4.16 Web APIにまつわる問題

JSON（JavaScript Object Notation）とは

JSONはJavaScriptから生まれたデータ記述フォーマットで，真偽値，数値，文字列，null値の組み合わせを持ったハッシュか配列かその両方の、ウェブ関連ではポピュラーなデータフォーマットです。
JavaScriptとの違いを挙げると、JavaScriptでは文字列リテラルを囲うのはシングルクォートでもダブルクォートでも構いませんが，JSONではダブルクォートのみです。
また，JSONではプロパティ名にもダブルクォートが必須です。JavaScriptにはNaNやundefinedなどの値がありますが，JSONには存在しません。
JSONはJavaScriptの曖昧さを排除して、データ記述言語としての精度と，解析のし易さを確保しています。
JavaScriptにネイティブなJSONサポート（JSON.parse，JSON.stringify）が定義されました。特にJSON.stringifyメソッドを使用すれば手軽にJavaScriptのオブジェクトからJSONを出力することができます。

サーバ側のJSON ⇔ オブジェクト（PHP）	
json_encode	PHP配列・オブジェクトからJSON文字列を生成
json_decode	JSON文字列をPHP配列に変換

JavaScript側のJSON ⇔ オブジェクト	
JSON.stringify	JavaScriptのオブジェクトからJSON文字列を生成
JSON.parse	JSON文字列をJavaScriptのオブジェクトに変換

JSONP（JSON with Padding）とは

Paddingは（本来は不要なものの）付け足しという意味です。（JSONPはJSONではないので，JSONPはJSONとしてパースできません）
JSONPはクロスドメインの制限を超えてデータをやり取りするために使用されます。
CORSができる前に異なるオリジンのサーバからデータを取得する方法の１つがJSONPでした。
JSONPはXMLHttpRequestではなく、script要素を用いて外部のJavaScriptを直接実行することにより、データを取得します。
JSON文字列をそのままscript要素で受け取ることができないので、関数呼び出しの形でデータを生成します。

JSONエスケープ不備による脆弱性

Web API においてJSON文字列生成時のエスケープ処理に不備があると、意図しないJavaScriptがJSONデータに混入する場合があります
JSONデコードにeval関数を使っている場合や、JSONPのようにscript要素でJSON文字列を呼び込んでいる場合には、不正なJavaScriptの実行に至ります

JSONエスケープ不備・脆弱性の影響

Webサイト利用者のブラウザ上でのJavaScriptの実行

JSONエスケープ不備・脆弱性の対策

①文字列連結によるJSONデータ生成をやめ、信頼できるライブラリを用いて、JSONを生成する
②eval関数ではなく、JSON．Parseなどの安全なAPIでJSONを解釈する

JSON直接閲覧によるXSS脆弱性

JSONを返すWeb APIは、通常XMLHttpRequestによるアクセスを想定したものですが、APIが返すレスポンスデータをブラウザで直接閲覧させることで、攻撃が可能になることがあります

JSON直接閲覧によるXSS脆弱性の影響

Webサイト利用者のブラウザ上でのJavaScriptの実行

JSON直接閲覧によるXSS脆弱性の対策

①MIMEタイプを正しく設定する
②レスポンスヘッダ X-Content-Type-Options:nosniff を出力する
③＜、＞ をUnicodeエスケープする

PHPの場合、json encodeのオプションパラメータで、エスケープが可能

オプション	エスケープ対象文字	エスケープ結果
JSON_HEX_TAG	＜ ＞	＼U003C ＼U003E
JSON_HEX_AMP	&	＼U0026
JSON_HEX_APOS	'	＼U0029
JSON_HEX_QUOT	"	＼U0022

④XMLHttpRequestなどCORS対応の機能だけから呼び出せるようにする
XMLHttpRequestからのリクエストには、「X-Request-With: XMLHttpRequest」が付与され、
script要素からのアクセスには付かないので、その違いによって、不正なアクセスをチェックできます

JSONPのコールバック関数名によるXSS脆弱性

JSONPはJSONデータを引数とするコールバック関数呼び出しの形式を取るが、このコールバック関数名を外部から指定できるように構築するケースが多い
これにより、コールバック関数名によるクロスサイトスクリプティングが可能になる場合があります

JSONPのコールバック関数名によるXSS脆弱性の影響

Webサイト利用者のブラウザ上でのJavaScriptの実行

JSONPのコールバック関数名によるXSS脆弱性の対策

①コールバック関数名の文字種と文字数を制限する　※　英数字とアンダースコア「＿」のみに制限しても問題ないと思われる。＜＞を使えなくする。
②MIMEタイプを正しく設定する　※　JSONPはJSONではなく、JavaScriptなので、text/javascriptですが、MIMEタイプを指定しないと、text/htmlと解釈され、script要素がJavaScriptとして解釈される
※　text/javascriptであれば、script要素は解釈されず、実行されることはありません

Web APIのクロスサイトフォージェリ脆弱性

Web APIでもCSRF脆弱性は混入します。

Web APIのクロスサイトフォージェリ脆弱性をつくHTTP通信経路について

①GETリクエスト
②HTTPフォームでMIMEタイプが（text/plain、application/x-www-form-urlencoded、multipart/form-data）であることを想定していて、MIMEタイプをチェックしていない場合
③プリフライトリクエストを必要としないシンプルなリクエストで、MIMEタイプをチェックしていない場合
④プリフライトリクエストを必要とするリクエストだが、プリフライトリクエストに対する処理が不適切

Web APIのクロスサイトフォージェリ脆弱性の対策

①CSRFトークン（セッション変数にトークンを保持）
②2重送信クッキー　　問題点：以下①～③が原因で、セッションフィクセイションされる可能性がある　①クッキーモンスターバグ悪用のリスクがある　②サイトにXSS脆弱性がある場合がある　③HTTP強制による漏洩リスクがある
③XMLHttpRequestなどCORS対応の機能だけから呼び出せるようにする
XMLHttpRequestからのリクエストには、「X-Request-With: XMLHttpRequest」が付与され、
script要素からのアクセスには付かないので、その違いによって、不正なアクセスをチェックできます
問題点：Adobe Flash Player に過去、カスタムリクエストヘッダを許可なく送信できる脆弱性があった（2014年7月に修正済み）

JSONハイジャックの脆弱性

罠サイトの工夫によって、JSONデータを罠サイトのJavaScriptで読み出す手法がJSONハイジャックの脆弱性として、考案されてきて、ブラウザ側の対策が取られてきました
今後も新しい手法が見つかることが考えられるので、以下の対策が必要です

JSONハイジャックによるXSS脆弱性の対策

①レスポンスヘッダ X-Content-Type-Options:nosniff を出力する
②XMLHttpRequestなどCORS対応の機能だけから呼び出せるようにする
XMLHttpRequestからのリクエストには、「X-Request-With: XMLHttpRequest」が付与され、
script要素からのアクセスには付かないので、その違いによって、不正なアクセスをチェックできます

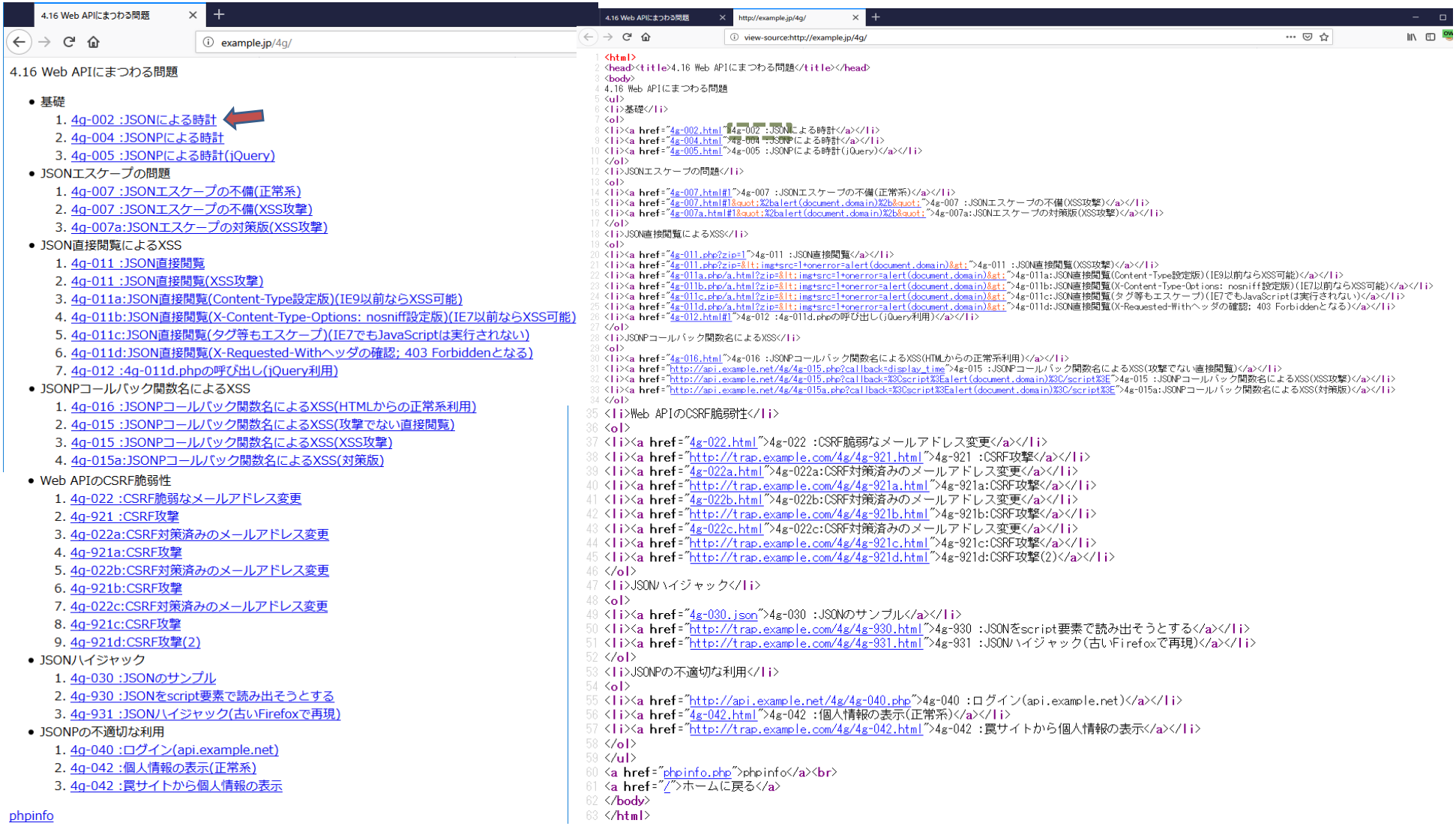
CORSの検証不備

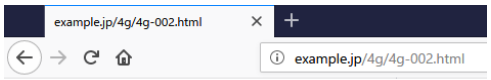
オリジンの制限をかけないのは情報漏洩のリスクになる	NG	Access-Control-Allow-Origin: *
オリジンを指定する	OK	Access-Control-Allow-Origin: example.jp

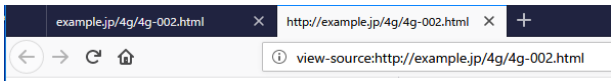
CORS規定では、XMLHttpRequestリクエストのWithCredentialsプロパティを指定した場合、Access-Control-Allow-Originヘッダにオリジンを指定しなければならないが、手抜きが多い

・基礎
4g-002 :JSONによる時計

【ブラウザ】








```
<body>
<p id="time"></p>
<script>
function disptime(json) {
  var obj = JSON.parse(json);
  var txt = document.createTextNode("時刻は" + obj.time + "です");
  var p = document.getElementById("time");
  p.appendChild(txt);
}

  var req = new XMLHttpRequest();
  req.open("GET", "4g-001.php");
  req.onreadystatechange = function() {
    if (req.readyState == 4 && req.status == 200) {
      disptime(req.responseText);
    }
  };
  req.send(null);
</script>
<body>
```

【サーバ： 4g/4g-002.html 】

/var/www/html/4g/4g-002.html - wasbook@example.jp - エディタ - WinSCP

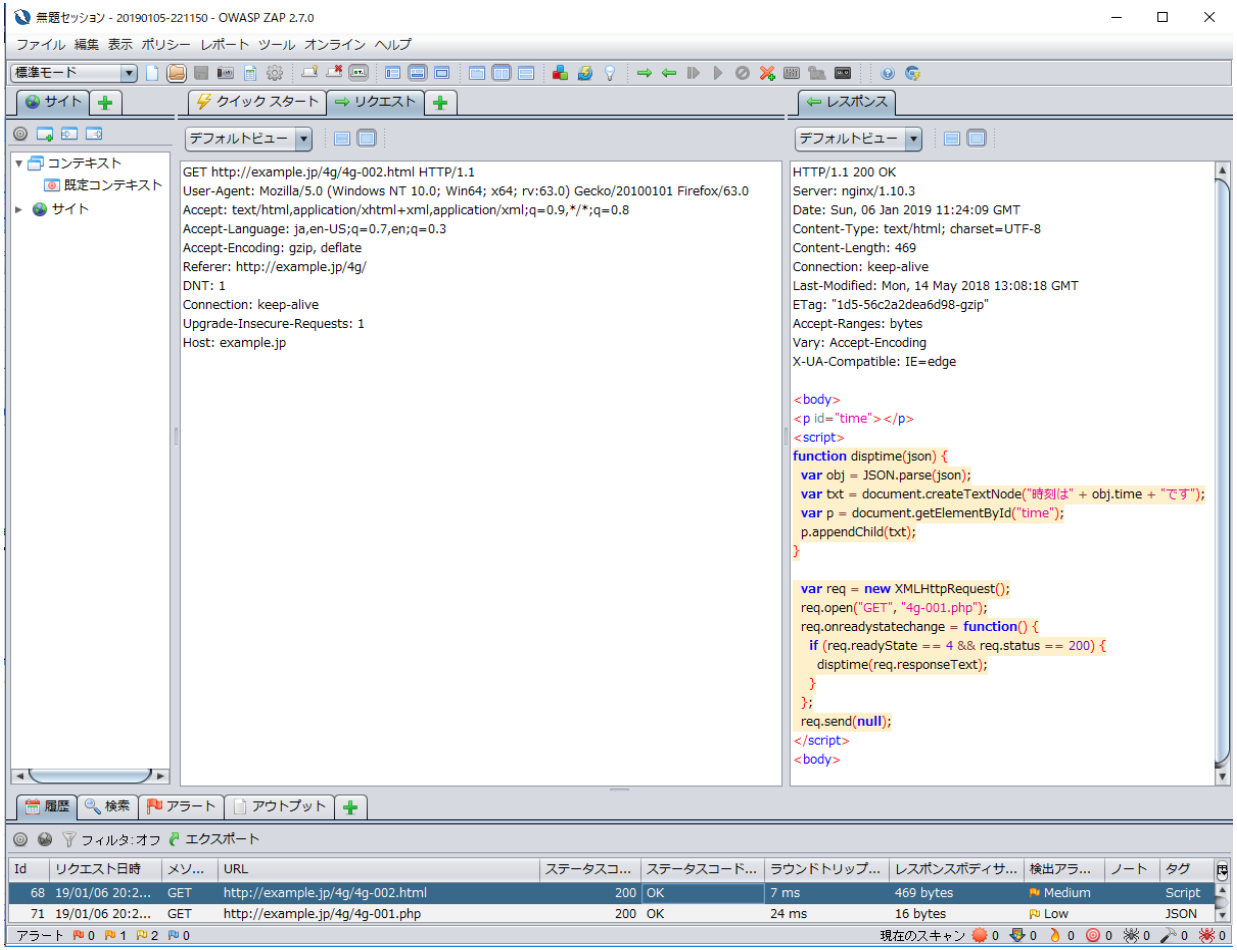
```
<body>
<p id="time"></p>
<script>
function disptime(json) {
  var obj = JSON.parse(json);
  var txt = document.createTextNode("時刻は" + obj.time + "です");
  var p = document.getElementById("time");
  p.appendChild(txt);
}

  var req = new XMLHttpRequest();
  req.open("GET", "4g-001.php");
  req.onreadystatechange = function() {
    if (req.readyState == 4 && req.status == 200) {
      disptime(req.responseText);
    }
  };
  req.send(null);
</script>
<body>
```

【サーバ： 4g/4g-001.php 】

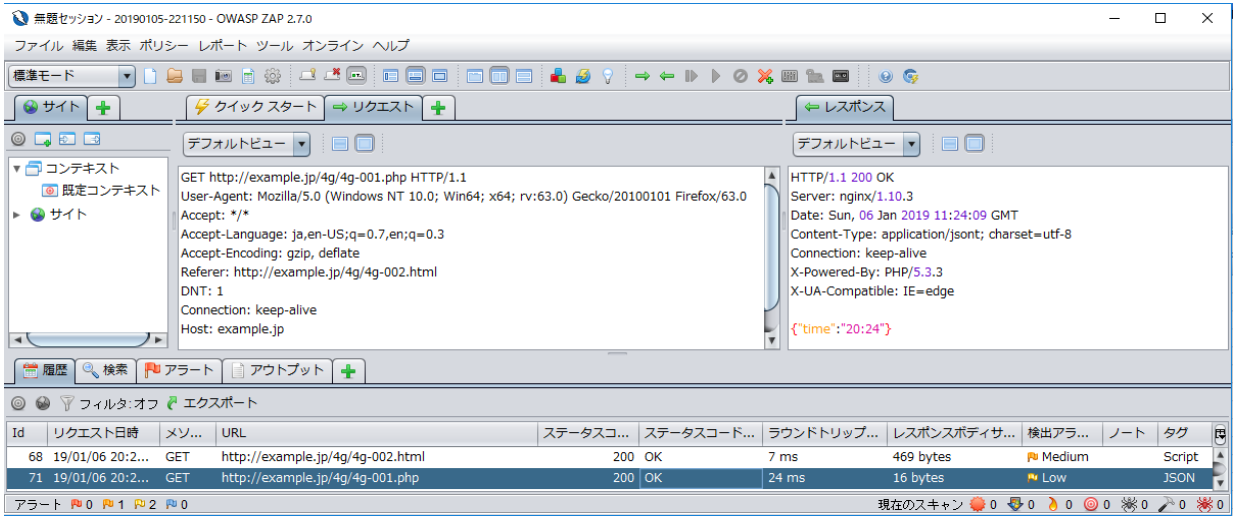
/var/www/html/4g/4g-001.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  header('Content-Type: application/jsont; charset=utf-8');
  echo json_encode(array('time' => date('G:i')));
```

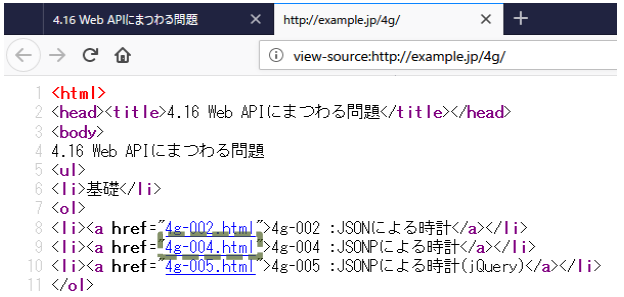
【ブラウザ→サーバ： リクエスト 4g/4g-002.html → レスポンス 】

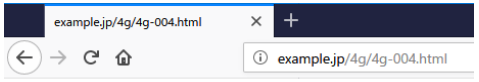


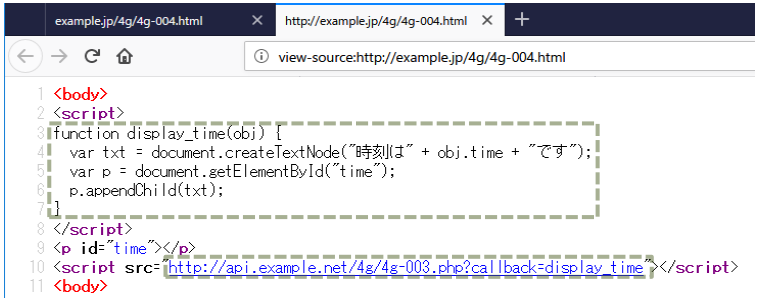
【ブラウザ→サーバ： リクエスト 4g/4g-001.php → レスポンス 】



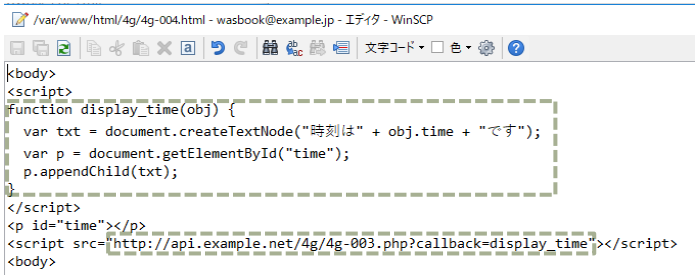
4g-004 :JSONPによる時計



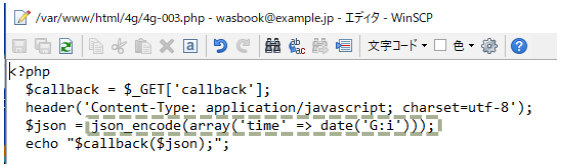




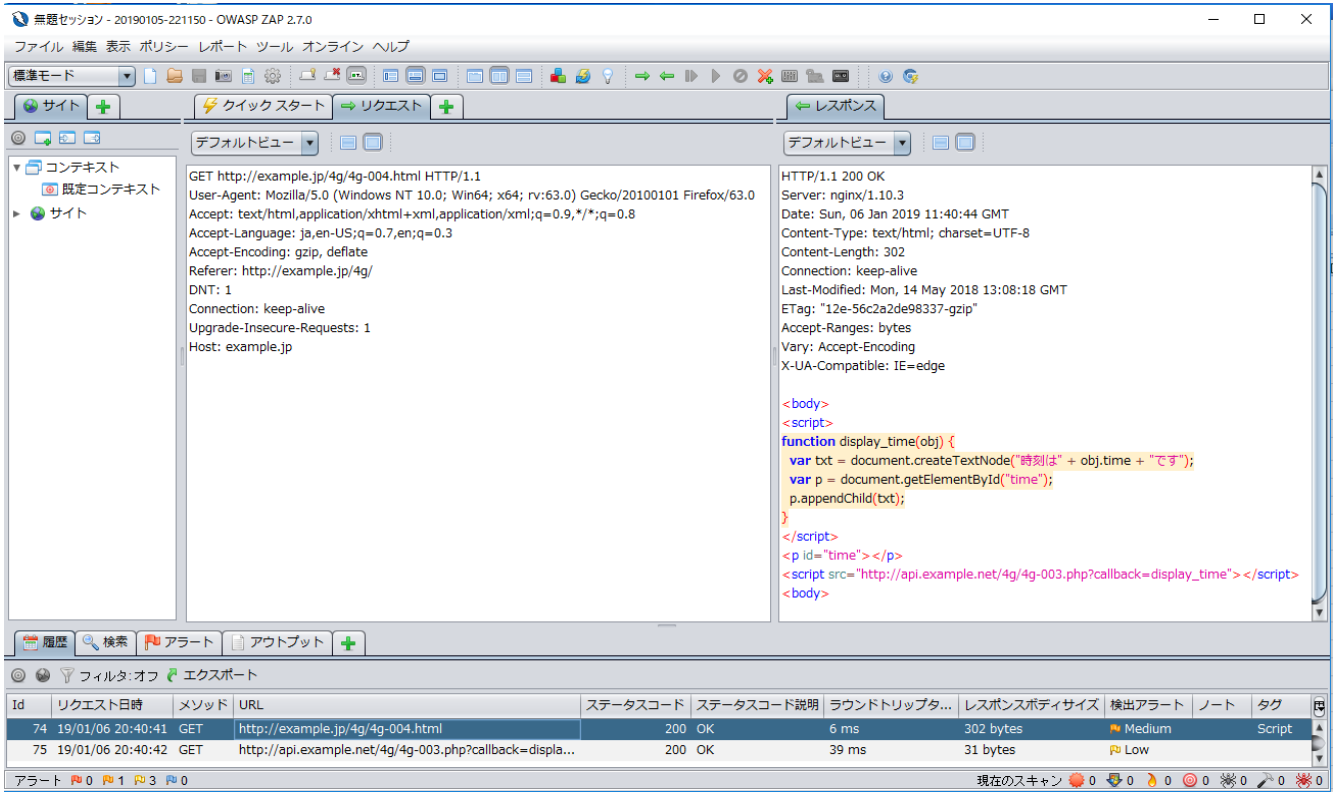
【サーバ： 4g/4g-004.html 】



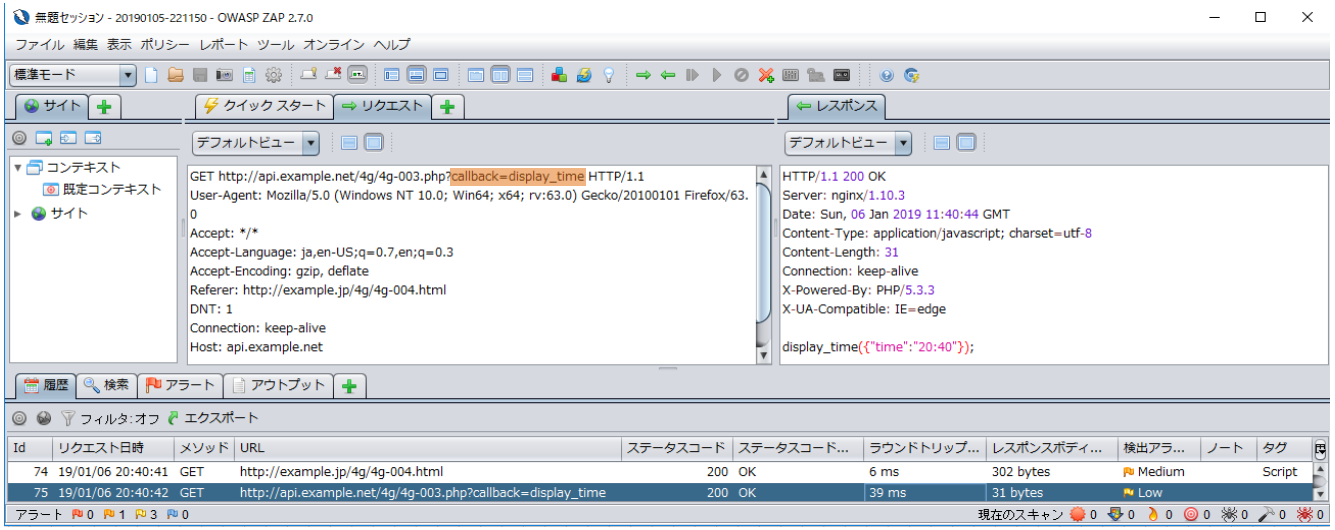
【サーバ： 4g/4g-003.php 】



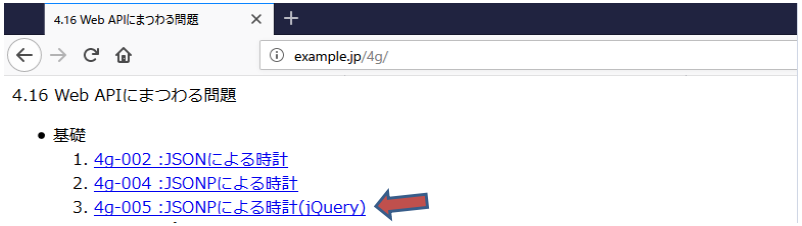
【ブラウザ→サーバ：リクエスト 4g/4g-004.html → レスポンス 】

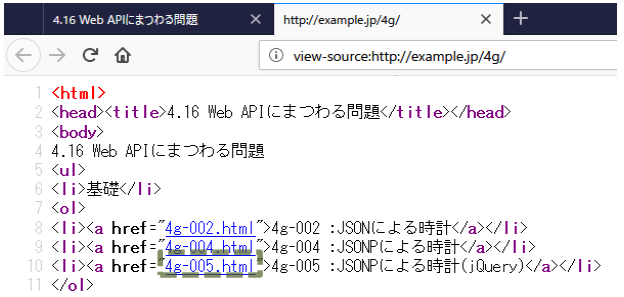


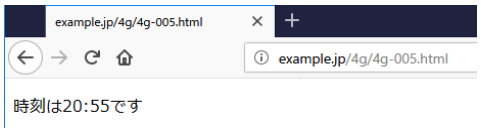
【ブラウザ→APIサーバ： リクエスト 4g/4g-003.php → レスポンス 】



4g-005 :JSONPによる時計(jQuery)







【サーバ： 4g/4g-005.html 】

/var/www/html/4g/4g-005.html - wasbook@example.jp - エディタ - WinSCP

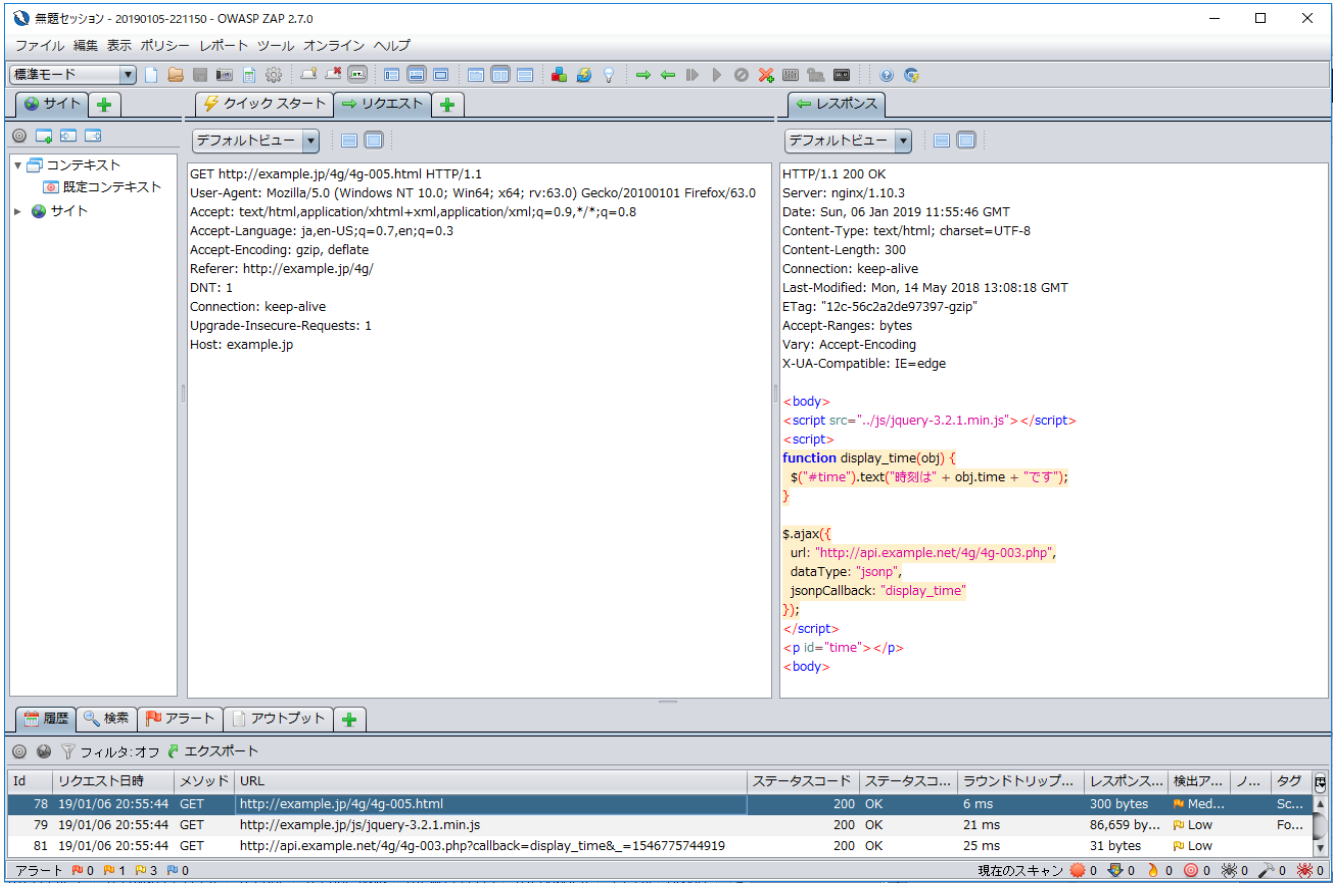
```
<body>
<script src="../js/jquery-3.2.1.min.js"></script>
<script>
function display_time(obj) {
  $("#time").text("時刻は" + obj.time + "です");
}
$.ajax({
  url: "http://api.example.net/4g/4g-003.php",
  dataType: "jsonp",
  jsonpCallback: "display_time"
});
</script>
<p id="time"></p>
<body>
```

【サーバ： 4g/4g-003.php 】

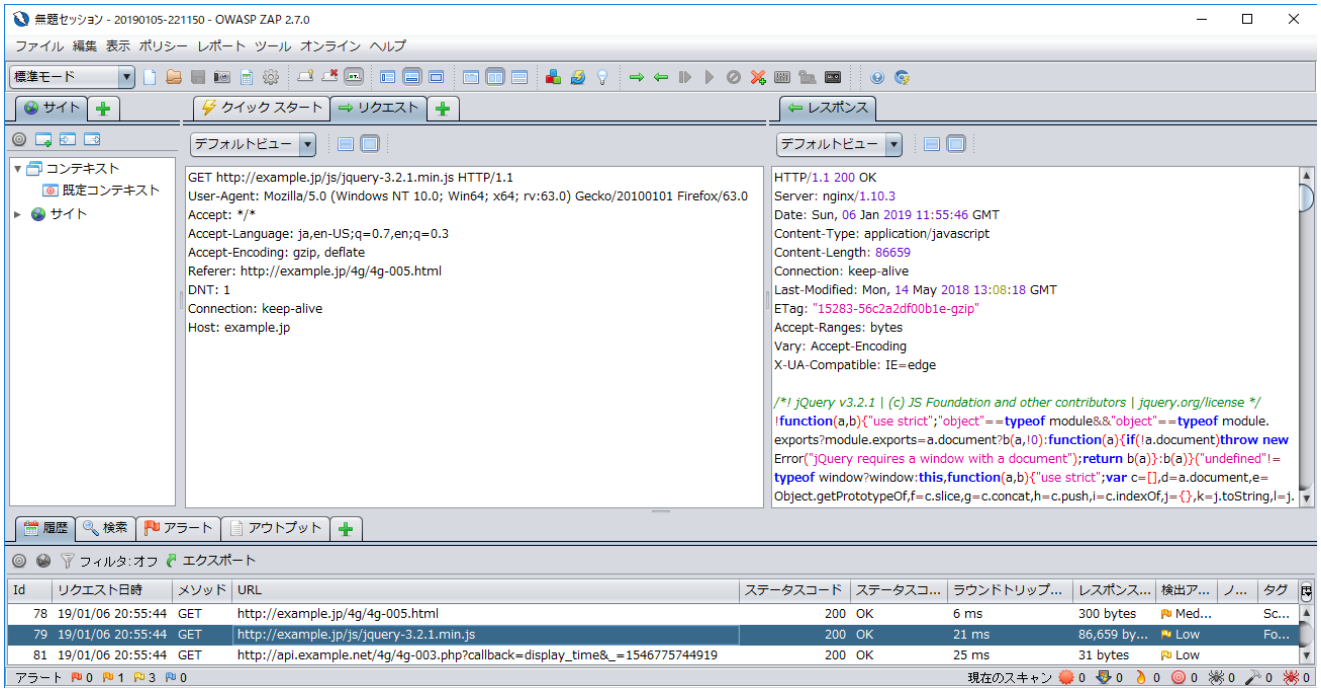
/var/www/html/4g/4g-003.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  $callback = $_GET['callback'];
  header('Content-Type: application/javascript; charset=utf-8');
  $json = json_encode(array('time' => date('G:i')));
  echo "$callback($json);";
```

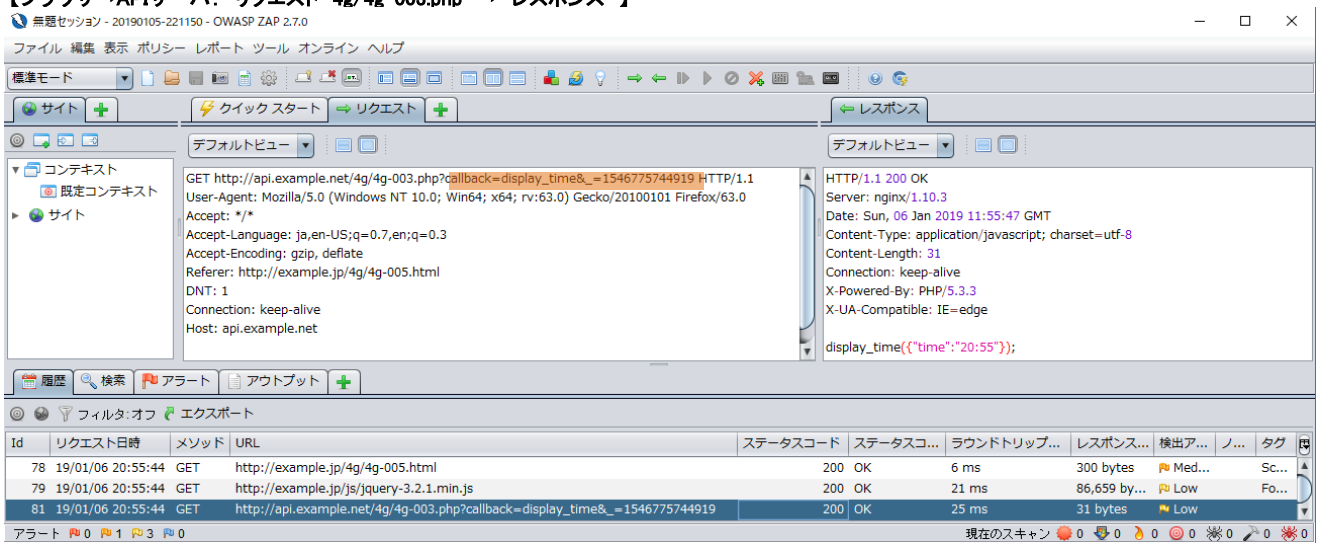
【ブラウザ→サーバ： リクエスト 4g/4g-005.html → レスポンス 】



【ブラウザ→サーバ： リクエスト js/jquery-3.2.1.min.js → レスポンス 】



【ブラウザ→APIサーバ： リクエスト 4g/4g-003.php → レスポンス 】



・JSONエスケープの問題
4g-002 :JSONによる時計

【ブラウザ】

- JSONエスケープの問題
  1. 4g-007 :JSONエスケープの不備(正常系)
  2. 4g-007 :JSONエスケープの不備(XSS攻撃)
  3. 4g-007a:JSONエスケープの対策版(XSS攻撃)

```
12 <li>JSONエスケープの問題</li>
13 <ol>
14 <li><a href="4g-007.html#1">4g-007 :JSONエスケープの不備(正常系)</a></li>
15 <li><a href="4g-007.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007 :JSONエスケープの不備(XSS攻撃)</a></li>
16 <li><a href="4g-007a.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007a:JSONエスケープの対策版(XSS攻撃)</a></li>
17 </ol>
```

example.jp/4g/4g-007.html#1

example.jp/4g/4g-007.html#1

郵便番号が見つかりません:1

example.jp/4g/4g-007.html#1 | http://example.jp/4g/4g-007.html#

view-source:http://example.jp/4g/4g-007.html#1

```
1  <body>
2  <script src="../js/jquery-3.2.1.min.js"></script>
3  <script>
4  $.ajax({
5    url: "http://api.example.net/4g/4g-006.php?zip=" + location.hash.slice(1),
6    dataType: "jsonp",
7    jsonpCallback: "callback_zip"
8  }).done(function(data) {
9    $('#message').text(data.message);
10 });
11 </script>
12 <p id="message"></p>
13 <body>
```

【サーバ: 4g/4g-007.html 】

/var/www/html/4g/4g-007.html - wasbook@example.jp - エディタ - WinSCP

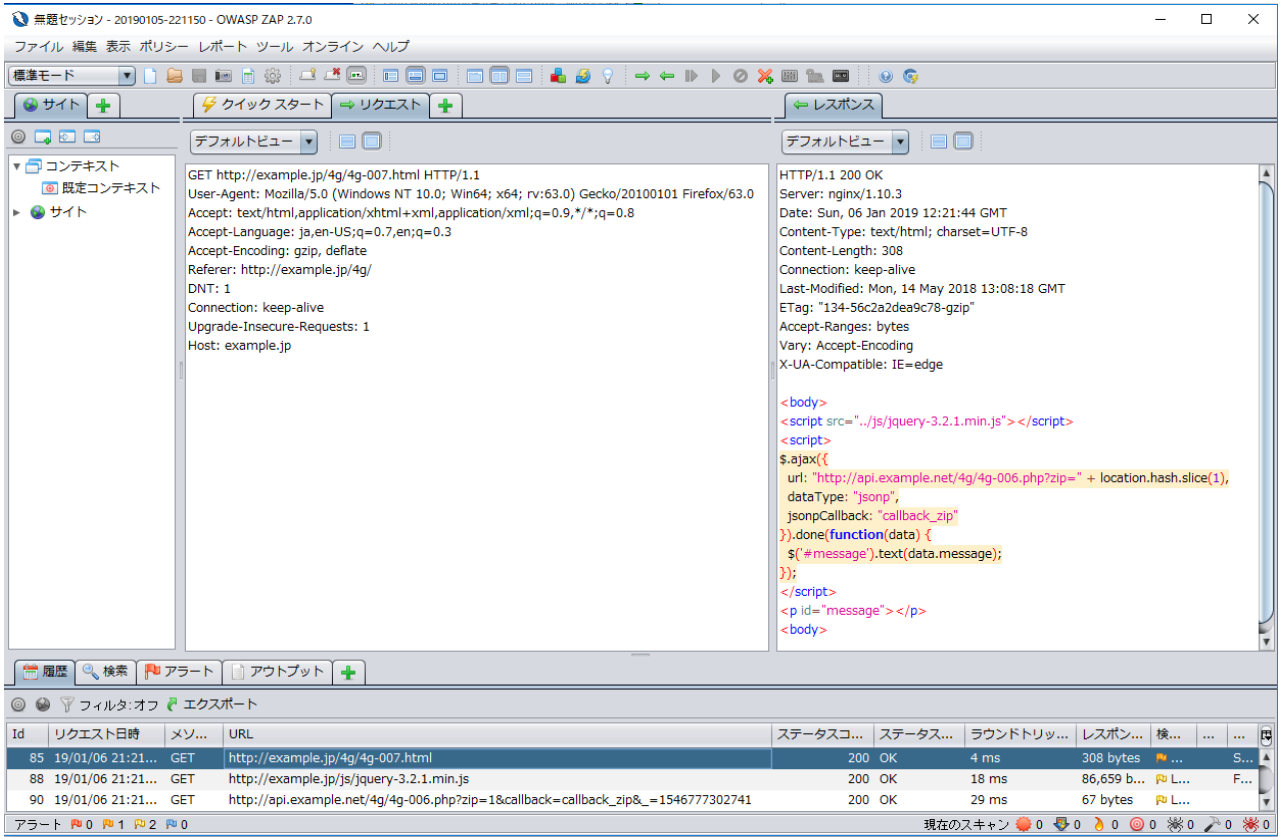
```
<body>
<script src="../js/jquery-3.2.1.min.js"></script>
<script>
$.ajax({
  url: "http://api.example.net/4g/4g-006.php?zip=" + location.hash.slice(1),
  dataType: "jsonp",
  jsonpCallback: "callback_zip"
}).done(function(data) {
  $('#message').text(data.message);
});
</script>
<p id="message"></p>
<body>
```

【サーバ: 4g/4g-006.php 】

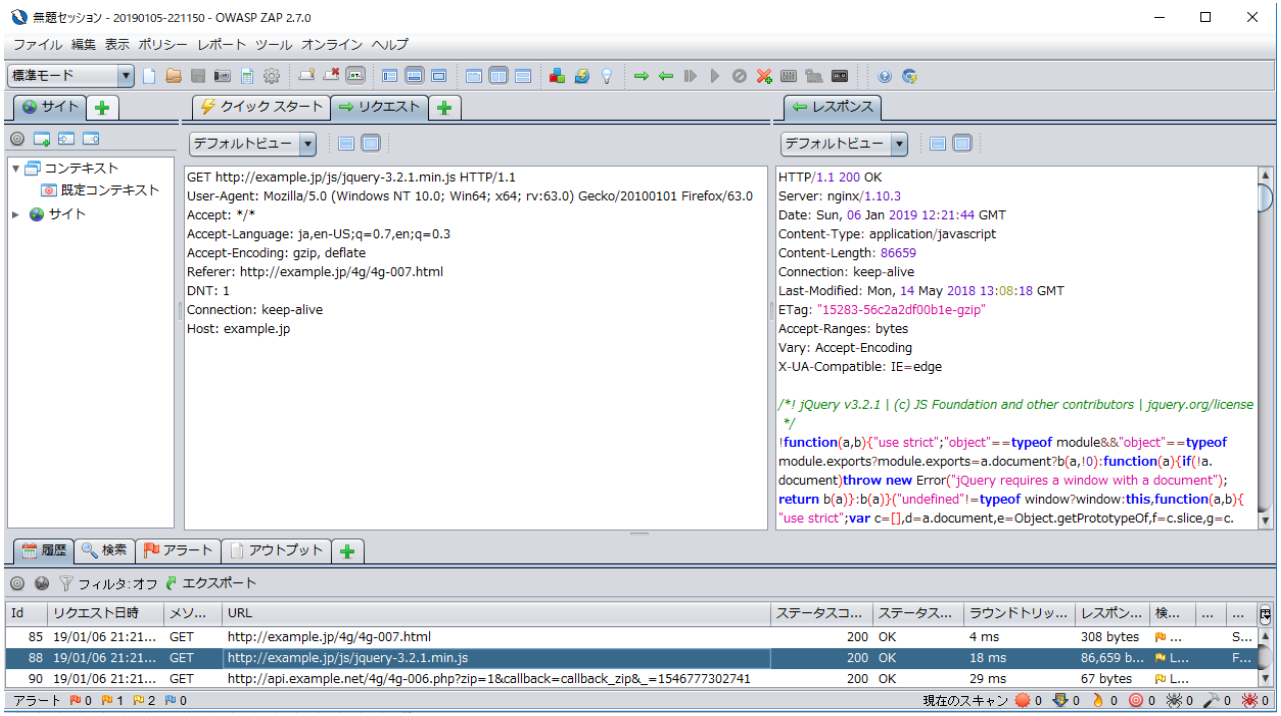
/var/www/html/4g/4g-006.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  $zip  = $_GET['zip'];
  // 以下は郵便番号が見つからなかった場合の処理
  $json = '{"message":"郵便番号が見つかりません:' . $zip . '"}';
  header('Content-Type: text/javascript; charset=utf-8');
  echo "callback_zip($json);";
```

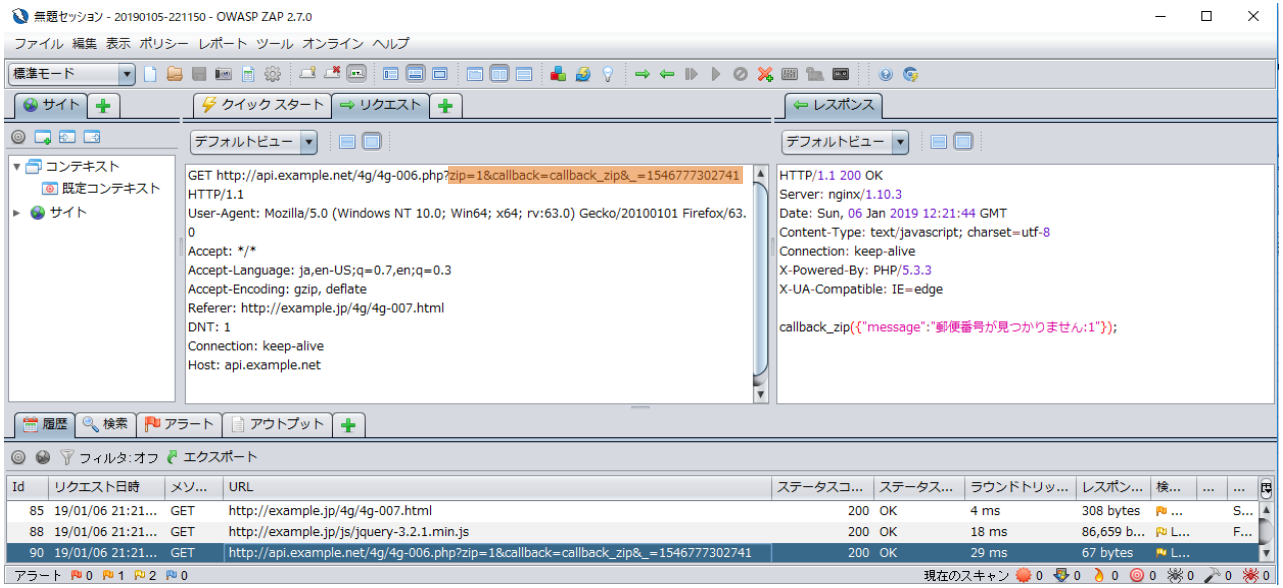
【ブラウザ→サーバ： リクエスト 4g/4g-007.html → レスポンス 】



【ブラウザ→サーバ： リクエスト js/jquery-3.2.1.min.js → レスポンス 】



【ブラウザ→APIサーバ： リクエスト 4g/4g-006.php → レスポンス 】

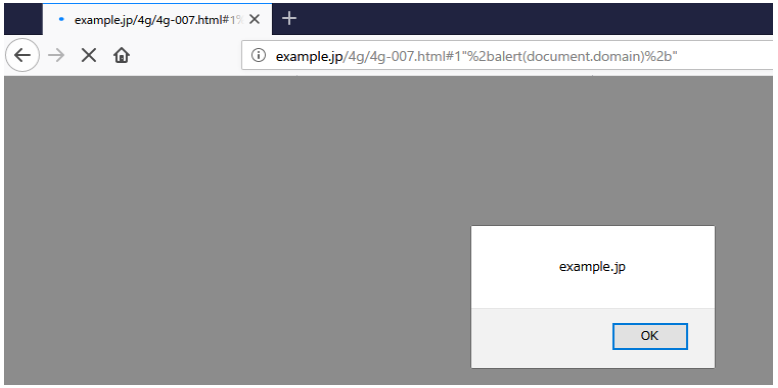


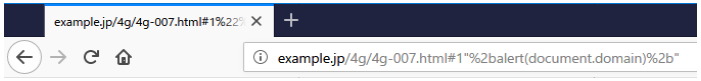
4g-002 :JSONによる時計

【ブラウザ】

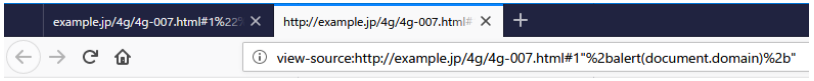
- JSONエスケープの問題
  1. 4g-007 :JSONエスケープの不備(正常系)
  2. 4g-007 :JSONエスケープの不備(XSS攻撃)
  3. 4g-007a:JSONエスケープの対策版(XSS攻撃)

```
12 <li>JSONエスケープの問題</li>
13 <ol>
14 <li><a href="4g-007.html#1">4g-007 :JSONエスケープの不備(正常系)</a></li>
15 <li><a href="4g-007.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007 :JSONエスケープの不備(XSS攻撃)</a></li>
16 <li><a href="4g-007a.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007a:JSONエスケープの対策版(XSS攻撃)</a></li>
17 </ol>
```





郵便番号が見つかりません:1undefined




```
1 <body>
2 <script src="../js/jquery-3.2.1.min.js"></script>
3 <script>
4 $.ajax({
5   url: "http://api.example.net/4g/4g-006.php?zip=" + location.hash.slice(1),
6   dataType: "jsonp",
7   jsonpCallback: "callback_zip"
8 }).done(function(data) {
9   $('#message').text(data.message);
10 });
11 </script>
12 <p id="message"></p>
13 <body>
```

【サーバ： 4g/4g-007.html 】

/var/www/html/4g/4g-007.html - wasbook@example.jp - エディタ - WinSCP

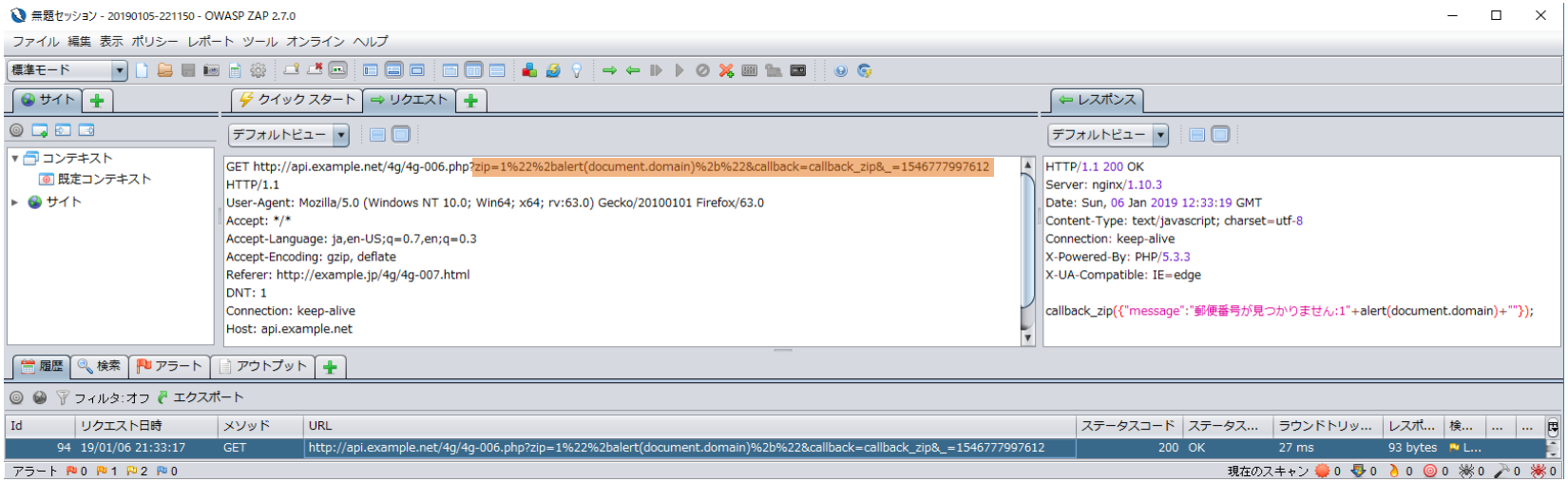
```
<body>
<script src="../js/jquery-3.2.1.min.js"></script>
<script>
$.ajax({
  url: "http://api.example.net/4g/4g-006.php?zip=" + location.hash.slice(1),
  dataType: "jsonp",
  jsonpCallback: "callback_zip"
}).done(function(data) {
  $('#message').text(data.message);
});
</script>
<p id="message"></p>
<body>
```

【サーバ： 4g/4g-006.php 】

/var/www/html/4g/4g-006.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  $zip  = $_GET['zip'];
  // 以下は郵便番号が見つからなかった場合の処理
  $json = '{"message":"郵便番号が見つかりません:' . $zip . '"}';
  header('Content-Type: text/javascript; charset=utf-8');
  echo "callback_zip($json);";
```

【ブラウザ→APIサーバ： リクエスト 4g/4g-006.php → レスポンス 】

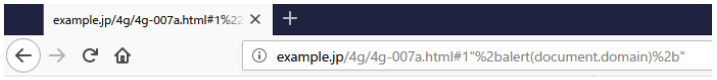


4g-007a:JSONエスケープの対策版(XSS攻撃)

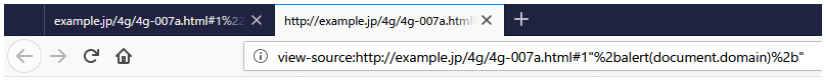
【ブラウザ】

- JSONエスケープの問題
  1. 4g-007 :JSONエスケープの不備(正常系)
  2. 4g-007 :JSONエスケープの不備(XSS攻撃)
  3. 4g-007a:JSONエスケープの対策版(XSS攻撃)

```
12 <li>JSONエスケープの問題</li>
13 <ol>
14 <li><a href="4g-007.html#1">4g-007 :JSONエスケープの不備(正常系)</a></li>
15 <li><a href="4g-007.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007 :JSONエスケープの不備(XSS攻撃)</a></li>
16 <li><a href="4g-007a.html#1&quot;%2balert(document.domain)%2b&quot;">4g-007a:JSONエスケープの対策版(XSS攻撃)</a></li>
17 </ol>
```



郵便番号が見つかりません:1"+alert(document.domain)+"




```
1 <body>
2 <script src="../js/jquery-3.2.1.min.js"></script>
3 <script>
4 $.ajax({
5   url: "http://api.example.net/4g/4g-006a.php?zip=" + location.hash.slice(1),
6   dataType: "jsonp",
7   jsonpCallback: "callback_zip"
8 }).done(function(data) {
9   $('#message').text(data.message);
10 });
11 </script>
12 <p id="message"></p>
13 <body>
```

【サーバ: 4g/4g-007a.html 】

/var/www/html/4g/4g-007a.html - wasbook@example.jp - エディタ - WinSCP

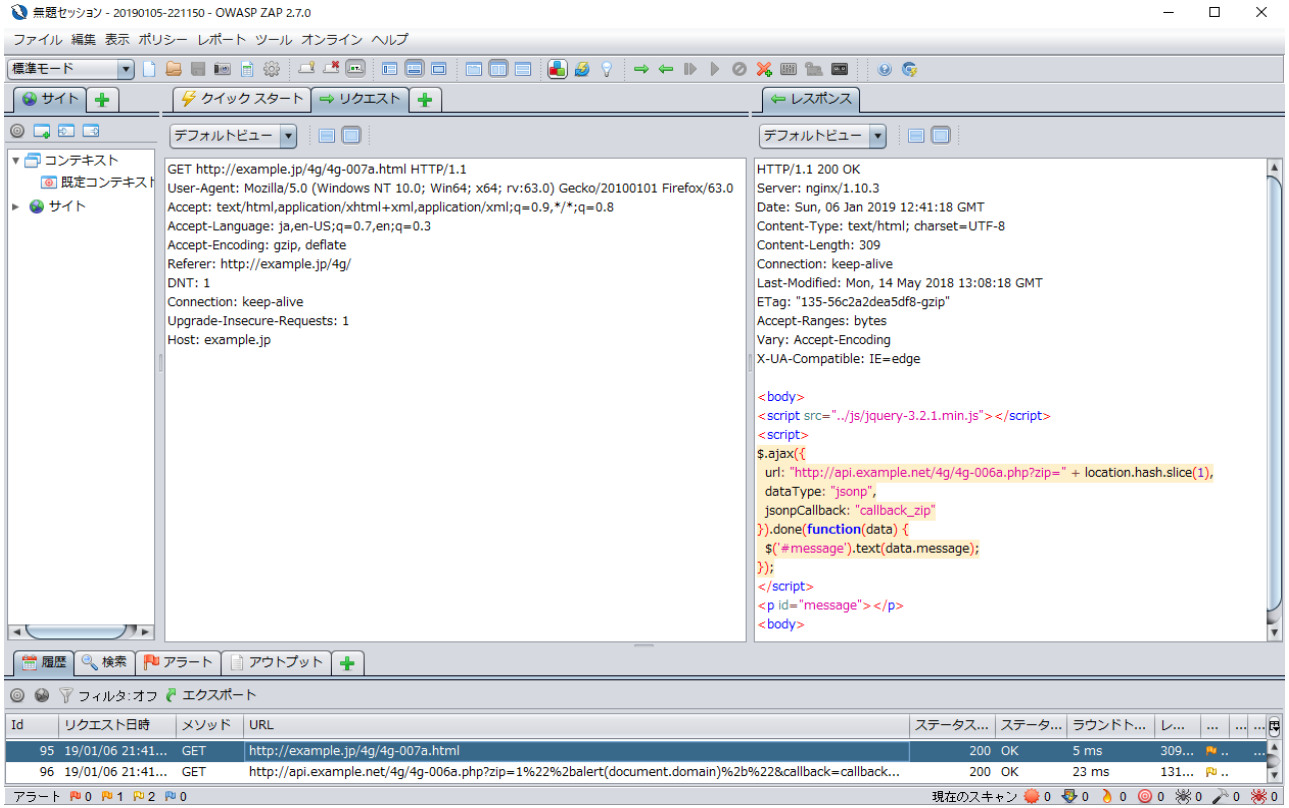
```
<body>
<script src="../js/jquery-3.2.1.min.js"></script>
<script>
$.ajax({
  url: "http://api.example.net/4g/4g-006a.php?zip=" + location.hash.slice(1),
  dataType: "jsonp",
  jsonpCallback: "callback_zip"
}).done(function(data) {
  $('#message').text(data.message);
});
</script>
<p id="message"></p>
<body>
```

【サーバ: 4g/4g-006a.php 】

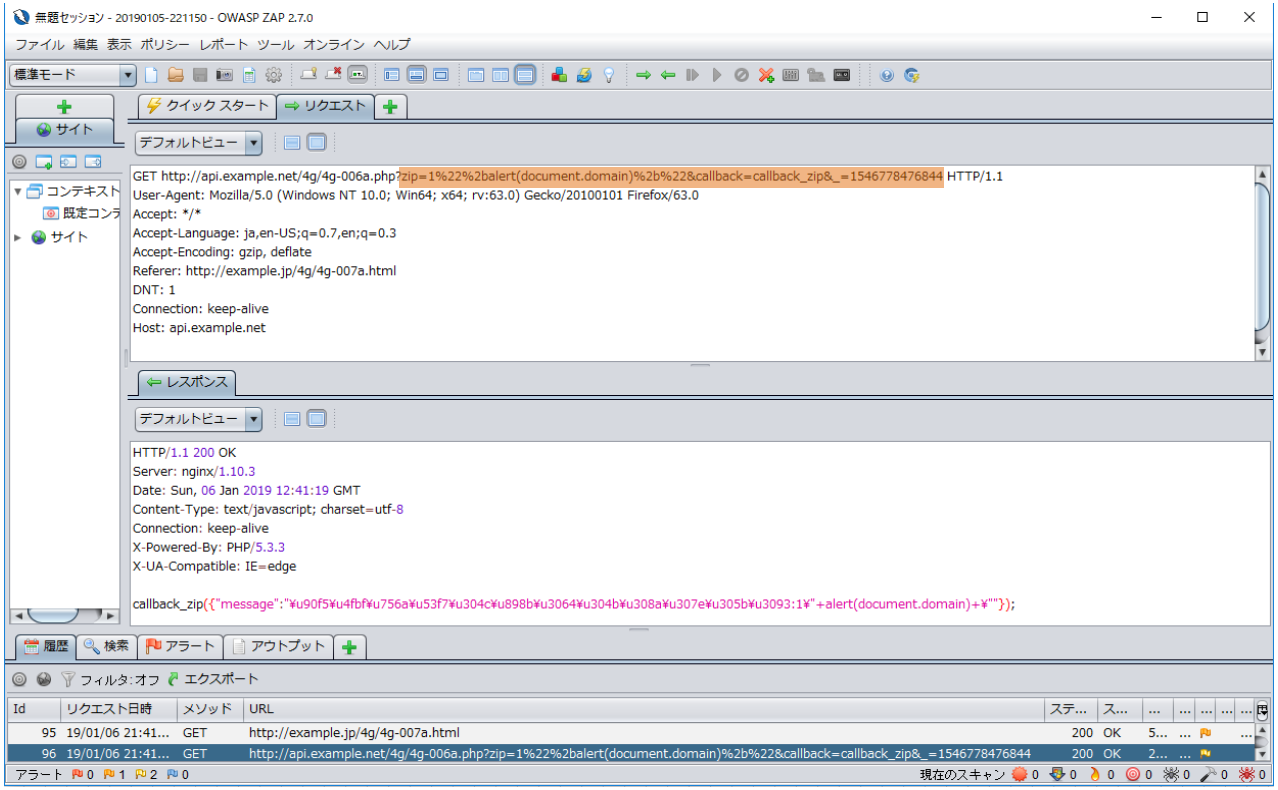
/var/www/html/4g/4g-006a.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
 $zip  = $_GET['zip'];
 // 以下は郵便番号が見つからなかった場合の処理
 $json = json_encode(array("message" => "郵便番号が見つかりません:" . $zip));
 header('Content-Type: text/javascript; charset=utf-8');
 echo "callback_zip($json);";
```

【ブラウザ→サーバ： リクエスト 4g/4g-007a.php → レスポンス 】



【ブラウザ→サーバ： リクエスト 4g/4g-006a.php → レスポンス 】

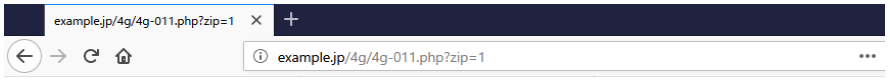


・JSON直接閲覧によるXSS
4g-011 :JSON直接閲覧

【ブラウザ】

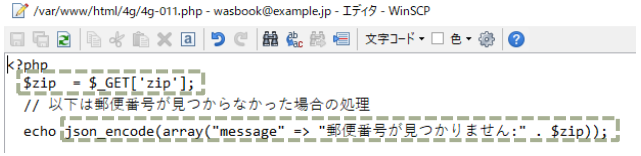
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
18 <li>JSON直接閲覧によるXSS</li>
19 <ol>
20 <li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
21 <li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
22 <li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
23 <li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
24 <li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
25 <li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
26 <li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
27 </ol>
```



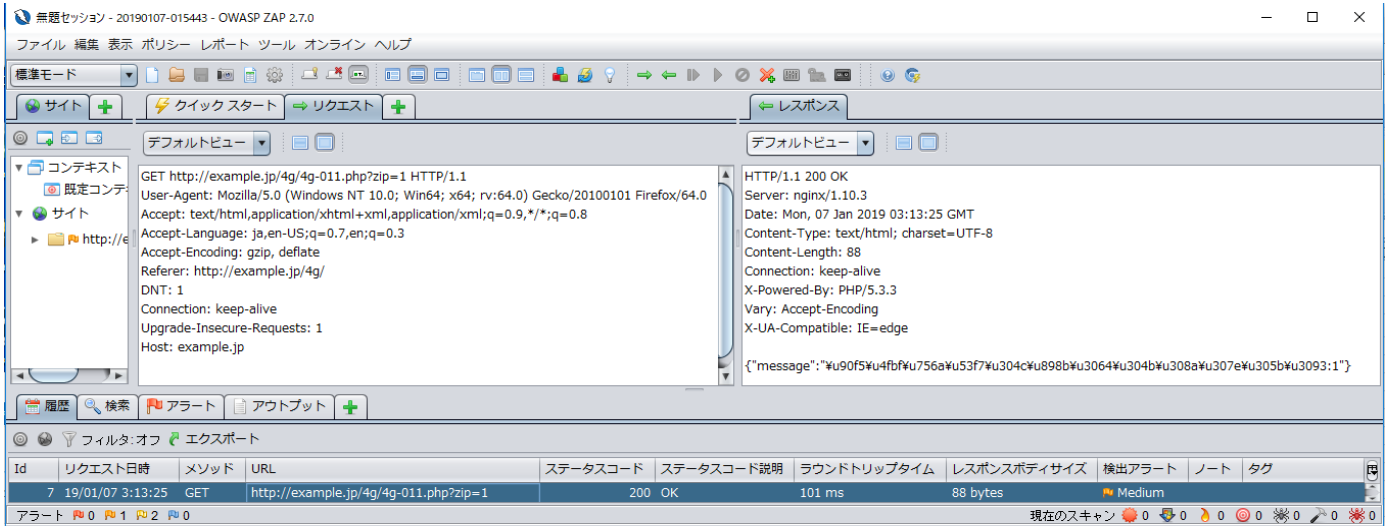
{"message":"¥u90f5¥u4fbf¥u756a¥u53f7¥u304c¥u898b¥u3064¥u304b¥u308a¥u307e¥u305b¥u3093:1"}

【サーバ: 4g/4g-011.php 】




```
<?php
 $zip  = $_GET['zip'];
 // 以下は郵便番号が見つからなかった場合の処理
 echo json_encode(array("message" => "郵便番号が見つかりません:" . $zip));
```

【ブラウザ→サーバ： リクエスト 4g/4g-011.php → レスポンス 】

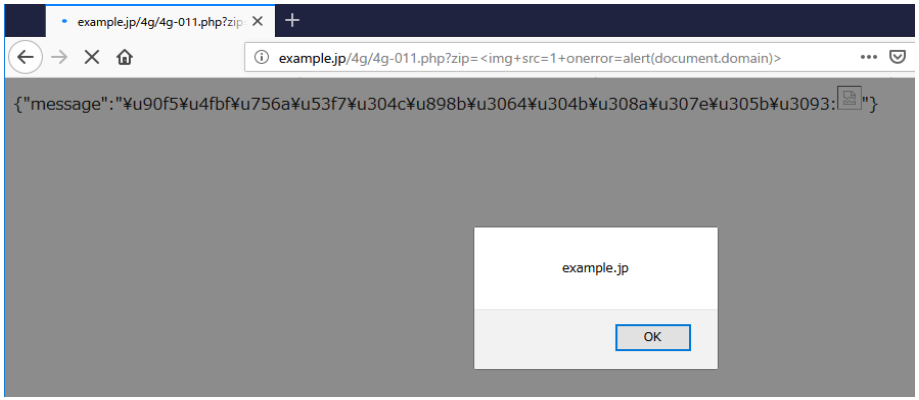


4g-011 :JSON直接閲覧(XSS攻撃)

【ブラウザ】

- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
18 <li>JSON直接閲覧によるXSS</li>
19 <ol>
20 <li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
21 <li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
22 <li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
23 <li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
24 <li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
25 <li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
26 <li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
27 </ol>
```



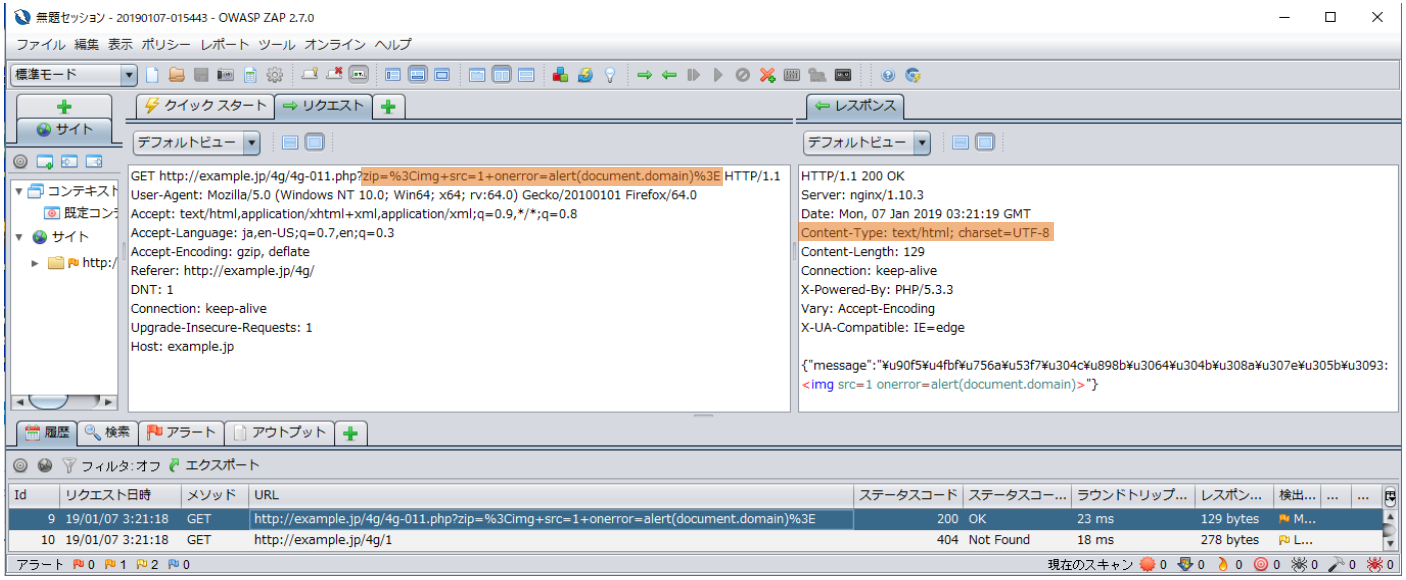
【サーバ: 4g/4g-011.php 】

/var/www/html/4g/4g-011.php - wasbook@example.jp - エディタ - WinSCP

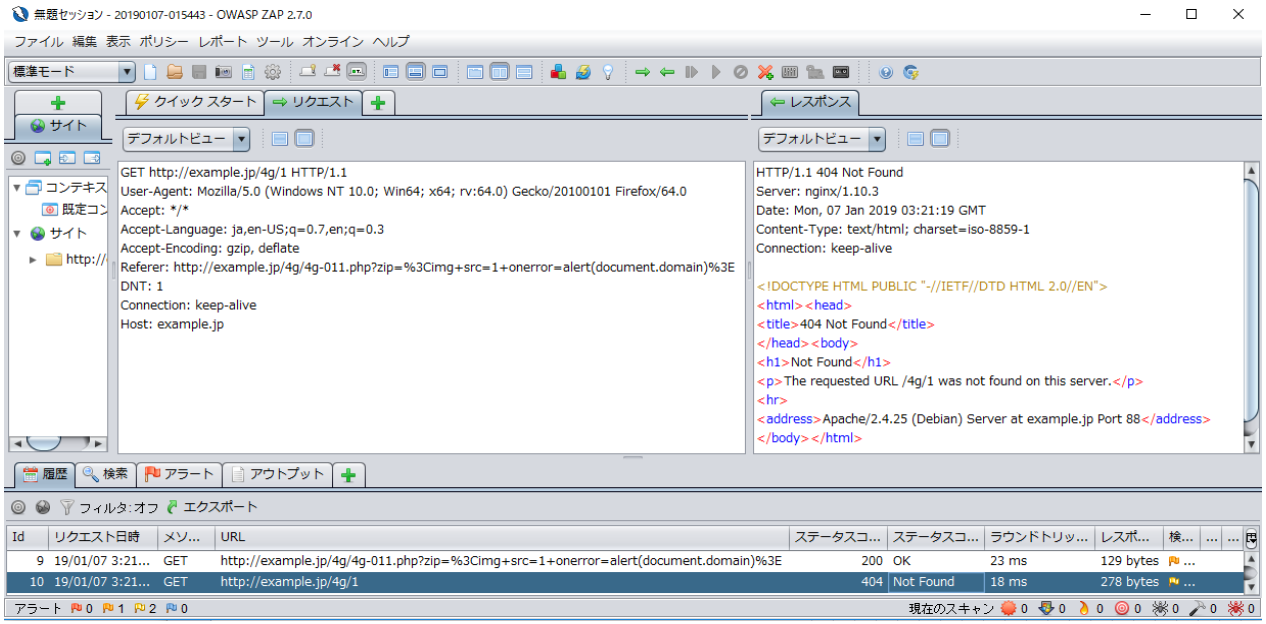
```
<?php
 $zip  = $_GET['zip'];
 // 以下は郵便番号が見つからなかった場合の処理
 echo json_encode(array("message" => "郵便番号が見つかりません:" . $zip));
```

JSONを返すので、MIMEタイプは application/json にすべきですが、
指定していないので、デフォルトの text/html を返します
そのため、JavaScriptが実行されてしまいます

【ブラウザ→サーバ： リクエスト 4g/4g-011.php → レスポンス 】



【ブラウザ→サーバ： リクエスト 4g/4g-011.php → レスポンス 】

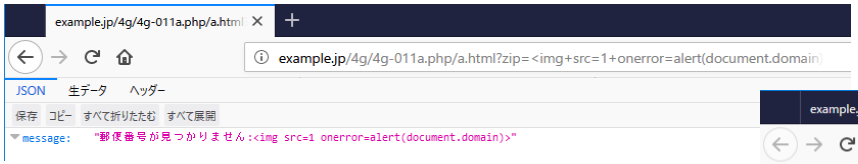


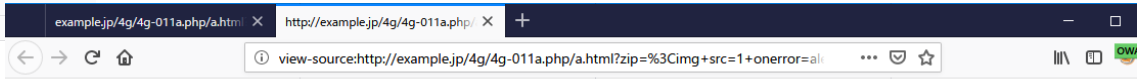
4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)

【ブラウザ】

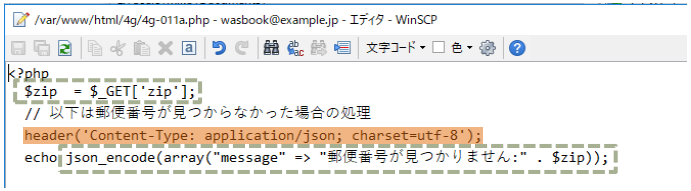
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
<li>JSON直接閲覧によるXSS</li>
<ol>
<li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
<li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
<li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
<li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
<li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
<li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
<li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
</ol>
```





【サーバ: 4g/4g-011a.php 】

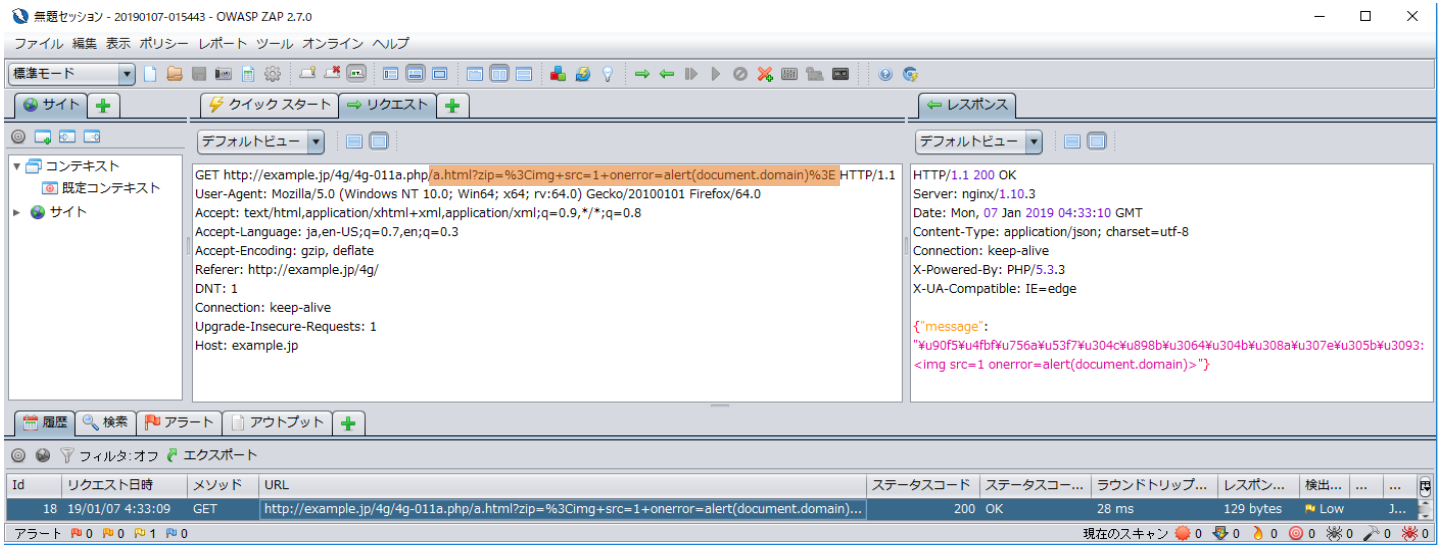


application/json を正しく指定しているが、/a.html(PATHINFO)を指定して、
IEがContent-Typeを正しく判断しない脆弱性をついています

IE9以前ではJavaScriptが実行されて、攻撃が成功します
レスポンスヘッダ X-Content-Type-Options:nosniff を出力するとJavaScript実行を回避できます

この設定でも、IE7以前ではJavaScriptは実行されてしまいます

【ブラウザ→サーバ: リクエスト 4g/4g-011a.php → レスポンス 】

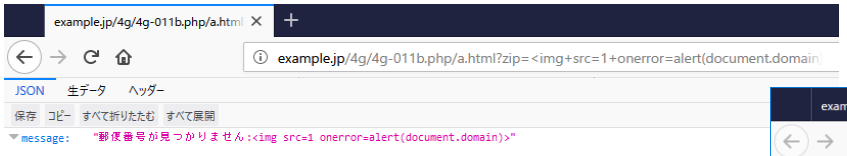


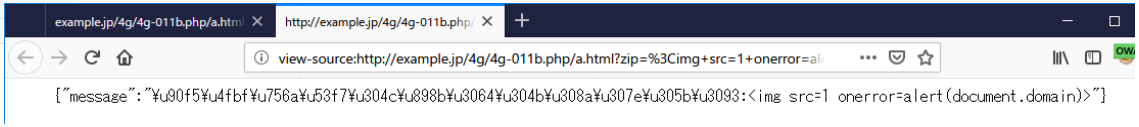
4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)

【ブラウザ】

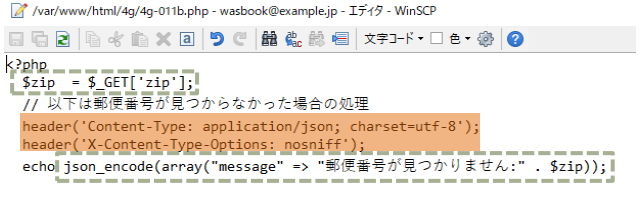
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
<li>JSON直接閲覧によるXSS</li>
<ol>
<li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
<li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
<li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
<li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
<li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
<li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
<li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
</ol>
```





【サーバ:4g/4g-011b.php 】



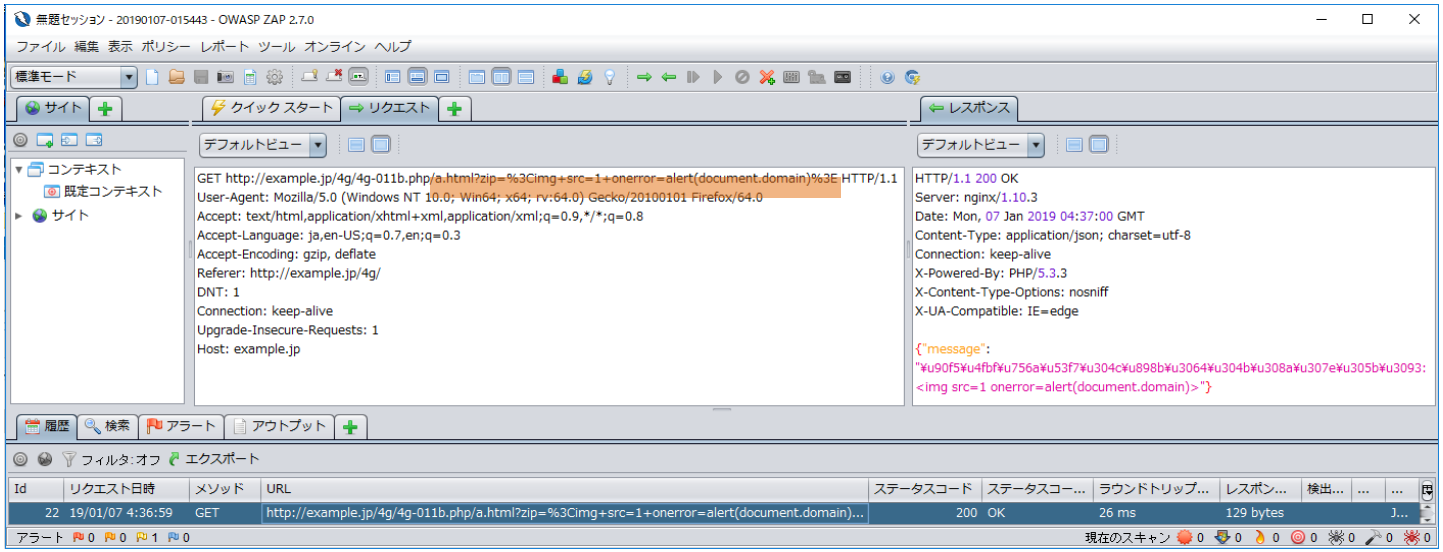

```
<?php
 $zip  = $_GET['zip'];
  // 以下は郵便番号が見つからなかった場合の処理
  header('Content-Type: application/json; charset=utf-8');
  header('X-Content-Type-Options: nosniff');
 echo json_encode(array("message" => "郵便番号が見つかりません:" . $zip));
```

application/json を正しく指定しているが、/a.html(PATHINFO)を指定して、
IEがContent-Typeを正しく判断しない脆弱性をついています
IE9以前ではJavaScriptが実行されて、攻撃が成功します

レスポンスヘッダ X-Content-Type-Options:nosniff を出力するとJavaScript実行を回避できます

この設定でも、IE7以前ではJavaScriptは実行されてしまいます

【ブラウザ→サーバ: リクエスト 4g/4g-011b.php → レスポンス 】

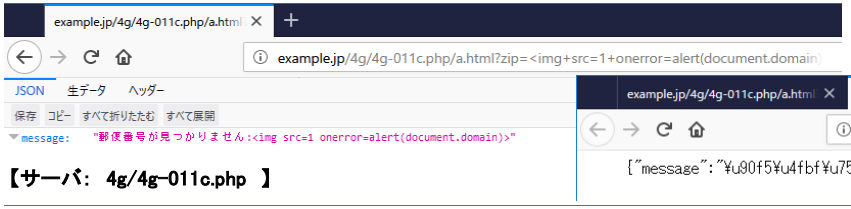


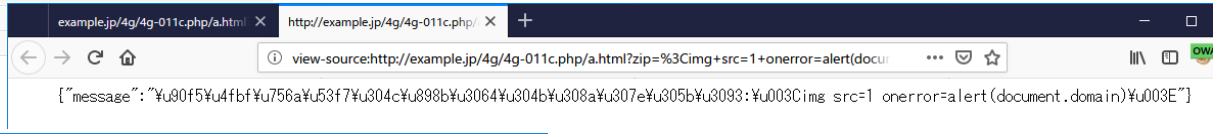
4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)

【ブラウザ】

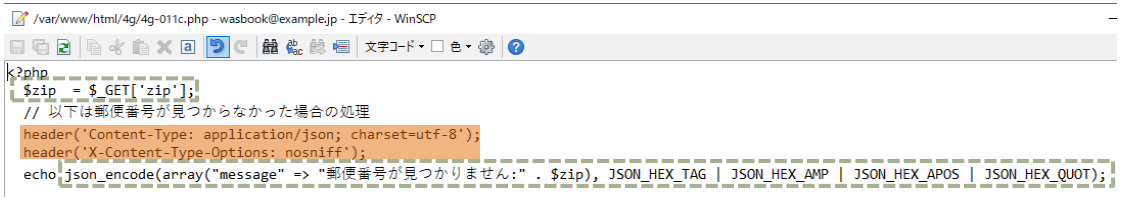
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
<li>JSON直接閲覧によるXSS</li>
<ol>
<li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
<li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
<li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
<li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
<li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
<li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
<li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
</ol>
```





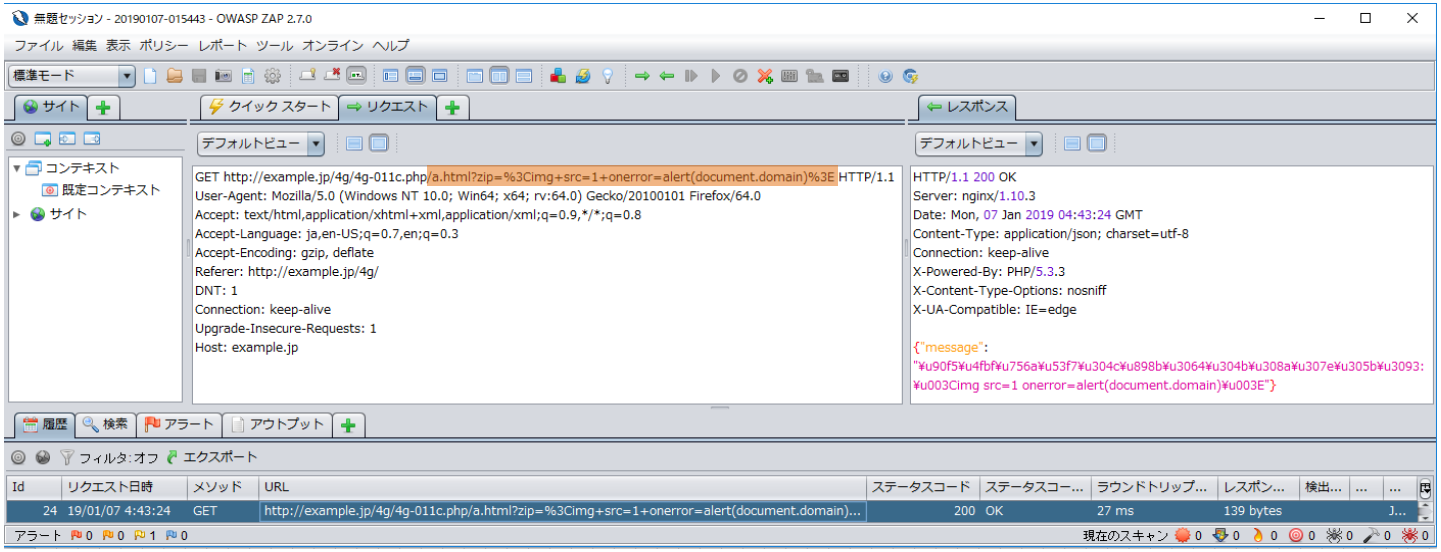
【サーバ: 4g/4g-011c.php 】



application/json を正しく指定しているが、/a.html(PATHINFO)を指定して、
IEがContent-Typeを正しく判断しない脆弱性をついています
IE9以前ではJavaScriptが実行されて、攻撃が成功します
レスポンスヘッダ X-Content-Type-Options:nosniff を出力するとJavaScript実行を回避できます
この設定でも、IE7以前ではJavaScriptは実行されてしまいます

PHPの場合、json_encodeのオプションパラメータで、[< >] をUnicodeエスケープできるので、
IE7以前でも JavaScript の実行を回避できます

【ブラウザ→サーバ: リクエスト 4g/4g-011c.php → レスポンス 】

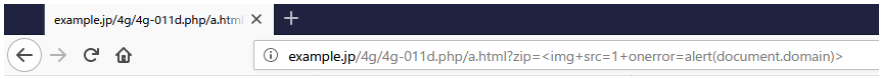


4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)

【ブラウザ】

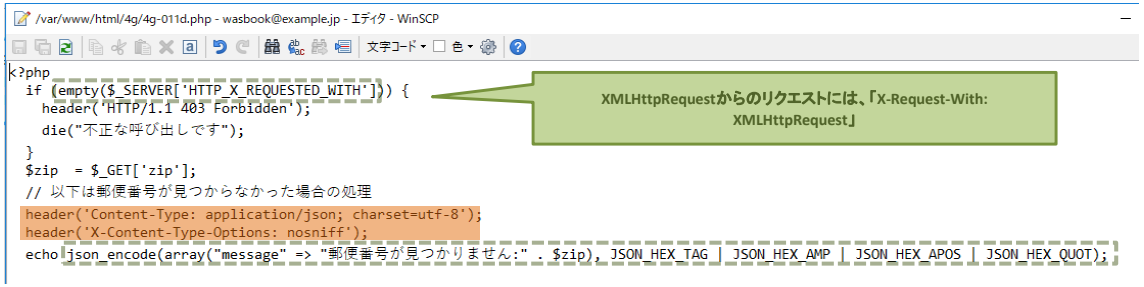
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
18 <li>JSON直接閲覧によるXSS</li>
19 <ol>
20 <li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
21 <li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
22 <li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
23 <li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
24 <li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
25 <li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
26 <li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
27 </ol>
```



不正な呼び出しです

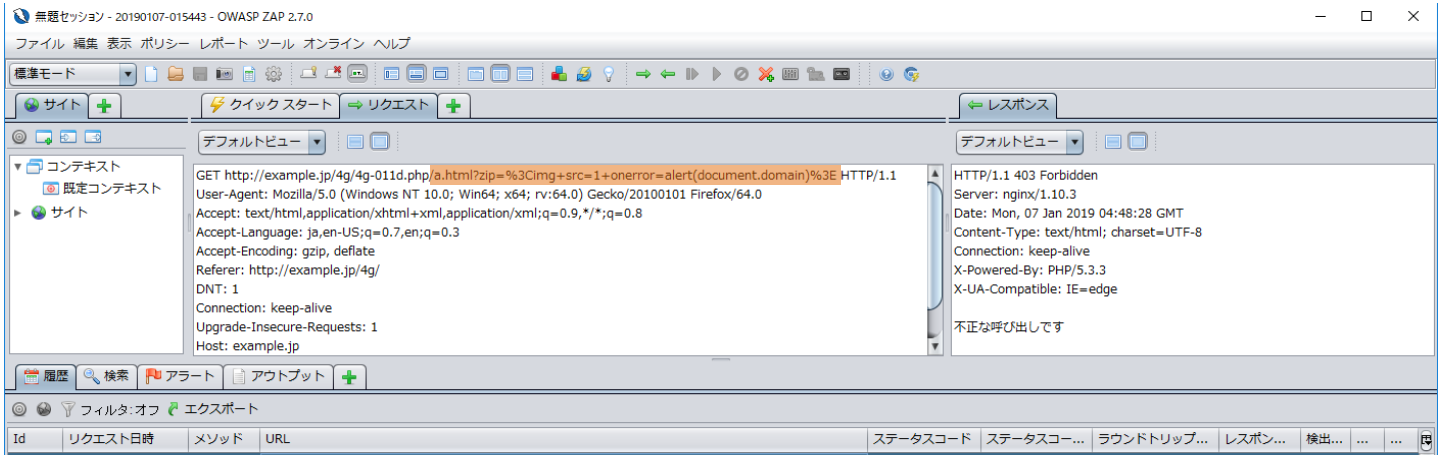
【サーバ: 4g/4g-011d.php 】

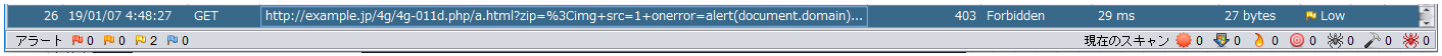


application/json を正しく指定しているが、/a.html(PATHINFO)を指定して、
IEがContent-Typeを正しく判断しない脆弱性をついています
IE9以前ではJavaScriptが実行されて、攻撃が成功します
レスポンスヘッダ X-Content-Type-Options:nosniff を出力するとJavaScript実行を回避できます
この設定でも、IE7以前ではJavaScriptは実行されてしまいます
PHPの場合、json_encodeのオプションパラメータで、[< >] をUnicodeエスケープできるので、
IE7以前でも JavaScript の実行を回避できます

さらに左吹き出しの対処を追加しています

【ブラウザ→サーバ: リクエスト 4g/4g-011d.php → レスポンス 】



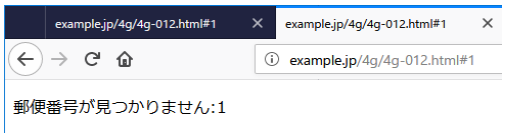


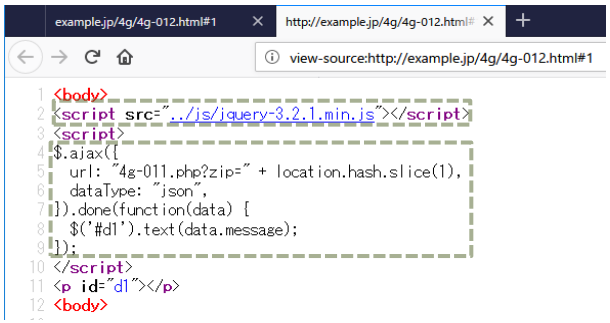
4g-012 :4g-011d.phpの呼び出し(jQuery利用)

【ブラウザ】

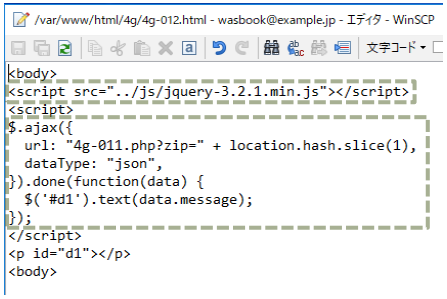
- JSON直接閲覧によるXSS
  1. 4g-011 :JSON直接閲覧
  2. 4g-011 :JSON直接閲覧(XSS攻撃)
  3. 4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)
  4. 4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)
  5. 4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)
  6. 4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)
  7. 4g-012 :4g-011d.phpの呼び出し(jQuery利用)

```
<li>JSON直接閲覧によるXSS</li>
<ol>
<li><a href="4g-011.php?zip=1">4g-011 :JSON直接閲覧</a></li>
<li><a href="4g-011.php?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011 :JSON直接閲覧(XSS攻撃)</a></li>
<li><a href="4g-011a.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011a:JSON直接閲覧(Content-Type設定版)(IE9以前ならXSS可能)</a></li>
<li><a href="4g-011b.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011b:JSON直接閲覧(X-Content-Type-Options: nosniff設定版)(IE7以前ならXSS可能)</a></li>
<li><a href="4g-011c.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011c:JSON直接閲覧(タグ等もエスケープ)(IE7でもJavaScriptは実行されない)</a></li>
<li><a href="4g-011d.php/a.html?zip=&lt;img+src=1+onerror=alert(document.domain)&gt;">4g-011d:JSON直接閲覧(X-Requested-Withヘッダの確認; 403 Forbiddenとなる)</a></li>
<li><a href="4g-012.html#1">4g-012 :4g-011d.phpの呼び出し(jQuery利用)</a></li>
</ol>
```

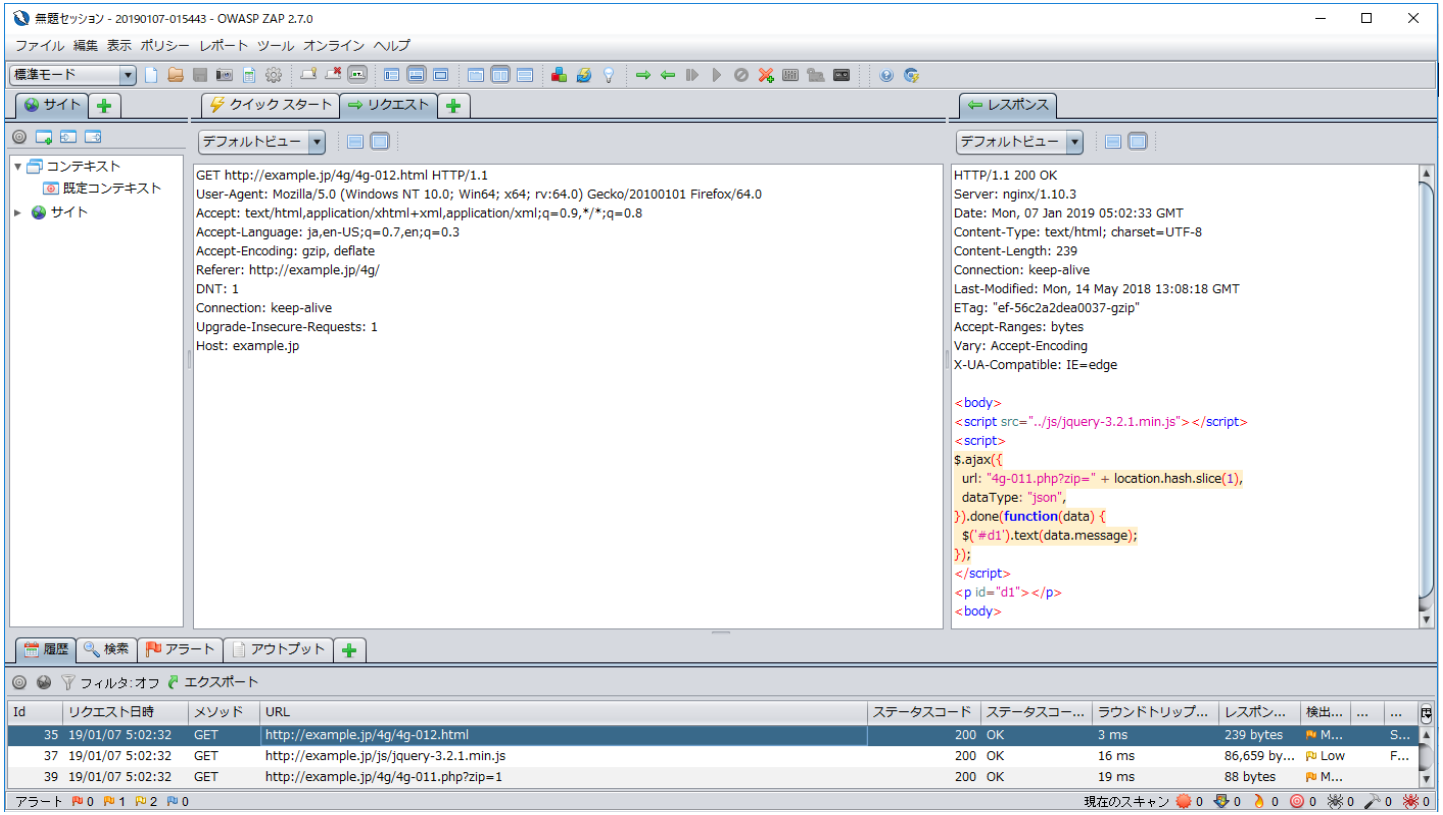




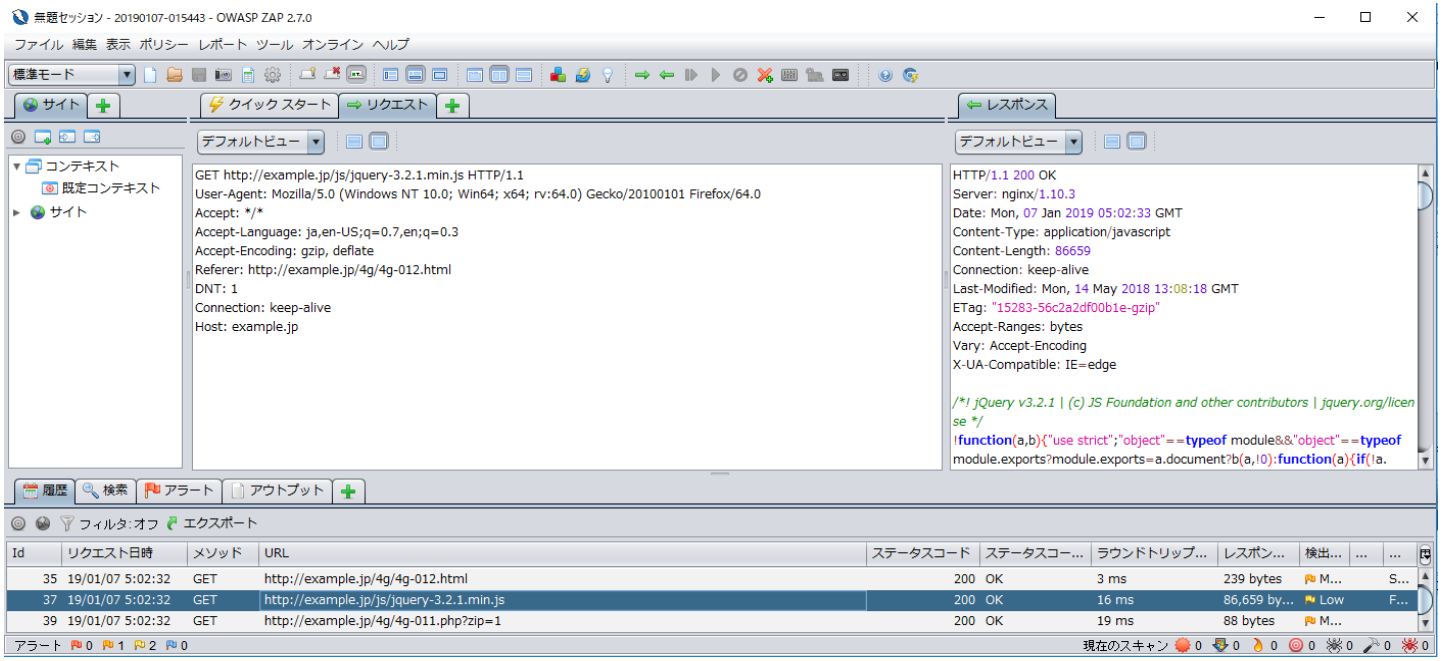
【サーバ: 4g/4g-012.html 】



【ブラウザ→サーバ： リクエスト 4g/4g-012.html → レスポンス 】

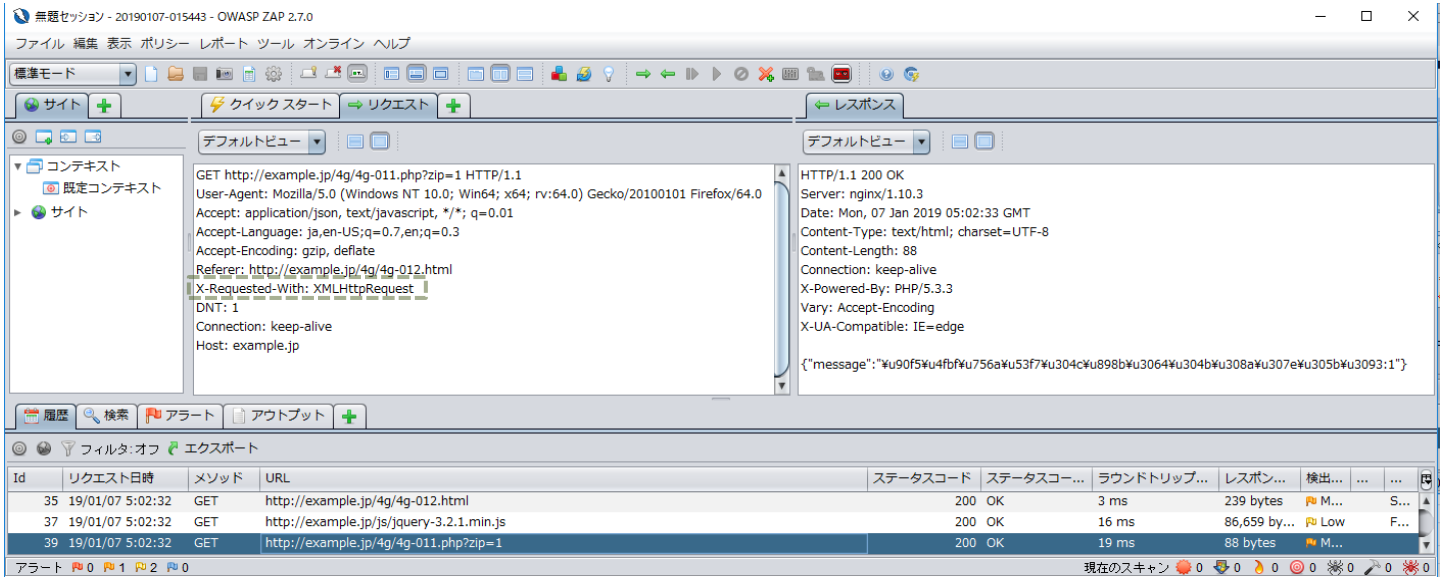


【ブラウザ→サーバ： リクエスト js/jquery-3.2.1.min.js → レスポンス 】



【ブラウザ→サーバ： リクエスト 4g/4g-011.php → レスポンス 】

jQueryなどの著名なJavaScriptライブラリは、XMLHttpRequertによるHTTPリクエストに、X-Request-Withヘッダを付与する

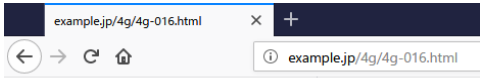


・JSONPコールバック関数名によるXSS
4g-016 ： JSONPコールバック関数名によるXSS(HTMLからの正常系利用)

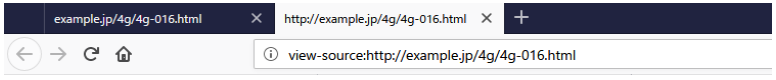
【ブラウザ】

- JSONPコールバック関数名によるXSS
  1. 4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)
  2. 4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)
  3. 4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)
  4. 4g-015a:JSONPコールバック関数名によるXSS(対策版)

```
28 <li>JSONPコールバック関数名によるXSS</li>
29 <ol>
30 <li><a href="4g-016.html">4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)</a></li>
31 <li><a href="http://api.example.net/4g/4g-015.php?callback=display_time">4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)</a></li>
32 <li><a href="http://api.example.net/4g/4g-015.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)</a></li>
33 <li><a href="http://api.example.net/4g/4g-015a.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015a:JSONPコールバック関数名によるXSS(対策版)</a></li>
34 </ol>
```



時刻は14:13です




```
1 <body>
2 <script>
3 function display_time(obj) {
4   var txt = document.createTextNode("時刻は" + obj.time + "です");
5   var p = document.getElementById("time");
6   p.appendChild(txt);
7 }
8 </script>
9 <p id="time"></p>
10 <script src="http://api.example.net/4g/4g-015.php?callback=display_time"></script>
11 <body>
```

【サーバ： 4g/4g-016.html 】

/var/www/html/4g/4g-016.html - wasbook@example.jp - エディタ - WinSCP

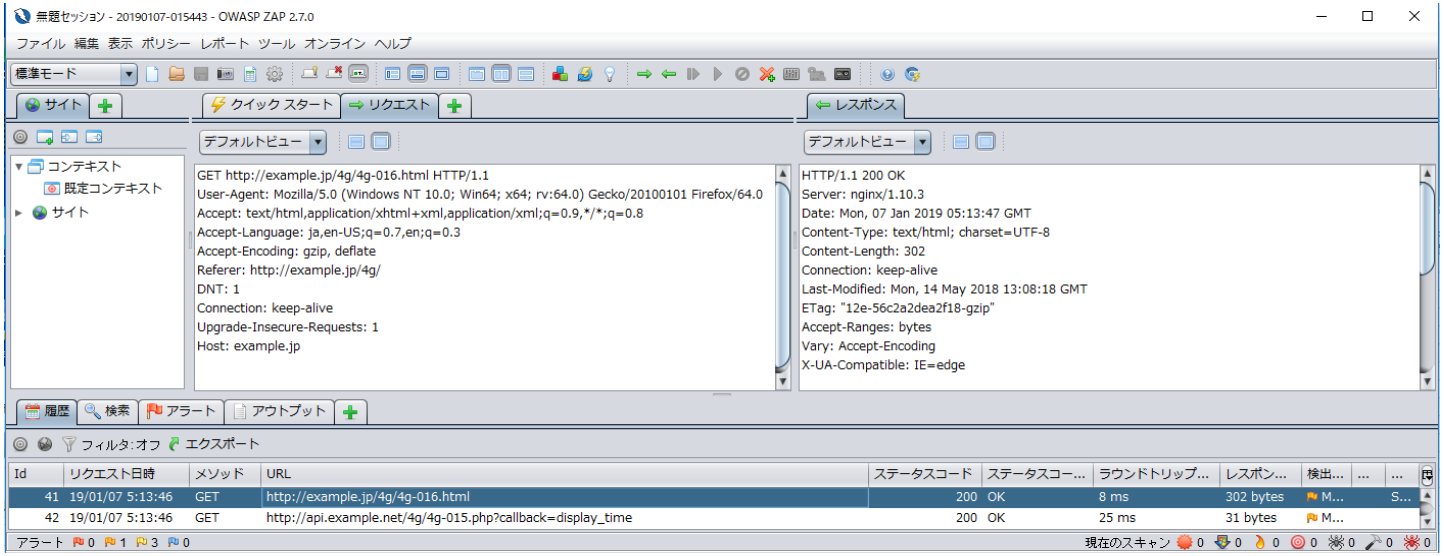
```
<body>
<script>
function display_time(obj) {
  var txt = document.createTextNode("時刻は" + obj.time + "です");
  var p = document.getElementById("time");
  p.appendChild(txt);
}
</script>
<p id="time"></p>
<script src="http://api.example.net/4g/4g-015.php?callback=display_time"></script>
<body>
```

【サーバ： 4g/4g-015.php 】

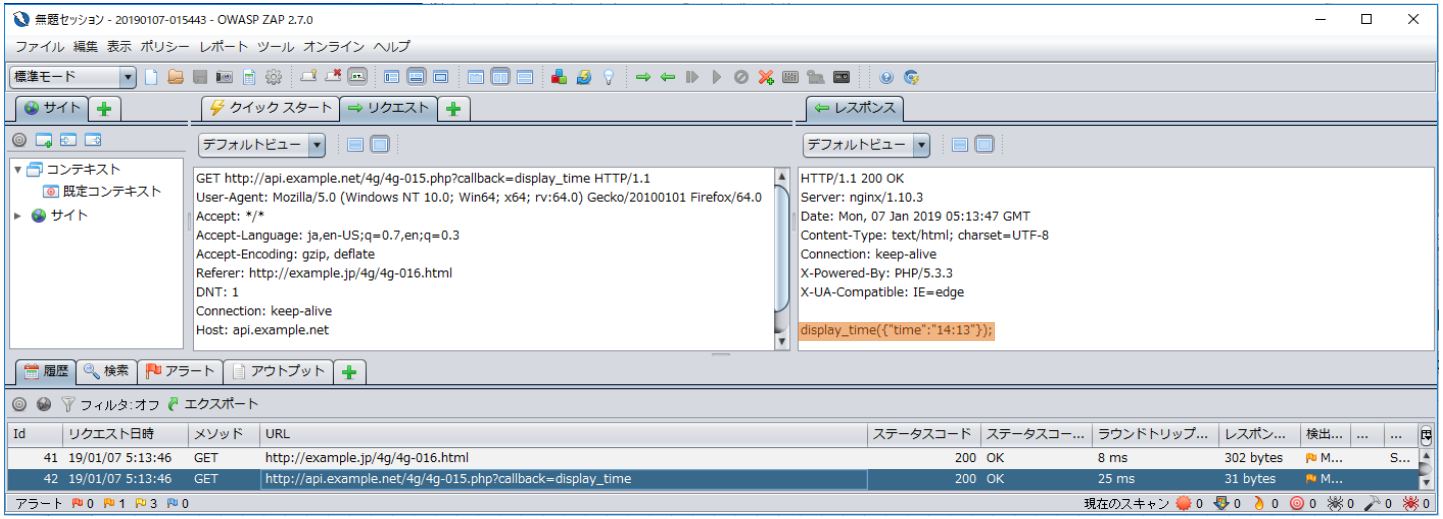
/var/www/html/4g/4g-015.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  $callback = $_GET['callback'];
  $json = json_encode(array('time' => date('G:i')));
  echo "$callback($json);";
```

【ブラウザ→サーバ： リクエスト 4g/4g-016.html → レスポンス 】



【ブラウザ→APIサーバ： リクエスト 4g/4g-015.php → レスポンス 】

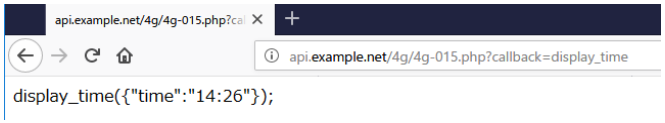


4g-015 ： JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)

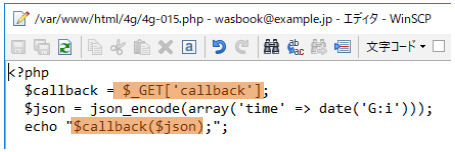
【ブラウザ】

- JSONPコールバック関数名によるXSS
  1. 4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)
  2. 4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)
  3. 4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)
  4. 4g-015a:JSONPコールバック関数名によるXSS(対策版)

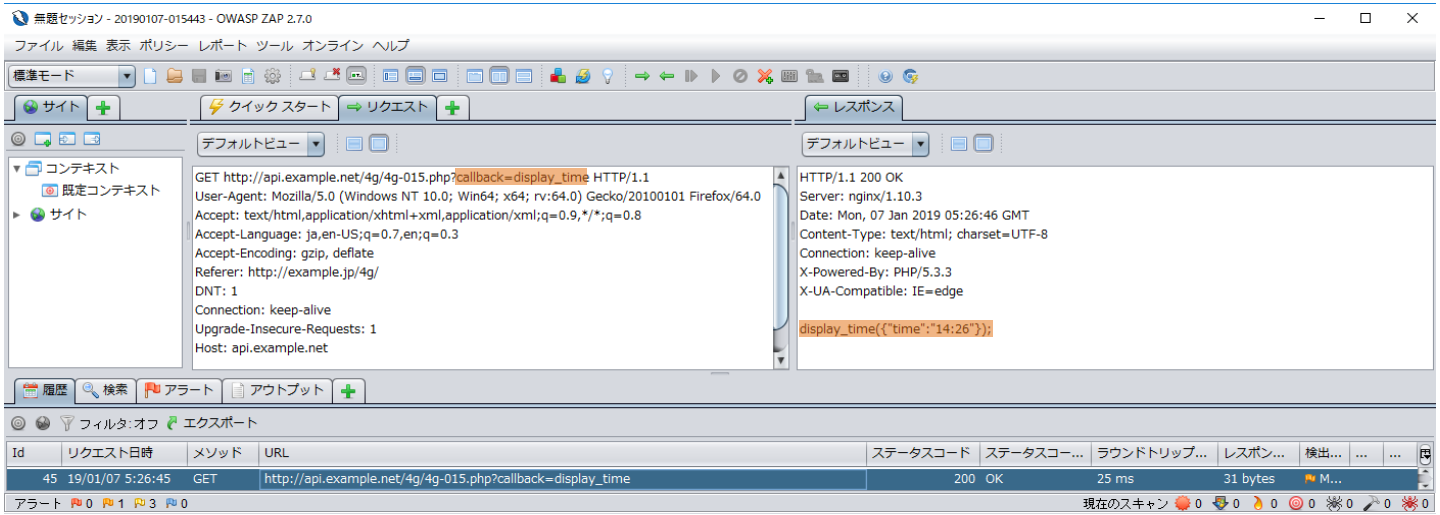
```
<li>JSONPコールバック関数名によるXSS</li>
<ol>
<li><a href="4g-016.html">4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)</a></li>
<li><a href="http://api.example.net/4g/4g-015.php?callback=display_time">4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)</a></li>
<li><a href="http://api.example.net/4g/4g-015.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)</a></li>
<li><a href="http://api.example.net/4g/4g-015a.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015a:JSONPコールバック関数名によるXSS(対策版)</a></li>
</ol>
```



【サーバ： 4g/4g-015.php 】



【ブラウザ→APIサーバ： リクエスト 4g/4g-015.php → レスポンス 】

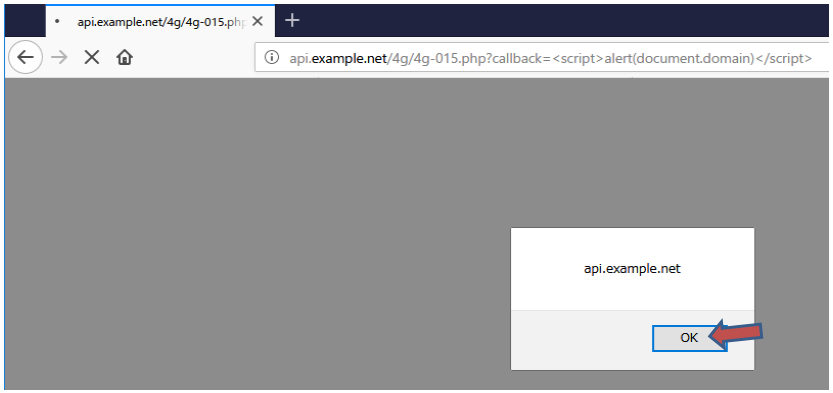


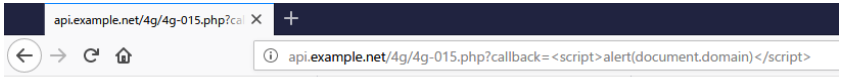
4g-015 ： JSONPコールバック関数名によるXSS(XSS攻撃)

【ブラウザ】

- JSONPコールバック関数名によるXSS
  1. 4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用
  2. 4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)
  3. 4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)
  4. 4g-015a:JSONPコールバック関数名によるXSS(対策版)

```
28 <li>JSONPコールバック関数名によるXSS</li>
29 <ol>
30 <li><a href="4g-016.html">4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)</a></li>
31 <li><a href="http://api.example.net/4g/4g-015.php?callback=display_time">4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)</a></li>
32 <li><a href="http://api.example.net/4g/4g-015.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)</a></li>
33 <li><a href="http://api.example.net/4g/4g-015a.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015a:JSONPコールバック関数名によるXSS(対策版)</a></li>
34 </ol>
```





({"time":"14:34"});

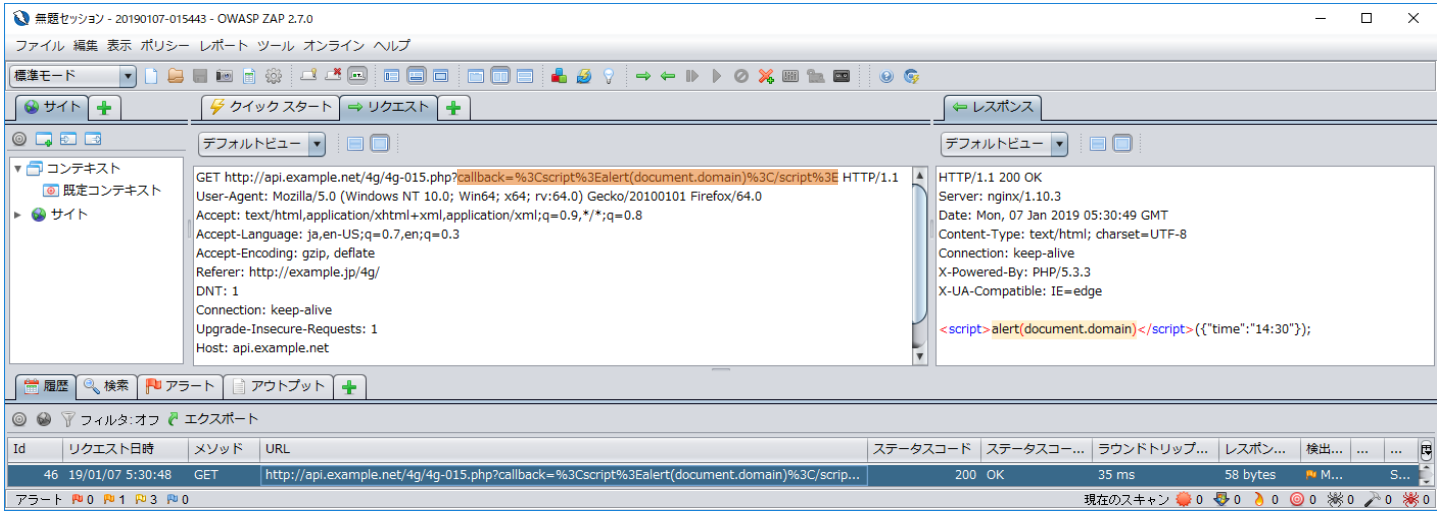
【サーバ： 4g/4g-015.php 】

/var/www/html/4g/4g-015.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  $callback = $_GET['callback'];
  $json = json_encode(array('time' => date('G:i')));
  echo "$callback($json);";
```

JSONPはJSONではなく、JavaScriptそのままなので、
MIMEタイプは text/javascript にすべきですが、
指定していないので、デフォルトの text/html を返します
そのため、JavaScriptが実行されてしまいます

【ブラウザ→APIサーバ： リクエスト 4g/4g-015.php → レスポンス 】

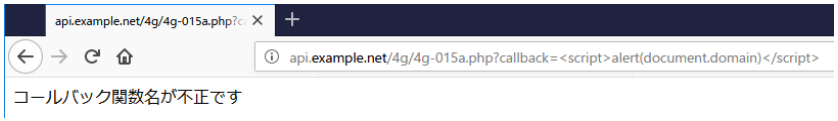


4g-015 ： JSONPコールバック関数名によるXSS(XSS攻撃)

【ブラウザ】

- JSONPコールバック関数名によるXSS
  1. 4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)
  2. 4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)
  3. 4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)
  4. 4g-015a:JSONPコールバック関数名によるXSS(対策版)

```
<li>JSONPコールバック関数名によるXSS</li>
<ol>
<li><a href="4g-016.html">4g-016 :JSONPコールバック関数名によるXSS(HTMLからの正常系利用)</a></li>
<li><a href="http://api.example.net/4g/4g-015.php?callback=display_time">4g-015 :JSONPコールバック関数名によるXSS(攻撃でない直接閲覧)</a></li>
<li><a href="http://api.example.net/4g/4g-015.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015 :JSONPコールバック関数名によるXSS(XSS攻撃)</a></li>
<li><a href="http://api.example.net/4g/4g-015a.php?callback=%3Cscript%3Ealert(document.domain)%3C/script%3E">4g-015a:JSONPコールバック関数名によるXSS(対策版)</a></li>
</ol>
```



【サーバ： 4g/4g-015a.php 】

/var/www/html/4g/4g-015a.php - wasbook@example.jp - エディタ - WinSCP

```
k?php
  $callback = $_GET['callback'];
  if (preg_match('/\A[_a-z][_a-z0-9]*\z/i', $callback) !== 1) {
    header('HTTP/1.1 403 Forbidden');
    die('コールバック関数名が不正です');
  }
  header('Content-Type: text/javascript; charset=UTF-8');
  $json = json_encode(array('time' => date('G:i')));
  echo "$callback($json);";
```

コールバック関数名の文字種と文字数を制限しています
※ 英数字とアンダースコア「 _ 」のみに制限しても問題ないと思われる。<>を使えなくする。

MIMEタイプを正しく設定する
text/javascript とすると、レスポンスをHTMLとして解釈しないので、script要素は解釈されず、
JavaScriptとしても実行されません

【ブラウザ→APIサーバ： リクエスト 4g/4g-015a.php → レスポンス 】