Detecting Rogue RDP

You won't believe what Sysmon missed with this new initial access technique!

March 27, 2022 · 9 min · Nameless

▶ Table of Contents

Introduction

Microsoft recently announced the disablement of VBA macros carrying the mark of the web, impacting the ease-of-use of this widespread initial access technique. BlackHillsInfosec wrote a blog post on a different technique that could help fill the void titled "Rogue RDP". This post examines signals generated by the attack, outlines detection opportunities, and discusses required sysmon configuration changes.

Rogue RDP Technique

The author Mike Felch describes the technique as:

> ... the ability to leverage a malicious RDP server, an RDP relay, and a weaponized .RDP connection file which forces unsuspecting victims into connecting and forwarding control over some parts of their machine. With the right ruse, an established connection will provide

The novelty of this attack leaves a lot of things that can go right for the attacker:

- .RDP file extensions are not blocked by default with most email gateways.
- .RDP files are not blocked by default with most Web proxies.
- The malicious code runs on the remote server instead of the target.
- DLL sideloading is a detection weak point in even the best EDR's.

Signals

Let's break down this new technique into a series of signals from a defender's perspective.

Delivery

- Email log with a .rdp file as an email attachment.
- Creation of a .rdp file in the user's Downloads/Desktop/etc. folder by a Outlook/Web Browser/Chat Application process.

Execution

- Process creation of mstsc.exe with a .rdp file specified as a process argument.
- Network connection initiated by a mstsc.exe process with a .rdp file specified specified as a process argument over a non-standard port.
- Outbound RDP traffic from the source over a mismatched port observed by a network sensor.

Persistence

- Creation of a new file on the system by a mstsc.exe process with a .rdp file specified as a process argument.

Detection Opportunities

There are two chokepoints we can monitor as defenders that particularly stand out.

1. The attacker is highly incentivized to set their .rdp connection file to connect over a non-standard port back to their PyRDP redirector.
2. In order to gain remote code execution on the target, they have to drop a file to disk with the mstsc.exe process.

1. Connection Over a Non-Standard Port

Why is the attacker highly incentivized to use a non-standard port?

As mentioned by Mike:

- To avoid the brute forcing of their infrastructure by the internet if port 3389 is exposed.
- To workaround the target having outbound traffic blocked over port 3389.

Network connections coming from the mstsc.exe process, the process handling the .rdp file execution, typically occur over port 3389. Any other port is a very noticeable anomaly. Even more so when the process that initiated the network connection was started with a .rdp file specified as a process argument.

2. Mstsc.exe File Writes

Mike notes several options for pushing a payload to the connecting client.

- LNK
- Binary in auto-run location
- DLL for sideloading

process. There may be times that the mstsc.exe process is writing files during a typical RDP session for a user. However, they are rarely .lnk, .exe, or .dll files that have been created by a mstsc.exe instance with a .rdp file specified as a process argument.

Let's look at potential implementation of these detection opportunities through two different tools - Microsoft Defender for Endpoint (with Advanced Hunting/KQL) and with Sysmon data ingested into Splunk.

Microsoft Defender for Endpoint

Rule #1 - MSTSC Connection Over a Non-Standard Port Initiated by a RDP File

```
DeviceNetworkEvents
| where InitiatingProcessFileName == "mstsc.exe" and InitiatingProcessCommandLine contains ".rd
```

Rule #2 - Suspicious MSTSC File Creation Initiated by a RDP File

```
DeviceFileEvents
| where InitiatingProcessFileName == "mstsc.exe" and InitiatingProcessCommandLine contains ".rd
| where FileName endswith ".exe" or FileName endswith ".dll" or FileName endswith ".lnk"
```

Splunk/Sysmon

Sysmon Visibility Gaps

Unfortunately the implementation in Sysmon/Splunk isn't as simple.

While conducting this attack in the lab, I noticed that there was missing Sysmon telemetry. I was able to isolate implicit exclusions in two popular Sysmon configurations SwiftOnSecurity's and

Olaf Hartong's that led to the anticipated telemetry being missed. So before explore [illegible] implementation with Sysmon/Splunk, let's breakdown how to get Sysmon to **[illegible]p see the event**.

Essentially there are two problems.

1. Both the SwiftOnSecurity and Olaf Hartong configurations will not generate a network connection event when the mstsc.exe process uses ports other than 3389 (with a few exceptions).
2. The Olaf Hartong configuration will not generate a file creation event for a DLL file placed into Teams/Slack for sideloading (which has greater impact than just in a Rogue RDP attack.)

Problem #1

There are two issues that exist in both configs that lead to **implicitly excluded** events when the mstsc.exe process initiates a network connection over a port other than 3389.

1. There is **not** a match condition for the Image "mstsc.exe".
2. There is a small list of DestinationPort conditions. Ports that are likely to be chosen for this attack are not covered.

The configurations will see standard "mstsc.exe" initiated network connections over 3389 because of the matching condition below.

```
<DestinationPort name="RDP" condition="is">3389</DestinationPort>
```

This condition does not take the protocol into account and only matches on the port. Therefore, RDP traffic is not what is being matched specifically, just any traffic over 3389.

But as we know, attackers are incentivized to go over non-standard ports in this attack.

As long as they do not choose another port that has a matching DestinationPort condition, the event will be missed - breaking the entire detection pipeline.

Excerpt from SwiftOnSecurity's Sysmon config

```
<!--SYSMON EVENT ID 3 : NETWORK CONNECTION INITIATED [NetworkConnect]-->
<RuleGroup name="" groupRelation="or">
    <NetworkConnect onmatch="include">
        ...
        <Image condition="image">mmc.exe</Image>
        <Image condition="image">msbuild.exe</Image>
        <Image condition="image">mshta.exe</Image>
        <Image condition="image">msiexec.exe</Image>
        <Image condition="image">nbtstat.exe</Image>
        ...
        <!--Ports: Suspicious-->
        <DestinationPort name="SSH" condition="is">22</DestinationPort>
        <DestinationPort name="Telnet" condition="is">23</DestinationPort>
        <DestinationPort name="SMTP" condition="is">25</DestinationPort>
        <DestinationPort name="IMAP" condition="is">143</DestinationPort>
        <DestinationPort name="RDP" condition="is">3389</DestinationPort>
        <DestinationPort name="VNC" condition="is">5800</DestinationPort>
        <DestinationPort name="VNC" condition="is">5900</DestinationPort>
        <DestinationPort name="Alert,Metasploit" condition="is">4444</DestinationPort>
        <!--Ports: Proxy-->
        <DestinationPort name="Proxy" condition="is">1080</DestinationPort>
        <DestinationPort name="Proxy" condition="is">3128</DestinationPort>
        <DestinationPort name="Proxy" condition="is">8080</DestinationPort>
        <DestinationPort name="Tor" condition="is">1723</DestinationPort>
        <DestinationPort name="Tor" condition="is">9001</DestinationPort>
        <DestinationPort name="Tor" condition="is">9030</DestinationPort>
    </NetworkConnect>
</RuleGroup>
```

Both configs can be patched for this implicit exclusion by adding an Image match condition for "mstsc.exe" under the NetworkConnect section.

```
<!--SYSMON EVENT ID 3 : NETWORK CONNECTION INITIATED [NetworkConnect]-->
<RuleGroup name="" groupRelation="or">
    <NetworkConnect onmatch="include">
```

```
...
<Image condition="image">mstsc.exe</Image>
```

Problem #2

There are three issues in the Olaf Hartong config specifically that combine to **implicitly exclude** file creation events for DLL files placed into Teams/Slack folders through a Rogue RDP attack.

1. There is **not** a match condition for a TargetFileName ending with ".dll"
2. There is **not** a match condition for the Image "mstsc.exe"
3. There is **not** a match condition for TargetFileNames in the installation path of Teams/Slack.

Teams and Slack typically live in the user's AppData folder, which allows for attackers to easily drop their own DLLs for sideloading.

For example, Slack's typical installation path is something like:

```
C:\Users\Administrator\AppData\Local\slack\app-4.24.0\
```

The "\AppData\Local" path is not covered by a match condition.

There is coverage for other user folder paths such as Public or AppData\Temp as seen below, but that is not where the file will be placed.

```
<TargetFilename name="technique_id=T1047,technique_name=File System Permissions Weakness" condi
<TargetFilename name="technique_id=T1047,technique_name=File System Permissions Weakness" condi
```

If the attacker chooses to take a different approach and plant a LNK file or binary in a startup location, the config would generate events for those. However, if you combine Rogue RDP with DLL sideloading the Olaf Hartong config is blind to two of the biggest chokepoints considering it also misses the network connections.

The reason for the difference between the two configs is that the SwiftOnSecurity config has an "include" statement for any TargetFileName ending in .dll.

<TargetFilename name="DLL" condition="end with">.dll</TargetFilename>

The Olaf Hartong config can be patched for this implicit exclusion by adding an image match condition for “mstsc.exe” under the FileCreate section. But adding a condition for TargetFileNames ending in “.dll” should also be strongly considered if you do not mind ingesting the extra telemetry against your SIEM license.

```
<!-- Event ID 11 == FileCreate - Includes -->
<RuleGroup groupRelation="or">
    <FileCreate onmatch="include">
        ...
        <Image condition="image">mstsc.exe</Image>
        ...
```

Splunk Rules

If you’ve managed to get your Sysmon config patched up and deployed, here are some implementations for detecting the events in Splunk.

Oh wait! One more problem.

While Microsoft Defender for Endpoint does a fantastic job of stitching together various telemetry to provide events with a more complete context, we will have to do some fiddling with Sysmon/Splunk to achieve the same events. We could skip that step, but we would lose fidelity as we wouldn’t be able to immediately see which mstsc.exe processes were started with a specified .rdp file due to missing CommandLine information.

We will solve this problem with some Splunk join statements to stitch together our own events.

Rule #1 - MSTSC Connection Over a Non-Standard Port Initiated by a RDP File

```
index=* source=WinEventLog:Sysmon EventCode=3 Image="*\\mstsc.exe" DestinationPort != 3389 Init
| table _time, ComputerName, EventCode, ProcessGuid, Image, CommandLine, DestinationIp, Destina
| join ProcessGuid ComputerName [search index=* source=WinEventLog:Sysmon EventCode=1 Image="*\
| search CommandLine="*.rdp*"
| iplocation DestinationIp | search Country=* | fields - City,Region,lat,lon
```

```
index=* source=WinEventLog:Sysmon EventCode=11 Image="*\\mstsc.exe" (TargetFilename="*.dll" OR T
| table _time, ComputerName, EventCode, Image, ProcessGuid, TargetFilename, CommandLine
| join ProcessGuid ComputerName [search index=* source=WinEventLog:Sysmon EventCode=1 Image="*\
| search CommandLine="*.rdp*"
```

Results

Let's put all this together by looking at the results of these Splunk rules using Sysmon data generated against the original and modified configurations.

You can download my modified version of the SwiftOnSecurity config here or the Olaf Hartong config here that have been patched with the required match conditions to A/B test against the originals.

Rule - MSTSC Connection Over a Non-Standard Port Initiated by a RDP File

```
EventCode: 3

Image: C:\Windows\System32\mstsc.exe

CommandLine: "mstsc.exe" "C:\Users\Administrator\Desktop\update.rdp"

DestinationIp: 10.10.30.50

DestinationPort: 443
```

Configuration	Version	Detected
SwiftOnSecurity	Original	No
SwiftOnSecurity	Patched	Yes
Olaf Hartong	Original	No

Olaf Hartong	Patched	Yes

Rule - Suspicious MSTSC File Creation Initiated by a RDP File

```
EventCode: 11

Image: C:\Windows\System32\mstsc.exe

CommandLine: "mstsc.exe" "C:\Users\Administrator\Desktop\update.rdp"

TargetFilename: "C:\Users\Administrator\AppData\Local\slack\app-4.24.0\cscapi.dll"
```

Configuration	Version	Detected
SwiftOnSecurity	Original	Yes
SwiftOnSecurity	Patched	Yes
Olaf Hartong	Original	No
Olaf Hartong	Patched	Yes

Summary

Thanks for sticking around through my first blog post.

To summarize,

- Adversaries can chain BlackHills Rogue RDP attack with DLL sideloading into common applications such as Teams to circumvent email/proxy controls, EDR, and even Sysmon.
- Detect Rogue RDP attacks by monitoring mstsc.exe for abnormal network connections and file write events.

- Consider blocking .rdp files at the email gateway.

Tweak the default Sysmon configs to obtain proper telemetry for this attack.

Authored by Nameless

View all posts by Nameless. Follow Nameless on Twitter @kxngcodes.

detection rogue_rdp sysmon mde evasion

« PREV PAGE
Realistic Phishing with PowerShell and a Mail Relay