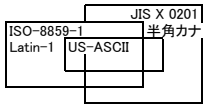
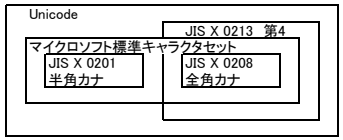
6. 文字コードとセキュリティ

文字集合

1バイト文字集合の包含関係



マルチバイト文字集合の包含関係



Unicode

当初16ビットですべての文字を表現する予定だったが、不十分で21ビットに拡張された。元の16ビット範囲をBMPと呼ぶ。(U＋XXXX or XXXXXX)

文字集合間の文字割り当ての違い

文字集合	0x5C	0xA5
ASCII	\	%
JIS X 0201	¥	・
ISO-8859-1	\	¥
Unicode	\	¥

¥(U＋0045)の状態で、エスケープ処理が行われた後で、バックスラッシュ(＼)に変換されると、エスケープ処理をすり抜けることになります。

文字エンコーディング

Shift－JIS　　ラリルレロの2バイト目が「宴」にマッチする

Shift－JISの各バイトの分布

	0	#	#	#	#	A0	C0	E0	FF
1バイト文字									
2バイト文字の先行バイト									
2バイト文字の後続バイト									

不正なShift－JIS

Shift－JISの先行バイトの後にデータがない場合(例:0x81)
Shift－JISの先行バイトに続くバイトが、後続バイトの範囲にない場合(例:0x81　0x21)

EUC－JP　　ラリルレロの4バイト目が「蛍」にマッチする

EUC－JPの各バイトの分布

	0	#	#	#	#	A0	C0	E0	FF
1バイト文字									
2バイト文字の先行バイト									
2バイト文字の後続バイト									

ISO－2022－JP

「ESC＄B」でJIS X 0208に切り替え、「ESC(B」でUS－ASCIIに切り替える。
「インターネットでは半角カナを使うな」というのは、ISO－2022－JPが半角カナ(JIS X 0201)をサポートしないことに由来します。

UTF－16

もともと16ビット内で設計されていたが、21ビットに拡張された。
UCS-2 は16ビットのスカラ値を扱う。
BMP外の文字をサポートする符号化方式が**UTF-16**
UTF-16でBMPをサポートするのが、サロゲートペアで、(0xD800～0xDBFF)、(0xDC00～0xDFFF)の2つの領域を予約し、組み合わせて2の20乗文字を表現する。

UTF－8

UTF-8のビットパターン

スカラ値の範囲	UTF-8ビットパターン				ビット長
0～7F	0xxxxxxx				7ビット
80～7FF	110 xxxxx	10 xxxxxx			11ビット
800～FFFF	1110 xxxx	10 xxxxxx	10 xxxxxx		16ビット
10000～10FFFF	11110 xxx	10 xxxxxx	10 xxxxxx	10 xxxxxx	21ビット

UTF-8の各バイトの配置

	0	#	#	#	#	A0	C0	E0	FF
1バイト文字									
2バイト文字の先行バイト									
2バイト文字の後続バイト									

文字の途中に別の文字がマッチする問題は起きません。

UTF-8の非最短形式の問題　　U+007F までの文字は、2バイト以上のビットパターンを当てはめて符号化できる。

「/」 の非最短形式による脆弱性

1バイト	0x2F			
2バイト	0xC0	0xAF		
3バイト	0xE0	0x80	0xAF	
4バイト	0xF0	0x80	0x80	0xAF

最短形式
非最短形式

UTF-8の非最短形式による脆弱性
- セキュリティチェックで、非最短形式　0xC0 0xAF がスラッシュ 0x2F でないと見なされる
- ファイル名などとして、非最短形式　0xC0 0xAF を使うと、スラッシュと見なされる
  ※ UTF-8の最新規格では、UTF-8の非最短形式は処理してはならないとされています

UTF-8の非最短形式を許していた処理系
- Java　SE6 Update 10以前のJRE（Java実行環境）
- PHP5．3．1以前の htmlspecialchars関数

その他の非正規なUTF-8
ISO/IEC 10646　Unicodeよりも広い31ビット空間で、UTF-8符号化後の1文字が最大6バイトだったが、
2006年改訂で、実質的にUnicodeと同じものとなり、UTF-8符号化後の1文字も最大4バイトとなった。

PHPの文字エンコーディング関数 mb_check_encoding
- PHP5.3　まではUTF-8の5バイト以上の形式を正しいエンコーディングと判断
- PHP5.4　以上は間違いと判断

サロゲートペア領域の文字の変換
- 正しい変換　：　①まず、UTF-32に変換　⇒　次にUTF－8に変換　⇒　結果4バイト
- 機械的に UTF-8 に変換すると、3バイトになる　　※　このような文字エンコーディングに　CESU-8　がある　（oracleDBの内部処理で使用）
- PHPの文字エンコーディング関数 mb_check_encoding
  ① PHP5.2　では正しいと判断
  ② PHP5.3.0　以降では間違いと判断

文字コードを正しく扱うために

● Java と ASP.NET は入力時に文字エンコーディングの変換処理が入るので、チェックされ、不正な文字は代替文字（U+FFFD）に変換される。
● Perl （5.8以降）の場合は、decode関数による内部形式への変換時に、不正文字エンコーディングは代替文字に変換されます。
● PHPの場合には、文字エンコーディングが自動的にチェックされないので、mb_check_encoding関数により、チェックします。

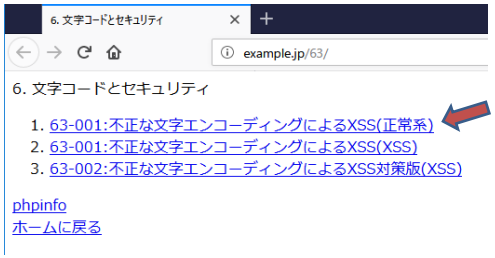
● マルチバイト文字の扱いは
Java、.NET、Perl（5.8以降）は問題ないが、PHPはマルチバイト文字に対応していないので、以下に従う必要があります。
①ソースコードはUTF-8で保存
② php.ini　の default_charset （PHP5.6以降） あるいは mbstring.internal_encoding （PHP5.5以前）を、UTF-8と設定する
③文字列処理は原則 mbstring系の関数を使う
④関数の引数として、正しく文字エンコーディングを指定する

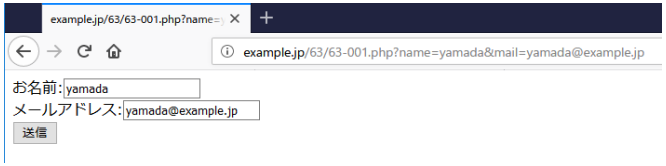
● HTTPレスポンスの　Content-Type　を正しく指定する

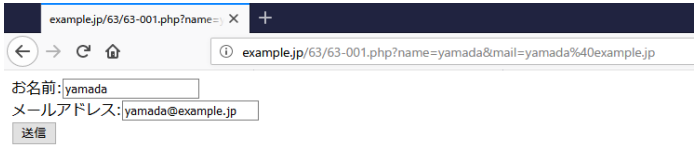
● データベースの文字エンコーディングを正しく指定する
① MySQLでは、「utf8」 は3バイトまでの形式で、BMP外の文字を含む場合は、「utf8mb4」 と指定する。
② 「𠮟咤」「吉」をデータベースに登録して、表示が崩れたら、Unicodeでの一貫した処理ではなく、途中にShift－JIS、EUC-JPの部分がある。

63-001:不正な文字エンコーディングによるXSS(正常系)

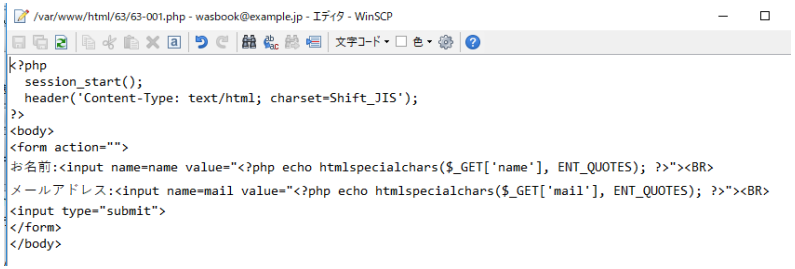
【ブラウザ】



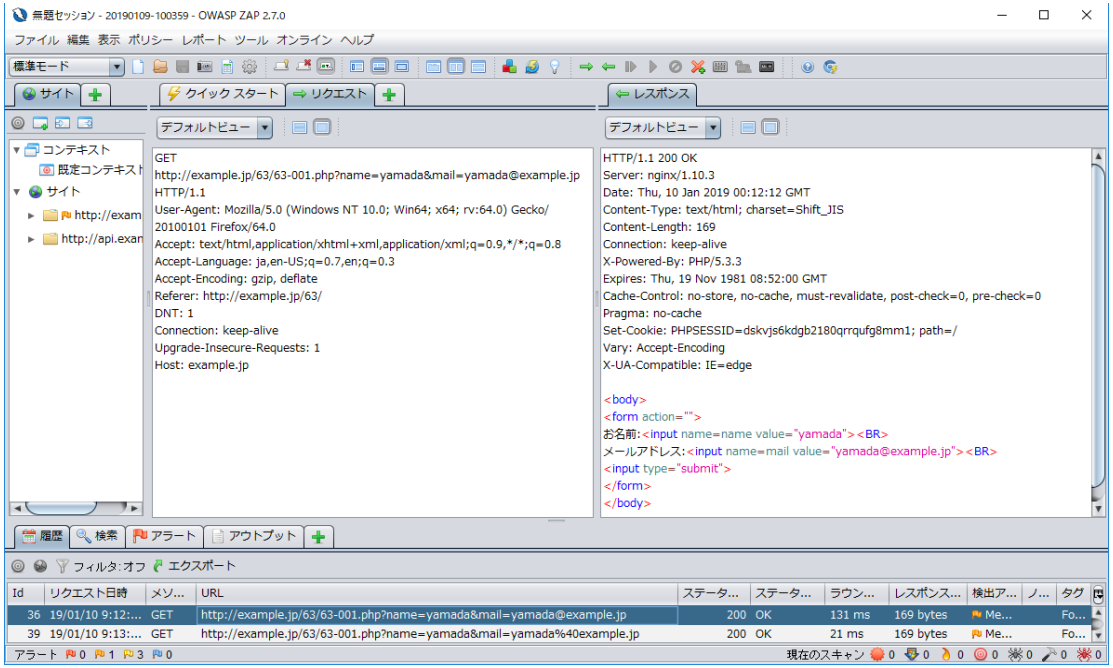




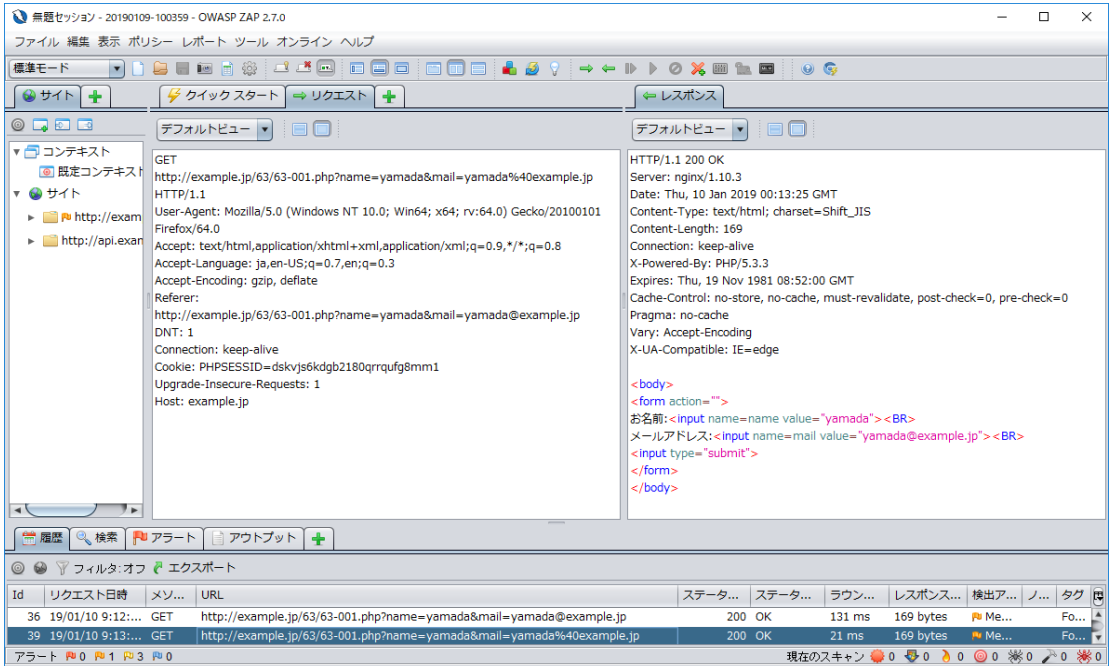
【サーバ: 63/63-001.php 】



【ブラウザ→サーバ： リクエスト 63/63-001.php → レスポンス 】（初期フォーム表示）

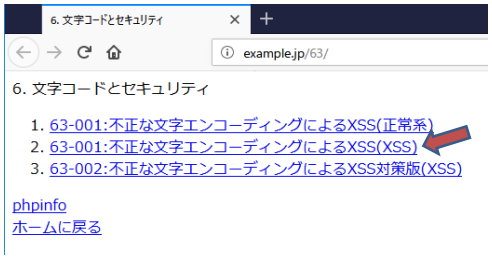


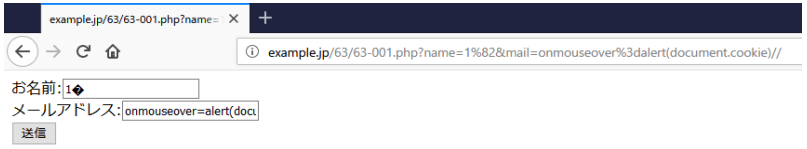
【ブラウザ→サーバ： リクエスト 63/63-001.php → レスポンス 】（送信クリック）

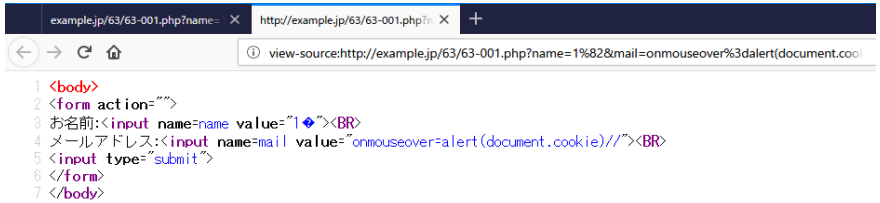


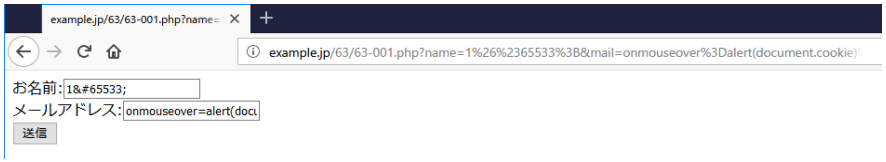
63-001:不正な文字エンコーディングによるXSS(XSS)

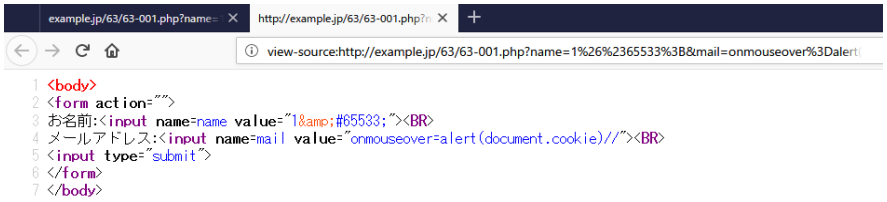
【ブラウザ】



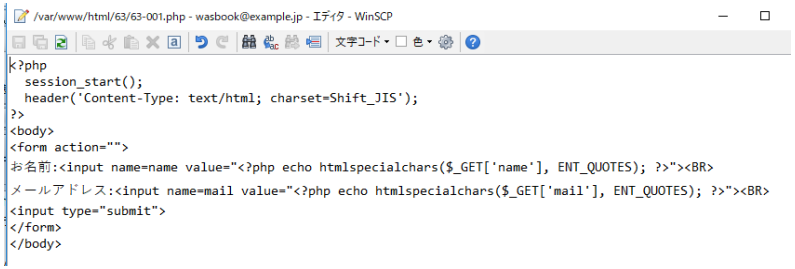




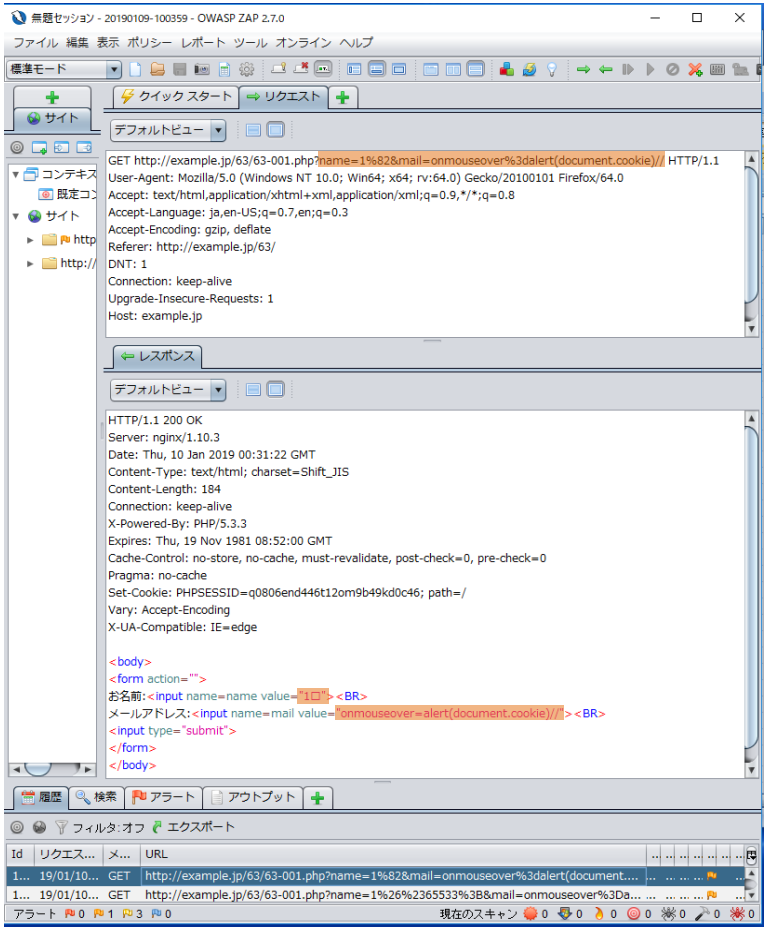




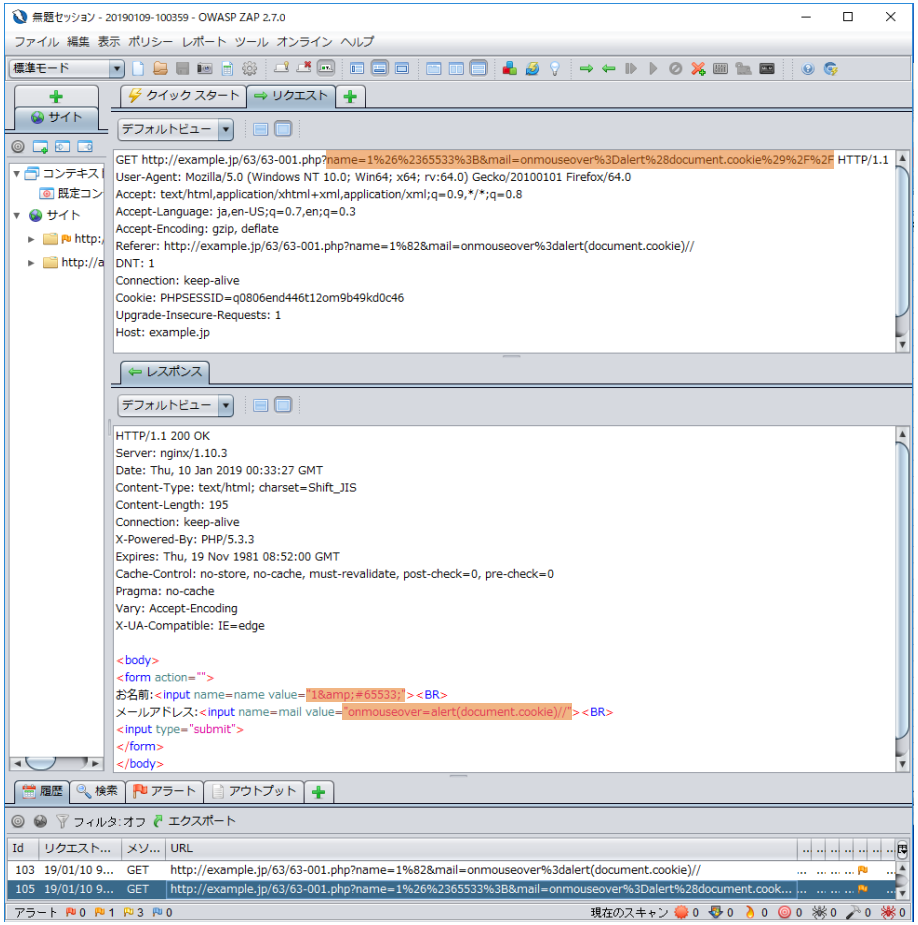
【サーバ: 63/63-001.php 】



【ブラウザ→サーバ： リクエスト 63/63-001.php → レスポンス 】（初期フォーム表示）

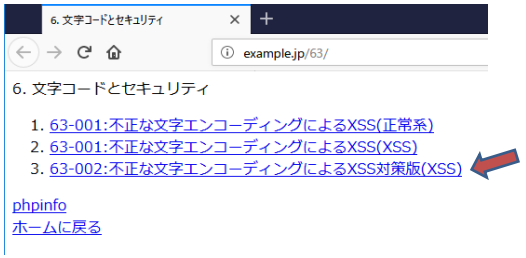


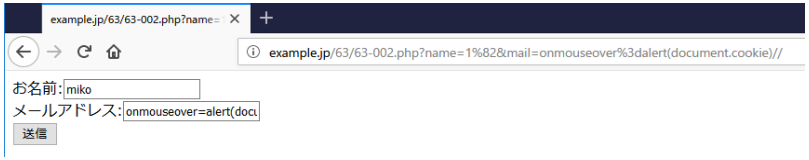
【ブラウザ→サーバ： リクエスト 63/63-001.php → レスポンス 】（送信クリック）

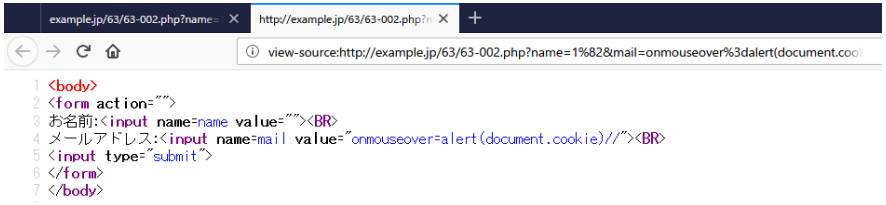


63-002:不正な文字エンコーディングによるXSS対策版(XSS)

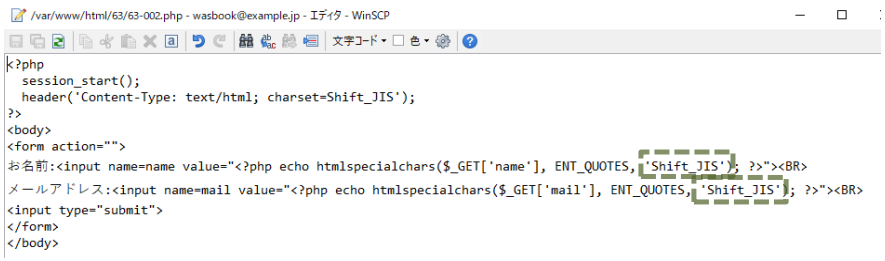
【ブラウザ】



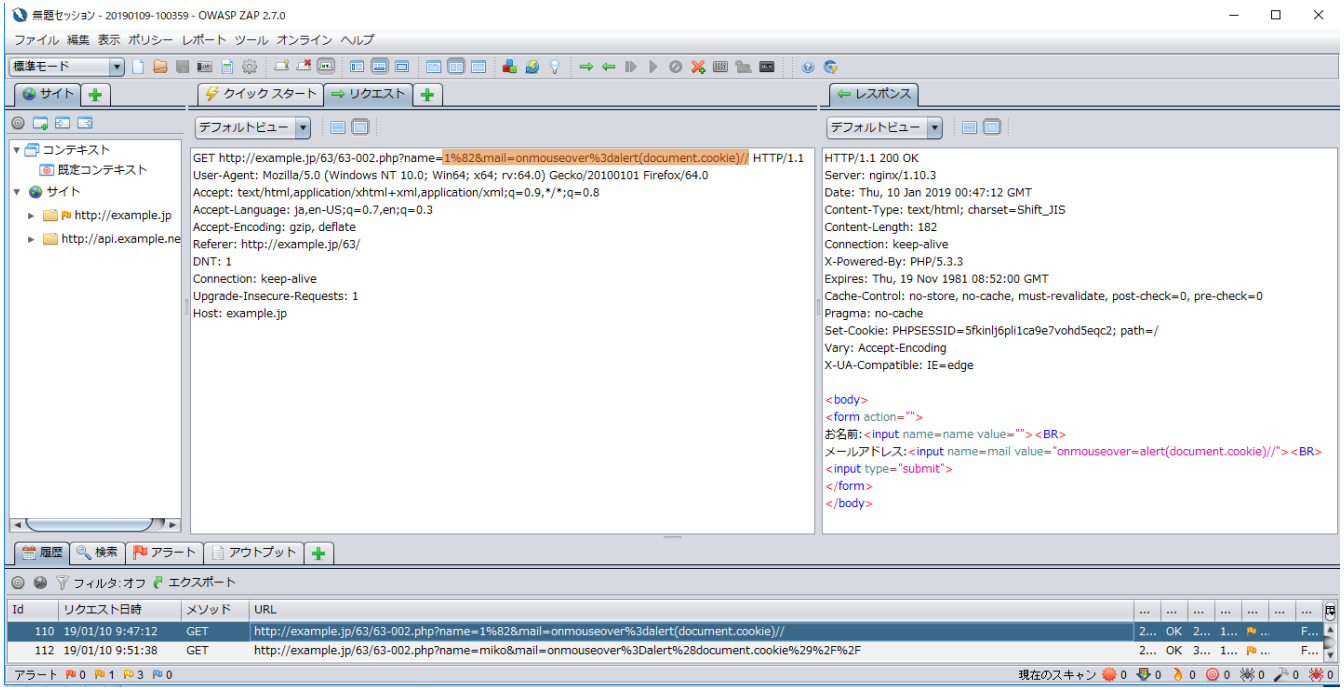




【サーバ: 63/63-001.php 】



【ブラウザ→サーバ： リクエスト 63/63-002.php → レスポンス 】（初期フォーム表示）



【ブラウザ→サーバ： リクエスト 63/63-002.php → レスポンス 】（送信クリック）