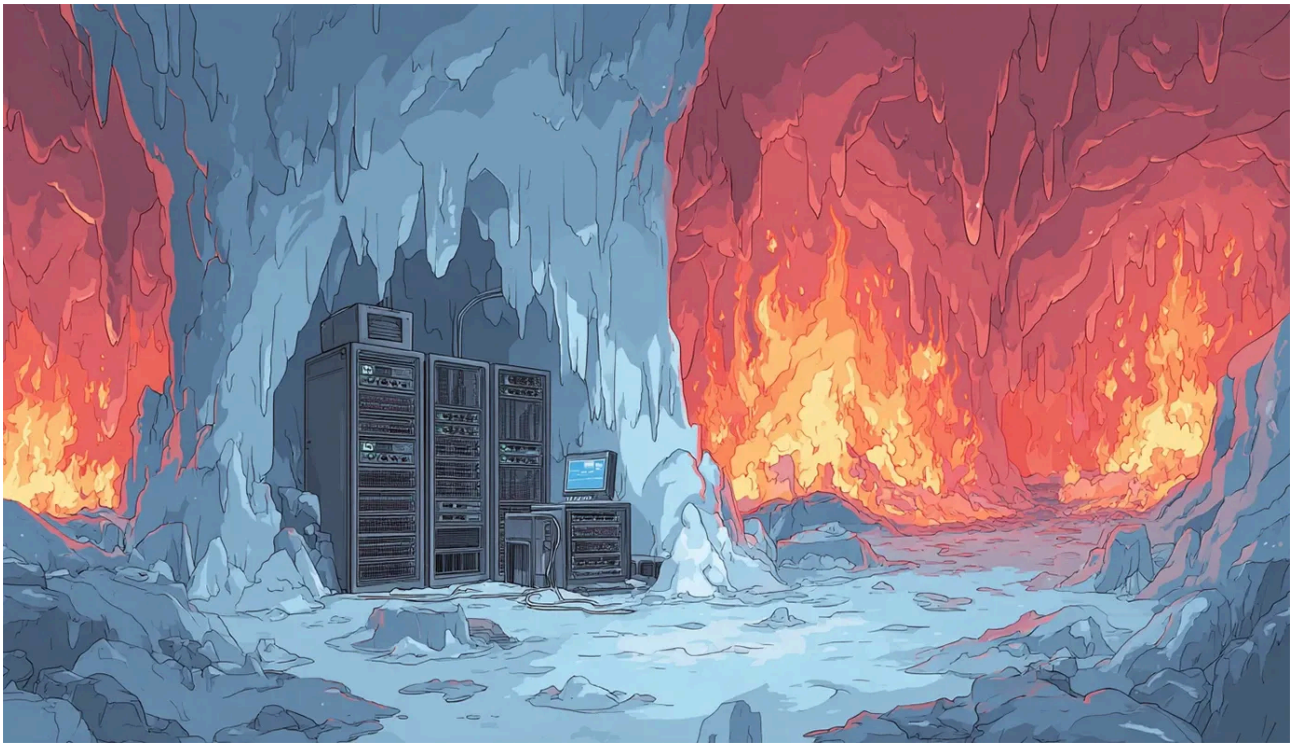
ChillyHell: A Deep Dive into a Modular macOS Backdoor

By Jamf Threat Labs

Archived: 2026-04-05 14:17:28 UTC

Jamf Threat Labs performs a deep dive on the modular malware that has been mysteriously maligning macOS since 2021.



Authors: Ferdous Saljooki, Maggie Zirnhelt

Introduction

During routine sample analysis on VirusTotal, Jamf Threat Labs discovered a file that stood out due to a notable method of process reconnaissance being used. Despite the malware family having been documented in the past, it remains unflagged by antivirus vendors.

The sample is developer-signed and successfully passed Apple's notarization process in 2021. Its notarization status remained active until these recent findings.

Background

The malware, known as ChillyHell, was originally unveiled in a private 2023 Mandiant report that loosely tied it to a threat actor targeting officials in Ukraine. This report documents a 2022 attack in which a threat actor, tracked by Mandiant as UNC4487, compromised a Ukrainian auto insurance website which government employees were

required to use for official travel. The compromised site delivered the MATANBUCHUS malware, where attackers were then able to sell access to the infected systems at a high price.

Also in its report, Mandiant discovered additional malware dubbed ‘ChillyHell’, while searching for other samples matching the signing certificate used to sign the MATANBUCHUS malware.

The technical details of the ChillyHell malware were not documented, but two macOS hashes were provided in the original report:

- `eDrawMaxBeta2023.app/Contents/macOS/eDrawMaxBeta2023`
  - c52e03b9a9625023a255f051f179143c4c5e5636
  - **TEAMID**: F645668Q3H
  - Notarized
- `chrome_render.app/Contents/MacOS/chrome_render`
  - 87dcb891aa324dcb0f4f406deebb1098b8838b96
  - **TEAMID**: R868N47FV5
  - Not notarized

These two samples are different from each other, despite being assigned the same family name. For example, the one titled eDrawMaxBeta came packaged with an SSH server compiled directly into it, whereas the one titled chrome_render did not.

Jamf Threat labs encountered a new ChillyHell sample uploaded to VirusTotal on May 2nd, 2025.

- `applet.app/Contents/MacOS/applet`
  - 6a144aa70128ddb6be28b39f0c1c3c57d3bf2438
  - **TEAMID**: R868N47FV5
  - Notarized

Despite not making it to VirusTotal until 2025, this sample was also notarized in 2021 and has remained notarized up until these findings. The teamID matches that of the ChillyHell sample reported by Mandiant, and its functionality appears to be nearly identical. Furthermore, this notarized sample has been publicly hosted on Dropbox since 2021 at https://dropbox[.]com/s/2fncbp2rv134z6y/applet.zip.

Technical analysis

The executable applet is a modular C++ backdoor developed for Intel architectures. Although packaged as applet.app, it does not function as a legitimate macOS applet. In genuine applets, `Contents/MacOS/applet` is paired with a compiled AppleScript within `Contents/Resources/Scripts/`. This sample omits any scripts, showing that both the app bundle and the executable name are used only as a disguise.

On execution, ChillyHell begins by invoking `OS::StartupLogic()`, which performs host profiling and attempts to establish persistence. If successful, it initializes command and control (C2) communication using `TransportInit()` and then enters its primary command loop in a function called `mainCycle()`.

Host profiling

The `OS::StartupLogic()` function is responsible for preparing the compromised system, both by collecting basic environmental details and establishing persistence.

As part of its host profiling logic, ChillyHell executes a helper function, `Utils::GetUsers()`, which queries the local directory service to enumerate all user accounts. This is implemented by shelling out the following command

This command returns detailed information about all user records on the system in plist format. An additional function obtains the contents of the environment variables `$HOME`, `$PATH` and `$SHELL`. The results of these two functions are used to provide insight into the user's home directory, available executable paths and preferred shell.

ChillyHell uses `proc_listpids()` to programmatically retrieve all active PIDs. It iterates over each one to collect structured metadata about running processes. In addition, it executes the below command calling the function `Utils::ParsePSCommand` to capture uid, pid and command-line arguments for every process:

After enumerating processes, ChillyHell collects the effective username under which it's running. It does this by instantiating a command object with the string "whoami" and executing it via `Utils::RunCommand()`. The result is retained if the command succeeds. An Env object is then initialized with default values setting the username to "root" followed by string "`$PATH`".

Next, applet performs another check related to the user context under which it's executing. It executes a system `call _getuid()` to obtain the current user's UID, then invokes `Utils::FindUserByUID()` to map this UID to a full user record. If this lookup fails, it falls back to the whoami output collected earlier.

Persistence

After resolving the active user, ChillyHell initializes its installation logic.

Three separate persistence mechanisms are supported depending on privilege level and installation state:

LaunchAgent (User Context)

If running as a regular user, ChillyHell installs itself as a LaunchAgent using the `StartupInstall::Install()` method. It writes a plist to `~/Library/LaunchAgents/com.apple.qtop.plist` and drops its main binary at `~/Library/com.apple.qtop/qtop`. The plist ensures execution on login, with RunAtLoad set to true.

LaunchDaemon (Root/System Context)

If executed with elevated privileges, ChillyHell attempts to install itself as a system LaunchDaemon via `StartupInstall::ReplaceInstallWithElevatedRights()`. Upon success, it writes a plist to `/Library/LaunchDaemons/com.apple.qtop.plist` and places a copy of the main binary `at/usr/local/bin/qtop`. This ensures execution at boot with system privileges.

Shell profile injection

As a fallback persistence mechanism, ChillyHell can modify the user's shell profile (.zshrc, .bash_profile or .profile). It uses `StartupInstall::GetRcFilePath()` to determine the appropriate shell configuration based on

the user's shell and home directory. The persistence logic is handled by `StartupInstall::InstallToShell()`, which calls `StartupInstall::InsertLineToShellRCIfNotExist()` to inject a launch command into the configuration file – only if the line doesn't already exist. This ensures the malware is executed on each new terminal session.

When ChillyHell is manually executed it daemonizes itself using `OS::ForkyDaemon()`. This function creates a daemon using the old school Unix approach by performing a double-fork: creating a new session and redirecting all standard I/O to `/dev/null` to fully detach from the parent process. It also opens a decoy URL (https://google.com) in the default web browser for reasons not fully known at this time, although the current belief is to minimize user suspicion.

Timestomping

After ChillyHell has created artifacts on the infected system, it replaces their associated timestamps to reduce suspicion. A call to `_utime()` updates the creation and modification times of any created directories, plists and binaries. If it does not have sufficient permission to update the timestamps by means of a direct system call, it will fall back to using shell commands `touch -c -a -t` and `touch -c -m -t` respectively, each with a formatted string representing a date from the past as an argument included at the end of the command. The `-a` in the first command refers to access time and the `-m` in the second command refers to modification time.

Due to the way macOS and the APFS file system work, adjusting the modified timestamp could inadvertently result in a backdated birth timestamp. Meanwhile, the "change" timestamp ends up reflecting the time that the manipulation took place.

C2 initialization and communication setup

The `TransportInit()` function, which comes after the initial function `OS::StartupLogic()`, is used to establish communication with the C2.

ChillyHell performs a network reachability check to determine if it has access to the internet. By calling a function named `WaitForNetworkReachability()` which leverages Apple's System Configuration framework, `applet` attempts to reach Google's DNS server at 8.8.8.8. Specifically, it waits for the `SCNetworkReachabilityFlags` to be set to `0x2` indicating the host is reachable. Between requests it sleeps for 60 to 120 seconds until the system is online.

Once an internet connection is confirmed, ChillyHell calls `GateServersInit()` to populate its internal list of C2 servers. Two hardcoded IP addresses are used: `93[.]88.75.252` and `148[.]72.172.53`. Each address is paired with multiple ports (53, 80, 1001, and 8080) and a transport type indicator (1 for DNS; 2 for HTTP). These combinations are stored in a global list for use in later C2 communications, such as retrieving attacker-issued tasks.

Main execution loop and tasking

After establishing a network connection, ChillyHell runs its `mainCycle()` loop, handling tasks and carrying out commands from the attacker-controlled C2 infrastructure.

The loop within the `mainCycle()` function does the following:

1. Retrieve tasks

Via the function `tasks::getTasks()`, ChillyHell fetches a list of task descriptors from the C2 server and populates them into a local vector.

2. Deduplicate

Via the function `Utils::pidExists()`, applet compares the contents of running tasks to the local record of already processed tasks. If the current task to execute does not duplicate work that is in progress or work that has been done already, then the task will be executed.

3. Execute

The function `tasks::execTask()` is called when a task is deemed new. It dynamically instantiates the appropriate module class (ModuleLoader, ModuleUpdater, ModuleBackconnectShell or ModuleSUBF), and invokes its `Execute()` method. Each module receives the original task string, which is first base64-decoded. Modules then parse the decoded content for parameters. Some modules also report status updates to the C2. Successfully executed tasks are tracked by storing their process ID and payload in memory to prevent duplicate execution in future polling cycles.

4. Sleep

At the end of each cycle, `randoms::doSleep()` is invoked with a randomized delay between 60 and 120 seconds to introduce variability in C2 polling.

A diagram describing the steps taken during each iteration of the loop within mainCycle().

Now that the outermost loop of the malware has been described, we can take a closer look at the elements that make up the most interesting parts of the cycle.

Task retrieval

As identified in the first step of the main execution loop, ChillyHell uses the `tasks::getTasks()` function to fetch new commands. This method constructs a C2 query string, sends the request and parses the response for task payloads.

Build the query

The function first calls `tasks::getPrefix()`, which constructs a prefix string composed of host-specific metadata in the following order: OS identifier (3), group label (shadmins), hardware UUID (lowercased and dashless), a static literal (0) and a static version marker (114). This full prefix is appended to the C2's base domain and used in a DNS TXT query to retrieve tasking commands.

Poll for a response

The function next repeatedly queries the C2 by invoking `Query()` until a response is received. Between attempts, it sleeps for a randomized interval between 60 and 120 seconds. The final response is stored in a local string buffer.

Parse response

ChillyHell iterates through each returned string, scanning for the marker TASK:. If found, it extracts the portion of the string following this marker as a task payload which is queued for execution. Each new task is dynamically executed by instantiating a corresponding module and invoking its `Execute()` method.

Add to task list

Each valid task is copied into a new string and appended to a vector passed into the function. This vector accumulates all discovered tasks during the current polling cycle is used in the main execution cycle to prevent task duplication.

Below is an example DNS query that contains host-derived metadata used for tasking.

C2 transport

ChillyHell communicates with its C2 infrastructure using the `Query()` function, which loops through a list of predefined gate servers, each associated with a transport type.

Each gate entry includes both an address and a transport type. If a previously successful (“stable”) gate exists, it is tried first. Otherwise, the malware loops through all available gates until one returns a valid response with the marker "[+] Ok". That gate is then saved as the new stable gate for future use.

The `QueryGate()` function dispatches based on the gate’s transport type:

- HTTP transport (type 2): Calls `QueryHTTP()` to fetch tasks from a remote server over standard HTTP(S).
- DNS transport (type 1): Calls `QueryTXTRecords()` to retrieve encoded task data via DNS TXT record lookups.

If the gate returns a valid response, it is parsed and stored into the string vector passed to `getTasks()`. If all gates fail, the stable gate is reset, and the polling logic will be retried in the next cycle.

Task execution and Module dispatch

Once a task payload is retrieved and parsed, ChillyHell calls `tasks::execTask()` to execute it. This function determines the task type and uses a switch statement to instantiate the corresponding module:

Type 0 - ModuleBackconnectShell: Connects to a C2 IP and port, spawns a pseudo-terminal using `forkpty()` and relays input/output over the socket to maintain an interactive reverse shell. We noticed it uses the banner "\r\n----===Welcome to Paradise!===---\r\n".

Type 1 - ModuleUpdater: Downloads a new version of the malware from the C2 server, replaces the current binary and restarts itself calling `ModuleUpdater::restartProcess()`.

Type 2 - ModuleLoader: Downloads a payload from the C2 server, writes it to `/tmp/kworker` and then attempts to execute it using `startProcess()`. If execution is successful, it waits five seconds before deleting the file.

Type 4 - ModuleSUBF: Enumerates user accounts and performs password cracking. We were unable to retrieve this tool from the server at the time of analysis and thus unable to guarantee its purpose. Despite this, we believe this module targets Kerberos-based authentication because of the observed filenames, downloaded wordlists and brute-force attempts.

Each module extends a shared AbstractModule base class and implements its own `Execute()` method. The first three module types, ModuleBackconnectShell, ModuleUpdater and ModuleLoadereach have their functionality described effectively in the statements above. ModuleSUBF, however, contains more complex logic and warrants additional details.

ModuleSUBF - Brute Forcing

This module (Type 4) performs local password cracking against user accounts. It consists of the following steps:

Retrieves usernames

To begin, ModuleSUBF uses the method `getUsernames()` to parse `/etc/passwd` and extract local usernames, which are then stored for use in subsequent password brute-force attempts.

Downloads tooling and wordlist

This module then calls `downloadModule()` to retrieve a brute-force tool from the C2 named ./kerberos. After downloading the tool, ChillyHell invokes `downloadWordlist()` to fetch a password wordlist used in the attack.

Launches credential attacks

The module calls `tryUser()` for each discovered username. This function begins by constructing configuration strings by replacing placeholders {USERNAME} and {WORDLIST_PATH}. The downloaded ./kerberos binary is then executed in a forked child process. Each process attempts to crack the user's password by brute-forcing it with the supplied username and wordlist. Successful guesses are likely written by the ./kerberos binary to a file named good.txt.

Validates and extracts results

The module calls `getFound()` to parse the brute-force output and extract valid credential pairs. If a match is found, applet verifies whether the associated cracking process is still running using `Utils::pidExists()`. If the PID is inactive, the module issues a SIGTERM to ensure cleanup and then removes the corresponding entry from memory.

The above is a pseudocode representation of the `Execute()` function of ModuleSUBF.

Conclusion

Between its multiple persistence mechanisms, ability to communicate over different protocols and modular structure, ChillyHell is extraordinarily flexible. Capabilities such as timestomping and password cracking make this sample an unusual find in the current macOS threat landscape. Notably, ChillyHell was notarized and serves as an important reminder that not all malicious code comes unsigned.

The Jamf Threat Labs team would like to thank Google Threat Intelligence for sharing details regarding the original ChillyHell discovery, and Apple for working with us to quickly revoke the developer certificates associated with this malware.

Indicators of Compromise (IoCs)
