AsyncRAT C2 Activity at Internet Scale - Censys

By Josue

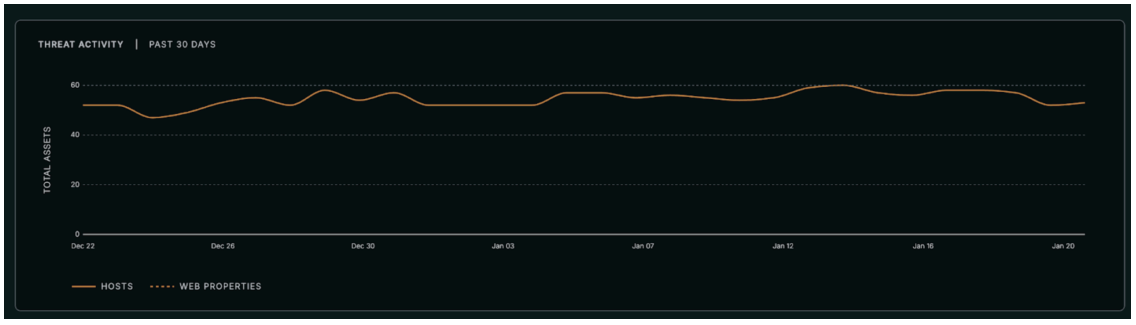
Published: 2026-01-29 · Archived: 2026-04-05 20:04:08 UTC

Executive Summary

AsyncRAT is an open-source .NET remote access trojan (RAT) implemented in C# and first released publicly in 2019. AsyncRAT has since become widely adopted by criminal operators for persistent remote access, surveillance, and data theft. The malware supports remote command execution, file transfer, keylogging, screen capture, and credential harvesting, typically communicating with command-and-control (C2) servers over a **custom TCP protocol** with traffic **encrypted via SSL/TLS**, often using **self-signed certificates** that may present **CN=AsyncRAT Server**.

As of January 2026, Censys is tracking **57 active AsyncRAT-associated hosts** exposed on the public internet. These hosts are primarily concentrated within a small number of VPS-focused autonomous systems and frequently reuse a distinctive self-signed TLS certificate identifying the service as an **"AsyncRAT Server,"** enabling scalable discovery of related infrastructure beyond sample-based detection.

Operationally, AsyncRAT enables long-lived unauthorized access and post-compromise control, making it a reliable tool for credential theft, lateral movement staging, and follow-on payload delivery.



Count of AsyncRAT assets in the Censys Platform Threat Hunt Module

Background

AsyncRAT was released publicly in 2019 by the developer known as **NYAN-x-CAT** and distributed via an open GitHub repository. Since its release, the codebase has been widely copied, modified, and redistributed across criminal communities, contributing to its persistent presence in commodity malware ecosystems.

The project has not undergone a formal rebrand in the way some families have, but it has spawned multiple closely related forks and derivatives, including **DCRat (DarkCrystal RAT)** and **VenomRAT**, which retain overlapping functionality and infrastructure patterns. This fragmentation complicates tracking when analysts rely solely on family names rather than shared technical artifacts.

AsyncRAT is most commonly delivered through malspam campaigns using compressed archives or document-based lures, as well as through loader chains that deploy the RAT as a secondary payload after initial execution. In some cases, AsyncRAT appears alongside other commodity tooling, enabling operators to blend its activity into high-volume background noise.

Capabilities

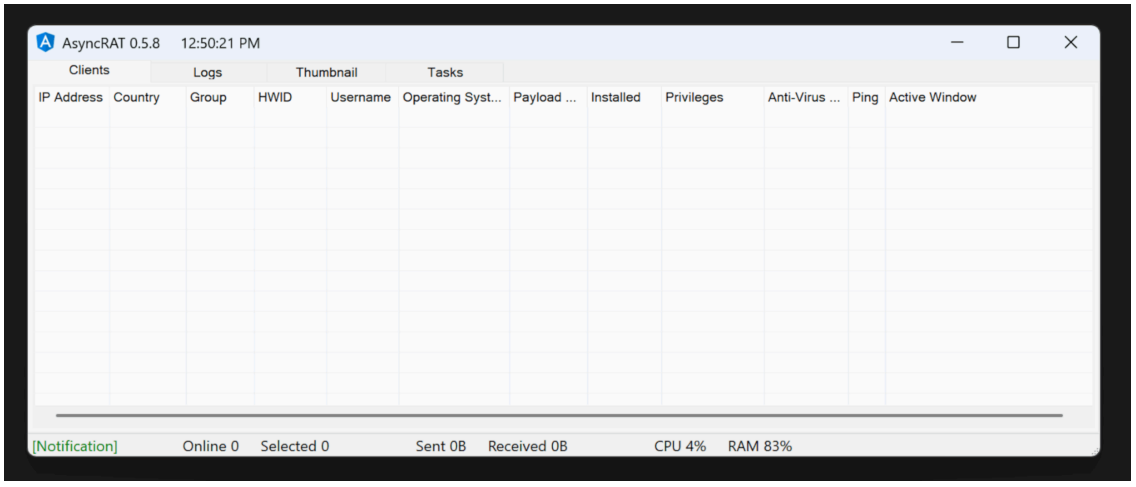
Once deployed, AsyncRAT provides operators with a broad set of post-compromise capabilities:

- Remote command execution and interactive shell access
- Credential theft via keylogging and memory access
- File upload, download, and arbitrary payload staging
- Persistence through scheduled tasks, registry run keys, or services
- Follow-on tooling deployment and lateral movement preparation

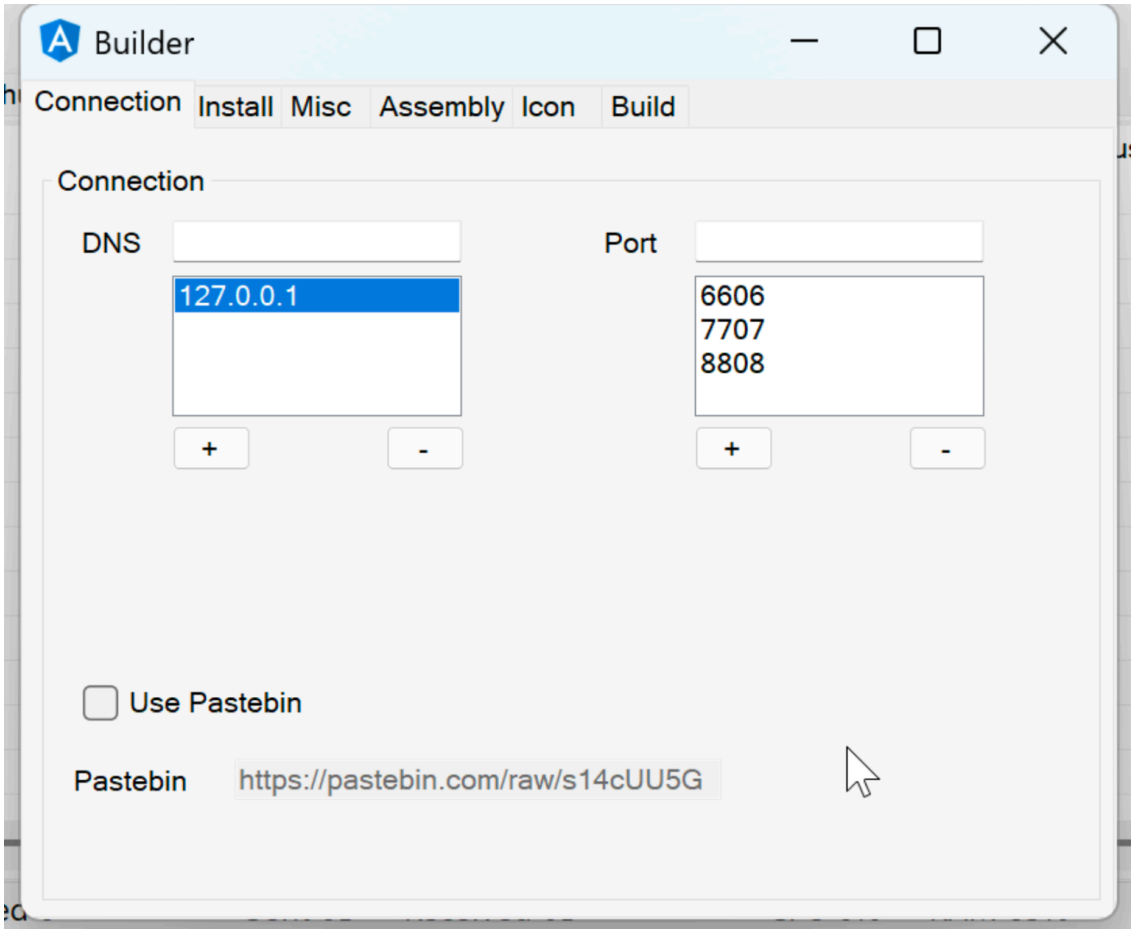
Technical Characteristics

AsyncRAT deployments exhibit several recurring technical traits observable across campaigns:

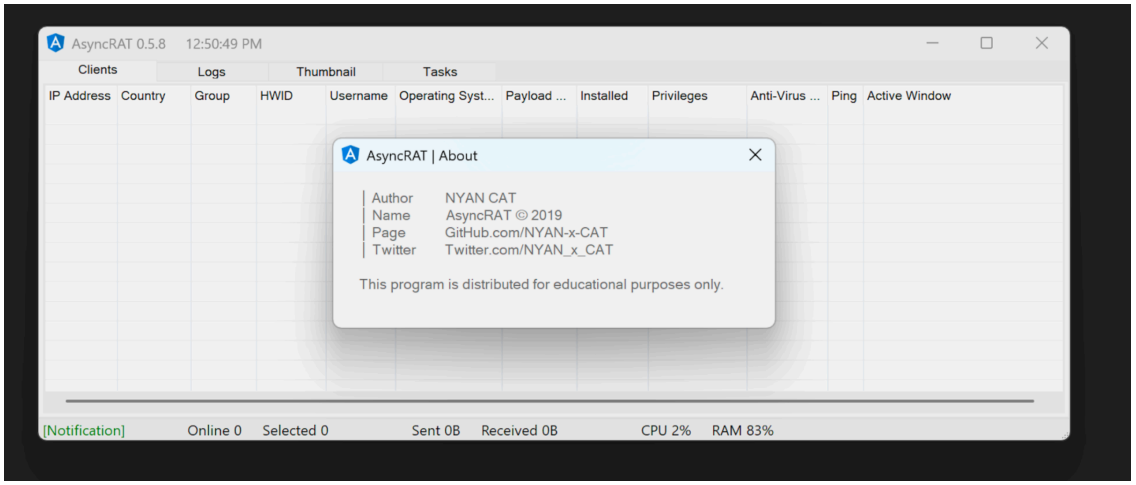
- **Common filenames and artifacts:** Though the AsyncRAT builder defaults to "**AsyncClient.exe**", operators frequently use generic or misleading executable names, or masqueraded system binaries; registry-based persistence commonly leverages standard Run key locations.
- **Network behavior:** AsyncRAT typically communicates with C2 servers over a **custom TCP protocol**, often exposed on non-standard ports such as 8808, 6606, and 7707, rather than embedding traffic within common application protocols
- **TLS characteristics:** Many deployments wrap C2 traffic in SSL/TLS using **self-signed certificates**, frequently presenting a common name such as **"AsyncRAT Server"**



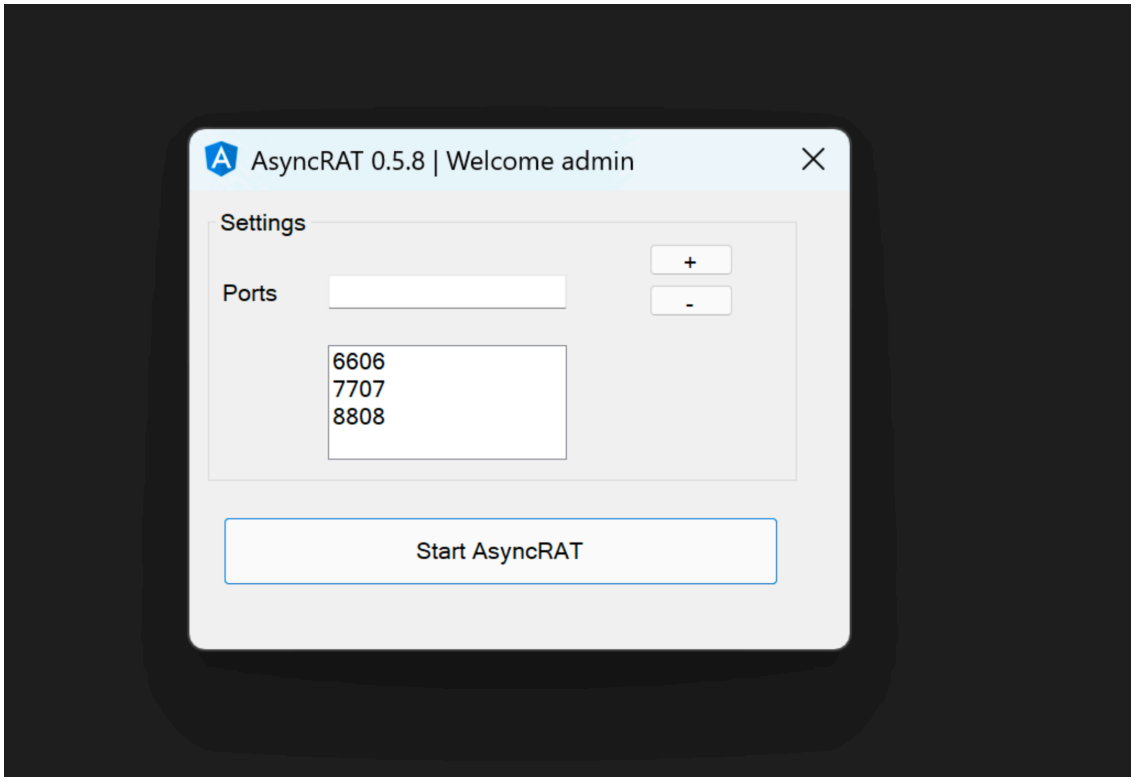
AsyncRAT Server Client



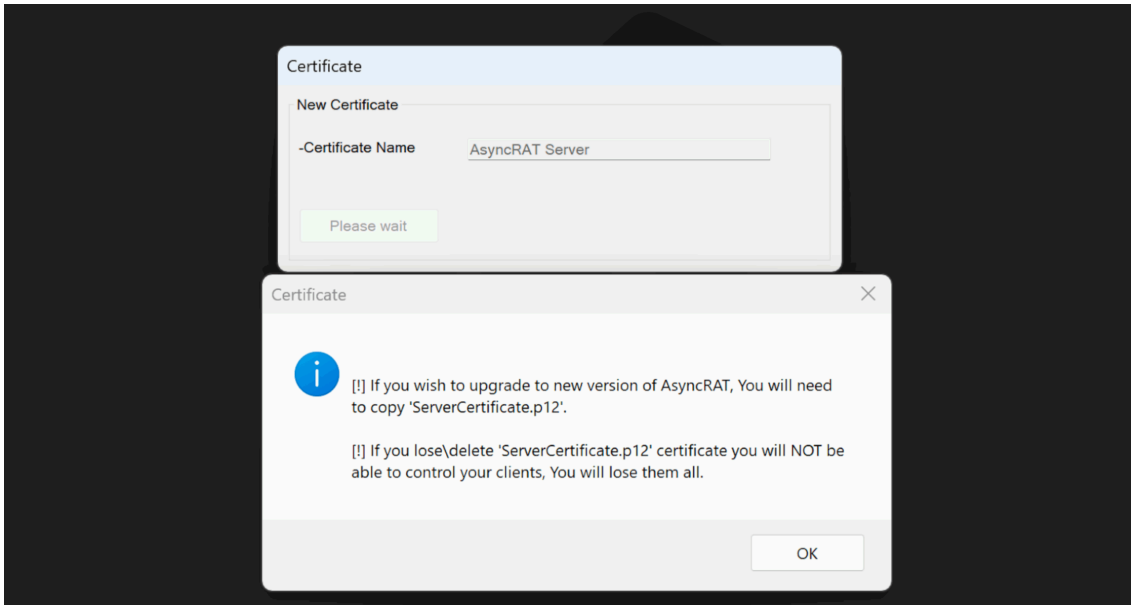
AsyncRAT client builder default connection options



AsyncRAT Server About Information



AsyncRAT Server Default Network Configuration



AsyncRAT Server Certificate Configuration and Default Certificate Name

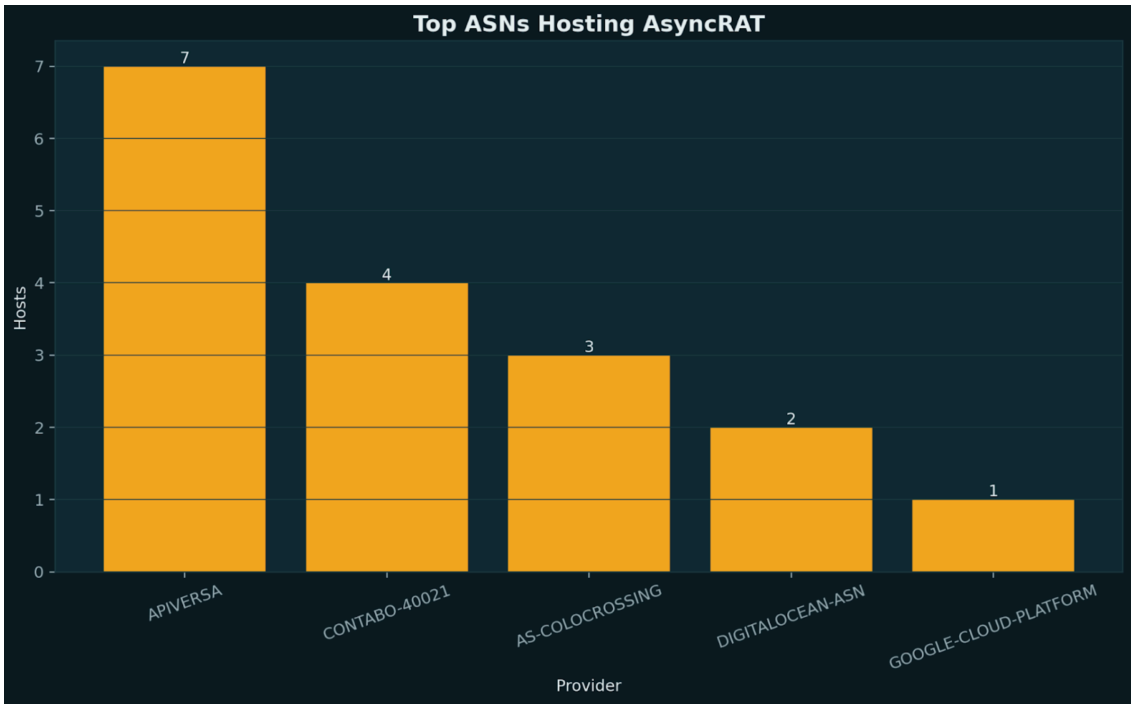
Operational Context

AsyncRAT remains relevant due to its low barrier to entry, ease of customization, and continued reuse across a wide range of operators. While often associated with opportunistic campaigns, its infrastructure and tooling have also appeared in more targeted activity, underscoring how commodity RATs continue to serve as building blocks for diverse threat models. Reuse of distinctive TLS artifacts and concentration within a limited set of VPS providers further amplifies its visibility at internet scale.

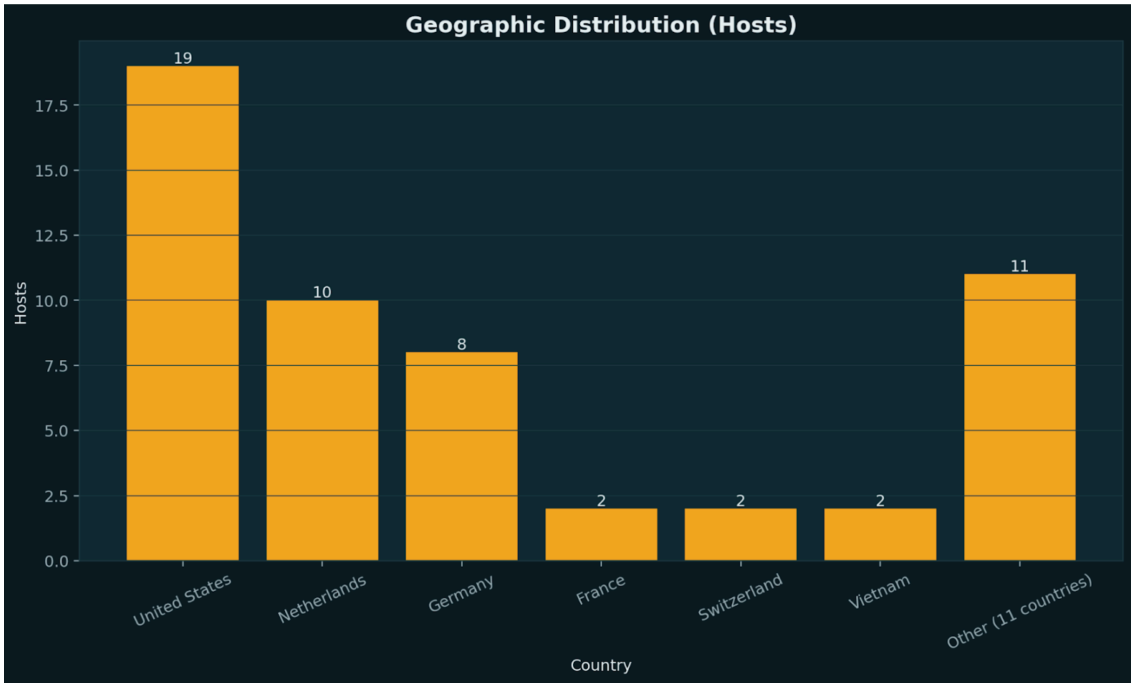
The following section summarizes Censys visibility into infrastructure and prevalence trends associated with AsyncRAT.

Censys Perspective

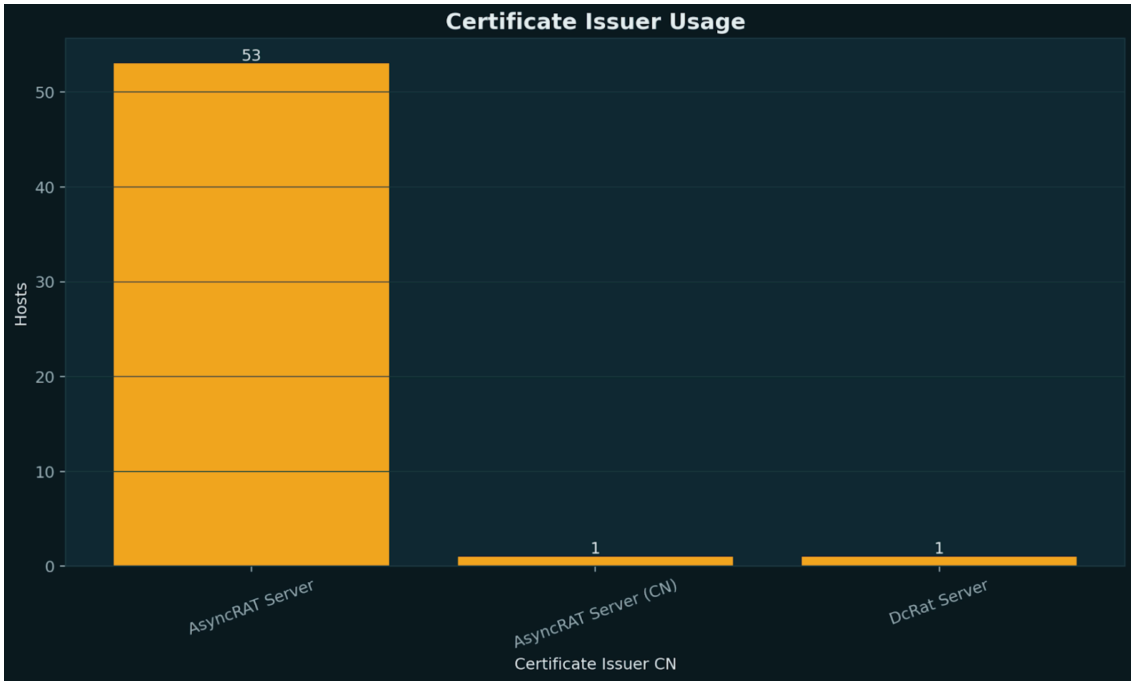
Out of the 57 total assets hosting AsyncRAT, we analyzed how they were distributed across infrastructure and found evidence of a decentralized hosting strategy favoring budget VPS providers and resellers. The dominance of APIVERSA (13% of hosts), Contabo networks (11% combined), and AS-COLOCROSSING (5.5%) indicates operators prioritize low-cost, abuse-tolerant hosting over major cloud providers.



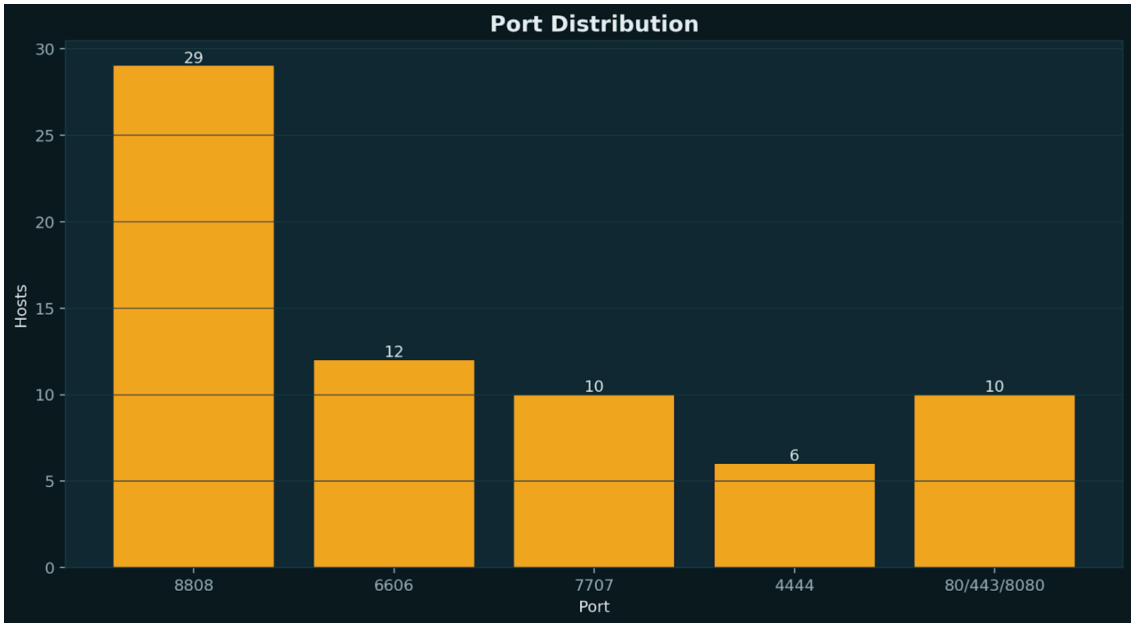
Geographic concentration in the US, Netherlands, and Germany aligns with data center density in these regions rather than operator location.



The near-universal use of the default “AsyncRAT Server” certificate (98%) suggests operators are deploying unmodified or lightly customized versions of the RAT, creating a highly reliable detection pivot.

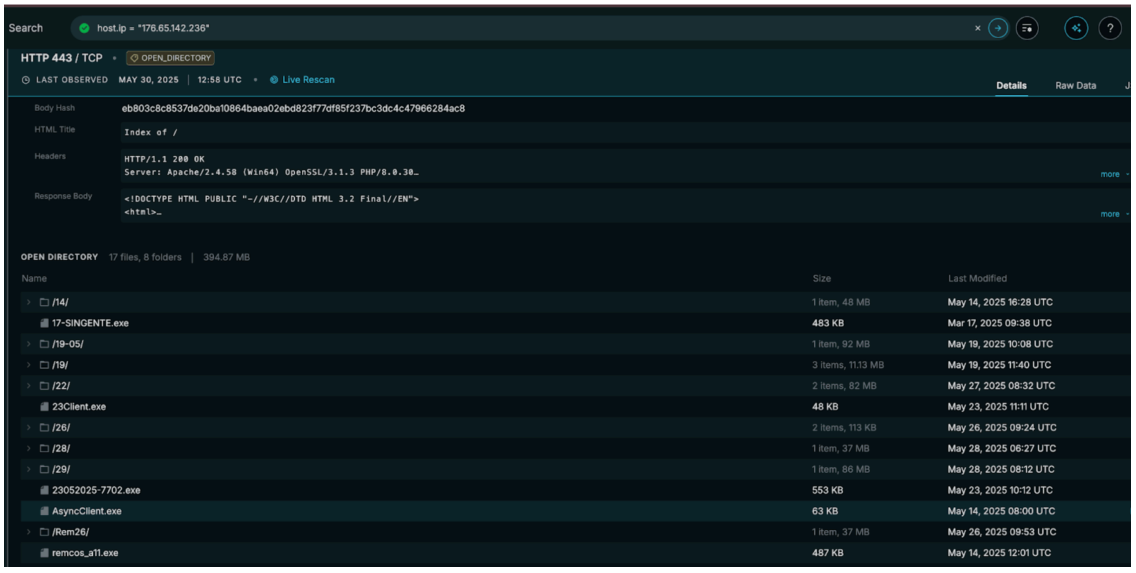


Multiple hosts running 3-5 AsyncRAT instances on sequential ports (e.g., 185.196.9.158 with ports 4501-4504) indicate either multi-campaign infrastructure or redundancy configurations.



Case Study

Another approach to hunting AsyncRAT with Censys Platform is to go after exposed clients. By searching for the known default name of the AsyncRAT client (AsyncClient.exe) we are able to discover AsyncRAT samples staged in **Open Directories** using the Censys Open Directory details view.



An AsyncRAT payload (client) hosted in an open directory

```
❯ rat-king-parser asynccl ient.exe | jq
[
  {
    "file_path": "asynccl ient.exe",
    "sha256": "f96745d8a8d89615cfcd9f8d8b9f6c34224d184f37c5203448eea9d4f9a39430",
    "yara_possible_family": "asyncrat",
    "key": "8451f4ddfa237d6ce804cf6cb2e6d893ea76f5ae976d49e2609432906cb2cf49",
    "salt": "bfeb1e56fbcd973bb219022430a57843003d5644d21e62b9d4f180e7e6c33941",
    "config": {
      "Hwid": "null",
      "Ports": "11005",
      "Hosts": "dvdpures1.duckdns.org",
      "Version": "| nelsontriana980",
      "Install": "false",
      "InstallFolder": "%AppData%",
      "InstallFile": "",
      "Key": "cnpSS0dieXhsUjNqS2VGWFZGWktqRUhHVjNkNkJXa28=",
      "MTX": "AsyncMutex_6SI8OkPnk",
      "Certificate": "MIIE8jCCAtqgAwIBAgIQAPeWQ4YJ3MvReCGwLzn7rTANBgkqhkiG9w0BAQ0FADAaMRgwFgYDVQQDDA9Bc3luY1JBVCBTZXJ2ZXIwIBcNMjIwNDI1MDA0MTA5WhgPOTk5OTEyMzEyM
zU5NTlaMBoxGDAWBgNVBAMMD0FzeW5jUkFUIFNlcnZlcjCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAKT9nYYTjYTZhY+g1tekZ8/F29gsEIDgf/8odvCbCmYKGGZZi2yND9NjtBXEMANM9PAXCyMa
pGvapDPbWgjYkL1Mw/Vwa3kZRg7kLpXMpzInLQufe7Q587viilcsGDoVXmnf51/SwsKPjSysZUpyayezUlJ1j6aXkZGnasiqJ7iKANdSneQducON6IwaEuJBmpXKWxhhq8R9JMfiWeOXL/hXoE/wCzwzvU/CrzP
XdJuMsLffMDHZJ+OQ9OXKU/CHZNCgSPs4VSgCgM4eK0YTbu1mLsWSo5th3/ingNFaTyYmGsmLIE2Jq5AR1A+xA+FEdCBzKL1bAwYQcRgIJs7QdedtAIufepPZ9D5HiOiy3ITYVonqwTiiIm20en7UICt+J8iDb4
M2Q2iLWA7Yi9PN2cr0Xrs8A4/RL29Qe5Ly2k35i74RiBTiT7Jbl2r7PcYlUGcjTCbdB9PWt3dYaTysuamoq2Zuo2HVRhhoZpwnajS9vNcjuZCYVoQvUQBUnHTeRZrtHXU5JV59ZBlu7flZneMZnbrWXTxob6Bdt
8+hrGoSDMWBFcO4jRzhT3hEfUpu4lSFeb9T3Vx4KWkHJhHtMvHuYgDTXERdEcI00sOUbVxgd/62LhGXNNommQKCyiAGj0V5uLD73Fyw8vJpm3jXf3NgNt/CjnlaMc40DJ+HlXE5AgMBAAGjMjAwMB0GA1UdDgQW
8BQsT2WvtxGUK29SWs4sHz1xYye0fzAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBDQUAA4ICAQCK5sVfnYyT5MqnCg3uHV2ojf12fIVFCY02Cc7gy3DVoE6/xZCPjr22V/xZunZ7DG1nt0kOJKDwdQYnGoM
c5UPh8jbNRoc1ojLOCaluaIYQyl8AGkmUSRA3Ltk0XetDescffrWT/nKuRvIEYU4Ra+B39f8ouGMCa7VXaxnGJ0z0BkUie8KsDLgNmJ7/kVfIYuRxl+YefoCsUTCogqf0fu3DuRHBpUVaSQQOf9YCbvFWH7Nupc
3UIwpH5D8kSdpKusEfbRp8nfWN/Fm+lzF3THeHU6vNJ+5UoAWHYFW8wfJCbzQ/0L8QZeOv4uy74oQP2Ed0RdrWCwUL6SSsDPZdDEOy4K4vYYkDTl1nL5tleATguELAEbbT42oLce85z4C7sKvpEfa4DPbU55xBL
wvHniILFfjB7VVsrgVckUL/lEf4Y92uJVKvLGruQt/mtKSqIuJjD8T9y7RIsk6g9624egV5UtLtv+36kLKhgIJlqC7Xx/PVwMc2yw8BiQlvxQZgqSd1k7QmV1AhV/3z2wqnYmb09ibTMYaMFjtamFegeFqc4jRL
ABhVQFEFv8z5E6G9vgKn5mQDWS/JykARBv9o2BjL/PTADfwAtc1b4nWo0l+CI8IjjYXu/mJOuwR+kFJ19INtwbffQvT9U12t4smpcZV+OK0opk4Yr9r1tZYm92ghXA==",
      "Serversignature": "Ge6Rs1BQKc2RESvo9WtDQ60PwAzpJXsrr+hG+nSg4o0g6SgXHtnLrS68wiBckIYAe2NaaU8JuyApBJ3I/BFDUJnvNnz1T7YN43Mvmawmyjqddf+TLYRLiIlqlFv8QXk96rlaw
I0kbXyK0MlxNUYgQMcPir8SRaUOWLOuKlmc6lLm4n2ekf1zxey/bNvNIu/SrrthziQKZBVohN+0ba4htpJojeccl0LEnzYEQ9SKSM9MaY7Ths434sOl1QzOIDpt1kZRtfMlxU/7lJQ8VbkSeUYjSKT7jI+GbeEZ
rZXO1NGZqY4oxpY0bpRLoUSWyT6qipeTvmmETAkTvJxzxp1Da+cjkRhPG998sUO+H2Zm+J2qDQRCLD2+HXa2YigF5h3YJin8kwC55CTsrBPoW6vY7b6D3ASF7CzvsZEp/rWzEbMGVsz8YFl592N1FV4sVbqm1oX
AqbvMTHDRUnzkdvaZr0OW393SJZBADGZzrA4D8a0CVCGjhCV7IcfO/NWU9oR6HzLpSNYyoAOw+osHzmXmnJUn02EUcMujHqPJpEbXOrZhvyjOiO+HwAYanIzlvu2pXOI9RfWajHpcaylxfJW7cPft8vtw3scJq0
EfR+Dai3A04xSeKjCqW7BpDSY3bKLqQ4y0pd+UWUsQGDsyrIXerUAG+4JcRYzo3xvW4L8BWt8=",
      "Anti": "false",
      "offlineKL": "false",
      "Pastebin": "null",
      "BDOS": "false",
      "Delay": "3",
      "Group": "LOADER NEW REVOLUTION"
    }
  }
]
```


Configuration using rat-king-parser validating this is an AsyncRAT sample

Static analysis confirms the payload is AsyncRAT. The sample implements the canonical AsyncRAT client architecture, including MessagePack-encoded command routing using a top-level Packet field, a plugin-based execution model that dynamically loads compressed assemblies in memory and invokes a standardized Plugin.Plugin.Run() entrypoint, and an AES-encrypted configuration schema containing the standard AsyncRAT fields (Hosts, Ports, MTX, Install, BDOS, Pastebin, Group). The client initializes a self-signed TLS channel using an embedded X.509 certificate whose decrypted subject is **"AsyncRAT Server"**, and passes that certificate directly into the client socket and plugin runtime.

To disambiguate between AsyncRAT and VenomRAT (which shares much of the original codebase) we searched for VenomRAT-specific modules. No VenomRAT-specific modules or configuration expansions (e.g., HVNC, clipper, Telegram control) were observed, supporting classification as AsyncRAT rather than a VenomRAT fork.


```
Settings
49            flag = ((RSACryptoServiceProvider)Settings.ServerCertificate.PublicKey.Key).VerifyHash(Sha256.ComputeHash(Encoding.UTF8.GetBytes(Settings.Key)), CryptoConfig.MapNameToOID("SHA256"),
                Convert.FromBase64String(Settings.Serversignature));
50            }
51            catch (Exception)
52            {
53                flag = false;
54            }
55            return flag;
56        }
57
58        // Token: 0x04000001 RID: 1
59        public static string Ports = "tY0YgeKth7EPhoa5ZDocGLv97zmyhjl+k/kE4OCgmgLwGimkK7X90PrwsRlH9dg7Q0zfFEwGtLPqReXCkk3RIg==";
60
61        // Token: 0x04000002 RID: 2
62        public static string Hosts = "caQh4eY9s9VmWZEN5rebf+zfitDcgCKyee/kapkIN1xnidfUV163vrkx2Lq0JrtCPaP1ynArR1zLRa5zNS4jikf3EV8ni4yl6/bN8tLaG08=";
63
64        // Token: 0x04000003 RID: 3
65        public static string Version = "A8VTnHK6ExF04WmPWO0p7I9EGYI9bgJmqam6kVxqYScgoIr7WLxeRo0d22EsM/dOgnkuc5DtsteNYL9Ray+S7OWuU5VxDAYUqFPwtBcOwN8=";
66
67        // Token: 0x04000004 RID: 4
68        public static string Install = "6lLDJvglg22WDXAlyTFeOy8VPpP2bSohyq9Tz+1jRs6mcEHtTOclceDzpQMsPZMDgrDUTYhg9DjKUAWKbk6PLA==";
69
70        // Token: 0x04000005 RID: 5
71        public static string InstallFolder = "%AppData%";
72
73        // Token: 0x04000006 RID: 6
74        public static string InstallFile = "";
75
76        // Token: 0x04000007 RID: 7
77        public static string Key = "cnpSS0dieXhsUjNqS2VGWFZGWktqRUhHVjNkNkJXa28=";
78
79        // Token: 0x04000008 RID: 8
80        public static string MTX = "tNf3s5XPgvPwaseowcCxmqyfvOYmKsRyZUm0cg/bMaqhLp66sRCkqOxKXt6OvKZcXYXmxcI3saLX6lcNVGVJQF9jPXaUZI0kagTXxIy24rg=";
81
82        // Token: 0x04000009 RID: 9
83        public static string Certificate = "42MKb3/zAjlxy4ucQ4uO0m7VTXt5XU41uJfdOkQVP1xq80Kwb93ggTCQh2TUey/Y796JF8Xl8XBUPz0lAYxnAhpbyRLI84kdMNB6qdEurQNRscl+usrgeP6oOQhPnAEoTZ2nw8PtU+12+Adjy0iVD5nefIuKklRQo
+pUWqfyukaiwvvK8P12eGocI/nIr5lFzPIKNA7/B0bTEFw+3XCmGfgeLmUxMeHvCIcRa/zWWLO30dAG4Wtz/DvEszR8JpZC1WX5U2jOwxdDWd8J5eRAS2N6Y6MENYpce+oYTaADRZgKydXi1BeVEH1RfiJzyQVewN9w7MIxTXt11DeDm
+pbPeGyvrfiRYCyyd170NUN1iRSHmWynJV5ZgQU9ajEiZgB3lCd4Z7xdb1uxhrJK24smqoupSe9i3cYVAqw7LUtMOlEGnNCLyfp8nc6j5iGOq7WzF48+UbmY9683TJyNkaCY/gVeRmkeNi2ycKEbmeVmzjjotHAVCHPexy87dFi+L2
notepad++
Analyzer
Client.Settings.Hosts : string @04000002
Assigned By
Client.Connection.ClientSocket.InitializeClient() : void @0600001B
Client.Settings..cctor() : void @06000005
Client.Settings.InitializeSettings() : bool @06000003
Read By
Client.Connection.ClientSocket.InitializeClient() : void @0600001B
Client.Settings.InitializeSettings() : bool @06000003
```

Manual validation of configuration in dnSpy showing the AES256 encrypted strings

```
PS C:\Users\admin > # Key and ciphertext
>> $keyB64 = "cnpSS0dieXhsUjNqS2VGWFZGWktqRUhHVjNkNkJXa28="
>> $cipherB64 = "caQh4eY9s9VmWZEN5rebf+zfitDcgCKyee/kapkINlxnidfUVi63vrkx2Lq0JrtCPaP1ynArR1zLRa5zNS4jikf3EV8ni4yl6/bN8tLaG08="
>>
>> # Decode the master key
>> $masterKey = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($keyB64))
>>
>> # AsyncRAT hardcoded salt
>> $salt = [byte[]]@(191, 235, 30, 86, 251, 205, 151, 59, 178, 25, 2, 36, 48, 165, 120, 67, 0, 61, 86, 68, 210, 30, 98, 185, 212, 241, 128, 231, 230, 195, 57, 65)
>>
>> # Derive keys using PBKDF2 (50000 iterations)
>> $pbkdf2 = New-Object System.Security.Cryptography.Rfc2898DeriveBytes($masterKey, $salt, 50000)
>> $aesKey = $pbkdf2.GetBytes(32)    # AES-256 key
>> $authKey = $pbkdf2.GetBytes(64)   # HMAC key
>>
>> # Decode ciphertext
>> $cipherBytes = [System.Convert]::FromBase64String($cipherB64)
>>
>> # Structure: [32 bytes HMAC] [16 bytes IV] [encrypted data]
>> $hmac = $cipherBytes[0..31]
>> $iv = $cipherBytes[32..47]
>> $encrypted = $cipherBytes[48..($cipherBytes.Length - 1)]
>>
>> # Decrypt AES-CBC
>> $aes = [System.Security.Cryptography.Aes]::Create()
>> $aes.KeySize = 256
>> $aes.BlockSize = 128
>> $aes.Mode = [System.Security.Cryptography.CipherMode]::CBC
>> $aes.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
>> $aes.Key = $aesKey
>> $aes.IV = $iv
>>
>> $decryptor = $aes.CreateDecryptor()
>> $decrypted = $decryptor.TransformFinalBlock($encrypted, 0, $encrypted.Length)
>> $plaintext = [System.Text.Encoding]::UTF8.GetString($decrypted)
>>
>> Write-Output $plaintext
dvdpures1.duckdns.org
```

Manual decryption of ‘Hosts’ value using Powershell to validate rat-king-parser extraction

Notable Findings

The presence of a Chinese-localized certificate variant (“AsyncRAT 服务器”) and a “bullet-proof” hostname in certificate data indicates geographic expansion of operator demographics beyond the typical Eastern European and Latin American threat actor communities historically associated with AsyncRAT.

These infrastructure patterns inform how defenders can prioritize detection and blocking, detailed in the following section.

Andrew Northern

Principal Security Researcher

Andrew Northern is a Principal Security Researcher with Censys ARC focused on tracking the apex predators of the initial-access e-crime landscape. His work targets the most capable operators, uncovering novel attack chains and dynamic web-delivered malware while mapping the infrastructure that enables them. He has earned multiple MITRE ATT&CK citations, discovered and named several espionage-focused malware families, and published research that exposes previously unknown tradecraft.

Source: https://censys.com/blog/asyncrat-c2-activity-at-internet-scale