Lab 4: MalloX Ransomware Analysis

Kill Chain Reconstruction & Behavioral Forensics

Mohammed-Rida El hani

Contents

1 Introduction:

Analyzing MalloX ransomware during late-night sessions proved both instructive and entertaining—the malware's creative approach to achieving its objectives was technically fascinating. This report reconstructs the complete attack lifecycle using a structured methodology: static reverse engineering in Ghidra combined with dynamic sandbox validation. The investigation followed a deliberate timeline from initial triage through granular function analysis to runtime behavioral correlation, demonstrating how hypothesis-driven workflows can achieve comprehensive understanding without exhaustive reverse engineering of every component.

2 Executive Summary

MalloX is a 32-bit Windows ransomware compiled with Visual Studio 2019 in October 2022. The binary embeds complete extortion infrastructure: Tor onion portal (`wtyafjyhwqrgo4a45wdvvwhen3cx4euie73qvlhkhvlrexljoyuklaad.onion`), operator email (`mallox@stealthypost.net`), and full ransom note instructions.

Static analysis revealed imports for HTTP beaconing (WININET), network share enumeration (NETAPI32), and privilege escalation (ADVAPI32). Ghidra reverse engineering isolated key routines: victim profiling (`FUN_00405c15`), multi-threaded encryption driver (`FUN_004054ba`), and ransom note writer (`FUN_00404d10`).

Dynamic validation confirmed the complete kill chain. Sandbox logs captured HTTP POST to `193.106.191.141/QWEwqdsvsf/ap.php`, execution of `vssadmin` and `bcdedit` for recovery inhibition (T1490), aggressive SQL Server termination via `taskkill` chains, and lateral propagation attempts to `\\192.168.60.1\admin$`.

The operational model demonstrates double-extortion (credential theft pre-encryption), living-off-the-land abuse of signed Windows utilities, and operator-controlled C2 for affiliate tracking.

3 Methodology & Analysis Timeline

The investigation followed a three-phase workflow designed to maximize signal extraction while avoiding the classic analyst trap of reverse-engineering every function in a 144KB binary.

3.1 Phase 1: Static Triage (Surface-Level Recon)

We started with automated tooling to build a high-level behavioral profile before touching a disassembler. Tools used included `rabin2` for PE metadata, Detect It Easy (DIE) for compiler fingerprinting, and manual string extraction. The goal was to answer: What does this binary want to do, and what infrastructure does it depend on?

The triage confirmed a 32-bit PE executable for x86 with a Windows GUI subsystem, compiled with MSVC 19.20 and linked with version 14.20. DIE flagged high entropy in the `.text` and `.reloc` sections, which is typical for ransomware that embeds large cryptographic constant tables or compressed configuration blobs. The import table immediately revealed high-signal capabilities: file and drive enumeration APIs from KERNEL32, privilege and registry manipulation from ADVAPI32, network share enumeration from NETAPI32, and HTTP functionality from WININET.

String extraction was the jackpot. The binary contains a complete ransom note with Tor instructions, an onion URL, a portal path (`/mallox/privateSignin`), and an operator email (`mallox@stealthypost.net`). It also contains living-off-the-land command strings for shadow copy deletion (`vssadmin delete shadows /all /quiet`) and boot configuration sabotage (`bcdedit /set {current} recoveryenabled no`). Another massive string block lists

SQL Server process and service names targeted for termination, which explains the operational focus on database workloads.

At the end of Phase 1, we had a working hypothesis: this is a Mallox-family ransomware that profiles victims, beacons to a C2 server, destroys recovery options, kills database services, encrypts aggressively, and provides Tor-based extortion instructions. Everything after this was validation.

3.2 Phase 2: Deep Static Analysis (Ghidra Reverse Engineering)

Phase 2 involved loading the binary into Ghidra and isolating the key functions that implement the kill chain. The challenge with ransomware binaries is separating compiler runtime boilerplate from actual malware logic. MSVC binaries start with CRT initialization and stack-cookie setup, which can look suspicious if you're not familiar with legitimate compiler behavior.

The entry stub routes through `___security_init_cookie()`, which is a standard MSVC anti-stack-smashing routine. This is not malware trickery—it's just the compiler doing its job. After CRT initialization completes, control transfers to the malware's `main()` function, where the real orchestration begins.

Main Orchestration: The `main()` routine handles early environment control. It calls `GetUserDefaultLangID()` and exits early for specific language IDs, which is a common affiliate behavior to avoid targeting certain regions. It then loads `PowrProf.dll` and attempts to call `PowerSetActiveScheme` to prevent the system from sleeping or hibernating during encryption. Next, it tries to enable `SeTakeOwnershipPrivilege` and `SeDebugPrivilege` via token manipulation, spawns multiple worker threads, and creates a hidden window class that calls `ShutdownBlockReasonCreate()` with a warning message to prevent user-initiated shutdowns. It also modifies registry-based Group Policy values to hide shutdown and restart UI elements.

Pivot Function (FUN_004061e8): This is the setup-and-dispatch hub. It parses command-line arguments including `-l`, `-d`, `-p`, and `-path`, which suggests the malware supports operator-driven targeting modes. It then generates per-run entropy using `QueryPerformanceCounter`, tick counts, process and thread IDs, `rdtsc`, and CryptoAPI's `CryptGenRandom`. This randomness is stored globally and used for cryptographic key material or victim identifiers. The function collects host fingerprints (computer name, Windows directory timestamps, CPU vendor strings via `CPUID`) and passes them into routines that resemble key-derivation or mixing functions. Finally, it triggers the encryption driver (`FUN_004054ba`) and the ransom note writer (`FUN_00404d10`).

Encryption Driver (FUN_004054ba): This routine implements the multi-threaded encryption workflow. It creates an I/O completion port using `CreateIoCompletionPort`, calculates a worker thread count based on CPU cores (with bounds), and enumerates drives from A: through Z: using `GetLogicalDrives` and `GetDriveTypeW`. It filters drives by type (fixed, removable, and network-relevant drives) and calls `GetDiskFreeSpaceExW` to gather storage metadata. Work items representing file paths are queued to the completion port, and worker threads pull items, traverse directories, and encrypt files. This design is efficient for high-throughput parallel encryption across large directory trees.

Ransom Note Writer (FUN_00404d10): This routine constructs the ransom note buffer once (protected by a global flag) and writes it to the specified path using `CreateFileW` and `WriteFile`. The note content matches the embedded Tor instructions, onion portal, and email contact extracted during triage.

Victim Profiling (FUN_00405c15): This routine explains both the `TargetInfo.txt` artifact and the HTTP beacon payload format. It computes aggregate disk sizes across logical drives, retrieves the computer name, and determines the process's access level (Low/Medium/Admin/System) by querying token integrity and group membership. It collects CPU architecture details and formats a `SystemInformation` string, URL-escapes it, and constructs an HTTP POST body matching the embedded format:

```
user=%s&TargetID=%s&SystemInformation=%s&max_size_of_file=%s&
    size_of_hdd=%d
```

It also writes a local `TargetInfo.txt` file with pipe-delimited victim metadata. This artifact serves both as local bookkeeping and as the data transmitted to the C2 endpoint.

By the end of Phase 2, we had isolated the pivot routines and understood the code-level workflow. The next step was proving these behaviors actually execute in a live environment.

3.3 Phase 3: Dynamic Validation (Sandbox Telemetry)

Phase 3 involved parsing sandbox JSON logs to validate our static hypotheses with runtime evidence. The sample executed from a Temp directory with the filename `1f793f973fd906f9736aa483.exe`. Early behavior included repeated calls to `NtDelayExecution`, which is anti-sandbox stalling designed to outlast short observation windows.

The process tree confirmed multiple child `cmd.exe` instances used to execute native Windows utilities, including `vssadmin.exe` and `taskkill.exe`. System components like `svchost.exe` and `dllhost.exe` also appeared, which is normal for Windows COM hosting but can also indicate stealthy execution patterns.

Network Beacon: The most critical dynamic confirmation was the outbound HTTP request to `http://193.106.191.141/QWEwqdsvsf/ap.php`. This matches perfectly with our static findings: the binary imports WININET functions, constructs a URL-encoded victim profile payload, and maintains `TargetID` and `SystemInformation` strings. This endpoint acts as the victim registration or beaconing service and represents the most direct proof of operator infrastructure.

Recovery Inhibition: Dynamic logs confirmed the execution of recovery-killing commands exactly as embedded in the binary strings. The malware ran:

```
bcdedit /set {current} bootstatuspolicy ignoreallfailures
bcdedit /set {current} recoveryenabled no
vssadmin delete shadows /all /quiet
```

This is textbook MITRE ATT&CK technique T1490 (Inhibit System Recovery). The correlation between static strings and runtime execution is perfect.

Process & Service Disruption: The sandbox captured aggressive `taskkill` and `sc delete` chains targeting SQL Server components. Observed commands included:

```
taskkill /f /im sqlservr.exe
taskkill /f /im sqlwriter.exe
sc delete "MSSQLSERVER"
sc delete "SQLSERVERAGENT"
```

This matches the embedded SQL service list from static analysis. The operational goal is to stop database workloads, release file locks, and maximize the number of encryptable files.

Lateral Movement: File telemetry showed attempts to copy the ransomware binary to remote administrative shares:

```
\\192.168.60.1\admin$\mallox.exe
\\192.168.60.1\c$\mallox.exe
```

This matches the expected behavior from NETAPI32 imports and suggests the malware tries to propagate within reachable networks using existing administrative share access.

Extortion Artifacts: Dynamic logs referenced a ransom note filename `FILE RECOVERY.txt`. While static analysis showed `C:\HOW TO RECOVER !!.TXT`, filename mismatches are common in ransomware campaigns where different builds or affiliate configurations produce variant note names. What matters is the behavior consistency: the malware writes victim-facing instructions referencing Tor infrastructure and operator contact channels.

Credential Harvesting: File access logs showed hits on paths consistent with browser cookie storage, supporting the ”double extortion” claim from the lab brief. While the logs don’t prove full exfiltration, they demonstrate that the malware searches for credential and session material before or during encryption, which is typical for modern ransomware operations.

4 Part 1: Static Analysis & Attribution

4.1 Sample Identification & Hashes

32-bit Windows PE executable (x86/i386), identified as `sample.defanged`:

- **MD5:** `f54d6e6bb32a872a9b3fdb5a585e7c07`
- **SHA-256:** `85438d6f45f7001a88b61a7383021dd12a3f481e0592b2b059284df858218130`

4.2 Ransom Note Extraction

Embedded in `.rdata` section, full extortion message extracted:

Tor Infrastructure:

- **Onion URL:** `wtyafjyhwqrgo4a45wdvvwhen3cx4euie73qvlhkhvlrexljoyuklaad.onion`
- **Portal Path:** `/mallox/privateSignin`

Contact Information:

- **Email:** `mallox@stealthypost.net`

Note instructs Tor installation, provides ”private ID” authentication, offers ”free test decryption,” and warns about data publication (double extortion).

4.3 Interesting Strings

Recovery Inhibition:

```
vssadmin delete shadows /all /quiet
bcdedit /set {current} recoveryenabled no
bcdedit /set {current} bootstatuspolicy ignoreallfailures
```

SQL Server Disruption:

```
taskkill /f /im sqlservr.exe
taskkill /f /im sqlwriter.exe
sc delete "MSSQLSERVER"
sc delete "SQLSERVERAGENT"
```

Victim Profiling:

```
user=%s&TargetID=%s&SystemInformation=%s&max_size_of_file=%s&
   size_of_hdd=%d
```

Local artifact: `TargetInfo.txt`

5 Part 2: Command & Control Identification

Dynamic sandbox captured outbound HTTP beacon:

`http://193.106.191.141/QWEwqdsvsf/ap.php`

Matches static findings: WININET imports, URL-encoded victim profile payload format, and `TargetID/SystemInformation` strings from `FUN_00405c15()`.

Beacon transmits victim registration data (computer name, privilege level, CPU arch, disk capacity). C2 IP `193.106.191.141` and path `/QWEwqdsvsf/ap.php` are high-confidence IOCs for network detection.

6 Part 3: Behavioral Analysis — The Destruction Phase

6.1 Process Tree & Execution Flow

Execution path: `C:\Users\Stati\AppData\Local\Temp\1f793f973fd906f9736aa483.exe`

Spawned children:

- Multiple `cmd.exe` instances (command interpreters)
- `vssadmin.exe` (shadow copy deletion)
- `taskkill.exe` (process termination)

Classic living-off-the-land pattern: parent ransomware binary spawns `cmd.exe` children that invoke signed Windows utilities for destructive tasks.

6.2 Command-Line Execution Analysis

Category	**Command**	**Purpose**
Recovery Inhibition	`bcdedit /set {current} bootstatuspolicy ignoreallfailures`	Ignore boot failures, prevent recovery mode
Recovery Inhibition	`bcdedit /set {current} recoveryenabled no`	Disable Windows Recovery Environment
Recovery Inhibition	`vssadmin delete shadows /all /quiet`	Delete Volume Shadow Copies (T1490)
Process Killing	`taskkill /f /im sqlservr.exe`	Terminate SQL Server, release DB file locks
Process Killing	`taskkill /f /im sqlwriter.exe`	Kill SQL Writer (VSS backup component)
Process Killing	`taskkill /f /im sqlbrowser.exe`	Terminate SQL Browser
Process Killing	`taskkill /f /im msmdsrv.exe`	Kill Analysis Services (OLAP cubes)
Process Killing	`taskkill /f /im Ssms.exe`	Terminate SQL Management Studio

Category	**Command**	**Purpose**
Process Killing	`taskkill /f /im SQLAGENT.EXE`	Stop SQL Server Agent
Service Termination	`sc delete "MSSQLSERVER"`	Delete main SQL Server service
Service Termination	`sc delete "SQLSERVERAGENT"`	Remove SQL Agent service
Service Termination	`sc delete "SQLBrowser"`	Delete SQL Browser service
Service Termination	`sc delete "SQLWriter"`	Remove SQL Writer (prevent VSS backup)

Recovery Inhibition: Three-pronged attack eliminates snapshots (`vssadmin`), disables Recovery Environment, and configures boot manager to skip recovery triggers. MITRE T1490.

SQL Disruption: Forceful termination (`/f` flag) prevents graceful shutdown. Service deletion (`sc delete`) prevents automatic restart. Goal: release file locks on databases/logs/backups for encryption.

7 Part 4: Credential Access — The Stealer Component

Dynamic file telemetry showed access to paths within `%APPDATA%` associated with browser cookie databases and credential stores. While sandbox logs don't prove full exfiltration, they confirm the malware searches for credential material before/during encryption.

Supports double-extortion model: steal credentials/session tokens before encryption for additional leverage. If victims restore from backups, operators can still threaten to publish stolen data or use tokens for account takeover.

Static analysis (`FUN_00405c15`) and WININET imports suggest potential HTTP exfiltration, though not conclusively captured in sandbox observation window.

8 Consolidated IOC Table

The following table consolidates all Indicators of Compromise extracted during static and dynamic analysis. These IOCs are suitable for threat hunting, SIEM rule creation, and defensive blocking.

IOC Type	**Indicator**	**Context**
Hash (MD5)	`f54d6e6bb32a872a9b3fdb5a585e7c07`	Static sample hash
Hash (SHA-256)	`85438d6f45f7001a88b61a7383021dd12a3` f481e0592b2b059284df858218130	Static sample hash
Filename	`sample.defanged`	Lab analysis filename
Filename	`1f793f973fd906f9736aa483.exe`	Dynamic execution name
Filepath	`C:\Users\Stati\AppData\Local\Temp` \1f793f973fd906f9736aa483.exe	Observed execution path

IOC Type	Indicator	Context
IPv4 Address	`193.106.191.141`	C2 beacon target
URL	`http://193.106.191.141/QWEwqdsvsf` /ap.php	C2 beacon endpoint
Onion Domain	`wtyafjyhwqrgo4a45wdvvwhen3cx4euie73` qvlhkhvlrexljoyuklaad.onion	Tor extortion portal
URL Path	`/mallox/privateSignin`	Portal authentication path
Email	`mallox@stealthypost.net`	Operator contact address
Filename	`TargetInfo.txt`	Victim profiling artifact
Filepath	`C:\Users\Stati\AppData\Local\Temp` \TargetInfo.txt	Observed artifact path
Filename	`C:\HOW TO RECOVER !!.TXT`	Static ransom note name
Filename	`FILE RECOVERY.txt`	Dynamic ransom note name
UNC Path	`\\192.168.60.1\admin$\mallox.exe`	Lateral propagation target
UNC Path	`\\192.168.60.1\c$\mallox.exe`	Lateral propagation target
Command	`vssadmin delete shadows /all /quiet`	Recovery inhibition (T1490)
Command	`bcdedit /set {current} recoveryenabled no`	Recovery inhibition (T1490)
Command	`bcdedit /set {current} bootstatuspolicy ignoreallfailures`	Recovery inhibition (T1490)
Command	`taskkill /f /im sqlservr.exe`	SQL Server termination
Command	`taskkill /f /im sqlwriter.exe`	SQL Writer termination
Command	`sc delete "MSSQLSERVER"`	SQL service deletion
Registry Key	`HKLM\System\CurrentControlSet\` Services\LanmanWorkstation\Parameters	Network configuration access

9 Kill Chain Reconstruction

Based on combined static and dynamic analysis, the MalloX ransomware kill chain can be reconstructed as follows:

Stage 1: Initial Execution & Environment Preparation

The malware executes from a staging location (observed in the Temp directory). It performs anti-sandbox stalling via repeated `NtDelayExecution` calls to outlast short observation windows. It checks the system locale using `GetUserDefaultLangID()` and exits early for blacklisted language IDs, likely to avoid targeting systems in operator-affiliated regions (common affiliate behavior).

Stage 2: Privilege Escalation & System Control

The malware loads `PowrProf.dll` and attempts to manipulate power settings via `PowerSetActiveScheme` to prevent sleep or hibernation during encryption. It enables high-impact privileges (`SeTakeOwnershipPrivileg` and `SeDebugPrivilege`) through token manipulation. It creates a hidden window class and calls `ShutdownBlockReasonCreate()` to block user-initiated shutdowns. It also modifies registry-based Group Policy values to hide shutdown, restart, and signout UI elements, effectively trapping the user in a running session.

Stage 3: Victim Profiling & Beacon

The malware collects host fingerprints including computer name, CPU architecture, disk capacity, and access privilege level. It generates per-run randomness using multiple entropy sources (`QueryPerformanceCounter`, `rdtsc`, process/thread IDs, CryptoAPI RNG) for cryptographic material and victim identifiers. It constructs a URL-encoded HTTP POST payload and transmits victim registration data to `http://193.106.191.141/QWEwqdsvsf/ap.php`. It also writes a local `TargetInfo.txt` artifact containing pipe-delimited victim metadata.

Stage 4: Recovery Inhibition

The malware spawns `cmd.exe` children to execute living-off-the-land commands. It deletes all Volume Shadow Copies using `vssadmin delete shadows /all /quiet`. It disables Windows Recovery Environment and configures boot failure handling via `bcdedit` commands. This three-pronged attack eliminates snapshot-based recovery, bootable recovery modes, and automatic recovery triggers (MITRE T1490).

Stage 5: Process & Service Disruption

The malware systematically terminates SQL Server processes using `taskkill /f` commands. It deletes SQL Server services using `sc delete` to prevent automatic restart. This releases file locks on database files, logs, and backups, maximizing the number of encryptable files and increasing operational pain for enterprise victims.

Stage 6: Multi-Threaded Encryption

The malware creates an I/O completion port and spawns worker threads based on CPU count. It enumerates logical drives (A: through Z:), filters by drive type (fixed, removable, network-relevant), and schedules filesystem traversal work. Worker threads pull items from the completion port, traverse directories, and encrypt files using modern symmetric cryptography (AES, ChaCha20 based on YARA hits). The architecture is designed for high-throughput parallel encryption across large directory trees.

Stage 7: Lateral Propagation

The malware attempts to copy itself to remote administrative shares (`\\192.168.60.1\admin$` and `\\192.168.60.1\c$`) using existing network access. This behavior matches the NETAPI32 imports and network share enumeration capabilities observed in static analysis. If successful, the malware can spread within the local network and encrypt additional systems.

Stage 8: Extortion & Ransom Demand

After encryption completes, the malware writes a ransom note to disk (observed names: `C:\HOW TO RECOVER !!.TXT` and `FILE RECOVERY.txt`). The note contains Tor Browser installation instructions, the onion portal URL, a portal authentication path, and an email contact address. Victims are instructed to use a "private ID" to log in and are offered a "free test decryption" to prove operator capability. The note also contains warnings about data publication (double extortion threat) and payment deadlines.

This kill chain demonstrates a mature ransomware operation with clear operator-driven infrastructure (C2 beacon for affiliate tracking), aggressive anti-recovery tactics, enterprisefocused targeting (SQL Server disruption), and multi-faceted extortion leverage (encryption plus credential theft).

10 Conclusion

This analysis dissected MalloX's lifecycle and confirmed it's a sophisticated bully: it checks your locale before attacking (polite ransomware!), kills your backups with vssadmin, phones home via HTTP, then encrypts your databases while you watch. The investigation proved that static analysis finds the smoking gun, Ghidra explains how it fires, and sandbox shows it actually goes bang—no need to reverse-engineer every function like some masochistic completionist. Defense is simple: hide your backups where SMB can't reach them, watch for vssadmin shenanigans, block sketchy C2 domains, and maybe don't let malware casually murder your SQL Server. Or maybe the best option is to not install it from the beginning :).

It was such an interesting experience that now I'm motivated to analyze more malware samples in the future—and maybe even build my own sandbox for dynamic analysis, because apparently I enjoy watching how those binaries could make chaos in such creative ways.