Monetra® Payment Software

POST Protocol Specification

Revision: 1.3.1
Publication date May 04, 2018

Copyright © 2018 Main Street Softworks, Inc.

POST Protocol Specification

Main Street Softworks, Inc.

Revision: 1.3.1

Publication date May 04, 2018
Copyright © 2018 Main Street Softworks, Inc.

Legal Notice

The information contained herein is provided *As Is* without warranty of any kind, express or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. There is no warranty that the information or the use thereof does not infringe a patent, trademark, copyright, or trade secret.

Main Street Softworks, Inc. shall not be liable for any direct, special, incidental, or consequential damages resulting from the use of any information contained herein, whether resulting from breach of contract, breach of warranty, negligence, or otherwise, even if Main Street has been advised of the possibility of such damages. Main Street reserves the right to make changes to the information contained herein at anytime without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Main Street Softworks, Inc.

Table of Contents

1 Monetra POST System

1.1 POST Overview

The Monetra POST feature enables web-based and mobile applications to perform transactions by sending sensitive card data directly to a Monetra server instead of routing card data through the merchant's application server. This approach can significantly reduce the scope of PCI compliance requirements for the merchant and the application developer..

Monetra currently supports two different methods for performing Indirect Transaction Requests:

POSTBACK: The Postback method uses a server redirect mechanism. The merchant's application generates a web form that instructs the client's browser to post the payment request directly to a Monetra server. The Monetra server then notifies the merchant's website, via a defined postback URL, of the payment status and redirects the client's browser back to the merchant's website.

CLIENT TICKET REQUEST: The Client Ticket Request method uses an out-of-band 'ticket' request. The client's browser sends the sensitive card data directly to a Monetra server and receives back a special response, called a 'ticket,' that is associated with the card data that was provided. The client's browser then passes that ticket back to the merchant's application server, which sends the ticket to the Monetra server to perform a transaction using the associated card data.

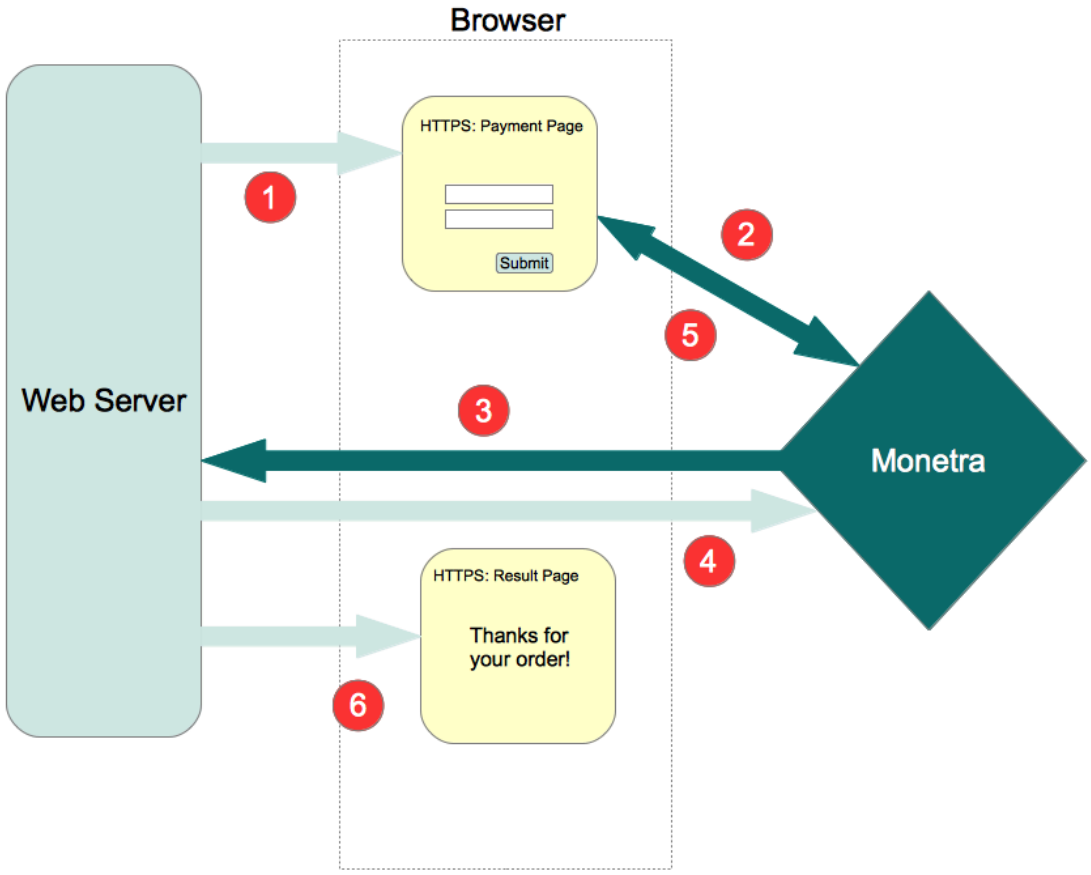
Both methods utilize secure HMACs with dual authentication to prevent known attack vectors including forgery and replay attacks.

Code Examples can be found in Appendix A

Copyright © 2018 Main Street Softworks, Inc. | POST Protocol Specification | CONFIDENTIAL

1.1.1 Postback Method Process Flow

Figure 1.1.



1. The merchant's website generates a payment form containing hidden form elements as specified, with 'action' pointing directly to the target Monetra server.

2. The client's browser POSTs the merchant-generated payment form to the Monetra server.

3. The Monetra server sends the corresponding transaction to the merchant's acquirer/ processing institution, receives the processor's response, and POSTs that response to the merchant's web server using the specified postback URL.

4. The Monetra server receives an acknowledgment response from the merchant's web server

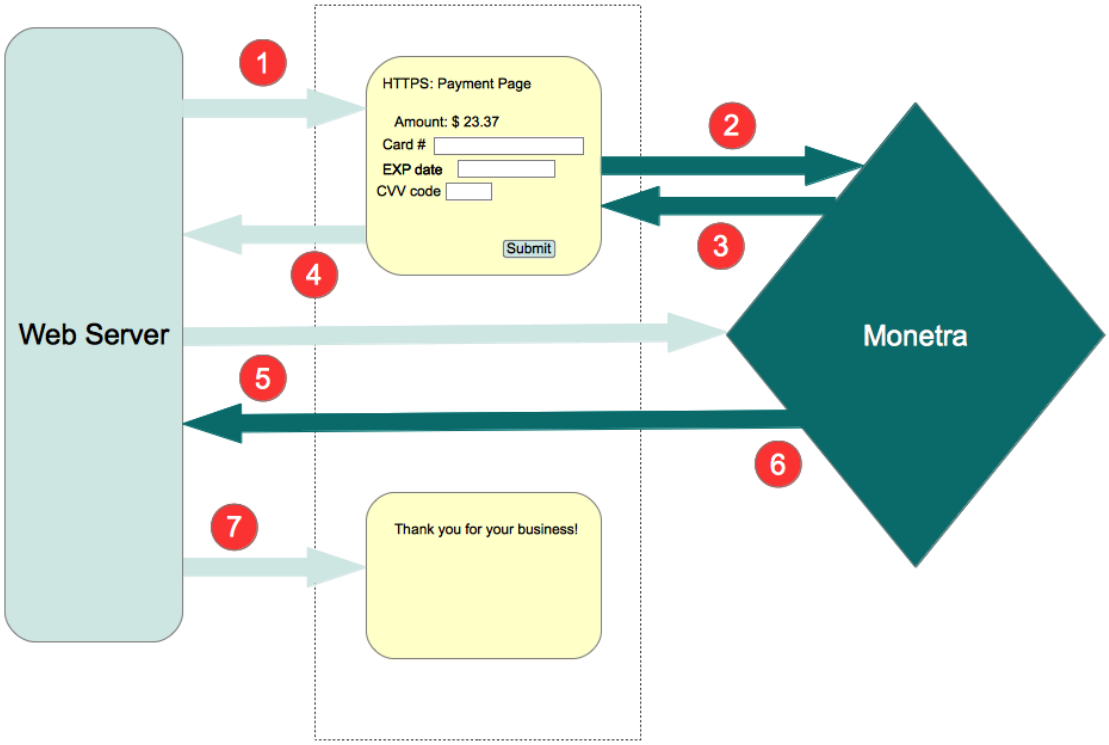
5. The Monetra server then sends a reply to the client's browser containing a redirect to the redirect URL provided in the payment form data. The redirect first attempts JavaScript, but falls back to a meta refresh if JavaScript is disabled.

Copyright © 2018 Main Street Softworks, Inc. | POST Protocol Specification |
CONFIDENTIAL

6. The client browser performs the redirect and sends a page request to the redirect URL. The Merchant's website then sends the client browser a reply/response page based on the postback data that was received from the Monetra server.

1.1.2 Client Ticket Request Method Process Flow

Figure 1.2.



1. The merchant's website generates a payment form containing hidden form elements as documented along with the sensitive card data entry fields. The form's "action" points at the merchant's own webserver as if it was to receive the sensitive cardholder data directly. .

2. The client submits the payment form. Merchant-provided Javascript will intercept the submission and first post the sensitive data directly to the Monetra server.

3. The Monetra server stores the sensitive card data and returns an associated 'ticket' to the client's browser.

4. The Merchant-provided javascript then inserts the 'ticket' response as a hidden form element and removes the sensitive data fields from the form then continues the standard form post action to the merchant's website.

5. The merchant's web server submits a transaction request to the Monetra server using the 'ticket' in place of sensitive card data.

Copyright © 2018 Main Street Softworks, Inc. | POST Protocol Specification |
CONFIDENTIAL

6. The Monetra server sends the corresponding transaction to the merchant's acquirer/ processing institution, receives the processor's response, and sends that response to merchant's web server.

7. The Merchant's web server then sends the client browser a reply/response page based on that response.

1.2 Transaction Structures

When using the Indirect Transaction Request feature, the data sent to a Monetra server via the client forms is standard url-encoded form data, and the key/value pairs are the same as documented in the Monetra Client Interface Protocol Specification, with only minimal exceptions. One such exception is that the Monetra User password must NOT be sent. Instead, HMAC is used for authentication, as described in the following sections.

1.2.1 Postback Structures

1.2.1.1 Postback Request

INPUT PARAMETER		NOTES
`username`	=	Username used for authentication by the Monetra server. This must be a sub user with 'obscure sensitive information' enabled.
`action`	=	Monetra action to perform such as SALE.
`amount`	=	Required. This is the transaction amount being requested.
`monetra_req_sequence`	=	Merchant-specified sequence number. May be alphanumeric. Should be stored for response validation.
`monetra_req_timestamp`	=	Standard unix timestamp. Must be within 15 minutes of server's time. Should be stored for response validation.
`monetra_req_fields`	=	This is a comma (',') separated list of keys that must be present in the post request. Missing fields will cause a rejection. For example, the iframe uses the following `'account, expdate, cardholdername, cv, street, zip'`. `'monetra_req_fields'` is part of the hmac and comes after `'monetra_req_timestamp'`.

`monetra_req_hmacsha256`	=	Generated HMAC-SHA256 using the merchant's password as the key. The message is the concatenation of the values contained within these fields with no delimiters in this exact order:
		`username`
		`action`
		`amount`
		`monetra_req_sequence`
		`monetra_req_timestamp`

		monetra_req_fields (if provided) monetra_url_postback monetra_url_redirect
monetra_url_postback	=	URL to post response to. There is a 30s timeout, so this must not trigger a long-running process.
monetra_url_redirect	=	URL to send client's browser to when complete.

Note: There are no 'echo' fields between the request and the response. The merchant should use the query string of the postback and redirect urls to pass back any unique order id used for tracking such as:
monetra_url_postback=https://www.mywebsite.com/order_handler.php?id=12345
monetra_url_redirect=https://www.mywebsite.com/receipt.php?id=12345

1.2.1.2 Postback Response Parameters

The response sent to the 'postback' URL will be standard application/x-www-form-urlencoded data using the same response key/value pairs as documented in the Monetra Client Interface Protocol Specification. However, there will be one additional response parameter:

RESPONSE PARAMETER		NOTES
monetra_resp_hmacsha256	=	This is a validation response parameter which the merchant MUST validate, otherwise a rogue customer could post a fake response to the post-back URL. It is a HMAC-SHA256 just like sent in the request, using the merchant's password as the key. The message is the concatenation of the values contained within these fields with no delimiters in this exact order: username action amount monetra_req_sequence monetra_req_timestamp code (from response) ttid (from response)

1.2.2 Ticket Structures

The Client Ticket Request method serves the same purpose as the Postback method, which is to eliminate the need for the merchant's web server to receive, process, or store sensitive card data as defined by the PCI Data Security Standard.

The Client Ticket Request method differs from the Postback method in that the Monetra server does not initiate a connection directly with the merchant's web server, and instead relies on the client to relay a 'ticket' to the merchant's web server, providing a reference to the sensitive card data that was previously sent to the Monetra server directly from the client's web browser.

Copyright © 2018 Main Street Softworks, Inc. | POST Protocol Specification | CONFIDENTIAL

In this method, the merchant's web server must also initiate a direct connection to the Monetra server, so two requests must be sent to the Monetra server to complete a payment, one from the client's web browser (containing the sensitive card data) and the other from the merchant's web server (to perform a transaction using the ticket associated with that card data).

Note: For security reasons it is advised to create a separate Monetra sub-user for performing these operations, with only 'ticketrequest' permissions.

1.2.2.1 Ticket Request

INPUT PARAMETER		NOTES
`username`	=	Username used for authentication in Monetra. This must be a sub user with 'obscure sensitive information' enabled.
`action`	=	'ADMIN'
`admin`	=	'cardshieldticket'
`monetra_req_sequence`	=	Merchant-specified sequence number. May be alphanumeric. Should be stored for response validation.
`monetra_req_timestamp`	=	Standard unix timestamp. Must be within 15 minutes of server's time. Should be stored for response validation.
`monetra_req_fields`	=	This is a comma (',') separated list of keys that must be present in the post request. Missing fields will cause a rejection. For example, the iframe uses the following `'account, expdate, cardholdername, cv, street, zip'`. `'monetra_req_fields'` is part of the hmac and comes after `'monetra_req_timestamp'`.

`monetra_req_hmacsha256`	=	Generated HMAC-SHA256 using the merchant's password as the key. The message is the concatenation of the values contained within these fields with no delimiters in this exact order:
		`username`
		`action`
		`admin`
		`monetra_req_sequence`
		`monetra_req_timestamp`
		`monetra_req_fields (if provided)`

1.2.2.2 Client Ticket Response Parameters

When performing a Client Ticket request, the response will be sent to the requestor in XML format and the postback/redirect URLs will not be used. The XML format used for the response is the same format found in the Monetra XML Protocol Specification.

RESPONSE PARAMETER		NOTES
`monetra_resp_hmacsha256`	=	This is a validation response parameter which the merchant MUST validate, otherwise a rogue customer

		could post a fake response to the post-back URL. It is a HMAC-SHA256 just like sent in the request, using the merchant's password as the key. The message is the concatenation of the values contained within these fields with no delimiters in this exact order:
		`username`
		`monetra_req_sequence`
		`monetra_req_timestamp`
		`ticket (from response)`

Note: In addition to the HMAC authentication value (as previously described), the following parameters may be sent in a Client Ticket request::

```
action=admin
admin=cardshieldticket
trackdata=(either this or account + expdate must exist)
account=(either this plus expdate or only trackdata must exist)
expdate=(either this plus account or only trackdata must exist)
street=(optional)
zip=(optional)
cvv2=(optional)
pin=(optional)
ksn=(optional)
```

The Ticket sent from a Monetra server in response to a Client Ticket request is valid for 5 minutes and is intended to be sent in place of the sensitive card data with a transaction request from the merchant's web server to the Monetra server.

1.3 HMAC Authentication

1.3.1 HMAC Overview

HMAC: Keyed-Hashing for Message Authentication as described by RFC-2104

HMAC provides a way to check the integrity of information transmitted over or stored in an unreliable medium is a prime necessity in the world of open computing and communications. Mechanisms that provide such integrity check based on a secret key are usually called "message authentication codes" (MAC). Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties.

Equation 1.1. HMAC Mathmatical Definition

HMAC (K,m) = H ((K $\oplus$ opad) # H ((K $\oplus$ ipad) # m))

Where:
H is a cryptographic hash function,
K is a secret key padded to the right with extra zeros to the input block size of the hash function, or the hash of the original key if it's longer than that block size,

m is the message to be authenticated,
# denotes concatenation,
⊕ denotes exclusive or (XOR),
opad is the outer padding (0x5c5c5c…5c5c, one-block-long hexadecimal constant),
and ipad is the inner padding (0x363636…3636, one-block-long hexadecimal constant).

For more information please see RFC 2104
http://www.ietf.org/rfc/rfc2104.txt

Also, Wikipedia has a nice overview here:
http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

Copyright © 2018 Main Street Softworks, Inc. | POST Protocol Specification |
CONFIDENTIAL

A POST Code Examples

A.1 Code Examples

A.1.1 Example POST using Postback (PHP)

```php
<?php
/* This tests the MONETRA POST (PostBack) implementation, and does
 * it via a single simple PHP page.
 * URLs used are:
 *   /                                  (base URL, new order request)
 *   /?method=postback&order_id=XXXX    (postback URL)
 *   /?method=response&order_id=XXXX    (response URL)
 */

error_reporting(E_ALL);

/* == Some basic configuration variables specific to our environment == */

/* Path to SQLite database holding order results */
$db_path     = "test.sqlite";

/* Location of Monetra to post payment request to */
$monetra_url = "https://testbox.monetra.com:8444/";

/* Authentication information for Monetra */
$username    = "test_ecomm:public";
$password    = "publ1ct3st";


/* == Helper functions == */

function get_data($order_id, $table)
{
 global $db_path;
 $rows = array();
 $out  = array();

 try {
  $dbh = new PDO("sqlite:$db_path");
  $stmt = $dbh->prepare("SELECT key,val FROM $table WHERE id=?");
  if (!$stmt->execute(array($order_id))) {
   echo 'SQL Query failed execute';
   return false;
  }
  $rows = $stmt->fetchAll();
 } catch (PDOException $e) {
  echo 'SQL Query failed: ' . $e->getMessage();
  return false;
 }

 /* Transform into a dictionary */
 for ($i=0;$i<count($rows);$i++) {
```

```
  48    $out[$rows[$i][0]] = $rows[$i][1];
  49   }
  50
  51   return $out;
  52  }
  53
  54
  55  /*! Retrieve order results from the database
  56   *  \param order_id Order id being queried
  57   *  \return an array of key/value pairs containing the result data
  58   *          from Monetra */
  59  function get_order_results($order_id)
  60  {
  61   return get_data($order_id, "results");
  62  }
  63
  64
  65  /*! Retrieve order request details from the database
  66   *  \param order_id Order id being queried
  67   *  \return an array of key/value pairs containing the request
  68   *          data stored for response verification */
  69  function get_order_request($order_id)
  70  {
  71   return get_data($order_id, "request");
  72  }
  73
  74
  75
  76  /*! Check to see the existance of an SQL table in the database
  77   *  handle provided.
  78   *  \param dbh   Open database handle
  79   *  \param table name of table checking existence of
  80   *  \return true if table exists, otherwise false */
  81  function sql_check_table($dbh, $table)
  82  {
  83   if ($dbh->exec("SELECT 1 FROM $table") === false)
  84    return false;
  85   return true;
  86  }
  87
  88
  89  function insert_order($data, $table, $order_id)
  90  {
  91   global $db_path;
  92
  93   try {
  94    $dbh = new PDO("sqlite:$db_path");
  95   } catch (PDOException $e) {
  96    echo 'SQL Query failed: ' . $e->getMessage();
  97    return false;
  98   }
  99
 100   /* Create the table if it doesn't exist */
 101   if (!sql_check_table($dbh, $table)) {
 102    $query = "CREATE TABLE $table (id INTEGER, key TEXT, val TEXT)";
 103    if ($dbh->exec($query) === false) {
 104     echo 'Failed to create $table table';
 105     return false;
```

```
   }
  }

  try {
   $stmt = $dbh->prepare("INSERT INTO $table VALUES (?, ?, ?)");
  } catch (PDOException $e) {
   echo 'SQL Prepare failed: ' . $e->getMessage();
   return false;
  }

  foreach ($data as $key => $value) {
   if (!$stmt->execute(array($order_id, $key, $value))) {
    echo 'failed to insert $table';
    return false;
   }
  }
  return true;
 }


 /*! Insert order results returned from Monetra into a database using the
  *  order_id as a key.
  *  \param results  an array of result data
  *  \param order_id order id associated with the results
  *  \return true on success, false on failure */
 function insert_order_results($results, $order_id)
 {
  return insert_order($results, "results", $order_id);
 }


 /*! Insert order request information.  This is used so we can validate
  *  response post-backs, so we need to store the sequence and timestamp
  *  we used the first time
  *  \param results  an array of request data
  *  \param order_id order id associated with the results
  *  \return true on success, false on failure */
 function insert_order_request($request, $order_id)
 {
  return insert_order($request, "request", $order_id);
 }


 /* == Implementation == */
 ob_start();

 $action = "sale";
 $amount = "12.00";

 if (isset($_GET['method']) && $_GET['method'] == 'postback') {
  /* If method == 'postback', this is Monetra directly posting the result
   * data to us to be stored */
  $requestdata = get_order_request($_GET['order_id']);
  if ($requestdata !== false) {
   $hmac_data = $username .
                $action .
                $amount .
                $requestdata["sequence"] .
```

```
                $requestdata["timestamp"] .
                $_POST["code"] .
                $_POST["ttid"];
  $hmacsha256 = hash_hmac("sha256", $hmac_data, $password);
  if (strcasecmp($hmacsha256, $_POST["monetra_resp_hmacsha256"]) != 0) {
   header("HTTP/1.0 500 Internal Server Error");
   echo "Unrecognized hmacsha256";
  } else {
   if (!insert_order_results($_POST, $_GET['order_id'])) {
    header("HTTP/1.0 500 Internal Server Error");
   } else {
    echo "Response recorded";
   }
  }
 }
} else if (isset($_GET['method']) && $_GET['method'] == 'redirect') {
 /* If method == 'redirect' this is the Client's web browser asking us
  * for the result page after Monetra has redirected it back to us.
  * The method = 'postback' is guaranteed to have been called _first_ */
 $results = get_order_results($_GET['order_id']);
 if ($results !== false) {
  echo "<pre>";
  print_r($results);
  echo "</pre>";
 }
} else {
 /* This is the default entry point, start a new order */

 $order_id               = mt_rand(); /* Use random order id */
 $monetra_req_timestamp  = time();
 $monetra_req_sequence   = $order_id;
 $myurl                  = "http://";
 if (!empty($_SERVER['HTTPS'])) {
  $myurl             = "https://";
 }
 $myurl                 .= $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'];
 $monetra_url_postback   = $myurl . "?method=postback&order_id=$order_id";
 $monetra_url_redirect   = $myurl . "?method=redirect&order_id=$order_id";
 $hmac_data              = $username . $action . $amount .
                           $monetra_req_sequence . $monetra_req_timestamp .
                           $monetra_url_postback . $monetra_url_redirect;
 $monetra_req_hmacsha256  = hash_hmac("sha256", $hmac_data, $password);

 $requestdata["sequence"]  = $monetra_req_sequence;
 $requestdata["timestamp"] = $monetra_req_timestamp;
 if (!insert_order_request($requestdata, $order_id)) {
  /* Do nothing, error was output */
 } else {
?>
<html>
<head><title>Enter your order</title></head>
<body>
 <form action='<?=$monetra_url?>' method='POST'>
 <!-- Required fields for Monetra POST -->
 <input type='hidden' name='username' value='<?=$username?>'/>
 <input type='hidden' name='monetra_req_timestamp'
  value='<?=$monetra_req_timestamp?>'/>
 <input type='hidden' name='monetra_req_sequence'
```

```
222    value='<?=$monetra_req_sequence?>'/>
223   <input type='hidden' name='monetra_req_hmacsha256'
224    value='<?=$monetra_req_hmacsha256?>'/>
225   <input type='hidden' name='monetra_url_postback'
226    value='<?=$monetra_url_postback?>'/>
227   <input type='hidden' name='monetra_url_redirect'
228    value='<?=$monetra_url_redirect?>'/>
229
230   <!-- Custom transaction input parameters -->
231   Account: <input type='text' name='account' value='4012888888881881'/><br/>
232   ExpDate: <input type='text' name='expdate' value='0514'/><br/>
233   Amount:  <input type='text' name='amount'  value='<?=$amount?>' readonly/>
234            <br/>
235   ZipCode: <input type='text' name='zip'     value='32606'/><br/>
236   <input type="submit" name="action" value="<?=$action?>" />
237   </form>
238  </body>
239  </html>
240  <?php
241   }
242  }
243
244  ob_end_flush();
245  ?>
246
247
```

A.1.2 Example POST using Tickets (PHP + JavaScript)

```
 1  <?php
 2   error_reporting(E_ALL);
 3   ob_start();
 4   require_once('libmonetra.php');
 5
 6   // The monetra_url must use the same connection type as the web page
 7   // the user is connecting to. Meaning if the web page is https then
 8   // the monetra_url must also be https. Otherwise the ajax request for
 9   // the ticket will fail.
10   $monetra_url    = 'https://testbox.monetra.com:8444';
11   $monetra_host   = 'testbox.monetra.com';
12   $monetra_port   = 8444;
13   $monetra_method = 'SSL';
14   $username       = 'test_ecomm:public';
15   $password       = 'publ1ct3st';
16   $amount         = '12.00';
17
18
19   if ($_SERVER['REQUEST_METHOD'] === 'POST') {
20    // Data we put into the form in the GET request.
21    $order_id               = $_POST['order_id'];
22    $monetra_req_timestamp  = $_POST['monetra_req_timestamp'];
23    // Data from the ajax ticket request we need.
24    // data from the ajax monetra request will be prefixed with mresp_
25    $monetra_resp_hmacsha256 =
26     strtolower($_POST['mresp_monetra_resp_hmacsha256']);
27    $ticket = $_POST['mresp_ticket'];
```

```php

    // Calculate the hmac so we can compare it against what was submitted.
    $hmac_data = $username . $order_id . $monetra_req_timestamp . $ticket;
    $hmac_to_verify = strtolower(hash_hmac('sha256', $hmac_data,
     $password));

    // If we didn't get a ticket then something is very wrong... Most
    // likely there was an error of some kind. We should check the response
    // from Monetra to see what happened.
    if (!$ticket) {
     $monetra_request  =
      'Did not recieve ticket from the ajax request to Monetra!';
     $monetra_response = 'No response because nothing can be sent.';
    } else {
     // Verify the hmac we calculated matches what was submitted.
     if ($hmac_to_verify !== $monetra_resp_hmacsha256) {
      // If it doesn't match then the data was tampered with or is
      // just plain invalid. Either way we can't trust/use it.
      $monetra_request =
       'HMAC SHA-256 failed verification: ' .
       "'$hmac_to_verify' !== '$monetra_resp_hmacsha256'";
      $monetra_response =
       'Not sending request to Monetra because data returned ' .
       'by client cannot be trusted!';
     } else {
      // The hmac was good so we're going to use the ticket to run
      // the transaction with Monetra.
      if (!M_InitEngine(NULL)) {
       echo 'Failed to initialize libmonetra';
       die;
      }

      $conn = M_InitConn();
      if ($conn === FALSE) {
       echo 'Failed to initialize connection resource';
       die;
      }

      if ($monetra_method === 'SSL') {
       if (!M_SetSSL($conn, $monetra_host, $monetra_port)) {
        echo 'Could not set method to SSL';
        die;
       }
      } else if ($monetra_method == 'IP') {
       if (!M_SetIP($conn, $monetra_host, $monetra_port)) {
        echo 'Could not set method to IP';
        die;
       }
      } else {
       echo 'Unknown Monetra connection method.';
       die;
      }

      if (!M_SetBlocking($conn, 1)) {
       echo 'Could not set non-blocking mode';
       die;
      }

```

```
 86      if (!M_Connect($conn)) {
 87       echo 'Connection failed: ' . M_ConnectionError($conn);
 88       die;
 89      }
 90
 91      $monetra_request = array();
 92      $monetra_request['username']         = $username;
 93      $monetra_request['password']         = $password;
 94      $monetra_request['action']           = 'sale';
 95      $monetra_request['cardshieldticket'] = $ticket;
 96      $monetra_request['amount']           = $amount;
 97
 98      $identifier = M_TransNew($conn);
 99      foreach ($monetra_request as $key => $value) {
100       M_TransKeyVal($conn, $identifier, $key, $value);
101      }
102
103      if (!M_TransSend($conn, $identifier)) {
104       echo 'Unable to send transaction.';
105       die;
106      }
107
108      $monetra_response = array();
109      foreach (M_ResponseKeys($conn, $identifier) as $key) {
110       $monetra_response[$key] = M_ResponseParam($conn,
111        $identifier, $key);
112      }
113     }
114    }
115  ?>
116    <h3>POST data:</h3>
117    <pre><?php print_r($_POST); ?></pre>
118    <h3>Monetra Request:</h3>
119    <pre><?php print_r($monetra_request); ?></pre>
120    <h3>Monetra Response:</h3>
121    <pre><?php print_r($monetra_response); ?></pre>
122    <a href=""><button type="button">Try Again</button></a>
123  <?php
124   } else {
125    // Get request.
126    $order_id               = mt_rand();
127    $monetra_action         = 'admin';
128    $monetra_admin          = 'cardshieldticket';
129    $monetra_req_timestamp  = time();
130    $monetra_req_sequence   = $order_id;
131    $hmac_data              = $username . $monetra_action . $monetra_admin
132     . $monetra_req_sequence . $monetra_req_timestamp;
133    $monetra_req_hmacsha256 = hash_hmac('sha256', $hmac_data, $password);
134  ?>
135  <html>
136   <head>
137    <title>Monetra Client Ticket Request Test</title>
138    <script type="text/javascript"
139  src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
140    <script type="text/javascript" src="jquery.ie.cors.js"></script>
141    <script type="text/javascript" src="monetra_post.js"></script>
142   </head>
143   <body>
```

```
144    <form method="post" class="monetra_ticket_form">
145     <div style="display:none;">
146      <input type="hidden" name="_ticket_monetra_url" value="<?=
147  $monetra_url ?>">
148      <input type="hidden" name="_ticket_username" value="<?=
149  $username ?>">
150      <input type="hidden" name="_ticket_monetra_req_timestamp"
151  value="<?= $monetra_req_timestamp ?>">
152      <input type="hidden" name="_ticket_monetra_req_sequence"
153  value="<?= $monetra_req_sequence ?>">
154      <input type="hidden" name="_ticket_monetra_req_hmacsha256"
155  value="<?= $monetra_req_hmacsha256 ?>">
156      <input type="hidden" name="_ticket_action" value="<?=
157  $monetra_action ?>">
158      <input type="hidden" name="_ticket_admin" value="<?=
159  $monetra_admin ?>">
160      <input type="hidden" name="order_id" value="<?= $order_id ?>">
161     </div>
162     Account: <input type="text" name="_ticket_account"
163     value="4242424242424242"><br>
164     Exp. Date: <input type="text" name="_ticket_expdate"
165     value="0514"><br>
166     Zip Code: <input type="text" name="_ticket_zip" value="32606"><br>
167     <input type="submit" value="Place Order">
168    </form>
169   </body>
170  </html>
171
172  <?php }
173
174  ob_end_flush();
175  ?>
176
177
```

The JavaScript example below is embedded into the web-page code.

```
 1  /**
 2   * Submits sensitive data directy to Monetra using the Monetra POST protocol
 3   * using Ajax. The sensitive data is then removed from the form and a
 4   * CardShield ticket (returned by the ajax call) is sent to the server in it's
 5   * place. This is to prevent the server from having to handle the senstitive
 6   * data itself.
 7   *
 8   * Attaches to a form which has the class 'monetra_ticket_form'. The form must
 9   * have the following hidden fields:
10   *
11   * _ticket_monetra_url
12   * _ticket_username
13   * _ticket_monetra_req_timestamp
14   * _ticket_monetra_req_sequence
15   * _ticket_monetra_req_hmacsha256
16   * _ticket_action
17   * _ticket_admin
18   *
19   * Anything with _ticket will be sent to Monetra.
20   * The response from Monetra will be put into the form and prefixed with mresp_.
21   * If an error occurs which prevents getting a response from Monetra an error
```

```
 * element will be added to the form.
 */
$(function() {
 var _STR_TICKET_LENGTH = '_ticket_'.length;

 var $form = $('.monetra_ticket_form');
 $form.submit(function(evt) {
  evt.preventDefault();

  var monetra_fields = {};
  var monetra_url = null;
  // Iterate over all inputs whose name attribute begins with "_ticket_"
  $('input[name^=_ticket_]').each(function() {
   // Strip the leading "_ticket_" from their name
   var field_name = $(this).attr('name').substr(_STR_TICKET_LENGTH);
   if (field_name == 'monetra_req_timestamp') {
    // We will use these to verify the HMAC server-side
    $(this).attr('name', field_name);
   } else {
    // Remove the name attribute, so no data is sent when we do the
    // regular submit after our POST
    $(this).attr('name', null);
    // Disable the field to notify the user that things are
    // happening
    $(this).prop('disabled', true);
   }

   // We don't need to pass along the monetra_url.
   if (field_name == 'monetra_url') {
    monetra_url = $(this).val();
   } else {
    monetra_fields[field_name] = $(this).val();
   }
  });

  if (!monetra_url) {
   $('<input>').attr({ type: 'hidden', name: 'error',
    value: 'monetra_url is empty' }).appendTo($form);
   $form.unbind('submit');
   $form.submit();
   return false;
  }

  $.ajax({
   type: 'POST',
   url: monetra_url,
   data: monetra_fields,
   dataType: 'xml',
   crossDomain: true,
   success: function(data) {
    $.each($(data).find('Resp').children(), function(_, node) {
     $('<input>').attr({ type: 'hidden',
      name: 'mresp_' + node.tagName,
      value: node.textContent}).appendTo($form);
    });
    // Remove our submit handler from the form, so we can submit
    // normally again
    $form.unbind('submit');
```

```
80      // This will continue with the POST to our webserver
81      $form.submit();
82     },
83     error: function(data) {
84      $('<input>').attr(
85       { type: 'hidden', name: 'error',
86        value: data.status + ': ' + data.statusText }
87       ).appendTo($form);
88      $form.unbind('submit');
89      $form.submit();
90     }
91    });
92
93    return false;
94   });
95  });
96
97
```