How UNC6692 Employed Social Engineering to Deploy a Custom Malware Suite

By Mandiant

Published: 2026-04-23 · Archived: 2026-05-01 02:03:35 UTC

Written by: JP Glab, Tufail Ahmed, Josh Kelley, Muhammad Umair

---

Introduction

Google Threat Intelligence Group (GTIG) identified a multistage intrusion campaign by a newly tracked threat group, UNC6692, that leveraged persistent social engineering, a custom modular malware suite, and deft pivoting inside the victim's environment to achieve deep network penetration.

As with many other intrusions in recent years, UNC6692 relied heavily on impersonating IT helpdesk employees, convincing their victim to accept a Microsoft Teams chat invitation from an account outside their organization. The UNC6692 campaign demonstrates an interesting evolution in tactics, particularly the use of social engineering, custom malware, and a malicious browser extension, playing on the victim's inherent trust in several different enterprise software providers.

Threat Details

In late December 2025, UNC6692 conducted a large email campaign designed to overwhelm the target with messages, creating a sense of urgency and distraction. Following this, the attacker sent a phishing message via Microsoft Teams, posing as helpdesk personnel offering assistance with the email volume.

Infection Chain

The victim was contacted through Microsoft Teams and was prompted to click a link to install a local patch that prevents email spamming. Once clicked, the user's browser opened an HTML page and ultimately downloaded a renamed AutoHotKey binary and an AutoHotkey script, sharing the same name, from a threat actor-controlled AWS S3 bucket.

```
 "url": "https://service-page-25144-30466-outlook.s3.us-west-2.amazonaws.com/update.html?email=<redacted>.com",
"description": "Microsoft Spam Filter Updates | Install the local patch to protect your account from email spamming",
```

Figure 1: Snippet from MS Team Logs

If the AutoHotkey binary is named the same as a script file in its current directory, AutoHotkey will automatically run the script with no additional command line arguments. Evidence of AutoHotKey execution was recorded immediately following the downloads resulting in initial reconnaissance commands and the installation of SNOWBELT, a malicious Chromium browser extension (not distributed through the Chrome Web Store). Mandiant was unable to recover the initial AutoHotKey script.

The persistence of SNOWBELT was established in multiple ways. First, a shortcut to an AutoHotKey script was added to the Windows Startup folder, which verified SNOWBELT was running and that a Scheduled Task was present.

```
if !CheckHeadlessEdge(){
  try{
     taskService:=ComObject("Schedule.Service")
     taskService.Connect()
     rootFolder:=taskService.GetFolder("\")
     if FindAndRunTask(rootFolder){
        Sleep 10000
```

```
         if CheckHeadlessEdge(){
         ExitApp
         }
      }
   }
   Run 'cmd /c start "" "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --user-data-dir="%LOCALAPPDATA%\Mic
}
ExitApp
```

Figure 2: Snippet from AutoHotKey script to verify SNOWBELT was running and to start it if not

Second, two additional scheduled tasks were installed. One task to start a windowless Microsoft Edge process that loads the SNOWBELT extension and another to identify and terminate Microsoft Edge processes that do not have CoreUIComponents.dll loaded.

```
 <Exec>
    <Command>
        "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
    </Command>
    <Arguments>
       --user-data-dir="C:\Users\<redacted>\AppData\Local\Microsoft\Edge\System Data"
       --no-first-run
       --load-extension="C:\Users\<redacted>\AppData\Local\Microsoft\Edge\Extension Data\SysEvents"
       --headless=new --disable-sync
    </Arguments>
</Exec>
```

Figure 3: Snippet from the scheduled task to start the SNOWBELT extension windowless Microsoft Edge

Microsoft Edge processes without CoreUIComponents.dll are typically headless. The threat actor uses this command to essentially “clean up” headless Edge processes that execute their malware.

```
 <Exec>
    <Command>cmd</Command>
    <Arguments>
    /c "for /f "tokens=2" %p in ('tasklist /M SHELL32.dll ^| findstr "msedge.exe"') do @(tasklist /M CoreUIComponents.dll
    </Arguments>
</Exec>
```

Figure 4: Snippet from the scheduled task to check for CoreUIComponents.dll

Using the SNOWBELT extension, UNC6692 downloaded additional files including SNOWGLAZE, SNOWBASIN, AutoHotkey scripts, and a ZIP archive containing a portable Python executable and required libraries.

Internal Recon and Lateral Movement

After gaining initial access, process execution telemetry recorded UNC6692 using a Python script to scan the local network for ports 135, 445, and 3389. Following internal port scanning, the threat actor established a Sysinternals PsExec session to the victims system via the SNOWGLAZE tunnel, and executed commands to enumerate local administrator accounts. Using the local administrator account, the threat actor initiated an RDP session via the SNOWGLAZE tunnel from the victim system to a backup server. Though not directly observed, the threat actor may have acquired the local administrator accounts credentials via multiple attack paths such as authenticated Server Message Block (SMB) share enumeration.

Escalate Privileges

After gaining access to the backup server the threat actor utilized the local administrator account to extract the system's LSASS process memory with Windows Task Manager. Microsoft Windows Local Security Authority Subsystem Service

(LSASS) process `lsass.exe` enforces security policy and contains usernames, passwords and hashes for accounts that have accessed the system. After extracting the process memory, UNC6692 exfiltrated it via LimeWire. With the process memory out of the victim environment UNC6692 is able to use offensive security tools to extract the credentials while not having to worry about being detected.

Complete Mission

Now armed with the password hashes of elevated users, UNC6692 used Pass-The-Hash to move laterally to the network's domain controllers. Pass-The-Hash is a common technique used by threat actors where the NTLM hash is passed to another system, instead of providing the account password, allowing for authentication via NTLM. Once authenticated to the Domain Controller, the threat actor opened Microsoft Edge, and downloaded a ZIP archive containing FTK Imager to the Domain Administrator's `\Downloads` folder. The threat actor executed FTK Imager and mounted the local storage drive. Subsequently, FTK Imager wrote the Active Directory database file (NTDS.dit), Security Account Manager (SAM) , SYSTEM, and SECURITY registry hives to the `\Downloads` folder. The extracted files were then exfiltrated from the network via LimeWire. Finally, EDR telemetry logged the threat actor performing screen captures on the Domain Controllers, specifically targeting in-focus instances of Microsoft Edge and FTK Imager.

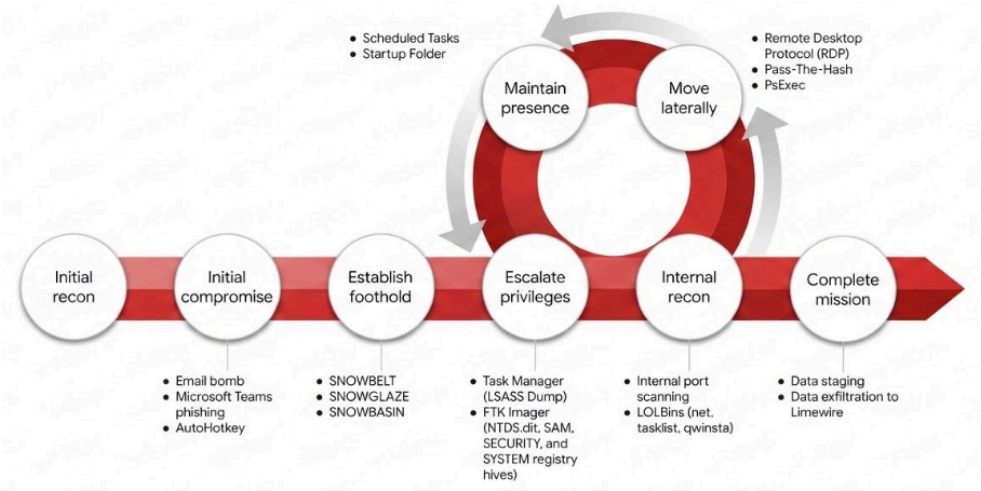

Figure 5: UNC6692 attack lifecycle

THE SNOW Ecosystem

Phishing Landing Page

The original phishing link (`https://service-page-25144-30466-outlook.s3.us-west-2.amazonaws.com/update.html?email=<redacted>.com`) delivered via Microsoft Teams directs the victim to a landing page masquerading as a "Mailbox Repair Utility." This interface is designed to elicit user engagement through various on-screen buttons.

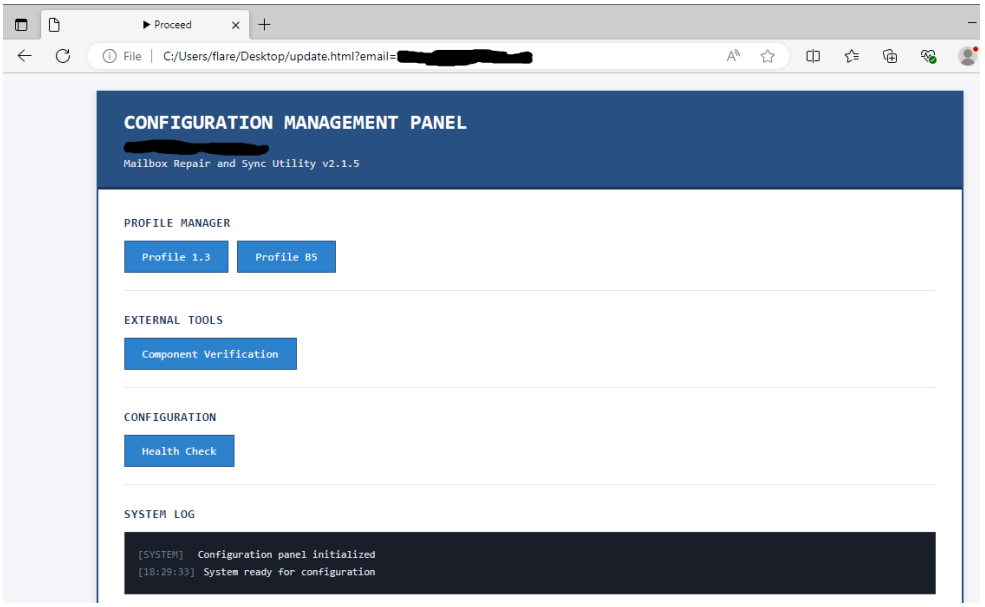

Figure 6: The landing page masquerading as an official "Mailbox Repair and Sync Utility v2.1.5."

Phase 1: Environment Enforcement and Anti-Analysis

The attacker used a gatekeeper script designed to ensure the payload is delivered only to intended targets while evading automated security sandboxes. Upon loading, the landing page executes an `init()` function that inspects the URL for a mandatory `?email=` parameter. If this parameter is absent, the page immediately redirects to `about:blank.`

The script also checks the victim's browser. If the user is not using Microsoft Edge, the page displays a persistent overlay warning. This forces the user to click an "Open in Edge" button, which triggers the `microsoft-edge:` URI scheme. This ensures the victim is moved from potentially secure mobile or third-party browser environments into a specific workspace where the attacker's exploits are most effective.

Phase 2: Credential Harvesting via Social Engineering

Once the environment is established, the page presents a professional-looking "Configuration Management Panel" masquerading as an official "Mailbox Repair and Sync Utility." The primary hook is a "Health Check" button that, when clicked, triggers an "Authentication Required" modal.

The harvesting script, handleAuthFormSubmit, employs a "double-entry" psychological trick. It is programmed to reject the first and second password attempt as incorrect. This serves two functions: it reinforces the user's belief that the system is legitimate and performs real-time validation, and it ensures that the attacker captures the password twice, significantly reducing the risk of a typo in the stolen data. A screenshot of authentication is shown in Figure 7, and the email supplied is entered by default.

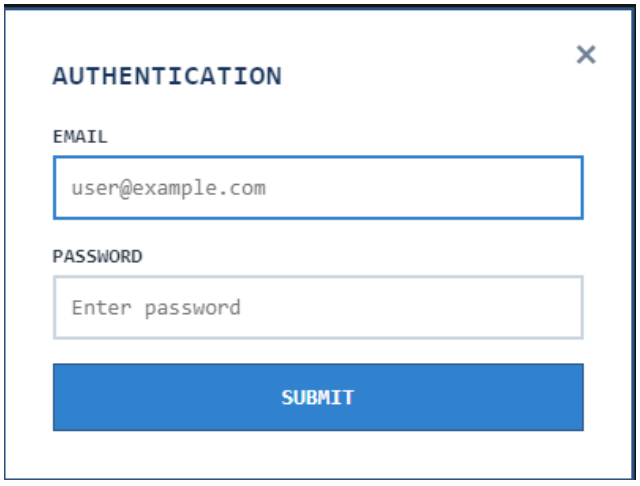

Figure 7: The credential harvesting prompt triggered by the "Health Check" button

Phase 3: Data Exfiltration and Distraction Sequences

Upon successful submission, the script executes an asynchronous PUT request using AWS URLs. The validated credentials and metadata are uploaded directly to an attacker-controlled Amazon S3 bucket (e.g., `service-page-18968-2419-outlook.s3.us-west-2.amazonaws.com`), which have since been taken down. These buckets serve as the command and control (C2) infrastructure and represent critical indicators of compromise (IOCs).

To mask this background activity and prevent user suspicion, the script initiates a startProgressBar function. This displays a scripted distraction sequence featuring fake technical tasks such as "Parsing configuration data" and "Checking mailbox integrity." This manipulation keeps the victim engaged until the data transfer is complete.

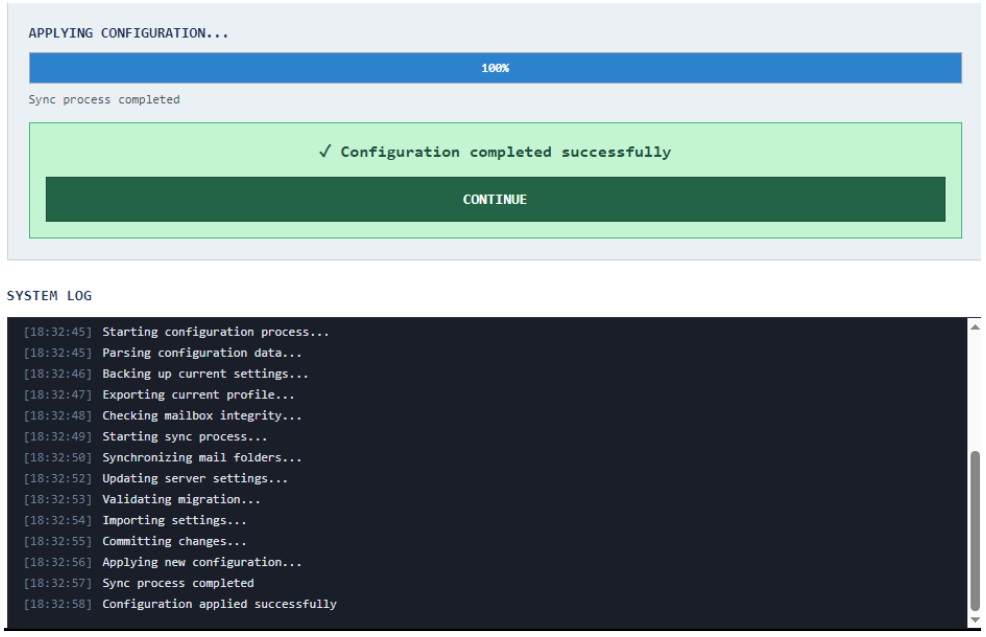

Figure 8: A scripted distraction sequence used to mask the background exfiltration of stolen data

Phase 4: Malware Staging and Endpoint Foothold

The final stage involves the delivery of secondary malicious payloads referenced within the CONFIG object of the script. While the progress bar runs, the site is prepared to deliver files seen in Table 1.

Button Clicked	File Downloaded	Type / Risk
Profile 1.3	Protected.ahk	**AutoHotKey Script:** Not found during the investigation, but suspected to install SNOWBELT.
Profile B5	profileB5.txt	Likely a configuration file for the malware.
Component Verification	RegSrvc.exe	**AutoHotKey Executable:** Masquerading as a "Registration Service."
Health Check	N/A	Prompts the user to input email credentials. Exfiltrates the credentials to Amazon S3 bucket.

Table 1: Buttons on the landing page

By the time the user receives a "Configuration completed successfully" message, the attacker has secured the credentials and potentially established a persistent foothold on the endpoint using these staged files.

The SNOW malware ecosystem, attributed to the threat cluster UNC6692, operates as a modular ecosystem comprising three primary components: SNOWBELT, SNOWGLAZE, and SNOWBASIN. Rather than functioning as isolated tools, these components form a coordinated pipeline that facilitates an attacker's journey from initial browser-based access to the internal network of the organization.

Figure 9: The SNOW ecosystem

1.SNOWBELT (Browser Extension)

SNOWBELT serves as the initial foothold and the primary "eyes" of the operation. It is a JavaScript-based backdoor delivered as a Chromium browser extension, often masquerading under names like "MS Heartbeat" or "System Heartbeat". Rather than being available through the Chrome Web Store, the extension is deployed through social engineering tactics.

- **Role:** It is designed to intercept commands and send them to SNOWBASIN for execution . It maintains persistence via the browser's extension registration system and uses Service Worker Alarms and Keep-Alive Tab Injection (via helper.html) to ensure it remains active whenever the browser is running.
- **Functionality:** By relaying commands from the threat actor to SNOWBASIN, SNOWBELT provides authenticated access to the environment. This allows the attacker to move laterally and escalate privileges without the need for

constant re-authentication.

2.SNOWGLAZE (Python Tunneler)

Once a foothold is established, SNOWGLAZE is deployed to manage the logistics of external communication. SNOWGLAZE is a Python-based tunneler that can operate in both Windows and Linux environments.

- **Role:** Its primary function is to create a secure, authenticated WebSocket tunnel between the victim's internal network and the attacker's command-and-control (C2) infrastructure, such as a Heroku subdomain. It facilitates SOCKS proxy operations, allowing arbitrary TCP traffic to be routed through the infected host.
- **Functionality:** SNOWGLAZE masks malicious traffic by wrapping data in JSON objects and Base64 encoding it for transfer via WebSockets. This makes the activity appear as standard encrypted web traffic. When attackers wish to interact with backdoors like SNOWBASIN or exfiltrate staged data, traffic is routed through this established tunnel.

3.SNOWBASIN (Python Bindshell)

While SNOWBELT monitors the user and SNOWGLAZE bridges the network gap, SNOWBASIN provides the functional interactive control over the infected system.

- **Role:** It acts as a persistent backdoor that operates as a local HTTP server (typically listening on port 8000). It enables remote command execution via cmd.exe or powershell.exe, screenshot capture, and data staging for exfiltration.
- **Functionality:** This component is where active reconnaissance and mission completion occur. Attacker commands (such as whoami or net user) are sent through the SNOWGLAZE tunnel, intercepted by the SNOWBELT extension, and then proxied to the SNOWBASIN local server via HTTP POST requests. SNOWBASIN executes these commands and relays the results back through the same pipeline to the attacker.

Malware Analysis

SNOWBELT

SNOWBELT is a JavaScript-based backdoor implemented as a Chromium browser extension. Its lifecycle begins with the execution of the background.js Service Worker upon installation, which leverages the browser's extension registration system for persistence. To ensure continuous operation while the browser is active, the malware utilizes Service Worker Alarms (agent-heartbeat) and Keep-Alive Tab Injection (helper.html).

Upon initialization, the malware generates a unique identity using the prefix fp-sw- followed by a UUID. It then employs a time-based DGA to calculate C2 URLs. Using a hard-coded seed value (`691f7258f212fa8908a8bf06bcf9e027d2177276e13e10ff56bd434ff3755cc4`), it generates a registry URL for an S3 bucket within 30-minute time slots. These URLs follow a specific structural pattern:

- `https://[a-f0-9]{24}-[0-9]{6,7}-{0-9}{1}.s3.us-east-2.amazonaws[.]com`

The manifest retrieved from this registry is decrypted via AES-GCM using a key derived from SHA256(SEED + "|" + timeslot).

For low-latency C2, SNOWBELT registers with the browser's Push Notification service. This is achieved using a hard-coded VAPID Public Key:

```
BJkWCT45mL0uvV3AssRaq9Gn7iE2N7Lx38ZmWDFCjwhz0zv0QSVhKuZBLTTgAijB12cgzMzqyiJZr5tokRzSJu0
```

This setup provides an asynchronous channel that allows attackers to "wake up" the Service Worker immediately via authenticated Push messages, bypassing standard polling. Additionally, the malware supports real-time interaction through a persistent REGISTRY_WEBSOCKET_URL connection.

SNOWBELT functions in coordination with SNOWBASIN, a backdoor acting as a local web server (typically on port 8000). It relays decrypted C2 commands—such as command, buffer, flush, and commit—to SNOWBASIN via HTTP POST requests, effectively proxying shell commands to the host system.

The malware also includes mechanisms to bypass the browser sandbox:

1. **Native Host Bridge (open_native_messaging):** Uses chrome.runtime.connectNative to establish I/O pipes with local applications for issuing privileged commands.
2. **Protocol Handler Abuse (open_uri):** Employs dream.html and dream.js to trigger custom URI schemes in new tabs, targeting vulnerabilities in third-party desktop applications.

Exfiltration is managed by the sendJsonDataToS3 function, which encrypts data with AES-GCM (Key: SHA256(SEED + "|ping|" + bucket + "|" + objectKey)) before uploading to S3. The backdoor's command set is summarized in Table 2.

Command Type	Description
command	**Relayed**: Decrypts and POSTs command text to SNOWBASIN; exfiltrates response to C2.
buffer	**Relayed**: Forwards file path payloads to local buffer endpoint.
flush	**Relayed**: Triggers a data flush on the local server.
commit	**Relayed**: Sends URL and path data for local processing.
stop_server	**Relayed**: Shutdown signal for the local SNOWBASIN instance.
screenshot	**Relayed**: Requests a screen capture from the host.
payload	**Internal**: Downloads files using chrome.downloads; supports URLs and base64 blobs.
open_native_messaging	**Internal**: Direct connection to native host apps via Chrome APIs.
open_uri	**Internal**: Triggers external protocol handlers via helper pages.
delete_cache	**Internal**: Removes downloaded files from the system.
websocket_control	**Internal**: Controls the state of WebSocket connectivity.
ping	**Internal**: Provides heartbeats and status updates to the C2.

Table 2: SNOWBELT commands

Finally, SNOWBELT implements a feedback loop by monitoring chrome.downloads.onChanged. If a download is blocked (e.g., FILE_VIRUS_INFECTED), the malware reports the error back to the S3-based C2.

SNOWBASIN

SNOWBASIN is a Python-based backdoor that operates as a local HTTP server on ports 8000, 8001, or 8002. Its core capabilities include command execution, screenshot capture, and data exfiltration. The malware also enables operators to manage files by downloading or deleting them, and it provides the capability to terminate active connections. SNOWBELT relays commands to this malware by sending HTTP requests to localhost:8000.

It turns the victim's computer into a command-and-control (C2) node that can be controlled via HTTP requests. It is designed to run on Windows (evidenced by os.chdir('C:\\') and cmd.exe calls) and allows a remote actor to execute commands, steal files, and take screenshots.

Endpoint	Function	Description
/stream	**Remote Shell**	Receives a command and executes it via cmd.exe or powershell.exe. It returns the STDOUT/STDERR results to the attacker.
/buffer	**File Exfiltration**	If a file path is provided, it reads the file, encodes it in Base64, and sends it back. If a folder is provided, it returns a full directory listing
/flush	**File Deletion**	Relayed. Signals http://localhost[:]8000/flush to flush buffered data.
/commit	**File Ingress**	Downloads a file from a provided URL and saves it to a specific path on the local disk. It bypasses SSL certificate verification (CERT_NONE).
/capture	**Take Screenshots**	Uses the mss and PIL libraries to take a screenshot of all monitors and send the image back as a Base64 string.
/gc	**Self-Termination**	Shuts down the server instance, effectively ""killing"" the backdoor's connection.

Table 3: SNOWBASIN endpoints

SNOWGLAZE

The network tunneler SNOWGLAZE, developed in Python, facilitates the routing of arbitrary TCP traffic through a compromised system by establishing a WebSocket connection to a static C2 host using hard-coded credentials.

The script is designed for cross-platform execution on both Windows and Linux, utilizing environment-specific behaviors for each. In Windows environments, it runs as a foreground process manageable via standard keyboard interrupts (Ctrl-C). Conversely, on Linux, it operates as a background daemon and includes specific logic to handle SIGINT and SIGTERM signals for orderly shutdowns.

To establish communication, the malware targets the C2 server at wss://sad4w7h913-b4a57f9c36eb[.]herokuapp[.]com:443/ws, masquerading its traffic with a Microsoft Edge User-Agent string. If the initial connection fails, the script employs an incremental backoff strategy, starting at 5 seconds and increasing by 5-second intervals up to a 300-second maximum. Upon a successful WebSocket handshake, it transmits the following Auth payload:

```
{
   "type": "auth",
   "login": "<redacted",
   "password": "<redacted",
   "uuid": "<redacted>"
}
```

Following authentication, the script sends a "register" type message with no payload, followed by an "agent_info" JSON record. Although the "info" field within this record is intended to carry the public IP address, it remains unpopulated due to improper implementation in the script.

Once fully connected, the malware listens for JSON-formatted commands. The supported "type" values include:

- ping
    - Prompts the script to return a "type": "pong" JSON object.
- agent_public_ip
    - Intended to report the host's public IP via an agent_info structure; however, the IP field is consistently blank in current versions.
- socks_connect
    - Requests a new SOCKS proxy connection using a unique conn_id provided by the operator to track the session. The request format is as follows:

```
{
   "type": "socks_connect",
   "conn_id": "<unique_connection_id>",
   "target_host": "example.com",
   "target_port": 80
}
```

    - Execution triggers an asynchronous worker thread that manages the TCP-to-WebSocket data transfer, utilizing Base64 encoding and JSON encapsulation with the socks_data type.
- socks_data
    - Facilitates bidirectional data exchange between the WebSocket and the TCP socket. Data is Base64-encoded within the data field of the following structure:

```
   {
      "type": "socks_data",
      "conn_id": "<unique_connection_id>",
      "data": "bG9yZW0gaXBzdW0="
   }
```

- socks_close
    - Terminates the specific proxy stream identified by the given conn_id.
- disconnect
    - Serves all active proxy connections and terminates script execution.

Outlook & Implications

The UNC6692 campaign demonstrates how modern attackers blend social engineering and technical evasion to gain a foothold into environments. A critical element of this strategy is the systematic abuse of legitimate cloud services for payload delivery and exfiltration, and for command-and-control (C2) infrastructure. By hosting malicious components on trusted cloud platforms, attackers can often bypass traditional network reputation filters and blend into the high volume of legitimate cloud traffic.

This "living off the cloud" strategy allows attackers to blend malicious operations into a high volume of encrypted, reputably sourced traffic, making detection based on domain reputation or IP blocking increasingly ineffective. Defenders must now look beyond process monitoring to gain clear visibility into browser activity and unauthorized cloud traffic. As threat actors continue to professionalize these modular, cross-platform methodologies, the ability to correlate disparate events across the browser, local Python environments, and cloud egress points will be critical for early detection.

Indicators of Compromise (IOCs)

To assist the wider community in hunting and identifying the activity outlined in this blog post, we have included IOCs in a free GTI Collection for registered users.

Network Indicators

Indicator	Description
`service-page-25144-30466-outlook.s3.us-west-2.amazonaws[.]com`	Hosted the phishing site and initial AutoHotKey payloads
`cloudfront-021.s3.us-west-2.amazonaws[.]com`	SNOWBELT C2
`wss://sad4w7h913-b4a57f9c36eb.herokuapp[.]com/ws`	Hard-coded WebSocket Secure URL within SNOWGLAZE
`service-page-11369-28315-outlook[.]s3[.]us-west-2[.]amazonaws[.]com`	Domain for URL used to upload a text file

File Indicators

File Name	Description	SHA-256 Hash
C:\ProgramData\log	SNOWGLAZE	`2fa987b9ed6ec6d09c7451abd994249dfaba1c5a7da1c22b8407c461e`
C:\ProgramData\log	SNOWBASIN	`c8940de8cb917abe158a826a1d08f1083af517351d01642e6c7f324d0`
C:\Users\<user>\AppData\Local\Microsoft\Edge\Extension Data\SysEvents\background.js	SNOWBELT Service worker	`7f1d71e1e079f3244a69205588d504ed830d4c473747bb1b5c520634c`

C:\Users\<user>\AppData\Local\Microsoft\Edge\Extension Data\SysEvents\dream.js	SNOWBELT JS resource	ca390b86793922555c84abc3b34406da2899382c617f9dcf83a74ac09
C:\Users\<user>\AppData\Local\Microsoft\Edge\Extension Data\SysEvents\dream.html	SNOWBELT HTML resource	6e6dab993f99505646051d2772701e3c4740096ff9be63c92713bcb7f
C:\Users\<user>\AppData\Local\Microsoft\Edge\Extension Data\SysEvents\helper.html	SNOWBELT HTML resource	de200b79ad2bd9db37baeba5e4d183498d450494c71c8929433681e84

YARA Rules

SNOWGLAZE

```
rule G_Tunneler_SNOWGLAZE_1 {
 meta:
  author = "Google Threat Intelligence Group (GTIG)"
  platforms = "Windows, Linux"

 strings:
   $r1 = /\.connect\(\s{0,25}WS_PROXY_URL/
   $r2 = /"data":\s{0,1}base64\.b64encode\(\w{1,10}\)\.decode\('ascii'\)/
   $r3 = /"type":\s{0,1}"socks_data"/
   $r4 = /await\s{0,1}reader\.read\(\d{2,4}\)/
   $r5 = /"login":\s{0,1}AGENT_LOGIN/
   $r6 = /"password":\s{0,1}AGENT_PASSWORD/
   $r7 = /"uuid":\s{0,1}AGENT_UUID/

   $s1 = ".socks_tcp_to_ws"

 condition:
   5 of ($r*)
   and $s1
}
```

SNOWBELT

```
rule G_Backdoor_SNOWBELT_1 {
   meta:
       author = "Google Threat Intelligence Group (GTIG)"
       platform = "Windows"

        strings:
                $str1 = ".importKey(\"raw\",keyMaterial,\"AES-GCM\",!1,[\"decrypt\"])"
                $str2 = ".importKey(\"raw\",keyMaterial,\"AES-GCM\",!1,[\"encrypt\"])"
                $str3 = "sendJsonDataToS3"
                $str4 = "processCommand"
                $str5 = "\"screenshot\"===cmdType"
                $str6 = "\"payload\"===cmdType"
                $str7 = "\"websocket_control\"===cmdType"
                $str8 = "\"open_uri\"===cmdType"
```

```
                $str9 = "\"delete_cache\"===cmdType"
                $str10 = "\"payload_download_complete\""
                $str11 = ".s3.us-east-2.amazonaws.com/"
        condition:
                all of them

}
```

SNOWBASIN

```
rule G_Backdoor_SNOWBASIN_1 {
 meta:
   author = "Google Threat Intelligence Group (GTIG)"
   platform = "Windows"

 strings:
   $path1 = "self.path == '/probe':"
   $path2 = "self.path == '/stream':"
   $path3 = "self.path == '/buffer':"
   $path4 = "self.path == '/flush':"
   $path5 = "self.path == '/commit':"
   $path6 = "self.path == '/capture':"
   $path7 = "self.path == '/gc':"

   $func1 = "self.handle_stream("
   $func2 = "self.handle_buffer("
   $func3 = "self.handle_flush("
   $func4 = "self.handle_commit("

   $s1 = "self.wfile.write(info_msg"
   $s2 = "selected_port), WebServerHandler) as httpd:"
   $s3 = "ThreadedTCPServer(socketserver.ThreadingMixIn"
   $s4 = "httpd.serve_forever()"


 condition:
   filesize<1MB and (
     (all of ($s*) and 6 of ($path*, $func*)) or
     (8 of ($path*, $func*)) or
     10 of them
   )
}
```

MITRE ATT&CK

Tactic	Techniques
Initial Access	T1566.002: Spearphishing Link
Execution	T1053: Scheduled Task/Job T1053.005: Scheduled Task T1059: Command and Scripting Interpreter T1059.001: PowerShell

Tactic	Techniques
	T1059.003: Windows Command Shell T1059.006: Python T1059.007: JavaScript T1059.010: AutoHotKey & AutoIT T1204.001: Malicious Link T1204.002: Malicious File T1559: Inter-Process Communication T1569.002: Service Execution
Persistence	T1176.001: Browser Extensions T1543: Create or Modify System Process T1543.003: Windows Service T1547.001: Registry Run Keys / Startup Folder T1547.009: Shortcut Modification
Privilege Escalation	T1068: Exploitation for Privilege Escalation
Defense Evasion	T1027: Obfuscated Files or Information T1027.010: Command Obfuscation T1027.015: Compression T1036.005: Match Legitimate Resource Name or Location T1055: Process Injection T1070.004: File Deletion T1112: Modify Registry T1134: Access Token Manipulation T1134.001: Token Impersonation/Theft T1140: Deobfuscate/Decode Files or Information T1202: Indirect Command Execution T1562.001: Disable or Modify Tools T1564.001: Hidden Files and Directories T1622: Debugger Evasion

Tactic	Techniques
Credential Access	T1003.001: LSASS Memory T1003.002: Security Account Manager T1003.003: NTDS T1110.001: Password Guessing T1110.003: Password Spraying T1552.001: Credentials In Files
Discovery	T1007: System Service Discovery T1012: Query Registry T1016: System Network Configuration Discovery T1018: Remote System Discovery T1033: System Owner/User Discovery T1046: Network Service Discovery T1057: Process Discovery T1082: System Information Discovery T1083: File and Directory Discovery T1087.001: Local Account T1518: Software Discovery
Lateral Movement	T1021.001: Remote Desktop Protocol T1021.002: SMB/Windows Admin Shares
Collection	T1005: Data from Local System T1074: Data Staged T1113: Screen Capture T1560: Archive Collected Data T1560.001: Archive via Utility
Exfiltration	T1020: Automated Exfiltration T1567: Exfiltration Over Web Service T1567.002: Exfiltration to Cloud Storage

Tactic	Techniques
Command and Control	T1071.001: Web Protocols T1090: Proxy T1105: Ingress Tool Transfer T1572: Protocol Tunneling
Impact	T1489: Service Stop
Resource Development	T1608.002: Upload Tool T1608.005: Link Target

Acknowledgements

This analysis would not have been possible without the assistance from several individuals within Mandiant Consulting, Google Threat Intelligence Group and FLARE who helped with analysis and reviewing this blog post. We also appreciate Amazon for their collaboration against this threat.

Posted in

- Threat Intelligence
