GhostLock: SMB Deny-Share Handles as a Zero-Privilege Availability Weapon

Kim Dvash

Offensive Security Team Leader

linkedin.com/in/kim-d-5b3114111

kimsecops@icloud.com | github.com/kimd155

May 2026

Abstract

We present GhostLock, a technique and accompanying open-source proof-of-concept tool demonstrating that a low-privileged Windows domain user with standard read access to an SMB file share can produce ransomware-equivalent organizational availability impact without writing, renaming, or encrypting any data. By invoking the documented Windows API `CreateFileW` with `dwShareMode = 0x00000000`, a caller acquires an exclusive deny-share handle that prevents all other network clients from accessing the affected file for any purpose until the handle is voluntarily released or the session is forcibly terminated by a storage administrator. Applied systematically across a recursive SMB directory tree using a 32-thread parallel work-stealing scanner, GhostLock acquires hundreds of thousands of exclusive handles within minutes, causing widespread `STATUS_SHARING_VIOLATION (0xC0000043)` failures across business applications, ERP systems, and shared workflow pipelines. Critically, the technique produces zero disk writes, zero file renames, zero extension changes, and zero encryption-related I/O — the precise signals upon which every modern behavioral ransomware defense is predicated. We trace the attack surface from the Windows I/O Manager through the SMB2 protocol wire format to the NAS session table, characterize the detection gap across seven enterprise security control categories, model organizational impact at scale, and identify the structural prevention paradox that makes this technique architecturally resistant to mitigation. No software vulnerability is exploited; the behavior is correct, mandatory, and cannot be restricted without breaking fundamental file integrity semantics across the Windows ecosystem. We conclude with concrete detection engineering guidance and call for storage-layer session telemetry to be treated as a first-class input to enterprise SIEM pipelines.

Keywords: SMB, file locking, ransomware, availability attack, Windows API, deny-share handles, red team, NAS security, enterprise threat modeling, NTFS, detection engineering

1. Introduction

Ransomware has become the defining enterprise cyber threat of the past decade. The security industry has responded with a substantial and increasingly sophisticated defense apparatus: honeypot file systems that detect unexpected writes [1], write-rate anomaly detectors that alert on bulk modification events [2], bulk-rename behavioral monitors [3], Shannon entropy analyzers that identify high-entropy write patterns characteristic of encryption [4], AI-based endpoint agents trained on ransomware execution traces [5], and network-layer controls that inspect SMB traffic for anomalous write volume [6]. The collective global investment in these controls exceeds tens of billions of dollars annually [7].

This paper demonstrates that the entire category of these defenses rests on a single foundational assumption that is empirically incorrect: to deny availability to files, an attacker must transform them. We show that a standard domain user account — the kind obtained through any successful phishing campaign, credential stuffing attack, or malicious insider action — can produce organizational availability impact functionally equivalent to ransomware without producing any of the signals these defenses monitor.

The mechanism is the Windows `CreateFileW` API invoked with `dwShareMode = 0x00000000`. This instructs the Windows I/O Manager to grant the caller an exclusive deny-share handle: a handle that prevents all other processes and network clients from opening the file for any purpose — read, write, or delete — for the duration of the handle's lifetime. The SMB2 protocol transmits this exclusivity request as the `ShareAccess` field of a `CREATE` request, and every major SMB server implementation is required by the protocol specification to honor it.

We call this technique and its associated tool GhostLock. The name reflects the central characteristic: the attack is invisible to every defense that monitors for writes, renames, or encryption. Files are locked. Nothing is written. The ghost leaves no trace.

1.1 Contributions

This paper makes the following technical contributions:

- A systematic threat model framing SMB exclusive file handles as a ransomware-class availability primitive, formally separating the *business objective* of ransomware (inaccessibility and leverage) from the *technical mechanism* (encryption I/O) that current defenses detect.
- A complete technical exposition of the attack surface, tracing the mechanism from the Windows I/O Manager API through the NTFS lock table, the SMB2 wire protocol, and the NAS session file table.
- GhostLock, an open-source Python proof-of-concept implementing the technique with a parallel 32-thread recursive file scanner, exponential-backoff handle acquisition, and file discovery caching for rapid session re-establishment.
- A controlled experimental evaluation measuring lock acquisition speed, blocking effectiveness, and detection evasion across seven enterprise security control categories.

- A quantitative organizational impact model for a representative 5,000-user enterprise with 50 TB of shared NAS infrastructure.
- A structural analysis of the prevention paradox and concrete detection engineering guidance for building storage-layer telemetry pipelines.

1.2 Scope and Ethics

This research was conducted exclusively under explicit written authorization as part of a formal red team engagement. All testing was performed against isolated test volumes provisioned for this purpose. The technique exploits no software defect; it uses documented, correct operating system behavior. The GhostLock tool incorporates a sentinel file safety mechanism that requires the operator to manually place an authorization marker in any target directory before the tool will acquire handles against existing files. We publish this research to enable defenders to understand, model, and detect this threat class.

2. Background and Prior Work

2.1 The Windows File I/O Stack

File access on Windows traverses a layered I/O subsystem. User-mode applications call `CreateFileW` in `kernel32.dll`, which is a thin wrapper over the native API `NtCreateFile` in `ntdll.dll`. The native API transitions to kernel mode via a system call gate, entering the I/O Manager component of the Windows Executive. The I/O Manager constructs an I/O Request Packet (IRP) with type `IRP_MJ_CREATE` and dispatches it through the driver stack associated with the target volume.

For network file access, the redirector driver (`mrxsmb.sys` for SMB) intercepts the IRP and translates it into an SMB2 protocol message. The `dwShareMode` parameter from the original `CreateFileW` call is carried through this entire chain and emerges as the `ShareAccess` field in the SMB2 `CREATE` request sent over the network.

The `dwShareMode` parameter accepts a bitmask of three flags, shown in Table 1. When all three bits are zero — `dwShareMode = 0x00000000` — the caller requests that no other entity be permitted to open the file while this handle remains open. The I/O Manager enforces this through the NTFS share access routines, which maintain a per-file data structure tracking all current openers and their requested share modes. Any incoming `IRP_MJ_CREATE` that conflicts with an existing exclusive handle is failed with `STATUS_SHARING_VIOLATION (0xC0000043)` before the request reaches the file system driver.

Flag Constant	Hex Value	Effect on Concurrent Openers
FILE_SHARE_READ	0x00000001	Allow concurrent read-only opens
FILE_SHARE_WRITE	0x00000002	Allow concurrent write opens
FILE_SHARE_DELETE	0x00000004	Allow concurrent delete or rename
(none set)	0x00000000	Deny all concurrent access — exclusive handle

Table 1: Windows CreateFileW dwShareMode flag definitions.

2.2 NTFS Share Access Internals

The NTFS driver maintains a `SHARE_ACCESS` structure for each open file control block (FCB). This structure tracks the count of openers requesting each combination of read, write, and delete access, and the sharing modes each opener has granted to others. When a new open request arrives, the kernel calls `IoCheckShareAccess` (or `IoCheckShareAccessEx` on modern Windows) to determine whether the requested access conflicts with any existing opener's share mode grant.

For an exclusive handle (`ShareAccess = 0`), the conflict check is simple: if any other handle is open on the file, the new open fails. Conversely, any attempt to open a file that already has an exclusive handle fails regardless of the new opener's requested access mode. This is enforced entirely in kernel memory, before any disk I/O occurs, and cannot be bypassed from user mode.

2.3 SMB2 Protocol Wire Format

The SMB2 protocol [MS-SMB2] transmits file opening parameters in the `SMB2 CREATE` request, defined in section 2.2.13 of the specification. The relevant fields for this research are:

Field	Offset	Size	Description
DesiredAccess	0x18	4 bytes	Requested file access (maps from dwDesiredAccess)
ShareAccess	0x20	4 bytes	Sharing mode bitmask (maps from dwShareMode)
CreateDisposition	0x24	4 bytes	Open/create behavior (maps from dwCreationDisposition)
CreateOptions	0x28	4 bytes	Additional open options and flags

Table 2: Relevant fields in the SMB2 CREATE request structure.

When a Windows SMB client transmits a `CREATE` request with `ShareAccess = 0x00000000`, the SMB server is required by the specification to: (1) record the exclusive open in its session file table, (2) reject any subsequent `CREATE` request targeting the same file with `STATUS_SHARING_VIOLATION`, and (3) maintain this state until the client sends a matching `CLOSE` request or the session is terminated. All tested NAS platforms implement this behavior correctly.

A critical implementation detail: the SMB server enforces share access based on the `ShareAccess` field alone, without regard to the client's identity, privilege level, or any administrator-configured policy. There is no standard SMB2 mechanism for a server to impose a maximum hold duration or a per-user

open file count limit at the protocol layer. These controls, where they exist, are vendor-specific extensions to the session management layer, not SMB2 protocol features.

2.4 NAS Platform Session Tables

Network-attached storage platforms maintain a server-side session table mapping each active client connection to the set of files it has opened, the access and sharing modes for each, and the SMB2 file identifier (`FileId`) assigned at open time. This table is the authoritative record of file lock state from the server's perspective. All NAS platforms tested expose this table through management APIs, but none surface it as a standard security telemetry feed to external monitoring systems.

2.5 Prior Work

Ransomware detection has been studied extensively through the lens of encryption I/O. Scaife et al. [8] introduced CryptoDrop, detecting ransomware through file type changes and Shannon entropy measurements of write operations. Kharraz et al. [9] proposed UNVEIL, using dynamic sandbox analysis to identify ransomware through desktop screenshot changes and file system write activity. Continella et al. [10] developed ShieldFS, a transparent filesystem shim that detects ransomware through write behavior modeling and provides file recovery from shadow copies. Morato et al. [11] analyzed ransomware detection through network traffic write volume patterns. All of these approaches share the foundational assumption that ransomware must write to disk to cause harm.

File locking semantics have been discussed in the distributed systems literature in the context of consistency and concurrency control [12, 13] but have not previously appeared in the offensive security literature as an availability attack primitive. SMB security research has focused on protocol vulnerabilities (MS17-010 [14], NTLM relay attacks [15], credential coercion via UNC path injection [16]) rather than the abuse of correct session-layer behavior. To our knowledge, no prior publication characterizes exclusive SMB file handles as a scalable availability weapon or analyzes the resulting detection gap across the enterprise security stack.

3. Threat Model

We consider the following attacker model, which corresponds to two of the most prevalent enterprise threat scenarios observed in practice.

3.1 Attacker Capabilities

- **Credential access:** A valid Windows domain user account obtained through phishing, credential stuffing, password spraying, or insider threat. No privileged account is required. Standard employee-level access to corporate file shares is sufficient.
- **Network access:** SMB connectivity (TCP port 445) from a Windows endpoint to one or more target file servers. This represents standard intranet access available to any authenticated domain user.

- **Code execution:** The ability to run a Python 3 script on a Windows host. This can be the attacker's own machine (insider scenario) or a compromised endpoint (external attacker post-phishing). No administrative rights on the endpoint are required.
- **Knowledge:** UNC paths to target file shares. In most enterprise environments, share paths are discoverable through DFS namespace enumeration, which is accessible to any authenticated domain user via standard Windows APIs.

3.2 Attacker Constraints

- No elevated privileges on the file server or domain controllers
- No administrative access to network infrastructure
- No ability to install kernel-mode drivers or modify system binaries
- No pre-positioned persistent access beyond the current session
- No ability to exfiltrate data (though the technique does not require exfiltration for leverage)

3.3 Real-World Correspondence

This threat model accurately represents two dominant enterprise attack scenarios. First, the **malicious insider**: a current or departing employee with legitimate credentials and network access who wishes to cause disruption. Second, the **post-initial-access external attacker**: a threat actor who has obtained domain credentials through phishing or credential theft and seeks to maximize impact before being detected. Mandiant's M-Trends 2024 report [17] indicates that credential phishing remains the leading initial access vector, meaning any successful phishing campaign immediately grants the attacker the capability described here.

The attack requires no lateral movement, no privilege escalation, no additional tooling beyond the proof-of-concept script, and no network traffic beyond standard authenticated SMB sessions. It is executable immediately upon obtaining any valid domain credential.

4. Technical Attack Description

4.1 API-Level Mechanism

The GhostLock technique is reducible to a single Windows API call repeated across a set of target files. The call signature and semantically significant parameters are:

```
HANDLE CreateFileW(
    LPCWSTR lpFileName, // UNC path: \\server\share\path\file.ext
    DWORD dwDesiredAccess, // 0x80000000 = GENERIC_READ
    DWORD dwShareMode, // 0x00000000 = deny all sharing <-- weapon
    LPSECURITY_ATTRIBUTES lp..., // NULL = default security
    DWORD dwCreationDisposition, // 0x00000003 = OPEN_EXISTING
    DWORD dwFlagsAndAttributes, // 0x00000080 = FILE_ATTRIBUTE_NORMAL
    HANDLE hTemplateFile // NULL
);
```

Listing 1: CreateFileW invocation with exclusive deny-share handle.

Several parameter choices deserve specific attention. `dwDesiredAccess = GENERIC_READ` requests only read permission, which standard users are granted on most shared files. Requesting write access would fail on read-only shares and leave a different audit trail. The `GENERIC_READ` request maps to a combination of `FILE_READ_DATA`, `FILE_READ_ATTRIBUTES`, `FILE_READ_EA`, and `SYNCHRONIZE` at the kernel level — all permissions routinely granted to standard users.

The critical parameter is `dwShareMode = 0x00000000`. This value does not represent a special mode or an abuse of an edge case. It is the natural zero value of the bitmask, meaning no sharing bits are set. This is the mode used by Microsoft Office when it opens a document for editing — it is the single most common exclusive file open operation in the Windows ecosystem. It is indistinguishable from that operation at every layer of the network and endpoint stack.

4.2 Python Implementation via ctypes

GhostLock implements the API call using Python's `ctypes` module, which provides direct access to Windows API functions from Python without requiring native compilation. This choice means the tool requires no external dependencies beyond the Python standard library and runs on any Windows Python 3.9+ installation.

```
import ctypes, ctypes.wintypes
kernel32 = ctypes.WinDLL('kernel32', use_last_error=True)
GENERIC_READ = 0x80000000
FILE_SHARE_NONE = 0x00000000 # deny all sharing
OPEN_EXISTING = 3
FILE_ATTRIBUTE_NORMAL = 0x00000080
def acquire_exclusive_handle(path: str) -> int:
    handle = kernel32.CreateFileW(
        path,
        GENERIC_READ,
        FILE_SHARE_NONE, # <-- the exclusive lock request
        None,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        None,
    )
    if handle == -1: # INVALID_HANDLE_VALUE
        err = ctypes.get_last_error()
        raise OSError(f'CreateFileW failed: error {err:#010x}')
    return handle
def release_handle(handle: int) -> None:
    kernel32.CloseHandle(handle)
```

Listing 2: Core GhostLock handle acquisition implementation (Python/ctypes).

4.3 Parallel File Discovery Architecture

The most significant engineering challenge in producing a large-scale lock operation against an enterprise NAS is the file discovery phase. A naive recursive `os.walk()` or `pathlib.rglob()` call traverses directories sequentially, each directory listing requiring a separate SMB2 `QUERY_DIRECTORY` round-trip to the server. On a large share with tens of thousands of directories, this produces a sequential chain of network round-trips that can take hours.

GhostLock uses a parallel work-stealing pattern built on `concurrent.futures.ThreadPoolExecutor`. Each worker thread calls `os.scandir()` on a single directory, collects files and subdirectories, and submits each subdirectory as a new task back to the executor. The main thread maintains a set of pending futures and continuously drains completed results, maintaining 32 concurrent directory enumerations at all times. This parallelizes the SMB2 round-trips across all available network bandwidth and reduces discovery wall-clock time by approximately one order of magnitude.

```
def discover_files_fast(root, sentinel, proof_dir, workers=32):
    all_files = []
    lock = threading.Lock()
    with ThreadPoolExecutor(max_workers=workers) as ex:
        pending = {ex.submit(scan_one_dir, root, sentinel, proof_dir)}
        while pending:
            done, pending = wait(pending, return_when=FIRST_COMPLETED)
            for fut in done:
                files, subdirs = fut.result()
                with lock:
                    all_files.extend(files)
                for d in subdirs:
                    pending.add(ex.submit(scan_one_dir, d, sentinel, proof_dir))
    return all_files
```

Listing 3: Parallel recursive file discovery with work-stealing thread pool.

4.4 Handle Acquisition with Exponential Backoff

Large shares frequently have files that are transiently locked by legitimate users — documents open in editing applications, files being processed by background jobs. A naive acquisition loop would permanently skip these files. GhostLock implements exponential backoff retry: each file is attempted up to three times with delays of 0.25s, 0.5s, and 1.0s before being recorded as a permanent skip. This pattern ensures that transient locks do not create persistent gaps in coverage.

```
MAX_RETRIES = 3
BASE_DELAY = 0.25 # seconds
def acquire_with_backoff(path):
    last_exc = None
    for attempt in range(MAX_RETRIES):
        try:
            return acquire_exclusive_handle(str(path))
        except OSError as exc:
            last_exc = exc
            if attempt < MAX_RETRIES - 1:
                time.sleep(BASE_DELAY * (2 ** attempt))
    raise last_exc
```

Listing 4: Exponential backoff handle acquisition.

4.5 File Discovery Caching

If an attacker's session is terminated and must be re-established, re-running the full parallel discovery scan wastes minutes that the victim's IT team may use for response. GhostLock serializes the discovered file list to a JSON cache file after each successful scan. On subsequent runs against the same share, the tool detects the cache, offers to load it, and proceeds directly to handle acquisition — reducing re-lock time from minutes to seconds.

```
def save_cache(files, proof_dir):
    cache = proof_dir / 'ghostlock_cache.json'
    cache.write_text(json.dumps([str(p) for p in files], indent=2))
    return cache
def load_cache(cache_path):
    data = json.loads(cache_path.read_text())
    # Filter to paths that still exist on the share
    return [Path(p) for p in data if Path(p).exists()]
```

Listing 5: Discovery result caching for rapid session re-establishment.

4.6 Operational Modes

GhostLock operates in two modes. **Interactive mode** presents a terminal UI with colored output, prompts the operator for a target UNC path, checks for a sentinel authorization file, runs discovery with a live file count display, and holds locks indefinitely until Ctrl+C. **CLI mode** accepts all parameters as arguments, supports timed holds, victim simulation workers that measure blocking effectiveness, and is suitable for use in automated red team toolchains.

5. Experimental Evaluation

5.1 Experimental Setup

All experiments were conducted under explicit written authorization against isolated test infrastructure provisioned for this purpose. The test environment consisted of a dedicated NAS volume mounted as an SMB share, populated with representative enterprise file content (Office documents, PDF files, project files, configuration files) at three scales: 10,000 files (small), 100,000 files (medium), and 500,000 files (large). The attacking host was a standard Windows 10 workstation connected via 10 GbE LAN. The domain user account had standard read/write permissions on the share.

5.2 Discovery Performance

Table 3 shows discovery times for parallel versus sequential scanning across the three test scales. The parallel scanner maintains approximately a 10x speedup over sequential scanning, with the advantage increasing at larger scales due to deeper directory trees producing longer sequential dependency chains.

Share Size	Files	Directories	Sequential (rglob)	Parallel (32 threads)	Speedup
Small	10,000	800	48s	5.1s	9.4x
Medium	100,000	7,200	7m 34s	48s	9.5x
Large	500,000	38,000	61m 2s	6m 22s	9.6x

Table 3: File discovery time comparison: sequential vs. parallel scanning.

5.3 Handle Acquisition Performance

After discovery, handle acquisition is computationally trivial — each acquisition is a single system call that returns within milliseconds if no conflict exists. Table 4 shows acquisition times and success rates across the three test scales. The small number of skipped files in each run corresponds to files that were genuinely open by legitimate background processes during the test period and could not be locked within the retry window.

Share Size	Files Discovered	Handles Acquired	Skipped	Acquisition Time	Success Rate
Small	10,000	9,971	29	3.2s	99.7%
Medium	100,000	99,614	386	31.4s	99.6%
Large	500,000	498,203	1,797	2m 37s	99.6%

Table 4: Handle acquisition results across three test scales.

5.4 Blocking Effectiveness

To measure the impact on victim clients, GhostLock includes a victim simulation subsystem that spawns configurable worker threads attempting to perform read, write, replace, and delete operations against the locked files. Table 5 shows blocking rates measured during a 60-second hold period across the three scales, with 16 victim workers performing 20 access rounds each.

Scale	Victim Attempts	Blocked	Passed	Block Rate
Small	3,200	3,187	13	99.6%
Medium	3,200	3,190	10	99.7%
Large	3,200	3,193	7	99.8%

Table 5: Victim access blocking rate during 60-second hold period.

The small number of passed attempts corresponds to files in the 0.4% skip category — files the tool was unable to lock due to pre-existing legitimate exclusive handles. For all files successfully locked, the block rate is effectively 100%.

5.5 Detection Evasion Validation

We validated GhostLock's evasion characteristics against a representative enterprise detection stack deployed in the test environment:

- **Honeypot files:** 50 canary files were distributed throughout the test share. None triggered during the GhostLock run. GhostLock opens files with read-only access; canary files are designed to alert on write events.
- **Write-rate detector:** A behavioral monitoring tool configured to alert when any endpoint writes more than 100 files per minute. Zero alerts fired. Write count: zero.

- **EDR (commercial):** The commercial EDR agent on the attacking endpoint produced no alerts during the GhostLock run. The process behavior — opening many remote files for reading — is identical to legitimate backup clients and search indexers.
- **Network monitoring:** NDR analysis of captured SMB2 traffic showed only CREATE and CLOSE requests with no WRITE, SET_INFO, or RENAME operations. The traffic pattern was consistent with a bulk file scan or backup operation.
- **SIEM correlation rules:** No existing correlation rules in the test SIEM triggered. No rule existed for per-session exclusive handle count — confirming the gap identified in the detection analysis.

6. Enterprise Impact Modeling

We model organizational impact against a generic large enterprise environment with thousands of employees and tens of terabytes of shared NAS infrastructure presented through a DFS namespace. The NAS platform supports between 16,000 and 64,000 simultaneously open file handles per SMB session, consistent with major enterprise NAS platforms in production deployment. The organization runs standard enterprise applications dependent on shared file infrastructure including ERP systems, document management platforms, and engineering project stores.

6.1 Single-Session Impact

A single GhostLock session from one compromised endpoint, using one standard user account, can lock up to 64,000 files simultaneously depending on the platform's per-session handle limit. A single session produces the following estimated impact:

Impact Category	Estimated Scope	Business Effect
Shared document store	Up to 64,000 locked files	Departmental file access failure across affected directories
ERP document roots	Selective targeting possible	ERP open and save operations fail immediately
Engineering data store	Selective targeting possible	Project files inaccessible to all concurrent users
Discovery time (large)	Under 10 minutes	Full namespace mapped before detection likely
Recovery time (no prep)	Hours to days	Storage admin expertise required for session termination

Table 6: Estimated single-session impact against a representative large enterprise.

6.2 Multi-Session Amplification

The handle limit is per-session, not per-user. An attacker with access to multiple endpoints — or the ability to open multiple simultaneous SMB sessions from a single endpoint using different connection parameters — can multiply lock coverage linearly with no additional privilege required beyond the initial domain credential. Three simultaneous sessions can maintain over 150,000 exclusive handles concurrently. Ten sessions exceed 500,000 handles — covering a significant fraction of all files in a large enterprise NAS deployment simultaneously.

6.3 Recovery Time Analysis

The standard recovery action for SMB sharing violations caused by a rogue session is to terminate the offending session at the storage platform management layer. This requires an administrator to: (1) identify that a sharing violation condition is the root cause of application failures, which requires familiarity with `STATUS_SHARING_VIOLATION` error codes; (2) access the NAS management interface and enumerate open sessions; (3) correlate the offending session to a source IP and user account; (4) execute a session termination command.

In organizations where the NAS management interface is administered by a separate storage operations team from the security operations team — which describes the majority of large enterprises — this recovery path requires cross-team coordination that is rarely pre-planned. Most incident response runbooks document procedures for isolating endpoints and revoking credentials, but do not include procedures for terminating NAS sessions. Mean time to recovery in a tabletop exercise without a pre-built runbook was estimated at 4-8 hours.

An additional complication: if the attacker's account credentials are revoked as part of the incident response, the existing SMB session and its associated file handles may persist until session timeout (typically 15-60 minutes depending on platform configuration), because the session was already authenticated before credential revocation. The handles do not drop immediately upon account disable in most configurations tested.

7. Detection Gap Analysis

We systematically evaluate GhostLock against each major enterprise detection control category. For each, we describe the detection mechanism, explain why it produces no signal for GhostLock, and note what signal it would require to detect the attack.

7.1 Honeypot and Canary File Systems

Canary files are purpose-placed files distributed throughout a share that alert when written, renamed, or deleted — operations a ransomware process would perform while processing a directory. GhostLock opens canary files with read-only access and an exclusive share mode. The canary file is locked (preventing the operator from opening it) but no write event is generated. The detection mechanism never fires. Detection would require canary files that alert on exclusive-open events — a design that would also trigger on legitimate exclusive opens by applications and is not implemented in any reviewed commercial product.

7.2 Write-Rate Anomaly Detection

These systems establish behavioral baselines of write operation rates per endpoint and alert when rates exceed configured thresholds. GhostLock produces exactly zero write operations against the target share. The metric that these systems monitor is simply absent. No tuning of existing thresholds can make a write-rate detector alert on a workload that contains no writes.

7.3 Behavioral AI Ransomware Detection

AI-based behavioral ransomware detectors in both EDR and network monitoring products are trained on features extracted from observed ransomware executions: Shannon entropy of written data (high for encrypted content), bulk file rename rates, new file extension appearances, and sequential read-then-write patterns on file content. GhostLock's behavioral profile — sequential read-opens across many files with no writes — matches the behavioral signature of a file system indexer, search crawler, or backup pre-scan agent. Commercial behavioral AI products tested produced zero alerts.

7.4 Endpoint Detection and Response (EDR)

EDR agents monitor process behavior, system calls, and network activity on the attacking endpoint. GhostLock's system call profile consists of `NtCreateFile` calls with `ShareAccess = 0` over SMB. This is identical to the system call profile of Microsoft Word opening documents. The GhostLock process is a Python interpreter invoking ctypes — indistinguishable from any other Python file-processing script. No shellcode injection, no code hollowing, no DLL sideloading. No tested EDR product produced an alert.

7.5 Network Detection and Response (NDR)

NDR products analyze SMB2 traffic at the packet level, looking for bulk WRITE operations, unusual CREATE patterns, or anomalous RENAME sequences. GhostLock generates only SMB2 `CREATE` (with `ShareAccess = 0`) and, when sessions end, `CLOSE` requests. The `ShareAccess = 0` value is present in every CREATE request for a document opened in Microsoft Word. At the wire level, a GhostLock session is indistinguishable from a user opening hundreds of Word documents in rapid succession — an operation that would not be considered anomalous in many enterprise environments.

7.6 Data Loss Prevention (DLP)

DLP systems monitor file access for bulk read operations that might indicate data exfiltration. GhostLock's read access pattern — many files opened but minimal data actually read — is consistent with a backup pre-scan, search index update, or antivirus scan. DLP rules typically require both bulk access and data transfer to trigger; GhostLock transfers essentially no data after opening each file.

7.7 Storage Platform Session Telemetry

The only reliable detection signal is the per-session open-file count with `ShareAccess = 0` at the NAS management layer. This metric is accessible via platform-specific management APIs, for example via session file enumeration on platforms that expose this capability. A single session holding thousands of exclusive handles is anomalous; legitimate applications rarely hold more than a few dozen exclusive handles simultaneously, and even a backup agent or search indexer closes handles after processing each file rather than accumulating them indefinitely.

The critical finding is that this metric is not part of the standard telemetry pipeline of any enterprise SIEM reviewed during this research. It is exposed through storage platform management interfaces that are monitored by storage operations teams, not security operations teams, and the two teams rarely share instrumentation pipelines in large enterprises.

Detection Layer	Signal Present?	Why It Fails	What Would Be Needed
Canary files	No	No writes generated	Alert on exclusive-open events
Write-rate anomaly	No	Zero writes	Open-rate anomaly detection
Behavioral AI	No	Profile matches indexer	Exclusive-hold count model
EDR	No	Normal file open syscalls	ShareAccess=0 heuristic
NDR / DPI	No	Identical to Word opens	Per-session open count rule
DLP	No	Minimal data transferred	Open-without-read detection
NAS session table	YES	Metric not SIEM-ingested	Storage telemetry pipeline

Table 7: Comprehensive detection gap analysis across enterprise security controls.

8. Prevention Analysis: The Structural Paradox

GhostLock presents a structural prevention challenge that distinguishes it from most offensive techniques. The four simultaneous obstacles to prevention are described below.

8.1 No Patch Is Possible

`CreateFileW` with `dwShareMode = 0` is specified behavior in the Win32 API, the Windows driver model, and the SMB2 protocol. Microsoft Office uses this mode when opening documents for editing. SQL Server uses exclusive handles for database files during write transactions. Version control systems use exclusive locks during commit operations. Build systems use exclusive handles on output artifacts. Restricting this behavior for standard users would break the file integrity model that the entire Windows application ecosystem depends on. No CVE will be filed; no patch will be issued.

8.2 Network Filtering Cannot Distinguish Attack Traffic

At the SMB2 wire level, a GhostLock `CREATE` request with `ShareAccess = 0` is byte-for-byte identical to a user opening a document in Microsoft Word. Both carry `DesiredAccess = GENERIC_READ` and `ShareAccess = 0x00000000`. The only distinguishing characteristic is *behavioral* — GhostLock accumulates thousands of such handles without closing them, while Word closes each handle when the document is closed. This behavioral difference is only detectable at the session level, not at the individual packet level where DPI operates.

8.3 Endpoint Controls Cannot Distinguish the Workload

From the attacking endpoint's perspective, GhostLock is a Python process making `NtCreateFile` system calls over the network redirector. This profile is indistinguishable from a file synchronization agent, a search indexer, or a backup pre-scan. No process behavior heuristic can flag this workload without simultaneously flagging legitimate applications that perform identical operations.

8.4 Platform-Level Controls Exist But Are Not Deployed

Some NAS platforms expose vendor-specific controls that can limit per-user or per-session open file counts, or intercept open operations via policy engines similar to anti-virus integration interfaces. These controls are documented as performance tuning features, not security controls, and are not enabled in any default configuration reviewed during this research. Enabling them requires storage administration expertise and careful tuning to avoid disrupting legitimate high-volume file access patterns such as those produced by database engines and backup systems.

The net result is a prevention paradox: preventing GhostLock requires restricting behavior that is mandatory for normal enterprise operations, using controls that either do not exist in the standard security product ecosystem or require significant expertise to configure without disrupting legitimate operations. The most practical path forward is detection, not prevention.

9. Detection Engineering Guidance

We provide concrete guidance for building detection capability against the GhostLock technique, ordered from highest to lowest reliability.

9.1 NAS Session Telemetry (Primary)

The most reliable detection signal is the per-session exclusive-handle count from the NAS management layer. Security teams should work with storage administrators to expose this metric and forward it to the SIEM. A detection rule should alert when any single SMB session accumulates more than a configurable threshold of simultaneously open exclusive handles. A reasonable starting threshold is 500 exclusive handles per session, which is well above the typical maximum for any legitimate single-user application but well below the GhostLock operational minimum.

Example SIEM query structure (Splunk SPL pseudocode):

```
index=nas_telemetry sourcetype=smb_session_files
| where share_mode="exclusive" OR share_access=0
| stats count as exclusive_handles by session_id, client_ip, username
| where exclusive_handles > 500
| eval risk_score = case(
    exclusive_handles > 5000, "CRITICAL",
    exclusive_handles > 1000, "HIGH",
    true(), "MEDIUM")
| table _time, username, client_ip, exclusive_handles, risk_score
```

Listing 6: SIEM detection rule for per-session exclusive handle accumulation.

9.2 SMB Open-Without-Write Pattern (Secondary)

A secondary signal detectable from SMB traffic analysis is bulk file opens from a single source IP with no corresponding WRITE, SET_INFO, or RENAME operations over an extended period. A backup agent or search indexer will show this pattern briefly and then stop; GhostLock maintains it indefinitely. A detection rule should alert when a single source IP generates more than a threshold number of SMB CREATE requests with no WRITE operations over a rolling window of 30 minutes or longer.

```
index=smb_traffic
| stats count(eval(smb_command="CREATE")) as opens,
        count(eval(smb_command="WRITE")) as writes
  by src_ip, span=30m
| where opens > 1000 AND writes == 0
| table _time, src_ip, opens, writes
```

Listing 7: NDR detection rule for bulk opens without writes.

9.3 Operational Response Runbook

Detection alone is insufficient without a pre-built response procedure. Organizations should develop and exercise a runbook covering: (1) identification of the offending session ID and source IP from the NAS session table; (2) correlation to a user account via Active Directory or RADIUS authentication logs; (3) execution of the session termination command on the NAS platform; (4) parallel account disable in Active Directory; and (5) verification that handle counts have returned to baseline. This runbook should be owned jointly by the security operations team and the storage operations team, with both teams having access to the NAS management interface during an incident.

10. Mitigation Recommendations

Mitigation	Difficulty	Effectiveness	Operational Impact
Enable per-session open file limits on NAS platform	Medium	High	Low — requires tuning for backup agents and indexers
Ingest NAS session telemetry into SIEM	High	High (detection only)	None
Implement SMB open-without-write rule in NDR	Medium	Medium	None
Restrict shares to read-only where write access is not required	High	Low — read-only exclusive handles still block other readers	Medium — requires workflow review
Build and exercise NAS session termination runbook	Low	High (recovery only)	None
Per-user SMB session limits at network layer	Very High	Medium	High — may disrupt legitimate bulk access patterns

Table 8: Mitigation options with difficulty, effectiveness, and operational impact ratings.

11. Responsible Disclosure

This research was conducted exclusively under explicit written authorization as part of a formal red team engagement. All testing was performed against isolated test volumes provisioned specifically for this research. Findings were disclosed to the assessed organization prior to any public communication.

GhostLock does not exploit a software defect. No CVE has been filed because there is no defect to patch. The technique uses correct, documented Windows API behavior that cannot be changed without breaking the file integrity model of the Windows application ecosystem.

The GhostLock tool incorporates two safety mechanisms designed to prevent unauthorized or accidental use. First, a sentinel file requirement: the tool will not acquire handles against existing files in a directory unless an authorization marker file (`.ghostlock_authorized`) is present in that directory, placed manually by the operator. Second, a generated-files mode that creates its own test files, acquires handles only against those files, and never touches existing data — suitable for demonstrating the technique without any risk to production content.

The tool and this paper are published in the interest of enabling defenders to understand, model, and build detection for this threat class. A threat that has not been publicly characterized cannot be detected.

12. Conclusion

We have demonstrated that ransomware-grade organizational availability impact is achievable by a standard domain user against enterprise shared file infrastructure using a single documented Windows

API call that has been available since Windows NT 3.1. The technique produces none of the signals upon which the modern ransomware defense industry is predicated. Detection requires storage-layer session telemetry that is currently absent from enterprise SIEM pipelines. Prevention is structurally constrained by the impossibility of restricting correct file integrity behavior.

The core finding is not that a new tool has been built. It is that the ransomware defense industry has collectively instrumented for encryption I/O while leaving an entire orthogonal class of availability attack uncharacterized and undetected. The attack surface documented in this paper is present in every enterprise running SMB-backed shared file infrastructure — which is to say, effectively every enterprise on earth.

We call on NAS vendors to expose per-session exclusive-handle counts as a standard security telemetry feed alongside existing syslog and audit outputs. We call on SIEM vendors to build storage platform integrations that ingest this signal. We call on enterprise security teams to evaluate their own exposure using GhostLock in authorized testing scenarios and to develop response runbooks before an attacker forces the issue.

The GhostLock tool and full source code are available at: `https://github.com/kimd155/ghostlock`. The companion research site is at: `https://ghostlock.io`.

References

[1] Al-rimy, B. A. S., Maarof, M. A., & Shaid, S. Z. M. (2018). Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. Computers & Security, 74, 144-166.

[2] Morato, D., Berrueta, E., Magana, E., & Izal, M. (2018). Ransomware early detection by the analysis of file sharing traffic. Journal of Network and Computer Applications, 124, 14-32.

[3] Continella, A., Guagnelli, A., Zingaro, G., et al. (2016). ShieldFS: A self-healing, ransomware-aware filesystem. Proceedings of ACSAC 2016, 336-347.

[4] Sgandurra, D., Munoz-Gonzalez, L., Mohsen, R., & Lupu, E. C. (2016). Automated Dynamic Analysis of Ransomware: Benefits, Limitations and Use for Detection. arXiv:1609.03020.

[5] Ganfure, G. O., Wu, C. F., Chang, Y. W., & Shih, W. K. (2022). DeepGuard: Deep learning-based ransomware detection using system-level analysis. Computers & Security, 118, 102728.

[6] Berrueta, E., Morato, D., Magana, E., & Izal, M. (2020). Open repository for the evaluation of ransomware detection tools. IEEE Access, 8, 65188-65205.

[7] Cybersecurity Ventures. (2023). 2023 Official Cybercrime Report. Cybercrime Magazine.

[8] Scaife, N., Carter, H., Traynor, P., & Butler, K. R. B. (2016). CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. Proceedings of IEEE ICDCS 2016.

[9] Kharraz, A., Arief, B., & Binsalleeh, H. (2016). UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. Proceedings of USENIX Security 2016.

[10] Continella, A., Guagnelli, A., Zingaro, G., et al. (2016). ShieldFS: A self-healing, ransomware-aware filesystem. ACSAC 2016.

[11] Morato, D., et al. (2018). Ransomware early detection by the analysis of file sharing traffic. J. Network and Computer Applications, 124, 14-32.

[12] Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), 558-565.

[13] Gray, J., & Reuter, A. (1992). Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers.

[14] Breen, S. (2017). EternalBlue: A Nation-State Tool Goes Commercial. Rapid7 Blog.

[15] Guo, F. (2017). SMB NTLM Authentication: Attack and Defense. DEF CON 25 Proceedings.

[16] Dvash, K. (2025). MSRC Case 107497: Word LINK Field Security Filter Bypass via Forward-Slash UNC Paths. Responsible disclosure report.

[17] Mandiant. (2024). M-Trends 2024: Special Report on Emerging Threat Trends. Mandiant Threat Intelligence.