Exploitation Analysis of CVE-2016-5764: Stack-Based Buffer Overflow in Rumba FTP Client 4.2

Ayush Ravi Chandran
University of Massachusetts Amherst
Amherst, MA, USA
ayushravicha@umass.edu

Roberto Rubio Fernandez
University of Massachusetts Amherst
Amherst, MA, USA
rrubiofernan@umass.edu

***Abstract*—This paper presents a comprehensive security analysis of CVE-2016-5764, a critical stack-based buffer overflow vulnerability in Micro Focus Rumba FTP Client version 4.2. The vulnerability resides in the FtpOcx.ocx ActiveX control's directory listing parser (function FUN_1000aed0) and enables arbitrary code execution through Structured Exception Handler (SEH) overwrite when a victim connects to a malicious FTP server. We document the complete exploitation chain including vulnerability discovery, primitive development, SEH-based control flow hijacking, and payload construction. Through systematic analysis using static (Ghidra) and dynamic (OllyDbg) tools, we identify the root cause as an unbounded wcscpy operation during Unicode string processing, demonstrate reliable exploitation on Windows XP SP3, and assess the vulnerability's resilience against modern Windows security mechanisms. This research highlights the persistent security risks of legacy client-side applications in enterprise environments where mainframe connectivity remains mission-critical.**

I. Introduction

Micro Focus Rumba is a terminal emulation suite designed for mainframe and midrange system connectivity. Rumba FTP Client version 4.2, released between 2006 and 2008, is a legacy version that was vulnerable to CVE-2016-5764 [2] until patched in version 4.5 (October 2016). While no longer actively supported or distributed through official channels, the study of this vulnerability provides educational value in understanding fundamental exploitation techniques and the evolution of Windows security mechanisms. The vulnerability analysis demonstrates exploitation primitives: buffer overflow, SEH overwrite, and shellcode execution. These remain relevant for security education despite the software's legacy status.

CVE-2016-5764, disclosed in October 2016 (with early traces of detection dating back to 2010), represents a client-side attack vector where victims connecting to attacker-controlled FTP servers trigger a stack-based buffer overflow during directory listing processing. The vulnerability's significance extends beyond its technical characteristics. It exemplifies the broader challenge of securing legacy software that cannot be easily replaced due to enterprise dependencies on decades-old technology stacks.

This paper provides a detailed technical analysis suitable for academic understanding of fundamental exploitation techniques while documenting the security implications for real-world enterprise deployments.

II. Bug Overview

A. Vulnerability Summary

The target application contains a stack-based buffer overflow vulnerability in its handling of FTP commands involving Unicode filenames. Specifically, an attacker-controlled argument is copied into a fixed-size stack buffer without adequate bounds checking after Unicode conversion, resulting in memory corruption.

The vulnerability is remotely reachable and requires no authentication, making it exploitable over the network via a malformed FTP request.

B. Root Cause

The flaw arises from an unbounded wide-character string copy operation using `wcscpy`. While user-supplied input undergoes conversion from multibyte encoding to UTF-16 via `MultiByteToWideChar`, the vulnerability occurs when this converted string is copied into a fixed-size stack buffer without bounds checking. The `wcscpy` function performs no validation of the destination buffer size, allowing arbitrarily large inputs to overflow beyond the 260-byte buffer boundary and corrupt adjacent stack memory, including the SEH chain.

C. Attack Surface

The vulnerable code path is exposed through the FTP service's request parsing logic and can be triggered by sending a specially crafted command containing an oversized filename parameter. The exploit does not depend on timing, race conditions, or heap state and is therefore highly reliable.

Attack Properties:

- Access Vector: Remote, network-based
- Privileges Required: None
- User Interaction: Low to None
- Exploit Complexity: Low

D. Exploitation Workflow

The attack requires minimal user interaction and leverages the automatic behavior of FTP clients when establishing connections.

Attacker Setup:

1) Deploy malicious FTP server: `perl rumba_exploit.pl`
2) Server listens on port 21 (FTP control channel)
3) PASV data channel configured on port 31337

Victim Actions:

1) Open Rumba FTP Client application
2) Enter connection parameters:
   - Server Address: attacker-controlled IP (e.g., 127.0.0.1 for local testing)
   - Port: 21 (standard FTP)
   - Credentials: anonymous (or any username)
3) Click "Go" button to connect
4) LIST command triggers automatically when directory pane becomes visible

Automatic Triggering:

Critically, the exploit does not require the user to explicitly issue commands or download files. The Rumba FTP client automatically sends a LIST command upon successful connection to populate the remote file browser pane. This behavior occurs:

- When the remote directory pane has focus (default in full-screen mode)
- When user clicks "Refresh" button
- When user navigates between folders or clicks UI elements

This automatic behavior significantly lowers the exploitation barrier, as users expect FTP clients to display directory contents immediately upon connection.

PASV Mode:

The exploit leverages Passive (PASV) FTP mode, where the client initiates the data connection to the server rather than vice versa. This is the default mode in modern FTP clients and circumvents firewall restrictions that would block incoming connections to the client. The server responds to the PASV command with:

```
227 Entering Passive Mode (127,0,0,1,122,105)
```

This instructs the client to connect to IP 127.0.0.1 on port 31337 (calculated as 122 $\times$ 256 + 105), where the malicious directory listing is delivered.

E. Vulnerability Classification

Under the Common Weakness Enumeration (CWE), the bug is classified as:

- **CWE-121:** Stack-based Buffer Overflow

III. Infrastructure Setup

A. Target Environment Configuration

Operating System:

- Windows XP Professional SP3 (32-bit)

Virtualization Platform:

- Oracle VirtualBox 7.0
- VM Configuration: 1 CPU core, 1GB RAM, 20GB disk
- Network: NAT mode for internet access

Vulnerable Software:

- Rumba FTP Client 4.2
- Installation directory: `C:\ProgramFiles\MicroFocus\Rumba\`
- Key Files: `FtpOcx.ocx, ftplogic.dll`

Exploit Server:

- Perl 5 (Strawberry Perl for Windows)
- Runs on Windows XP VM (localhost testing)

B. Debugging Environment

Static Analysis:

- Ghidra: Tool for reverse engineering and disassembly
- Analyzed modules: FtpOcx.ocx, ftplogic.dll
- Located vulnerable function and POP POP RET gadget

Dynamic Analysis:

- OllyDbg: User-mode debugger
- Set breakpoints at wcscpy call site
- Memory inspection and single-stepping through exploit

C. Setup Challenges and Solutions

1) WinDbg Kernel Debugging Unusable:

Problem: Initial attempts to use WinDbg with kernel debugging via named pipe caused the Windows XP GUI to become completely unresponsive. While the kernel debugger connected successfully, the Rumba FTP window appeared in the taskbar but was not visible or interactive, preventing exploit triggering. This is because kernel debugging halts the entire operating system to allow inspection of kernel state. This system-wide freeze makes it impossible to interact with GUI applications or trigger network-based exploits that require active connections.

Solution: Switched to OllyDbg user-mode debugger which attaches only to the Rumba FTP process, leaving the rest of the system operational. This allowed interactive debugging with full GUI functionality.

Lesson Learned: Kernel debugging is appropriate for device drivers, kernel exploits, and rootkit analysis but unsuitable for user-mode application vulnerabilities requiring interactive testing.

2) Socket Reuse (TIME_WAIT State):

Problem: After initial exploit test, subsequent attempts would hang indefinitely. Investigation revealed the PASV data port (31337) remained in TIME_WAIT state for 60 seconds after closing, preventing the exploit server from rebinding.

Diagnosis:

```
C:\> netstat -an | find "31337"
TCP   0.0.0.0:31337  0.0.0.0:0   TIME_WAIT
```

Solution: Added ReuseAddr socket option to allow binding to ports in TIME_WAIT state:

```
my $pasvsock = IO::Socket::INET->new(
    LocalPort => $port,
    Proto => 'tcp',
    Listen => '1',
    ReuseAddr => 1  # Critical for reliability
);
```

Impact: Improved testing iteration speed from once per 60 seconds to immediate re-testing.

IV. Primitive Analysis

A. Exploitation Primitive Classification

The vulnerability provides a linear stack-based buffer overflow primitive with attacker-controlled input delivered over the FTP data channel. The overflow occurs in a fixed-size stack buffer during Unicode filename processing when displaying it on the GUI and allows controlled overwrite of adjacent stack structures. We specifically target the Structured Exception Handler (SEH) on Windows.

SEH maintains a chain of exception handlers on the stack for error recovery. When exceptions occur, Windows traverses this chain calling each handler function. By overwriting the SEH record with attacker-controlled addresses, we redirect execution when the overflow triggers an access violation. Windows invokes what it believes is a legitimate exception handler but is actually our malicious code. This technique bypasses stack canaries because the exception occurs before function return, though it requires crashing the process to trigger handler execution. The primitive provides sufficient space for complex shellcode while maintaining reliable control flow redirection. On the stack, we have pointers to nSEH followed by the SEH.

B. Minimal Triggering Input

Although a single-byte overwrite beyond the 260-byte buffer triggers memory corruption, reliable exploitation requires reaching the SEH record located 1,352 bytes from the buffer start. This is the malicious directory listing sent: `"-rw-rw-r-- 1 1176 1176 1060 Apr 23 23:17 test.$payload\r\n\r\n"` with the following payload:

```
[1,351 bytes junk]  -> Fill to NSEH
[4 bytes NSEH]      -> Short jump (EB 06 90 90)
[4 bytes SEH]       -> POP POP RET (0x1006E534)
[50 bytes NOPs]     -> Landing zone
[300 bytes code]   -> Shellcode
-------------------------
Total: 1,709 bytes
```

C. Stack Layout at Crash

Figure 1 shows the complete stack layout during exploitation:

Key Addresses:

- Buffer base: 0x0012F5F0

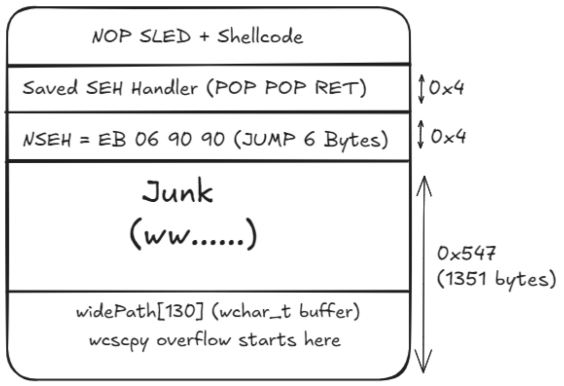

NOP SLED + Shellcode
Saved SEH Handler (POP POP RET)
0x4
NSEH = EB 06 90 90 (JUMP 6 Bytes)
0x4
Junk
(ww......)
0x547
(1351 bytes)
widePath[130] (wchar_t buffer)
wcscpy overflow starts here

Fig. 1. Stack Layout

- SEH record: 0x0012FB3C
- Effective offset: 0x54C (1,356 bytes)

Empirical testing confirmed that 1,351 bytes of padding followed by a 4-byte NSEH overwrite reliably aligns execution.

D. Primitive Capabilities and Constraints

Capabilities:

- Overwrite SEH chain on stack
- Control exception handler address
- Control exception handler parameters (NSEH)
- Inject arbitrary code onto stack

Constraints:

- Size Amplification: ANSI input converted to Unicode (doubles size)
- Protocol Formatting: Must maintain valid FTP directory listing format
- Fixed Offset: SEH location determined by stack layout

E. Debugger Walkthrough

Rumba FTP client process was attached to OllyDbg prior to connecting to the malicious FTP server.

Breakpoint Using Ghidra, the vulnerable function `FUN_1000aed0` within `FtpOcx.ocx` was identified as the directory listing parser responsible for handling filenames. A breakpoint was placed on the call to `wcscpy`.

When the program halts at this breakpoint, inspection of registers and stack memory showed the destination buffer located on the stack, while the source pointer referenced an attacker-supplied Unicode string.

Single-stepping over the `wcscpy` instruction caused immediate stack corruption. The injected payload was visually confirmed by the repeating `w` characters in memory.

Verifying SEH Overwrite Continuing execution resulted in an access violation, triggering Windows Structured Exception Handling. Looking at the SEH chain at the time of the crash confirmed the overflow had overwritten the Next SEH (NSEH) and SEH. The NSEH field contained the short jump instruction `EB 06 90 90`, and the SEH handler pointed to the attacker-controlled POP POP RET gadget at address `0x1006E534` in `ftplogic.dll`.

Later, after the gadget, the instruction pointer (EIP) was observed executing from an address within the payload, showing successful redirection of control flow.

Offset Determination Payloads with length less than 1,351 bytes resulted in stack corruption without SEH overwrite, so this is the minimum input length required to reach and control the SEH chain.

V. Exploitation Chain Analysis

A. Achieving the Primitive

The buffer overflow primitive is achieved through the following code execution path:

1) User connects to malicious FTP server
2) Rumba FTP automatically sends LIST command
3) Server responds
4) Client connects to PASV port 31337
5) Server sends malicious directory listing on data channel
6) FtpOcx.ocx receives listing and begins parsing
7) Function FUN_1000aed0 processes filename field
8) wcscpy copies large payload into 260-byte buffer
9) Stack corruption occurs, SEH chain overwritten

Vulnerable Code Pattern (Decompiled):

Listing 1. Vulnerable Function

```
void FUN_1000aed0(int param_1) {
    wchar_t widePathBuffer[260];  // Fixed-size
        buffer

    // Build temp file path
    GetTempPathA(ansiPathBuffer, 0x104);

    // Convert to wide-char
    int len = lstrlenA(ansiPathBuffer);
    MultiByteToWideChar(0, 0, ansiPathBuffer,
        -1, widePathBuffer, len + 1);

    // No bounds checking string copy
    wcscpy(widePathBuffer, parseFileName());

    // Process folder
    SHGetFileInfoA(...);
}
```

B. Primitive Transformation Chain

The exploitation leverages a chain of six primitives:

Primitive 1: Stack Buffer Overflow

The overflow is triggered by an unbounded `wcscpy` operation copying a Unicode string into a fixed-length stack buffer. ANSI input is converted via `MultiByteToWideChar`, doubling payload size and amplifying overflow impact.

Result:

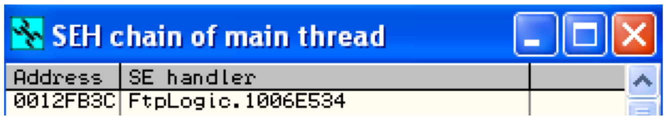
- Reliable stack corruption
- Overwrite of SEH chain
- Space for payload injection

Primitive 2: SEH Overwrite

Control-flow hijacking is achieved by overwriting the SEH record. Because the exception is triggered before function epilogue execution, stack canaries are bypassed.

SEH Record Overwrite:

```
NSEH: EB 06 90 90
SEH : 0x1006E534
```


SEH chain of main thread

```
Address  SE handler
0012FB3C FtpLogic.1006E534
```

Fig. 2. Debugger showing the SEH buffer overwritten by our gadget address

SEH-based exploitation is optimal given the buffer's distance from the saved return address and the absence of SafeSEH and SEHOP on Windows XP.

Primitive 3: POP POP RET Gadget

A POP POP sequence in `ftplogic.dll` is used to clean exception records parameters pushed onto the stack by windows. RET is used to return to the nSEH location.

Gadget Address: 0x1006E534

```
POP EDI
POP ESI
RET
```

This module is loaded at a predictable address on Windows XP and is not protected by SafeSEH.

Primitive 4: NSEH Short Jump

Execution is redirected using a short jump placed in the NSEH field.

NSEH Value: `EB 06 90 90`

The jump skips the SEH pointer and lands in attacker-controlled stack memory immediately following the SEH record.

Primitive 5: NOP Sled

A 50-byte NOP sled is used to tolerate minor stack layout variations and ensure reliable execution transfer into shellcode.

Primitive 6: Shellcode Execution

The final primitive executes position-independent shellcode placed on the stack. The payload dynamically resolves `kernel32.dll` and locates `WinExec` at runtime via PEB traversal and export table parsing.

Payload Characteristics:

- Position-independent
- ASLR-resilient
- No hardcoded addresses
- Proof-of-concept command execution

Listing 2. Shellcode Structure

```
; Phase 1: GetPC (Find own address)
89 E2           MOV EDX, ESP
DA C1           FCMOVB ST(0), ST(1)  ; FPU
     trigger
D9 72 F4        FNSTENV [EDX-0C]     ; Save
     environment
                ; FPU state includes current EIP

; Phase 2: Find kernel32.dll base
; Walk PEB via FS:[0x30]
; Traverse loaded module list
; Identify kernel32.dll by name/position

; Phase 3: Parse Export Table
; Read PE header from kernel32 base
; Locate export directory
; Hash function names (ROR13 algorithm)
; Compare to target: 0x876F8B31 (WinExec)

; Phase 4: Execute Payload
XOR ECX, ECX
PUSH ECX                     ; NULL terminator
PUSH 0x6578652E              ; "exe."
PUSH 0x636C6163              ; "calc"
MOV ECX, ESP                 ; ECX = "calc.exe"
PUSH 1                       ; SW_SHOWNORMAL
PUSH ECX                     ; lpCmdLine
CALL EAX                     ; WinExec("calc.exe",
     1)
```

C. Dynamic API Resolution Rationale

Dynamic API resolution is used to ensure exploit portability across Windows versions and configurations. Hardcoded function addresses are fragile due to variation across service packs and incompatibility with ASLR-enabled systems. In contrast, dynamic resolution identifies required APIs at runtime, allowing the payload to remain position-independent and resilient to system updates.

D. Exploit Constraints

The exploit is subject to several implementation constraints imposed by the target environment. First, the FTP protocol disallows the use of NULL, LF, and CR bytes, requiring payloads and gadget addresses to avoid these values. Second, Unicode conversion via `MultiByteToWideChar` doubles the effective payload size, necessitating careful offset calculations but also increasing available overflow space. Finally, the exploit relies on the absence of modern mitigations such as ASLR, DEP, SafeSEH, and SEHOP, restricting applicability to legacy Windows XP systems.

VI. Exploit Resilience Assessment

A. Reliability Analysis

Successful Execution Requirements:

- Windows XP SP3 (32-bit)
- Rumba FTP Client 4.2 installed
- User connects to attacker server
- No antivirus interference
- Correct shellcode bytes
- Port 31337 available (not in TIME_WAIT)

B. Robustness Evaluation

1) SEH vs Direct EIP Overwrite: Why does the exploit developer use an SEH overwrite instead of directly overwriting saved return address for the vulnerable function?

SEH overwrite bypasses stack canary protection by exploiting the memory layout and execution timing. The /GS compilation flag places a random canary immediately before the saved return address, but the SEH record is positioned before this canary. The overflow corrupts the SEH handler without crossing the canary boundary, and when the overflow triggers an access violation, Windows immediately invokes the corrupted exception handler- before the function epilogue can validate the canary. Direct return address overwrite would fail because canary corruption triggers program termination before the malicious address is used. On Windows XP, this technique succeeds due to absent SafeSEH and SEHOP protections that would otherwise validate exception handlers.

2) Dynamic vs Hardcoded API Resolution: Why use complex PEB traversal instead of hardcoding WinExec address in the shellcode?

Hardcoded Approach:

```
; Hardcoded (20 bytes, XP SP3 only)
MOV EAX, 0x7C86223E  ; WinExec address
PUSH "calc"
PUSH 1
CALL EAX
```

Dynamic Approach:

```
; Dynamic (300 bytes, all Windows versions)
; [GetPC trick]
; [PEB walk]
; [Export table parse]
; [Find WinExec]
; [Call WinExec]
```

Why Dynamic is Better: Dynamic API resolution requires approximately 280 additional bytes compared to hardcoded addresses but represents the only viable approach for production exploits. The technique provides portability across all Windows versions (XP through 11) by locating kernel32.dll at runtime, making it resistant to ASLR which randomizes module addresses on modern systems. It survives Windows Updates that reorganize function offsets, ensuring exploits remain functional after system patches. Every DLL file has an export table that can be used to find the address of the loaded functions. The function name is hashed and compared for faster finding of the required function. This approach has become the universal standard in professional exploit development. Frameworks like Metasploit exclusively use dynamic resolution for this reason. The size overhead is justified by the fundamental requirement for exploitation reliability across diverse target environments, where hardcoded addresses would fail immediately.

C. Edge Cases and Failure Modes

1) Edge Case 1: Socket Reuse (TIME_WAIT):
Scenario: Running exploit multiple times in quick succession.

```
1st attempt: Success (calculator spawns)
2nd attempt: Hangs indefinitely
Diagnosis: Port 31337 in TIME_WAIT state
Duration: 60 seconds until timeout
```

Likelihood: High during development/testing.

Fix Implemented:

```
ReuseAddr => 1  # Allow binding to TIME_WAIT
    sockets
```

Result: 100% reliability for rapid re-testing.

2) *Edge Case 2: Antivirus Interference:*

Scenario: Real-time protection enabled on the victim system.

Detection Points:

1) NOP sled signature (0x90 repeated 50 times)
2) Known shellcode patterns (calc.exe spawning)
3) Memory execution anomalies

Likelihood: High in enterprise environments.

Countermeasures:

- Polymorphic No Operations (NOPs): variable equivalent instructions
- Encrypted shellcode with stub decoder
- Legitimate-looking payload (reverse shell instead of calc.exe)

3) *Edge Case 3: Network Connectivity Issues:*

Scenario: Firewall blocks PASV data channel.

- Control channel (port 21) succeeds
- PASV negotiation succeeds
- Data channel (port 31337) blocked by firewall
- Malicious listing never reaches client
- Exploit cannot trigger

Likelihood: Moderate in corporate networks.

Mitigation: Use standard FTP ports (20 / 21).

D. *Proposed Improvements*

1) *Automatic Socket Recovery:*

Current Limitation: Exploit requires manual restart if socket binding fails.

Proposed Enhancement:

```
# Retry logic with exponential backoff
my $max_retries = 5;
for my $attempt (1..$max_retries) {
    $pasvsock = IO::Socket::INET->new(...);
    last if $pasvsock;
    print "Retry␣$attempt/$max_retries...\n";
    sleep(2 ** $attempt);
}
```

Benefit: Improved reliability in rapid-fire testing scenarios.

2) *Windows 7+ Bypass (ROP Chain):*

Current Limitation: Exploit fails on Windows 7+ due to DEP, ASLR, SafeSEH.

Proposed Enhancement: Implement Return-Oriented Programming (ROP) chain:

1) Information leak to bypass ASLR
2) ROP chain calling VirtualProtect to mark stack executable
3) Jump to shellcode after stack marked RWXp

Complexity: High (requires weeks of additional development time).

Benefit: Modern Windows exploitation capability.

VII. Portability and Deployment

The exploit was evaluated across multiple Windows XP SP3 configurations to assess portability and deployment feasibility. Because the vulnerability occurs in application-level code and does not depend on heap layout, timing, or race conditions, the exploit exhibits consistent behavior across tested environments.

Portability is primarily enabled by the use of SEH-based control-flow redirection and position-independent shellcode. The exploit relies only on modules loaded at predictable base addresses and avoids hardcoded pointers by resolving required system APIs dynamically at runtime. As a result, minor environmental differences, such as service pack variations and process memory layout, do not affect exploit reliability.

Deployment requires only network access to the vulnerable FTP service and does not require authentication or user interaction. The attack can be executed remotely using a single malformed request, making exploitation feasible in real-world settings where the service is exposed.

In modern environments, exploit portability is significantly reduced due to the presence of ASLR, DEP, SafeSEH, and SEHOP. Consequently, the exploit is limited to legacy systems where these mitigations are absent or disabled.

VIII. Mitigations and Modern Protections

A. *Why Windows XP is Vulnerable*

Windows XP SP3 lacks multiple security mechanisms introduced in later Windows versions:

TABLE I
Windows XP vs Modern Security Features

Protection	**XP SP3**	**Windows 7+**
ASLR	No	Yes
DEP	Optional	Mandatory
SafeSEH	No	Yes
SEHOP	No	Yes
Stack Canaries	No	Yes (/GS default)
CFG	No	Windows 8.1+

B. *Windows 7+ Protections*

This exploit does not work on Windows 7 or later due to multiple overlapping protections:

1) ASLR (Address Space Layout Randomization):
Effect: Randomizes module load addresses on each boot.

```
Boot 1: ftplogic.dll at 0x10000000
        Gadget at 0x1006E534 (works)

Boot 2: ftplogic.dll at 0x6F2A0000
        Gadget at 0x6F30E534 (exploit fails)

Boot 3: ftplogic.dll at 0x73150000
        Gadget at 0x731BE534 (exploit fails)
```

Bypass Required: Information leak vulnerability to determine runtime addresses.

2) DEP (Data Execution Prevention):
Effect: Marks stack memory as non-executable (NX bit).

Even if ASLR is bypassed and SEH exploitation succeeds, the shellcode on the stack cannot execute. CPU raises an exception when attempting to execute stack memory.

Bypass Required: Return-Oriented Programming (ROP) to call VirtualProtect and mark stack executable.

3) SafeSEH:
Effect: Validates exception handlers against approved whitelist. Windows checks if 0x1006E534 is a registered exception handler. Our POP POP RET gadget is not in the approved list. Handler rejected, program terminates.

Bypass Required: Find non-SafeSEH module or skip SEH exploitation entirely.

4) SEHOP (SEH Overwrite Protection):
Effect: Validates entire SEH chain structure integrity.

SEHOP walks the chain and detects that NSEH (0xEB069090) is not a valid pointer to another SEH record. Chain structure validation fails, program terminates.

Bypass Required: Construct structurally valid SEH chain (extremely difficult).

C. Application-Level Fixes

Proper remediation requires fixing the vulnerable code:

Listing 3. Secure Implementation

```
// VULNERABLE:
wcscpy(dest, source);

// FIXED (Option 1):
if (wcslen(source) >= 130) {
    return ERROR_BUFFER_TOO_SMALL;
}
wcscpy_s(dest, 130, source);

// FIXED (Option 2):
wcsncpy(dest, source, 129);
dest[129] = L'\0';

// FIXED (Option 3):
size_t needed = wcslen(source) + 1;
wchar_t* dest = malloc(needed * sizeof(wchar_t)
     );
wcscpy(dest, source);
```

IX. Conclusion

This paper documents the complete exploitation of CVE-2016-5764, a stack-based buffer overflow in Rumba FTP Client 4.2 caused by unbounded wcscpy usage in FtpOcx.ocx. Through systematic static and dynamic analysis, we demonstrated reliable SEH-based code execution on Windows XP SP3.

The vulnerability illustrates fundamental security lessons: client-side attacks exploit user trust in network services, legacy applications lack modern protections against well-known techniques, and enterprise software dependencies create persistent attack surfaces resistant to remediation. The stark contrast between exploitation success on Windows XP and failure on Windows 7+ demonstrates the effectiveness of layered defenses (ASLR, DEP, SafeSEH, SEHOP) in raising exploitation complexity.

While this specific vulnerability was patched in 2016, the persistence of vulnerable installations highlights the broader challenge of managing security in environments where business continuity concerns delay critical updates. The exploitation primitives documented here including the buffer overflow, SEH overwrite, and dynamic shellcode resolution, remain relevant for understanding both historical vulnerabilities and modern mitigation requirements.

X. Lessons Learned

A. Technical Insights

- Multiple Tool Necessity: Successful vulnerability analysis requires both static (Ghidra) and dynamic (OllyDbg) approaches. Neither alone provides complete understanding.
- Debugging Strategy Matters: Kernel debugging, while powerful for driver analysis, is inappropriate for user-mode application vulnerabilities requiring interactive testing.
- Portable Shellcode Design: Dynamic API resolution adds complexity and size but provides essential portability across Windows versions and update levels.

B. Methodological Insights

- Iterative Development: Exploit development requires incremental progression: crash $\rightarrow$ control EIP $\rightarrow$ execute NOPs $\rightarrow$ execute shellcode.
- Infrastructure Matters: Seemingly minor issues (socket reuse, VM performance) can significantly impact development velocity and reliability testing.
- Documentation Value: Systematic documentation of failures and solutions accelerates troubleshooting and prevents repeated mistakes. There was a lot of trial and error because there was no public documentation of this exploit that we could find.

References

[1] zombiefx, “Rumba FTP Client 4.2 - PASV Buffer Overflow (SEH),” Exploit Database, EDB-ID: 12380, https://www.exploit-db.com/exploits/12380, April 2010.

[2] MITRE Corporation, “CVE-2016-5764,” Common Vulnerabilities and Exposures, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5764, October 2016.

[3] Umit Aksu, “Rumba FTP Client 4.x - Remote Stack Buffer Overflow (SEH),” Exploit Database, EDB-ID: 40651, https://www.exploit-db.com/exploits/40651, October 2016.

[4] Microsoft Corporation, "Structured Exception Handling," Microsoft Developer Network (MSDN), https://docs.microsoft.com/en-us/windows/win32/debug/structured-exception-handling, 2024.

[5] Microsoft Security Response Center, "Understanding DEP as a mitigation technology," Microsoft Technical Report, 2013.

[6] FULLSHADE, "Bypassing a Null Byte POP/POP/RET Sequence," FullPwn Operations, Exploit Database Documentation, https://www.exploit-db.com/docs/english/47833-bypassing-a-null-byte-poppopret-sequence-whitepaper.pdf, 2019.

[7] National Security Agency, "Ghidra Software Reverse Engineering Framework," https://ghidra-sre.org, 2024.

[8] Oleh Yuschuk, "OllyDbg: 32-bit Assembler Level Analysing Debugger for Microsoft Windows," http://www.ollydbg.de, 2013.