


## [Source] x64 return address spoofing (source + explanation)

---

 [unknowncheats.me/forum/anti-cheat-bypass/268039-x64-return-address-spoofing-source-explanation.html](https://unknowncheats.me/forum/anti-cheat-bypass/268039-x64-return-address-spoofing-source-explanation.html)

C++:

Code:

```

1. #include <type_traits>
2. namespace detail
3. {
4.     extern "C" void* _spoofer_stub();
5.     template <typename Ret, typename... Args>
6.     static inline auto shellcode_stub_helper(
7.         const void* shell,
8.         Args... args
9.     ) -> Ret
10.    {
11.        auto fn = (Ret (*)(Args...))(shell);
12.        return fn(args...);
13.    }
14.    template <std::size_t Argc, typename>
15.    struct argument_remapper
16.    {
17.        // At least 5 params
18.        template<
19.            typename Ret,
20.            typename First,
21.            typename Second,
22.            typename Third,
23.            typename Fourth,
24.            typename... Pack
25.        >
26.        static auto do_call(
27.            const void* shell,
28.            void* shell_param,
29.            First first,
30.            Second second,
31.            Third third,
32.            Fourth fourth,
33.            Pack... pack
34.        ) -> Ret
35.        {
36.            return shellcode_stub_helper<
37.                Ret,
38.                First,
39.                Second,
40.                Third,
41.                Fourth,
42.                void*,
43.                void*,
44.                Pack...
45.            >(
46.                shell,
47.                first,
48.                second,
49.                third,
50.                fourth,
51.                shell_param,

```

```

52.             nullptr,
53.             pack...
54.         );
55.     }
56. };
57. template <std::size_t Argc>
58. struct argument_remapper<Argc, std::enable_if_t<Argc <= 4>>
59. {
60.     // 4 or less params
61.     template<
62.         typename Ret,
63.         typename First = void*,
64.         typename Second = void*,
65.         typename Third = void*,
66.         typename Fourth = void*
67.     >
68.     static auto do_call(
69.         const void* shell,
70.         void* shell_param,
71.         First first = First{},
72.         Second second = Second{},
73.         Third third = Third{},
74.         Fourth fourth = Fourth{}
75.     ) -> Ret
76.     {
77.         return shellcode_stub_helper<
78.             Ret,
79.             First,
80.             Second,
81.             Third,
82.             Fourth,
83.             void*,
84.             void*
85.         >(
86.             shell,
87.             first,
88.             second,
89.             third,
90.             fourth,
91.             shell_param,
92.             nullptr
93.         );
94.     }
95. };
96. }
97. template <typename Ret, typename... Args>
98. static inline auto spoof_call(
99.     const void* trampoline,
100.     Ret(*fn)(Args...),
101.     Args... args
102. ) -> Ret

```

```

103. {
104.     struct shell_params
105.     {
106.         const void* trampoline;
107.         void* function;
108.         void* rbx;
109.     };
110.     shell_params p{ trampoline, reinterpret_cast<void*>(fn) };
111.     using mapper = detail::argument_remapper<sizeof...(Args), void>;
112.     return mapper::template do_call<Ret, Args...>((const
        void*)&detail::_spoofer_stub, &p, args...);
113. }

```

## *MASM64:*

Code:

```

1.  COMMENT ~
2.  PUBLIC _spoofer_stub
3.  .code
4.  _spoofer_stub PROC
5.      pop r11 ~ popping without setting up stack frame, r11 is the return address (the
        one in our code)
6.      add rsp, 8 ~ skipping callee reserved space
7.      mov rax, [rsp + 24] ~ dereference shell_param
8.      mov r10, [rax] ~ load shell_param.trampoline
9.      mov [rsp], r10 ~ store address of trampoline as return address
10.     mov r10, [rax + 8] ~ load shell_param.function
11.     mov [rax + 8], r11 ~ store the original return address in shell_param.function
12.     mov [rax + 16], rbx ~ preserve rbx in shell_param.rbx
13.     lea rbx, fixup
14.     mov [rax], rbx ~ store address of fixup label in shell_param.trampoline
15.     mov rbx, rax ~ preserve address of shell_param in rbx
16.     jmp r10 ~ call shell_param.function
17.  fixup:
18.      sub rsp, 16
19.      mov rcx, rbx ~ restore address of shell_param
20.      mov rbx, [rcx + 16] ~ restore rbx from shell_param.rbx
21.      jmp QWORD PTR [rcx + 8] ~ jmp to the original return address
22.  _spoofer_stub ENDP
23.  END

```

## **Example usage:**



This image has been resized. Click this bar to view the full image. The original image is sized 667x419.

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

int main();

// Somewhere in a legit module
#pragma section(".text")
__declspec(allocate(".text")) const unsigned char jmp_rbx_0[] = { 0xFF, 0x23 };

int function(int a, int b, int c, int d, int e)
{
    const auto ret = _ReturnAddress();
    printf("%d\t%d\t", ret == main, ret == jmp_rbx_0);
    return a + b + c + d + e;
}

int main()
{
    const auto ret = spoof_call(jmp_rbx_0, &function, 1, 2, 3, 4, 5);
    printf("%d\n", ret);
}
```

C:\Windows\system32\cmd.exe

0 1 15  
Press any key to continue . . .