

## 局域网监控软件WFilter ICF 鸡肋0day RCE漏洞挖掘

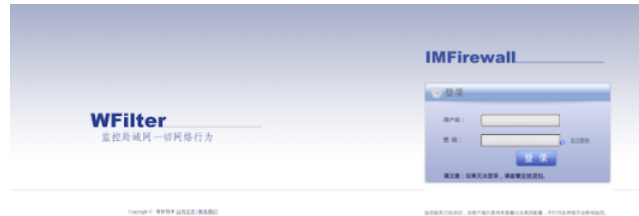
一月 16, 2021



## 0x00关于本文

逛freebuf的时候看到了这篇文章，[关于PDD员工发帖溯源联想到相关技术与实现](#)，其中提到了一个叫做wfilter的局域网监控软件。

这个软件官网提供了下载链接，支持10天实用，于是我就坐下来研究一番。

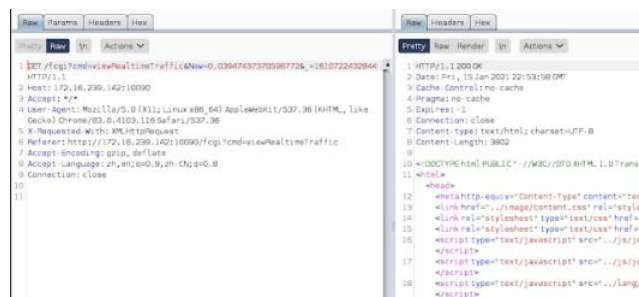


作为一个搞安全的人，我最感兴趣的是这个软件本身是否存在缺陷，毕竟这类用于监控的安全软件自身出现漏洞是最有戏剧性的事情了

## 0x01不同寻常Web管理

Wfilter通过串联/旁路的方式进行部署，同时提供一个Web管理接口供管理员进行远程管理。而作为一个Web狗，我自然把重心放在了在这个Web管理接口上了。

这个Web管理接口不同寻常的第一点就是登录之后不会返回任何Cookies，而且请求接口的时候也不会带Cookies或者任何特殊的Headers



那它是如何进行鉴别的呢？

测试了一番之后发现居然是根据IP地址来鉴别的，一旦管理者登录，wfilter就会记录下管理者的IP地址，然后允许其访问这些接口。

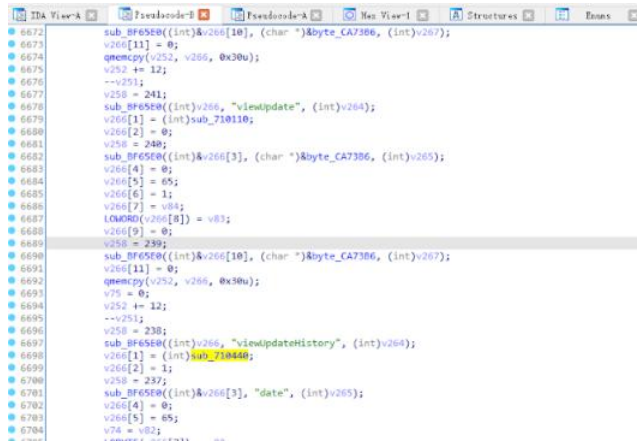
这么诡异的鉴别方式大概率搭配了一个诡异的后端实现，结果我用火绒剑一看，发现后台是通过CGI运行的。也就是说想要审计这个软件还得去逆向这个EXE！

websrvd.exe	2832	2832	IMFirewall Software	Web UI server of
cgiproj.exe	1292	2832	IMFirewall Software	Web UI cgi parse
Conhost.exe	5480	2832	Microsoft Corporation	控制台窗口主进程
startSys.exe	2720	2720	IMFirewall Software	Monitoring proc
Conhost.exe	3092	0	Microsoft Corporation	控制台窗口主进程
userAgent.exe	3076	2832	IMFirewall Software	Username detect
Conhost.exe	3264	0	Microsoft Corporation	控制台窗口主进程
webcategory.exe	3084	2832	IMFirewall Software	Websites auto ca
Conhost.exe	3128	0	Microsoft Corporation	控制台窗口主进程
svchost.exe	2948	0	Microsoft Corooration	Windows 服务丰沛

算了，逆向就逆向吧，正好试试IDA7.5效果怎么样

## 0x01对于cgiproj.exe的逆向分析

这个Web接口的特点是通过/fcgi?cmd=【功能名】来调用，因此在IDA里面搜索相关的字符串就可以找到实现通过功能名来执行不同功能的函数了。



这个函数非常的大，以至于我需要修改IDA的hexrays支持的最大函数大小来让IDA能够成功反编译这个函数（Web狗实在是不想啃汇编了）。

经过一番猜测和实验，我发现这个调用sub\_BF65E0中的参数包含了功能名字，而v266[1]中存放的则是其对应的函数。通过这些信息我可以查看具体接口是如何实现。

### 0x02 思路1：找到无需鉴权的功能，直接拿下服务器！

实现这个思路，简单来说就两个步骤，第一个是找到无需鉴权的功能，第二个是逆向分析然后找到有漏洞功能然后利用。为了找到无需鉴权就能使用的功能，简单起见，我搜集了那个大函数中的所有的字符串，然后拿到接口中去fuzz，接着发现viewHelp,helpSearch,login,mobileCode,mlogin,logout,remoteLogin,viewToolTip,ExecuteNoLogin,forgetPassword,resetPassword,viewHTML这些功能对应的接口无需登录。

但是很快发现他们都无助于入侵。

比如mobileCode功能虽然调用了system()但是所有的参数都是写死的，要么写死在程序里面，要么从配置文件里面读取

```
int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}
```

而viewHTML功能虽然可以玩路径穿越读取文件，但是后缀写死了htm，我尝试了00截断和Windows路径长度截断来绕过但是均未成功

```
int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}
```

ExecuteNoLogin功能看名字很诱人，看样子似乎还可以直接给一个IP让它信任，但是我无论如何调用都显示错误，也没法直接RCE了

```
int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}

int __cdecl sub_8F65E0(int a1, int a2, int a3)
{
    int v1; // eax
    sub_8F65E0(a1, a2, a3);
    v1 = 1;
    return v1;
}
```

而其余的功能更是没法利用，因此目前看来这条路就堵死了。

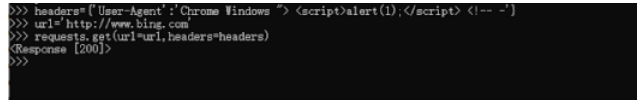
### 0x02 思路2：利用XSS/CSRF配合后台RCE ---XSS挖掘

实现一键日天看起来是不现实的，但是XSS/CSRF配合后台RCE并不是没有可能的。

首先我需要找一个靠谱的XSS漏洞。很快我就发现一个似乎可以利用的点，在“网页浏览”记录查看中，一旦请求带了User Agent，系统就会把User Agent放在前端



为了验证我这个猜测，我在安装了wfilter的机器上再装了个python requests库，以便于调试（简单起见，我并没有用串联/并联的方式将其接入我的网络），接着利用requests库构造一个包含XSS Payload的User-Agent，就像这样



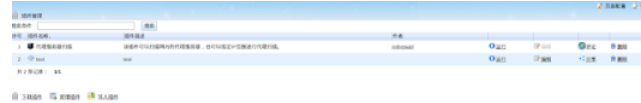
接着再去翻历史



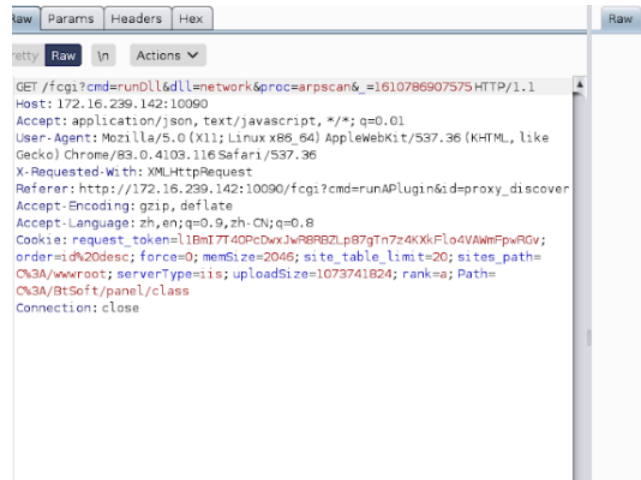
## 0x03 思路2: 利用XSS/CSRF配合后台RCE ---后台RCE挖掘

最开始我想的是直接搜system函数，查看引用，寻觅一波命令注入，但是并没有明显容易注入的点（主要是因为反编译出来的函数太大，而且一些不太清晰的反编译让我无法理清逻辑）。因此我采取另一种办法，即查看web界面有哪些有趣的功能，然后针对性地反编译。

在翻找了一阵子后，我发现了这个系统包含了利用插件扩展的功能。



执行插件的时候抓个包，我发现了一个叫做runDll的功能，这不明显的让我为所欲为？



到IDA上一看，发现这个runDll的逻辑是根据接收到的dll参数加载dll

```
GetParam((int)lpProcName, &dword_D71290, "dll");
v25 = 38;
sprintf(libFileName, "..\\plugins\\%s.dll", lpProcName[0]);
v21 = (void *) (lpProcName[0] - 12);
if ( (int *) lpProcName[0] - 3 != &dword_CA6370 )
{
    v25 = 36;
    if ( _InterlockedExchangeAdd((volatile LONG *) lpProcName[0], 1) != 0 )
    {
        Val = (int) v38;
        free__(v21);
    }
}
v25 = 39;
hModule = LoadLibraryA(libFileName);
```

接着再去获取的proc参数作为函数名，通过GetProcAddress获取函数地址。

最后再去把获取的para参数和另一个不知道干什么的v36传入到刚刚获取的函数中

```
GetParam((int)lpProcName, &dword_D71290, "proc");
v25 = 35;
FuncAddress = GetProcAddress(hModule, lpProcName[0]);
v20 = FuncAddress;
v19 = (void *) (lpProcName[0] - 12);
if ( (int *) lpProcName[0] - 3 != &dword_CA6370 )
{
    v25 = 33;
    if ( _InterlockedExchangeAdd((volatile LONG *) lpProcName[0], 1) != 0 )
    {
        Val = (int) v38;
        free__(v19);
    }
}
if ( v20 )
{
    v36[0] = (int) &unk_CA637C;
    v25 = 32;
    GetParam((int)lpParameter, &dword_D71290, "para");
    v25 = 31;
    v18 = ((int (__cdecl *) (int, int *)) FuncAddress)(lpParameter[0], v36);
```

首先这个dll参数并没有任何过滤，这意味着我们可以穿越到任意目录中去调用任意位置的DLL，因此Windows的各种API我们都可以想办法去调用了。

件相当的苛刻，但是真的有一个函数可以做到，那就是WinExec

## WinExec function (winbase.h)

12/05/2018 • 2 minutes to read

Runs the specified application.

**Note** This function is provided only for compatibility with 16-bit Windows. Applications should use the

## Syntax

```
C++

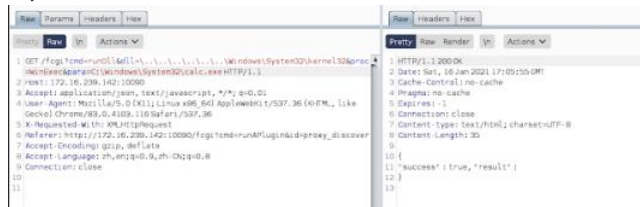
UINT WinExec(
    LPCSTR lpCmdLine,
    UINT    uCmdShow
);
```

而根据微软的描述，这个函数在Kernel32.dll中

## Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll
API set	ext-ms-win-kernel32-process-11-1-0 (introduced in Windows 10, version 10.0.14393)

于是构造如下Payload打过去



函数成功执行，但是并没有弹出计算器，不过这个执行的过程已经被火绒剑记录了下来，这证明我们成功地执行了 shell


[illegible]


至此，整个利用链已经清晰了，就是通过XSS让管理员执行恶意JS，从而触发一个CSRF过去让服务器执行shell

## 0x04 完整的利用过程展示

首先用CS生成一个powershell的payload，这样就可以做到一执行命令就弹shell，接着编写CSRF Payload,用我写的[CSRF 原理 利用 防御](#)就可以了。







DRIVERTOM

访问个人资料

归档

▼

举报滥用情况