SecTopRAT: Updates and Encrypted C2 communications

By Karsten Hahn

Published: 2021-09-03 · Archived: 2026-04-05 14:06:42 UTC

02/17/2021

SectopRAT: New version adds encrypted communication

Reading time: 5 min (1226 words)

SectopRAT, also known as 1xxbot or Asatafar, had been an unknown, in-development threat when we discovered it a year ago. Now it infects systems in Germany. What is the new version capable of?

Infections and aliases

New appearances of SectopRAT infection attempts in our telemetry prompted me to investigate the threat that seemed in its infancy at the time of the first article. The malware has been refined and gotten more features since. To sum up the first article: SectopRAT uses a second, hidden desktop to allow remote control. Parts of it seem unfinished.

While the previous article states that SectopRAT was first mentioned in 15. November 2019, I have now discovered that earlier tweets by @nao_sec from March 2019 use a different name for the malware: **ArechClient**. Other aliases are **1xxbot**, **ArechSoft** and **Asatafar**. Most names stem from the module name of the RAT or the PDB path. This name has changed in the course of development, likely to evade detection and identification. For the sake of consistency and clarity I will stick with **SectopRAT** because the use of a **sec**ond desk**top** seems to be the most notable core-feature. Many antivirus naming policies also forbid using a name that the malware developer has chosen.

Three packing layers

The analysed sample has three layers which need to be unpacked. The first one is obfuscated with SmartAssembly. The method with token **0x060001C6** invokes a .NET injection library (see picture of the deobfuscated code below).

```
// Token: 0x060001C6 RID: 454 RVA: 0x000064F4 File Offset: 0x000046F4
static void smethod_99()
{
    Class2 @class = new Class2();
    foreach (Type type in @class.Assembly.GetExportedTypes())
    {
        foreach (MemberInfo memberInfo in type.GetMembers())
        {
            if (memberInfo != null && memberInfo.Name == "LKrZTd0Le")
            {
                object target = Activator.CreateInstance(memberInfo.DeclaringType, false);
                @class.o = type.InvokeMember(memberInfo.Name, BindingFlags.Instance | BindingFlags.Public | BindingFlags.InvokeMethod, null, target, null);
                if (@class.o != null)
                {
                    Console.WriteLine((string)@class.o);
                }
                else
                {
                    Console.WriteLine("Err");
                }
            }
        }
    }
}
```

The injection library[2] has a configuration that allows multiple options, one of them being RunPE for native files. However, only a small portion is used which will decompress a file that is embedded as byte array and execute that. The code in the screenshot below shows the method reposonsible for the decompression stub.

```
// Token: 0x06000009 RID: 9
private void loadAndInvokeInjectionDLL()
{
    AppDomain domain = Thread.GetDomain();
    Assembly assembly = domain.Load(this.decompressGZIP(FnyvenjdJ176c5OPFg.compressedAssembly));
    Type type = assembly.GetExportedTypes()[0];
    MethodInfo object_ = type.GetMethods()[0];
    object obj = Activator.CreateInstance(type);
    if (obj == null)
    {
        int num = 0;
        if (TCmHVbF3Yo4eAsl5TU.vH12cwGuCQLNhfRvP6() != null)
        {
            switch (num)
            {
            }
        }
    }
    else
    {
        TCmHVbF3Yo4eAsl5TU.objInvokeW(object_, obj, new object[]
        {
            FnyvenjdJ176c5OPFg.H767c5OPF,
            Configuration.mainModuleFilename,
            ""
        });
    }
}
```

This decompressed file turns out to be another injection library[3]. At first glance the two libraries[2][3] look different because they use a different obfuscation. But they are actually the same injection library. This does not make sense from an attacker's perspective as adding more layers of the same provide ample opportunity to detect the code of the packer stub, at least moreso than having just one. If one obfuscation evades the antivirus software, the other might not.

The decompressed library[3] is the last packed layer and finally performs injection of SectopRAT[4] into its own child process via classic RunPE.

Configuration and encrypted CnC communication

The analyzed sample[1] saves configuration data as well as the IP of the server in a different class. The class that contained the IP in previous versions now shows the localhost. This is most likely an attempt to evade automatic

extraction of the command and control (CnC) server. If such extration tools are static, they might now yield 127.0.0.1 which won't raise as much suspicion as a non-working IP extraction. Dynamic analysis of course still shows the actual IP.

The configuration has now additional entries, such as a build ID and an encryption key for the CnC communication. The build ID shows "Build 3".

The CnC communication encryption key is saved in a 32 byte array named **rawData**. This key changes with different build versions. The CnC communication data is encrypted and decrypted with AES256 using said key and a randomly generated 16 byte intitialization vector (IV). This IV is prepended to the encrypted data before it is sent.

New commands

Besides added encryption, the server also supports a number of new commands. These are triggered via a JSON (a data-interchange format) string that contains one of the following command strings.

Command string	**Descryption**
ReceiveStopCapture	Not implemented
ReceiveParticipantList	Updates the list of participants; the purpose is unclear, the list doesn't seem to be used for anything else
ReceiveBotURL	Downloads a file from a given URL to %APPDATA%\<randomfilename>.exe
ReceiveSessionID	Sets the client's session ID
ReceiveTestTest	Sends test JSON strings.
ReceiveCaptureRequest	Not implemented
ReceiveSetColorDep	Not implemented
ReceiveControlInt	Not implemented
ReceiveSetApp	Restarts the client
ReceiveServerAfkSystem	Not implemented
ReceiveEncryptionStatus	Sets up encrypted communication to the server. If the connection type is "ElevatedClient", it will use the client's session ID set via ReceiveSessionID If the connection type is "Client", it will use a session ID created via hashing system information of the infected system. The system info is comprised of processor, RAM and graphics card information.
ReceiveFullscreenRequest	Not implemented

Most of these command strings are not implemented yet. The ones that are there mainly support the CnC encryption. The non-implemented strings paint a picture of planned updates to SectopRAT which may include screen capturing, controlling the screen resolution and fullscreen support. The purpose of the participant list as well as the ReceiveControlInt command is not clear. ReceiveServerAfkSystem may be a notification that the user is "away-from-keyboard", which means the system is currently not used.

Other changes

SectopRAT has no code related to persistence anymore. This part was probably outsourced to the loader.

Yara hunt rule evasion

In my last article about SectopRAT I published a Yara rule that contained three method names, an enum, and the module name. All but one of the method and module strings have been renamed in the newest build:

- EnoughSpace -> EnoghtSpace
- RemoteClient -> rsddsrrg
- InitHDesktop -> WaitReq

This might not be a coincidence. We know that malware developers read the news. This is why public Yara rules are often only suitable to collect past samples. I recommend malware hunters after publishing detection rules to have a second set of entirely different private rules for staying up-to-date on current samples.

Conclusion

SectopRAT is actively used to infect systems which is evident by our telemetry as well as researchers like @nao_sec who found the malware being distributed via RIG exploit kit. Stub methods in the code show which functionality is likely planned to be added soon. That may include screen capturing, full screen support and screen resolution settings.

Malware naming is, once again, a difficult topic to navigate. Earlier mentions of the malware were unknown due to the different naming by researchers. It often comes down to having a bit of luck and good contacts to other researchers to see the connection. At the moment there is no other solution to that than documenting aliases in blogs and public resources.

Fact summary and IoCs

Aliases: 1xxbot, ArechClient, ArechSoft, Arech, AsataFar, AsataFarClient

First known appearance: March 2019 https://twitter.com/nao_sec/status/1103474183676125185

TCP IP and port: 54.194.254.16:15647

Build id: Build 3

Compilation date: 2020-10-19 19:07:33

Compiler: .NET Framework 4

Current server IP saved in: %LOCALAPPDATA%\Microsoft\crds.rtt

AES key for CnC communication: 64 6a 14 63 72 7f d8 7d c5 f5 30 46 8f 7f b0 59 28 85 67 d2 38 64 e 5e 8d 91 d2 4b 45 d1 25 d5

Description	Module name	SHA256
[1] SectopRAT, analysed sample, packed	videolan.exe	df4f0960c97e2ad0697aacfd608ea2b66b732406dee28267e09234b3bad334a3
[2] injection library, unpacked from [1]	Kapsscmrduikk.dll	df04b3b2ada5fd7856696f0c7c60146aff8bbd69598e591aa2d8261dd21ddb4a
[3] injection library, unpacked from [1]	ClassLibrary3.dll	c77b8e0c86ae52d19d31d70bd7add6f85b3baee6f39250b06242344937bfa3df
[4] SectopRAT, unpacked payload from [1]	hjguh.exe	03fad40b096f42b90e30687dec1a8c085008399d8c0d3f45ffbf27828ac79e5e

Related articles:

Share Article

Content

- Related articles
