HUNT & HACKETT

REvil: the usage of legitimate remote admin tooling

KRIJN DE MIK @ JUN 10, 2021 12:42:03 PM

1. Introduction

Recently, Hunt & Hackett did an incident response engagement involving Sodinokibi (also known as REvil) ransomware. During the incident, the adversary installed a ScreenConnect service on several systems, functioning as a backdoor. This gave the adversary the possibility to connect directly to those systems, without the need of using the Remote Desktop Protocol (RDP), or the need to authenticate (this is required of course for installation of ScreenConnect). This is a rather efficient and effective technique used by more threat actors, next to other types of legitimate Remote Administration Tools like TeamViewer and AnyDesk. Other examples of threat actors that have been using ScreenConnect in the past are the Iranian actor named Static Kitten [1] and another targeted ransomware group called Zeppelin [2].

In this blog post, we will look into the traces left behind by the usage of ScreenConnect remote administration software and how these traces can help defenders with building custom detection.

2. ScreenConnect traces and detection

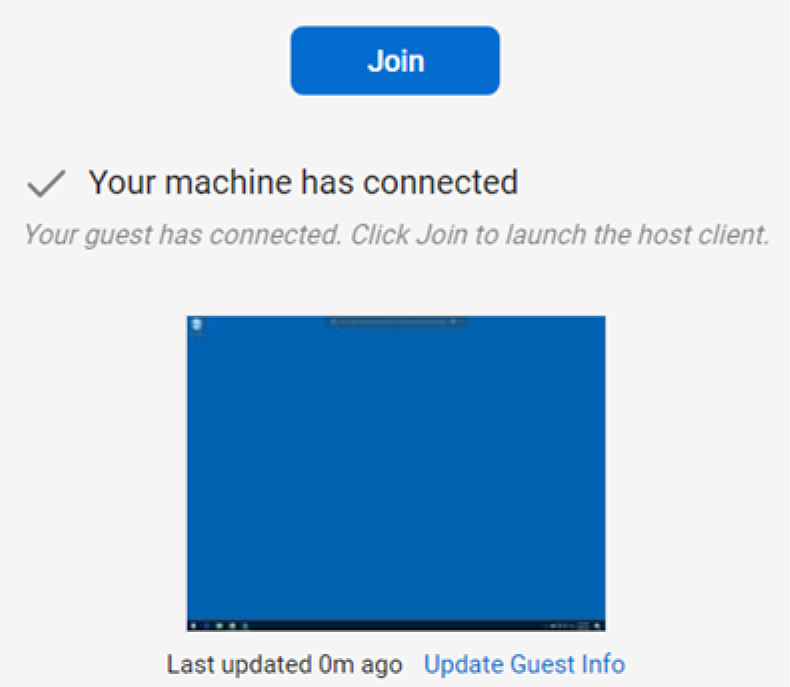
This chapter describes the different traces left by ScreenConnect when it is actively used. Furthermore, some rules are provided in order to detect the usage of ScreenConnect on a system, or in an infrastructure.

2.1 ScreenConnect funtionality

ConnectWise Control [**3**] (formerly known as ScreenConnect) is advertised as a solution that "gives your techs full remote access to remotely control, troubleshoot, and update client devices". It should not come as a surprise that ScreenConnect can thus also be used for malicious purposes.
Via the web interface, ConnectWise Control (among others) offers functionality to remotely:

- Execute arbitrary commands;
- Terminate processes;
- Uninstall software;
- View event logs;
- Start / Stop services;
- Install Windows updates.

Additionally, ConnectWise Control allows an operator to take control of a machine's desktop session. During a recent incident response case, the File Transfer functionality was among others used to upload MimiKatz [**4**] to a compromised system, as well as to upload other tools like Advanced IP Scanner and the actual ransomware.



✓ Your machine has connected

Your guest has connected. Click Join to launch the host client.

Last updated 0m ago

Figure 1- Machine with status connected in ConnectWise Control.

All events related to ScreenConnect can be found in the Windows event logs and are logged with the provider name '`ScreenConnect Client (<hex string>)`'. More specifically in the `Application.evtx` and `System.evtx` log files, which can generally be found at the following location:

- `C:\Windows\System32\winevt\logs\<event log file>.evtx`

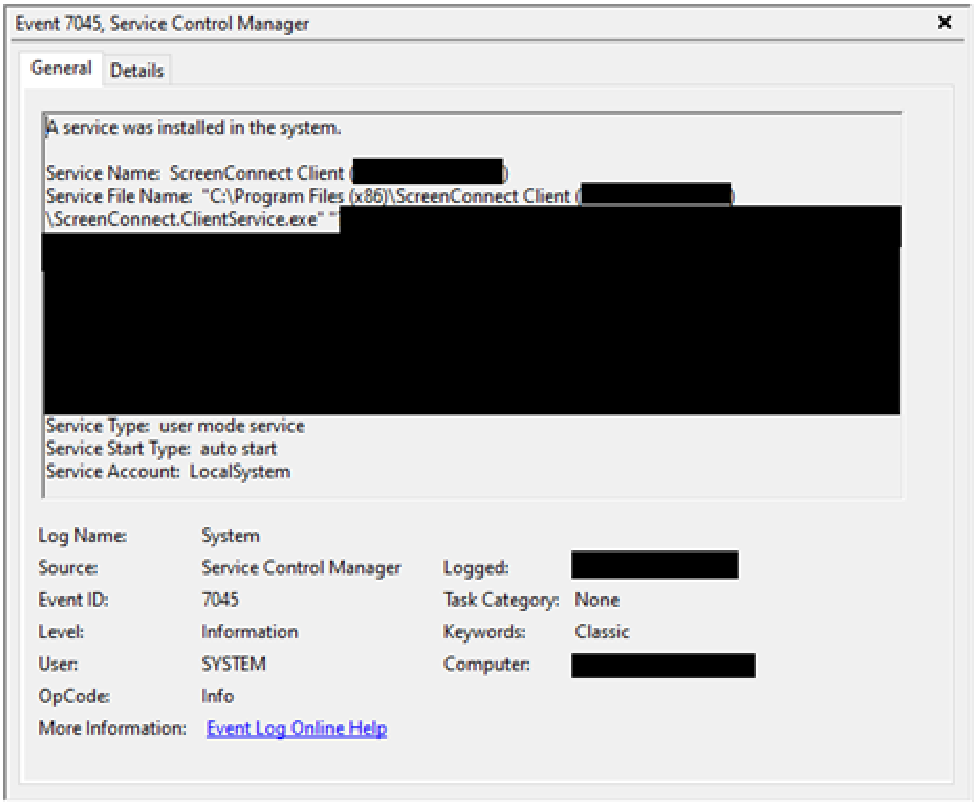
In *Table 2* an overview is given of the different events that are being logged in the Windows event logs, what is being logged, in which log file the event can be found and what the corresponding EventID Is.

Event	**Value**	**Log**	**EventID**
Service being installed	A service was installed in the system	System	7045
Start of remote session	Cloud Account Administrator Connected	Application	0
Closing of remote session	Cloud Account Administrator Disconnected	Application	0
File upload/ transfer	Transferred files with action 'Transfer': <list of file names>	Application	0
Command execution	Executed command of length: <size>	Application	0

Table 2- Windows Event log event information and variables.

2.2 ScreenConnect installation of service

When ScreenConnect is being installed, it installs itself as a service. Services that are being installed show up in the Windows event logs and can therefore be detected. More specifically, these events can be found in the 'System' event log and get the Event ID 7045. An example of such an event is shown in *Figure 2*.



A service was installed in the system.

Service Name: ScreenConnect Client ([redacted])
Service File Name: "C:\Program Files (x86)\ScreenConnect Client ([redacted])\ScreenConnect.ClientService.exe" "[redacted]

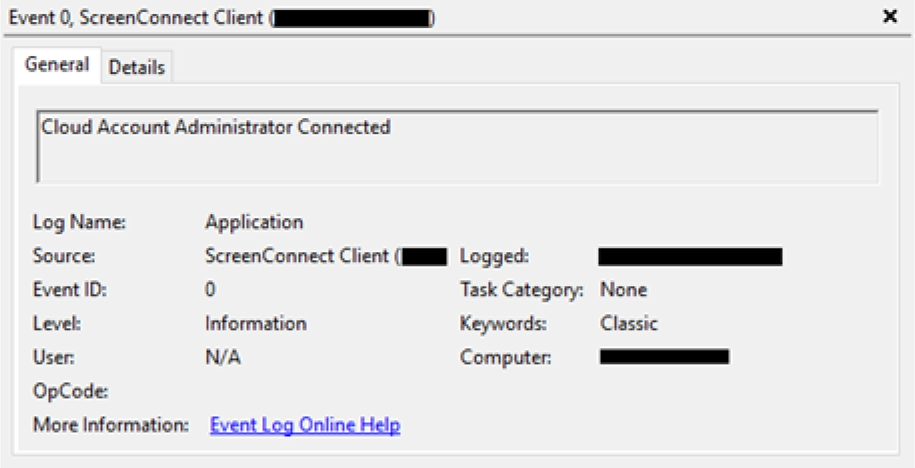
Service Type: user mode service
Service Start Type: auto start
Service Account: LocalSystem

Log Name: System
Source: Service Control Manager
Logged: [redacted]
Event ID: 7045
Task Category: None
Level: Information
Keywords: Classic
User: SYSTEM
Computer: [redacted]
OpCode: Info
More Information: Event Log Online Help

Figure 2 - ScreenConnect being installed as a service Windows event.

2.3 ScreenConnect start and ending of a session

Once a user decides to '*Join*' an endpoint (as shown in *Figure 1*) and to interact with it, a new event is being logged in the Windows Application event log. An example of such a recorded event is shown in *Figure 3*. A session disconnect is recorded as well and an example is shown in *Figure 4*.

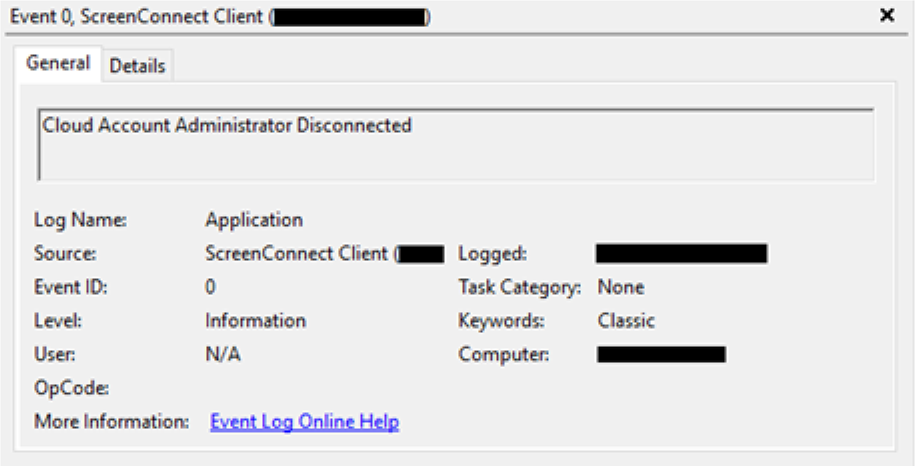


Event 0, ScreenConnect Client ([redacted])

Cloud Account Administrator Connected

Log Name: Application
Source: ScreenConnect Client ([redacted]
Logged: [redacted]
Event ID: 0
Task Category: None
Level: Information
Keywords: Classic
User: N/A
Computer: [redacted]
OpCode:
More Information: Event Log Online Help

Figure 3 - Cloud Account Administrator Connected event.



Event 0, ScreenConnect Client ([redacted])

Cloud Account Administrator Disconnected

Log Name: Application
Source: ScreenConnect Client ([redacted]
Logged: [redacted]
Event ID: 0
Task Category: None
Level: Information
Keywords: Classic
User: N/A
Computer: [redacted]
OpCode:
More Information: Event Log Online Help

Figure 4 - Cloud Account Administrator Disconnected event.

2.4 ScreenConnect file transfers

ScreenConnect offers different ways of interacting with the endpoint on which the ScreenConnect agent is installed. File Transferring is one of them. Files can be both send as well as being retrieved from an endpoint as shown in *Figure 5*.

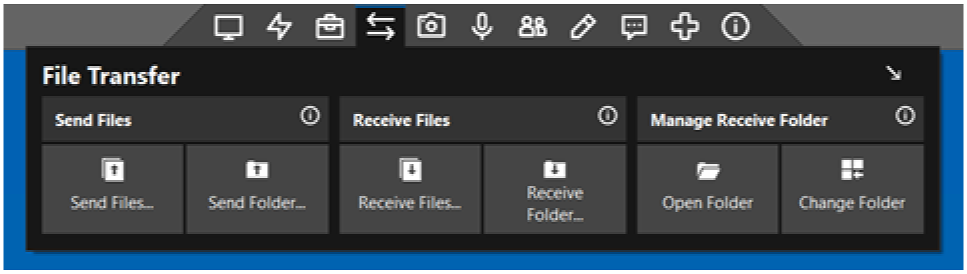
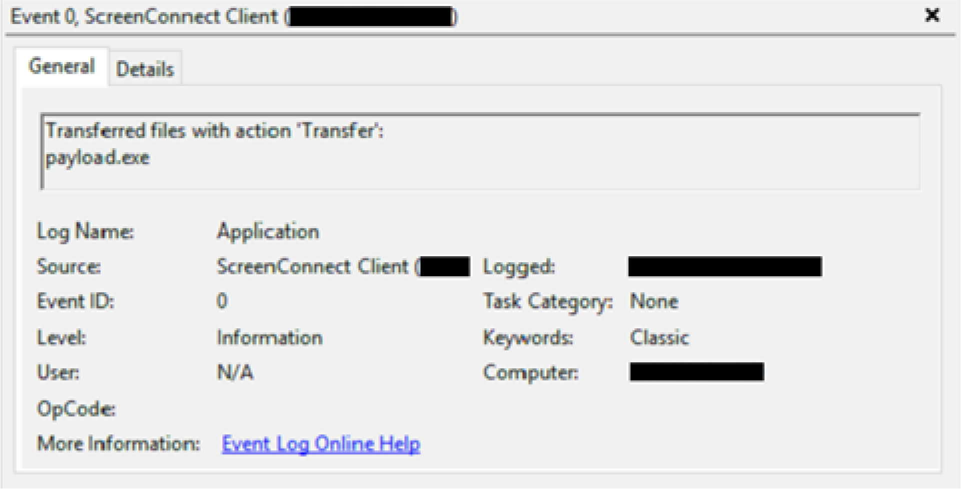


Figure 5 - ConnectWise Control file transfer functionality.

When files are transferred, the Windows Application event log not only records this as an event, but also registers the file that is being exchanged. In *Figure 6* the file payload.exe is transferred to the endpoint. Do note that the retrieval of files is not logged in the Windows Application event log.


Event 0, ScreenConnect Client ([redacted])

```
Transferred files with action 'Transfer':
payload.exe
```

Log Name:	Application		
Source:	ScreenConnect Client ([redacted]	Logged:	[redacted]
Event ID:	0	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	[redacted]
OpCode:			
More Information:	Event Log Online Help		

Figure 6 - Windows event log event indicating a file has been transferred.

2.5 ScreenConnect command execution

Another form of interaction with an endpoint is command execution. Via the ConnectWise Control center it's possible to type a command, hit the '*Run Command*' button, after which the command Is executed. The commands that are allowed to be used are the commands that are generally supported by the Windows Command Prompt. An example of the interface in ConnectWise Control is shown in *Figure 7*.

Figure 7 - ConnectWise Control command execution functionality.

Upon execution of an operator invoked task, a Windows Event is generated that indicates a command of a certain length has been executed, as shown in *Figure 8*. The type of executed task cannot be derived from the Windows Event Logs. However, manually-executed shell commands are launched from `ScreenConnect.ClientService.exe` as command (`.cmd`) scripts, whereas tasks like process listing and termination are executed via Powershell (`.ps1`) scripts.

Figure 8 - Windows Eventlog event indicating an executed command.

The process of a command task being launched by `ScreenConnect.ClientService.exe`, the actual execution of the command and the result are outlined in a bit more detail as shown in *Table 3*.

```
C:\Program Files (x86)\ScreenConnect Client (<hex
string>)\ScreenConnect.ClientService.exe
    --> "cmd.exe" /c "C:\Windows\TEMP\ScreenConnect\<version>\
    <uuid>run.cmd"
        --> <executed shell command>
```

Table 3 - Execution chain of command via the ConnectWise Control interface.

2.6 ScreenConnect detection and response

Installation of and interaction with ScreenConnect can be detected. We have developed Carbon Black and YARA-L rules to support detection efforts. These rules can be found in appendix 1, at the bottom of this page. Furthermore, from a forensics perspective, the fact that `ScreenConnect.ClientService.exe` writes either the command(s) to a `.cmd`, or a `.ps1` file before executing the command, might make it possible (hasn't been tested yet by Hunt & Hackett) to carve for these files. This potentially shines a bit more light on the actual commands that have been executed by the adversary.

3. Anti-forensics and robust detection

ScreenConnect event logs can indicate that an operator has connected to a machine or performed certain actions like executing commands or transferring files.

At first glance, it might appear that monitoring for ScreenConnect events might be enough to detect malicious usage of ScreenConnect. This is however not something we can always assume. For robust detection we cannot take for granted that all information is always available.

For example, we cannot just rely on the Windows event logs. ScreenConnect events are generated by the application itself, meaning that a determined attacker can prevent any logs from being created. The ScreenConnect client uses the `TryWriteInformationToEventLog()` function to log certain events.

Figure 9 - ScreenConnect event log operation.

`TryWriteInformationToEventLog()` is implemented in the unsigned `ScreenConnect.Core.dll` module and, below the surface, is merely a wrapper around `EventLog.WriteEntry()`. If we can modify the application in such a way that `EventLog.WriteEntry()` is no longer ever executed, we can prevent the generation of event logs.

Figure 10 - ScreenConnect write to EventLog.WriteEntry.

To demonstrate that we can prevent ScreenConnect from logging events, we can patch the bytecode of `ScreenConnect.Core.dll`, using a tool like *dnSpy* [**5**], and overwrite the call to `EventLog.WriteEntry()` in `TryWriteInformationToEventLog()`, as shown in *Figure 11*. This ensures that ScreenConnect can no longer generate any events and will create less evidence.

For this proof-of-concept, we manually patched out the call to `EventLog.WriteEntry()`. A determined attacker could of course implement this anti-forensics technique in a different way.

Figure 11 - Patching the write to event log functionality of ScreenConnect.

In case an attacker would use such a modified version of ScreenConnect, we cannot just rely on the Windows event logs. Instead, detection should also focus on the execution of suspicious executables, whether it being ScreenConnect being launched / installed by the attacker or executable files being started via the build-in Run Command functionality.

In Appendix 1, we have included detection rules for Chronicle and Carbon Black that can detect the initialization of ScreenConnect and the execution of tasks via the control panel, so that you can use these in your SOC or MDR setup. Additionally, a Sigma rule is publicly available that can detect the initialization of ScreenConnect [**6**].

Post Incident, carving of `.ps1`, or `.cmd` files might the way to go, in case you would like to have an understanding of the command that might have been executed by the adversary.

4. Sources

[1] https://www.anomali.com/blog/probable-iranian-cyber-actors-static-kitten-conducting-cyberespionage-campaign-targeting-uae-and-kuwait-government-agencies

[2] https://blog.morphisec.com/connectwise-control-abused-again-to-deliver-zeppelin-ransomware

[3] https://www.connectwise.com/platform/unified-management/control

[4] https://github.com/gentilkiwi/mimikatz

[5] https://github.com/dnSpy/dnSpy

[6] https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/win_susp_screenconnect_access.yml

Appendix 1 - Detection rules

This appendix contains an overview of different rules in both Carbon Black as well as Yara-L format. Additionally, you can find an already existing Sigma rule in the Sigma rule repository '`SigmaHQ/sigma/rules/windows/process_creation/win_susp_screenconnect_access.yml`', For completeness the Sigma rules is listed in this appendix as well.

Sigma

Action	Sigma rule

ScreenConnect Remote Access detection	`title: ScreenConnect Remote Access`
	`id: 75bfe6e6-cd8e-429e-91d3-03921e1d7962`

```
status: experimental
description: Detects ScreenConnect program starts that establish a remote access to that system (not meeting, not remote support)
references:
         -   https://www.anomali.com/blog/probable-iranian-cyber-actors-static-kitten-conducting-cyberespionage-campaign-targeting-uae-and-kuwait-government-agencies
author: Florian Roth
date: 2021/02/11
logsource:
   category: process_creation
   product: windows
detection:
   selection:
       CommandLine|contains|all:
           - 'e=Access&'
           - 'y=Guest&'
           - '&p='
           - '&c='
```

	`- '&k='`
	`condition: selection`
	`falsepositives:`
	`- Legitimate use by administrative staff`
	`level: high`

Carbon Black

Action	Carbon Black query
ScreenConnect initialisation	`(process_cmdline:"y=" OR process_cmdline:"?y=") AND (process_cmdline:"h=" OR process_cmdline:"?h=") AND (process_cmdline:"p=" OR process_cmdline:"?p=") AND (process_cmdline:"s=" OR process_cmdline:"?s=") AND (process_cmdline:"k=" OR process_cmdline:"?k=")`
Task execution through ScreenConnect	`(process_name:*screenconnect* OR process_publisher:*ConnectWise*) AND childproc_cmdline:*run.ps1`
Cmd command execution through ScreenConnect	`(process_name:*screenconnect* OR process_publisher:*ConnectWise*) AND`

	childproc_cmdline:*run.cmd

Yara-L

<table>
<tr><th>Action</th><th>Rule</th></tr>
<tr><td><b>ScreenConnect initialisation</b></td><td>rule screenconnect_initialisation {
    meta:
        author = "Hunt & Hackett"
        description = "Detects ScreenConnect initialisation"

    events:
        $e.metadata.event_type = "PROCESS_LAUNCH"
        $e.principal.process.command_line = /(\?|\&amp;|\;)y=/ nocase
        $e.principal.process.command_line = /(\?|\&amp;|\;)h=/ nocase
        $e.principal.process.command_line = /(\?|\&amp;|\;)p=/ nocase
        $e.principal.process.command_line = /(\?|\&amp;|\;)s=/ nocase
        $e.principal.process.command_line = /(\?|\&amp;|\;)k=/ nocase

    condition:</td></tr>
</table>

	$e }

ScreenConnect task execution	rule screenconnect_task_execution {
	meta:
	author = "Hunt & Hackett"
	description = "Detects task execution through ScreenConnect"
	events:
	$e.metadata.event_type = "PROCESS_LAUNCH"
	
	(
	$e.principal.process.file.full_path = /ScreenConnect/ nocase and
	$e.target.process.file.full_path = /powershell.exe/ nocase
	)
	or
	(
	$e.principal.process.command_line = /(powershell\.exe)(.*)([a-f0-9-]{36}run\.ps1)/ nocase
	)
	
	condition:
	$e
	}

ScreenConnect command	rule screenconnect_cmd_command_execution { meta:

<table>
<tr>
<td><b>execution</b></td>
<td>

```
        author = "Hunt & Hackett"
        description = "Detects cmd command execution through ScreenConnect"

    events:
        $e.metadata.event_type = "PROCESS_LAUNCH"

        (
                $e.principal.process.file.full_path = /ScreenConnect/ nocase and
                $e.target.process.file.full_path = /cmd.exe/ nocase
        )
        or
        (
                $e.principal.process.command_line = /(cmd\.exe)(.*)(\/c)(.*)([a-f0-9-]{36}run\.cmd)/ nocase
        )

    condition:
        $e
}
```

</td>
</tr>
</table>

DETECTION TECHNOLOGY DETECTION LOGIC MDR DETECTION DATA SOURCES TACTICS, TECHNIQUES & PROCEDURES

Keep me informed

SIGN UP FOR THE NEWSLETTER

MADE IN EUROPE