5.1 認証

認証

認証の種類

- HTTP認証　：　Basic認証、NTLM認証、Digest認証
- フォーム認証
- クライアント認証

不正ログインの原因となるセキュリティバグ

- SQLインジェクション
- セッションIDの固定化
- クッキーのセキュア属性不備
- オープンリダイレクト脆弱性
- HTTPヘッダ・インジェクション

パスワードに関するセキュリティ要件

● 文字種　：　英数字（大文字、小文字を区別）　　制約が厳しすぎる　　文字種　：　US-ASCII文字すべて（0X20～0X7E）
　桁数　　：　8桁まで入力可能　　桁数　　：　128桁以下

● パスフレーズ（文の形をした長いパスワード）が有効　　andesudetaberukusamochiisveryhapp(アンデスで食べる草餅　is very happy ）
● パスワードポリシーのチェック

基本的なアカウントロック

① パスワード間違いの回数、3回では少なすぎで、10回程度が良い
② アカウントロックした場合にはメールなどで、対象利用者と管理者に通知する
③ アカウントロックから30分経過した場合、自動的に再有効化する
　管理者が何らかの方法で本人確認した後に再有効化する

パスワード攻撃　　※　adobe社サイトからの漏洩パスワードの統計では、123456というパスワードは集計対象の5%だった

- 辞書攻撃
- ジョーアカウント探索　　IDとPWが同じ
- リバースブルートフォース攻撃
- パスワードスプレー攻撃　　ID、PWとも固定しないで、少数のパスワード候補をIDを変えながら試していく攻撃
- パスワードリスト攻撃　　成功率は高い場合には、数%から10%を超える場合があり、高い成功率

パスワード攻撃に対する対策　　※　adobe社サイトからの漏洩パスワードの統計では、123456というパスワードは集計対象の5%だった

① 2段階認証の実装　　利用者の負荷が高い　　認証タイミング調整　①初回ログイン時のみ2段階認証を行い、その後は一定期間一段階の認証とする
　②利用者の通常の使い方から外れたログイン（地域、時間帯、ブラウザなど）　：　リスクベース認証
　③決済やパスワード変更などの重要な処理の時
② 積極的なパスワードチェック　　実装が簡単だが、十分ではない
③ ログイン失敗率の監視　　監視人員が必要で、コストが高い

メッセージダイジェストによるパスワード保護のために必要なハッシュ関数の要件

・現像計算困難性　　一方向性　　※ GPUを使った総当たり攻撃の結果、
・第2現像計算困難性　　弱衝突耐性　　8桁で英数字・数字・記号を使った総当たり攻撃は、MD5で約14分、SHA-1で約39分で完了する
・衝突困難性　　強衝突耐性

パスワードのメッセージダイジェストに対する攻撃

● オフラインブルートフォース攻撃　　GPUを使った総当たり攻撃の結果、8桁で英数字・数字・記号を使った総当たり攻撃は、MD5で約14分、SHA-1で約39分で完了する。

● レインボーテーブル　　RainbowCrack ProjectのWebサイトには、8文字までのUS-ASCII文字全部に対応したMD5用テーブルや
10文字までの英小文字と数字に対応したSHA-1用テーブルが売られています。
単純なハッシュ関数による保存パスワードは市販レインボーテーブルで解読されます。

● ユーザDB内にパスワード辞書を作られる　　Webアプリに多数のユーザ、パスワードを登録し、同じハッシュ値を持つデータから元パスワードを突き止める。

パスワードのメッセージダイジェスト攻撃に対する対策

・ ソルト
・ ストレッチング
・ 処理速度の遅いハッシュ関数を使用する

自動ログインの安全な実装

・ セッションの寿命を延ばす
・ トークンを利用する（セッション寿命を延ばせない場合）
・ 認証チケットを利用する（Kerberos認証、ASP.NETのフォーム認証、複数サーバまたがった認証の場合には、SSO製品、OpenID Connectなどのオープンな認証基盤の利用）

ログアウト機能の要件

・ ログアウト処理は副作用があるので、POSTメソッドでリクエストする
・ ログアウト処理ではセッションを破棄する
・ 必要な場合、CSRF対策を実施する

5.2 アカウント管理

認証

メールアカウントの受信確認（実際にメールを送信して確認）

① メールにトークンつきURLを添付して、そのURLから処理を継続する　メール送信→メールのURLをクリック→URLのページから登録を再開　（利用者にメールURLを閲覧させるのは好ましくない習慣）
② メールアドレスを入力後、トークン（確認番号）入力画面に遷移する　メールアドレス入力→トークン番号入力→その他の登録内容入力　（推奨）
（トークンは指定したメールアドレスにメール送信する）　（トークン番号・メール送信）

ユーザIDの重複

・ パスワードが違えば同じIDで登録できるサイト
・ ユーザIDに一意制約をつけられないサイト

ユーザIDの自動登録への対処

・ CAPTCHA　reCAPTCHA

パスワード変更

● パスワード変更時に、現在のパスワードを入力し確認する　①セッションハイジャック対策　②CSRF脆弱性対策
● パスワード変更時にはメールで通知する
● パスワード変更機能に生じやすい脆弱性　①SQLインジェクション脆弱性　②CSRF脆弱性

メールアドレス変更

メールアドレス変更時には、新旧両方のメールアドレスに通知します。
・ 旧アドレスに送るのは、不正変更では正規ユーザのアドレスは元アドレスになるから
・ 旧アドレスは既に受信できない場合があるので、受信確認までは行わない

パスワードリセット　パスワードリマインダ←現在のパスワードを教える　パスワードリセット←リセットしたものを教えるか、パスワード変更を利用者にさせる

● 管理者向けパスワードリセット機能　①問合せを受け、利用者の本人確認を行う　②パスワード変更だけができる仮パスワードをメールで発行
● 利用者向けパスワードリセット機能
①現在のパスワードをメールで通知　盗聴リスク
②パスワード変更画面のURLをメールで通知　フィッシング被害に会いやすい習慣
③ メールアドレス入力→スマホアプリ数値の2段階認証→登録メールアドレスに仮パスワードを送信
パスワード変更通知で不正利用された場合にも気が付く。
スマホアプリ数値の2段階認証によって、不正にパスワード無効されることを防ぐ。
仮パスワードと本番パスワードは別管理で、仮パスワード発行後も元パスワードでログインできるようにすべき。
④ メールアドレス入力→トークン確認番号を登録メールアドレスに送信→トークン確認番号を画面から入力→スマホアプリ数値の2段階認証→新しいパスワードを画面から入力
トークンのブルートフォース攻撃のために、回数制限を設け、アカウントロックするか、パスワードリセット機能を一定期間ロックする旨、メール通知します

53 認可

認可

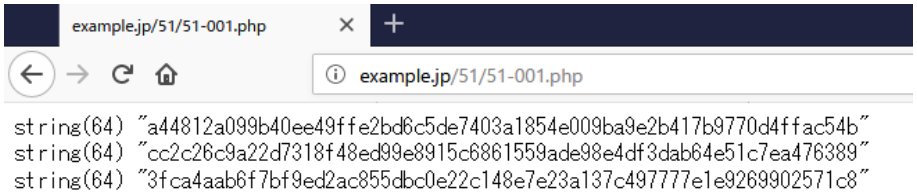
認可不備

情報リソースのURLを知っていると認証なしで情報を閲覧できる。
情報リソースのIDを変更すると権限外の情報が参照できる。
メニューの表示・非表示だけで制御している。
hiddenパラメータ、クッキーに権限情報を保持している。
⇒　URLやhiddenパラメータ、クッキーを書き換えると権限を不正利用できてしまう。

認可不備の対策

権限情報をセッション変数に保持することで、書き換えできないようにし、処理や表示の直前に権限の確認を行う。

51-001:パスワードのメッセージダイジェスト



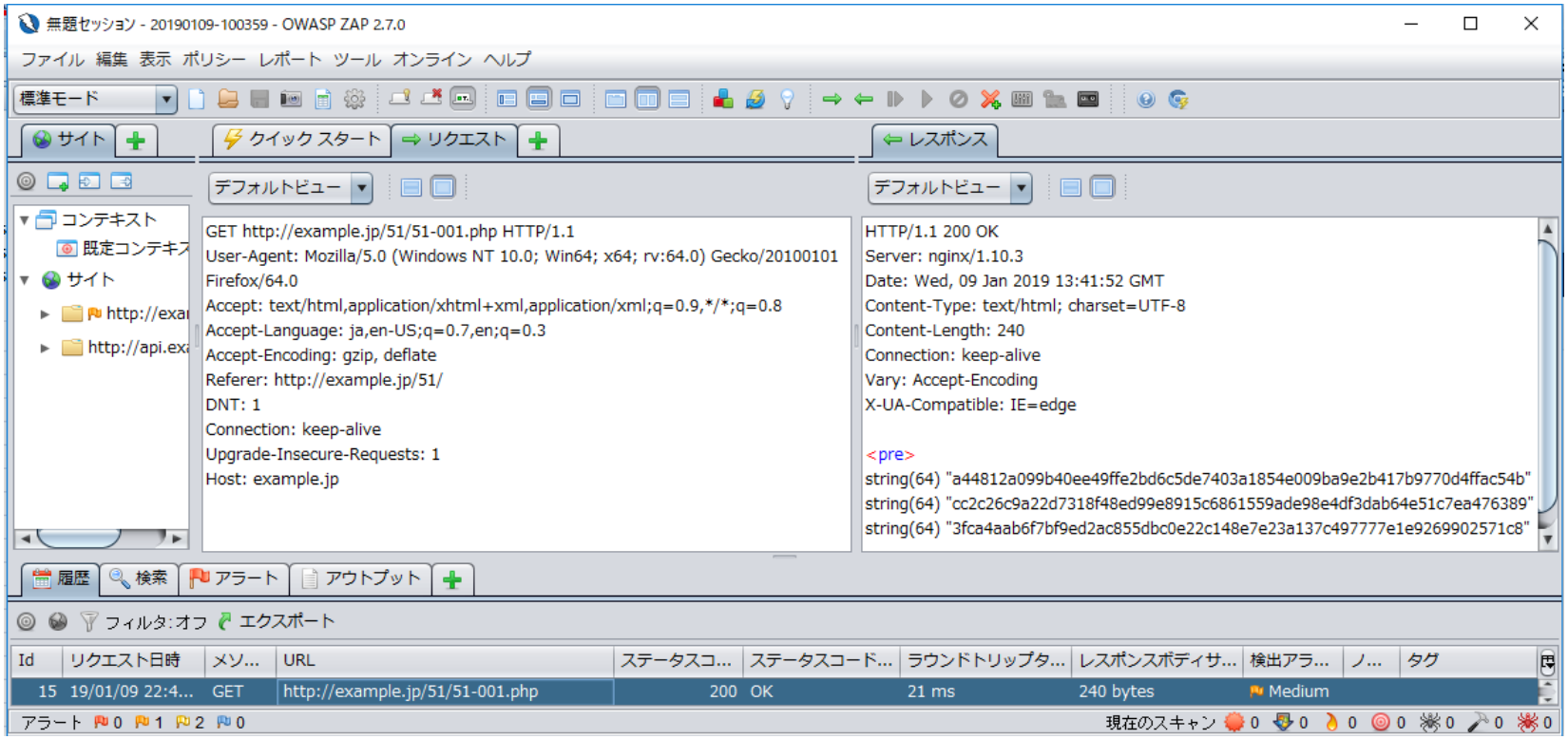
【サーバ: 51/51-001.php 】

/var/www/html/51/51-001.php - wasbook@example.jp - エディタ - WinSCP

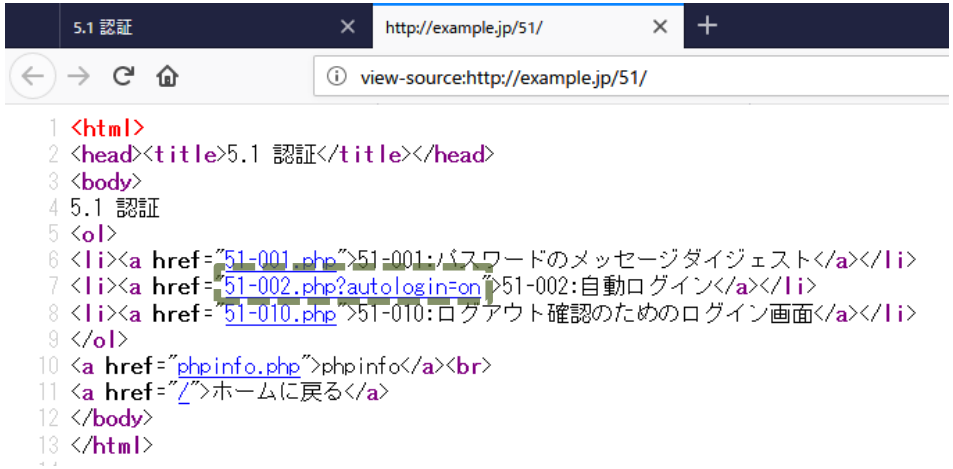
文字コード 色

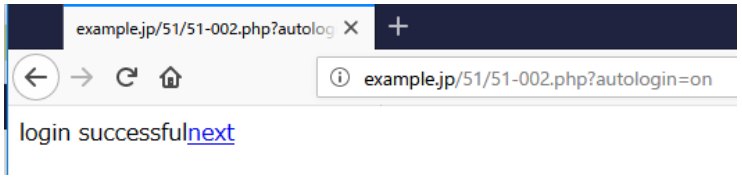
```
<pre>
<?php
  // FIXEDSALTはサイト毎に変更してください
  define('FIXEDSALT', 'bc578d1503b4602a590d8f8ce4a8e634a55bec0d');
  define('STRETCHCOUNT', 1000);

  // ソルトを生成する
  function get_salt($id) {
    return $id . pack('H*', FIXEDSALT);  // ユーザIDと固定文字列を連結
  }

  function get_password_hash($id, $pwd) {
    $salt = get_salt($id);
    $hash = '';  // ハッシュの初期値
    for ($i = 0; $i < STRETCHCOUNT; $i++) {
      $hash = hash('sha256', $hash . $pwd . $salt); //
    }
    return $hash;
  }
  // 呼び出し例
  var_dump(get_password_hash('user1', 'pass1'));
  var_dump(get_password_hash('user1', 'pass2'));
  var_dump(get_password_hash('user2', 'pass1'));
```

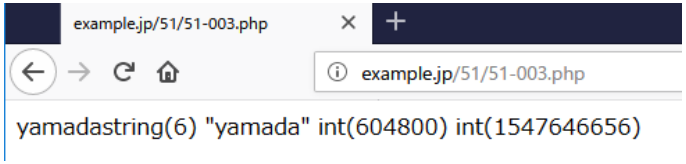
【ブラウザ→サーバ： リクエスト 51/51-001.php → レスポンス 】

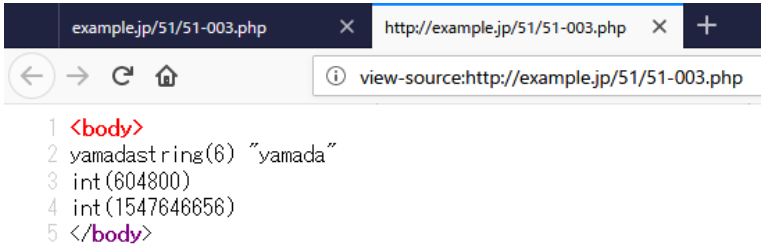


51-002:自動ログイン









【サーバ： 51/51-002.php 】

/var/www/html/51/51-002.php - wasbook@example.jp - エディタ - WinSCP

```php
<?php
  $autologin = (@$_GET['autologin'] === 'on');
  $timeout = 30 * 60;
  if ($autologin) {
    $timeout = 7 * 24 * 60 * 60;
    session_set_cookie_params($timeout);
  }
  session_start();
  session_regenerate_id(true);
  $_SESSION['id'] = 'yamada';   // ログイン機能がないので yamada で仮置き
  $_SESSION['timeout'] = $timeout;
  $_SESSION['expires'] = time() + $timeout;
?>
<body>
login successful<a href="51-003.php">next</a>
</body>
```

セッションの有効期間を1週間に
クッキーのExpires属性

本番ソースではログイン中ユーザIDセット
タイムアウト時間
タイムアウト期間

【サーバ： 51/51-003.php 】

/var/www/html/51/51-003.php - wasbook@example.jp - エディタ - WinSCP

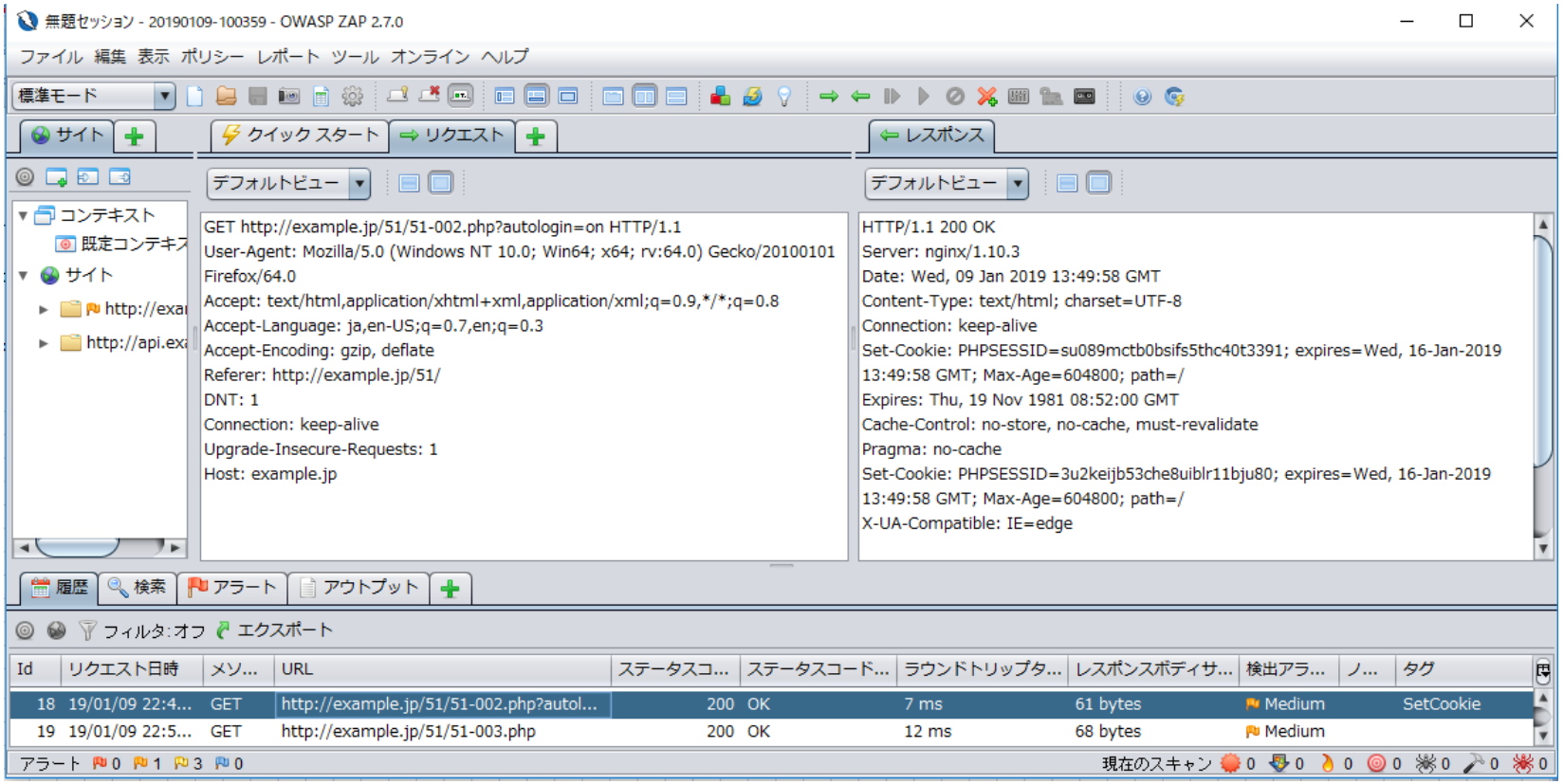
```php
<?php
  session_start();
  function islogin() {
    if (! isset($_SESSION['id'])) {
      return false;
    }
    if ($_SESSION['expires'] < time()) {
      session_destroy();
      return false;
    }
    $_SESSION['expires'] = time() + $_SESSION['timeout'];
    return true;
  }
  if (islogin()) {
    $id = $_SESSION['id'];
  } else {
    $id = '???';
  }
?>
<body>
<?php echo htmlspecialchars($id, ENT_NOQUOTES, 'UTF-8');  ?>
<?php
  var_dump($_SESSION['id']);
  var_dump($_SESSION['timeout']);
  var_dump($_SESSION['expires']);
?>
</body>
```

IDがセットされていない場合

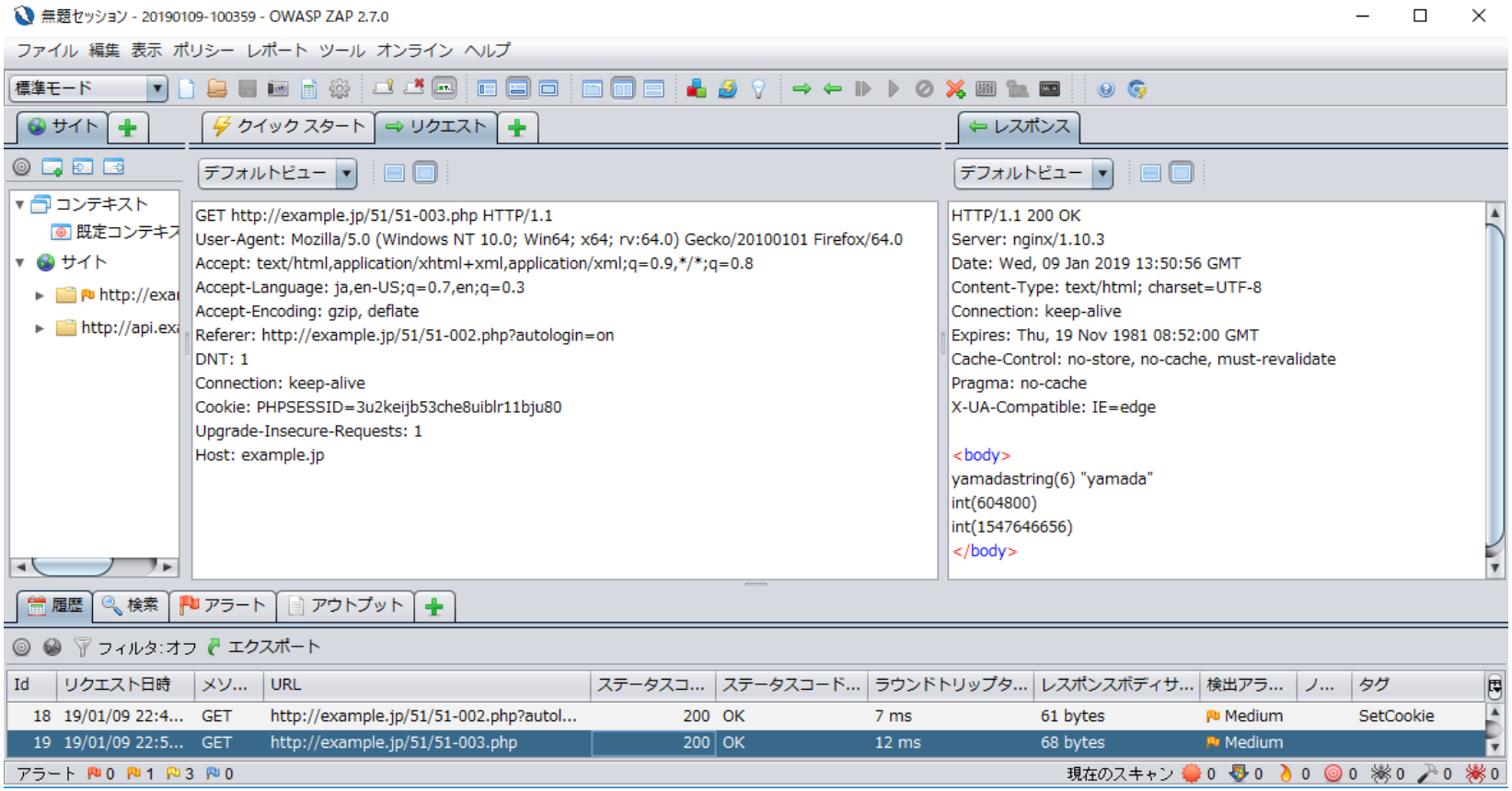
タイムアウトしている場合
セッション破棄

タイムアウト時刻を更新

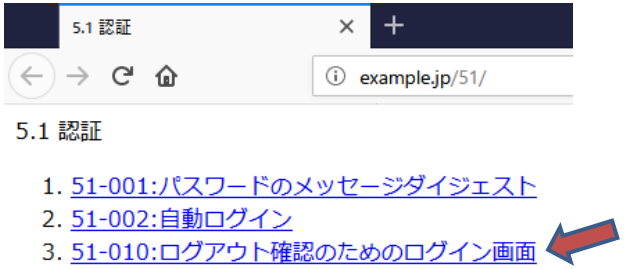
【ブラウザ→サーバ： リクエスト 51/51-002.php → レスポンス 】

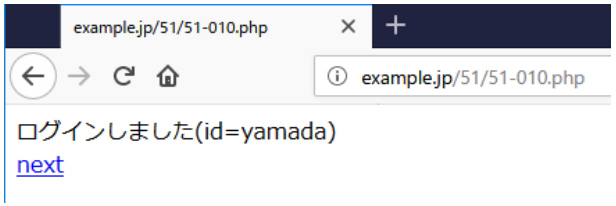


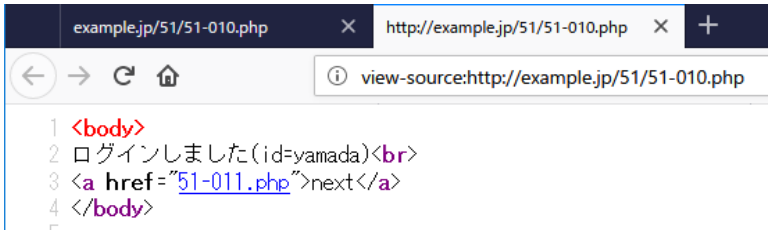
【ブラウザ→サーバ： リクエスト 51/51-003.php → レスポンス 】

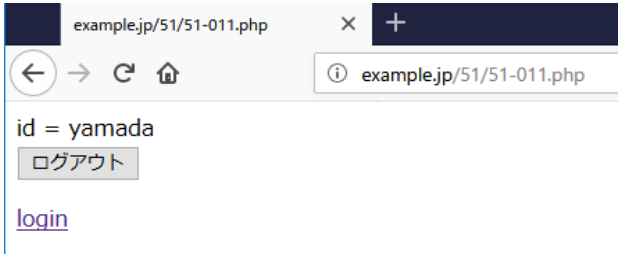


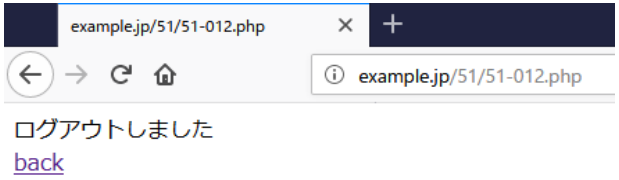
51-010:ログアウト確認のためのログイン画面

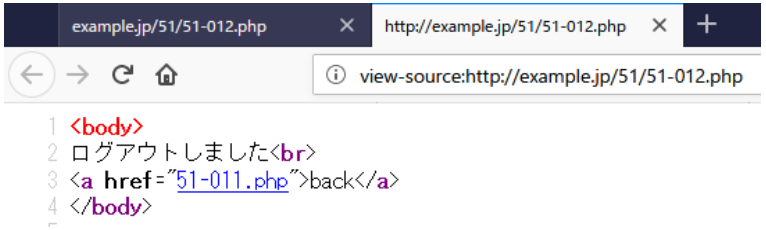


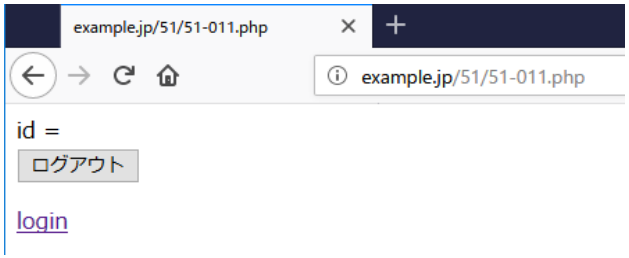




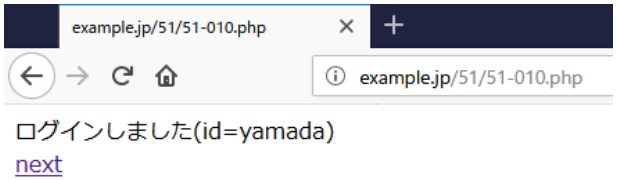


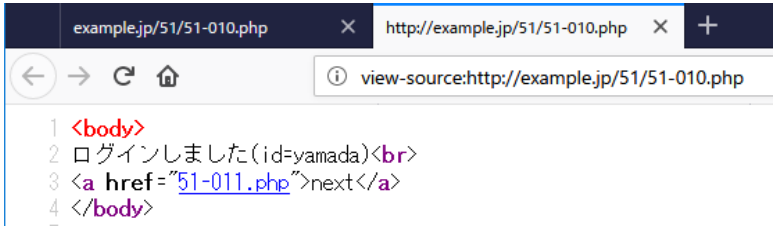

example.jp/51/51-012.php
example.jp/51/51-012.php
ログアウトしました
back


example.jp/51/51-012.php
http://example.jp/51/51-012.php
view-source:http://example.jp/51/51-012.php
<body>
ログアウトしました
back
</body>


example.jp/51/51-011.php
example.jp/51/51-011.php
id =
ログアウト
login


example.jp/51/51-011.php
http://example.jp/51/51-011.php
view-source:http://example.jp/51/51-011.php
<body>
id =
<form action="51-012.php" method="POST">
<!-- 以下はCSRF防止用トークン -->
<input type="hidden" name="token" value="f5992d2e71974de3d3b03721635dd491815fb6bf76773b25">
<input type="submit" value="ログアウト">
</form>
login
</body>


example.jp/51/51-010.php
example.jp/51/51-010.php
ログインしました(id=yamada)
next


example.jp/51/51-010.php
http://example.jp/51/51-010.php
view-source:http://example.jp/51/51-010.php
<body>
ログインしました(id=yamada)
next
</body>

【サーバ: 51/51-010.php 】

/var/www/html/51/51-010.php - wasbook@

```
<?php
  session_start();
  $_SESSION['id'] = 'yamada';
?>
<body>
ログインしました(id=yamada)<br>
<a href="51-011.php">next</a>
</body>
```

【サーバ: 51/51-011.php 】

/var/www/html/51/51-011.php - wasbook@example.jp - エディタ - WinSCP

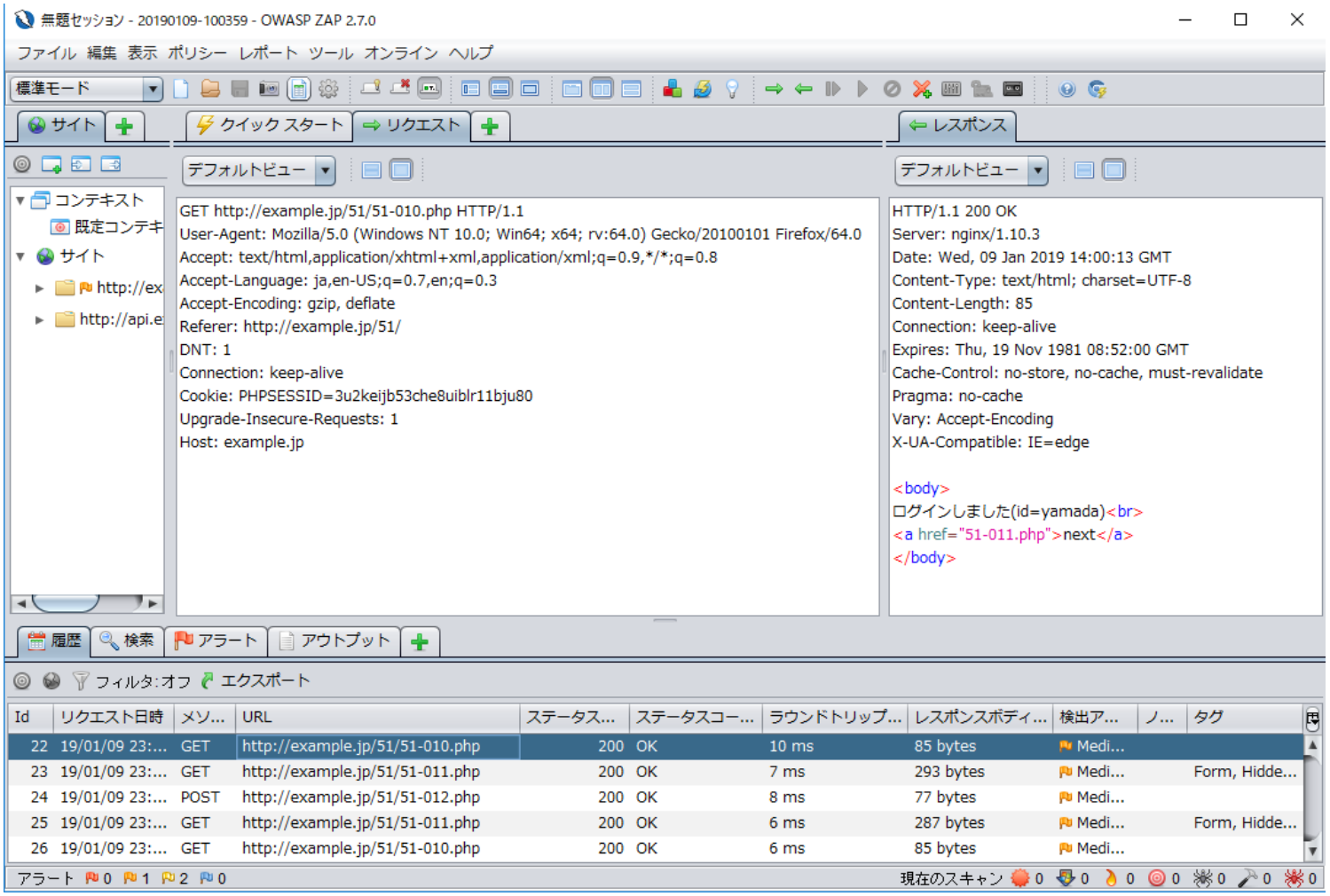
```
<?php
  session_start();
  $id = $_SESSION['id'];
  if (empty($_SESSION['token'])) {
    $token = bin2hex(openssl_random_pseudo_bytes(24));
    $_SESSION['token'] = $token;
  } else {
    $token = $_SESSION['token'];
  }
?>
<body>
id = <?php echo htmlspecialchars($id); ?><br>
<form action="51-012.php" method="POST">
<!-- 以下はCSRF防止用トークン -->
<input type="hidden" name="token" value="<?php echo
  htmlspecialchars($token); ?>">
<input type="submit" value="ログアウト">
</form>
<a href="51-010.php">login</a>
</body>
```

【サーバ: 51/51-012.php 】

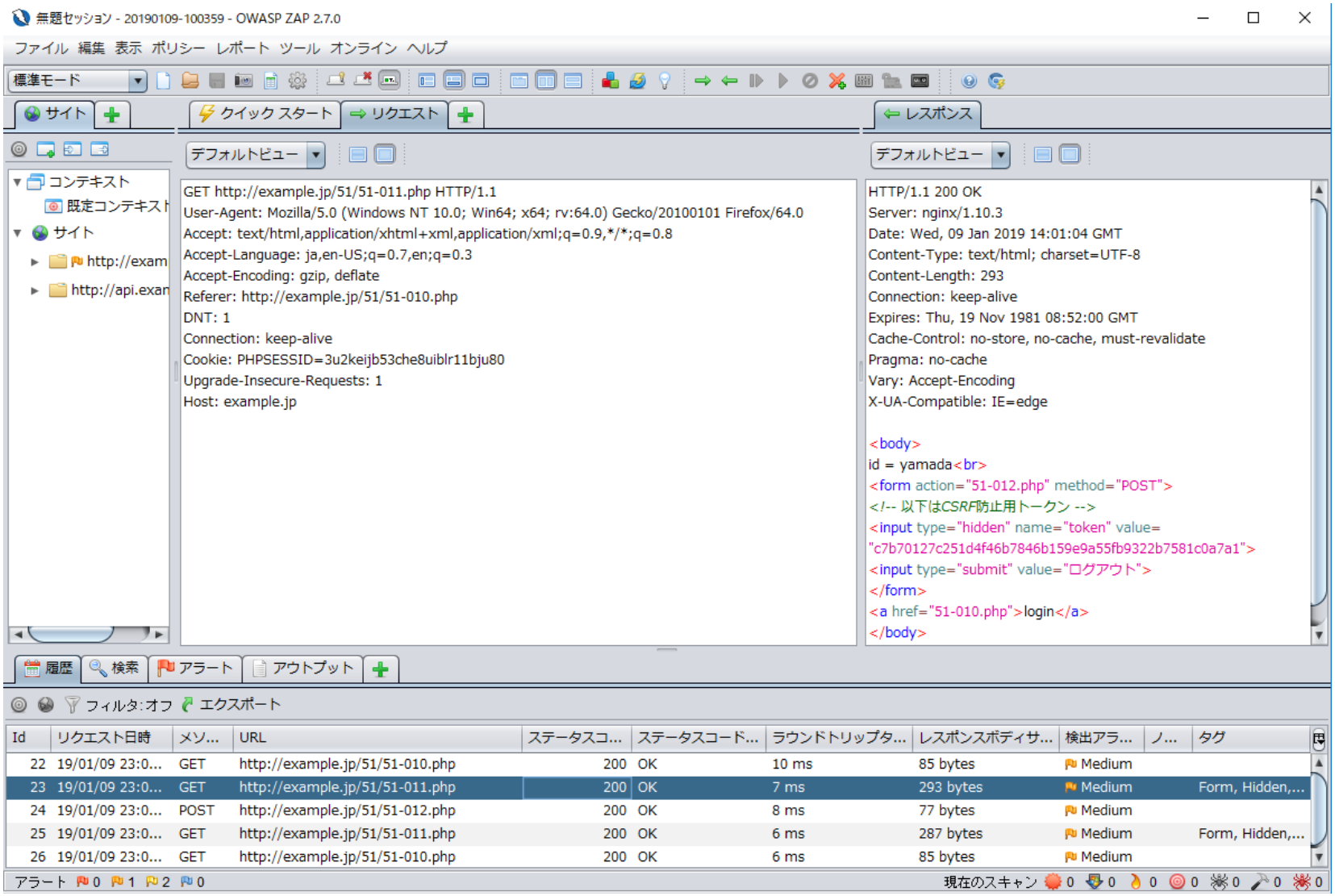
/var/www/html/51/51-012.php - wasbook@example.jp - エディタ - WinSCP

```
<?php
  session_start();
  $p_token = filter_input(INPUT_POST, 'token');
  $s_token = @$_SESSION['token'];
  if (empty($p_token) || $p_token !== $s_token) {
    die('ログアウトボタンからログアウトしてください');
  }
  $_SESSION = array();
  session_destroy();
?><body>
ログアウトしました<br>
<a href="51-011.php">back</a>
</body>
```

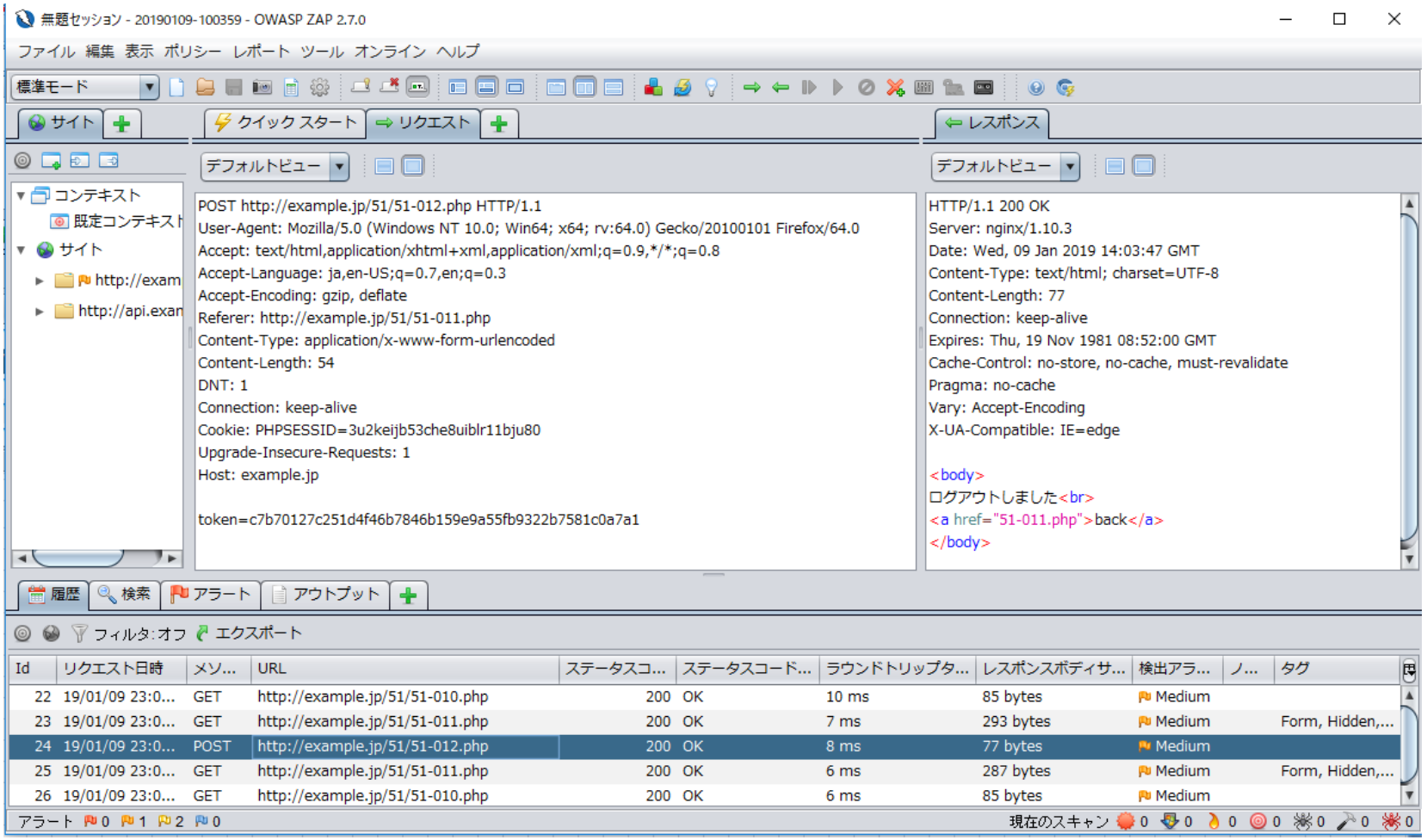
【ブラウザ→サーバ： リクエスト 51/51-010.php → レスポンス 】



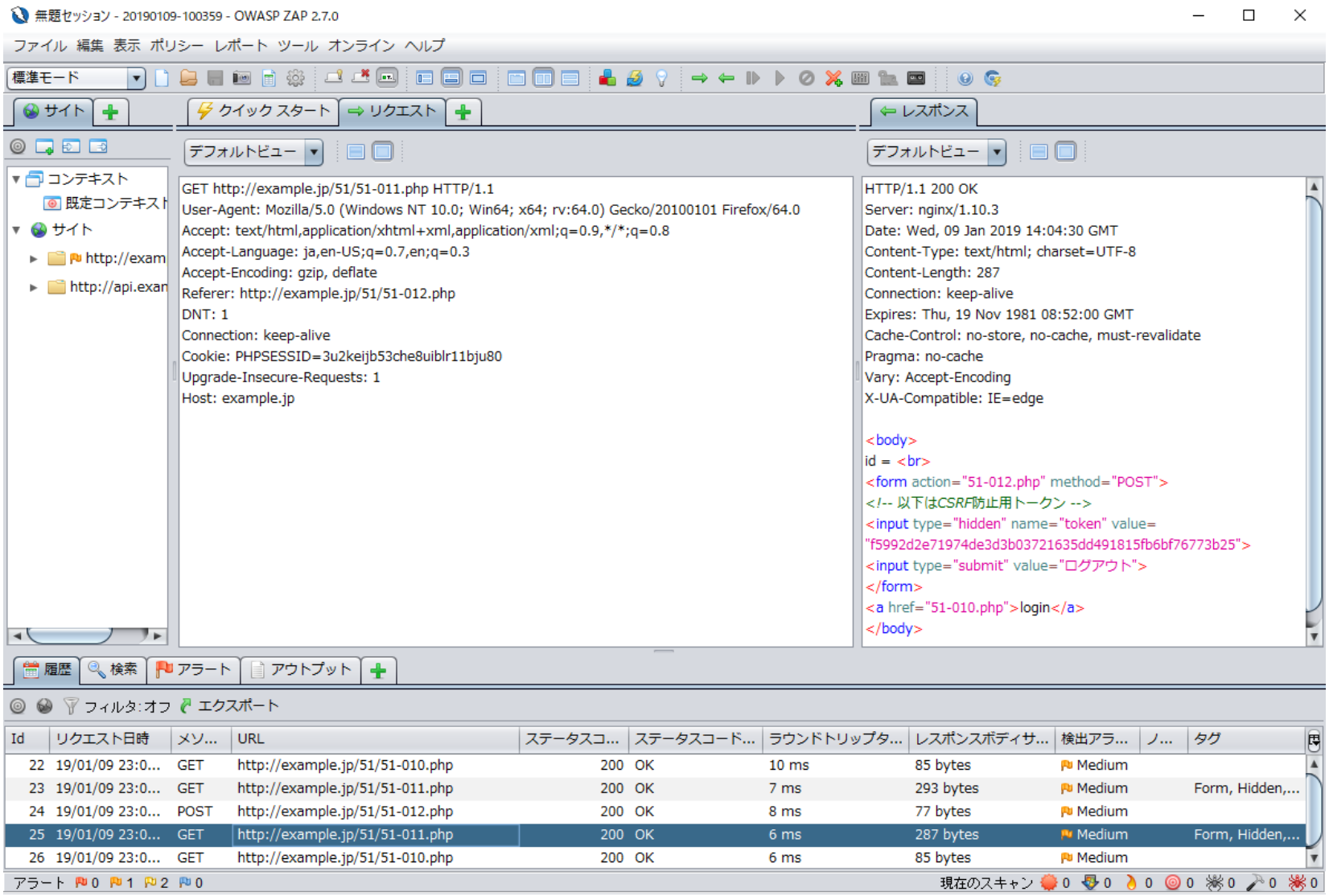
【ブラウザ→サーバ： リクエスト 51/51-011.php → レスポンス 】



【ブラウザ→サーバ： リクエスト 51/51-012.php → レスポンス 】



【ブラウザ→サーバ： リクエスト 51/51-011.php → レスポンス 】



【ブラウザ→サーバ： リクエスト 51/51-010.php → レスポンス 】