Wordpress の WebAPI を利用した記事改ざんの脆弱性について，提案方式により検出可能であることを示した．今後は，OWASP API Secyrity Top 10 の No.1 と No.5 の項目以外の脆弱性診断を可能にする手法の検討を行う予定である．また，実際に運用されている Wordpress を用いた評価実験を実施する必要がある．

参考文献

[1] Wordpress, “WP-API,” https://developer.wordpress.org/rest-api/, 2021/8/21.

[2] Marc-Alexandre Montpas, “Content Injection Vulnerability in WordPress,” https://blog.sucuri.net/2017/02/content-injection-vulnerability-wordpress-rest-api.html, 2021/8/21.

[3] OWASP, “OWASP API Security TOP 10 2019,” https://raw.githubusercontent.com/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] 高松, 勇輔, 小菅, 祐史, 河野, 健二, “Web アプリケーションの開発時におけるセッション管理の脆弱性の自動化,” 情報処理学会 コンピュータセキュリティシンポジウム (CSS 2011), pp.113 - 118, Oct. 2011.

[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.