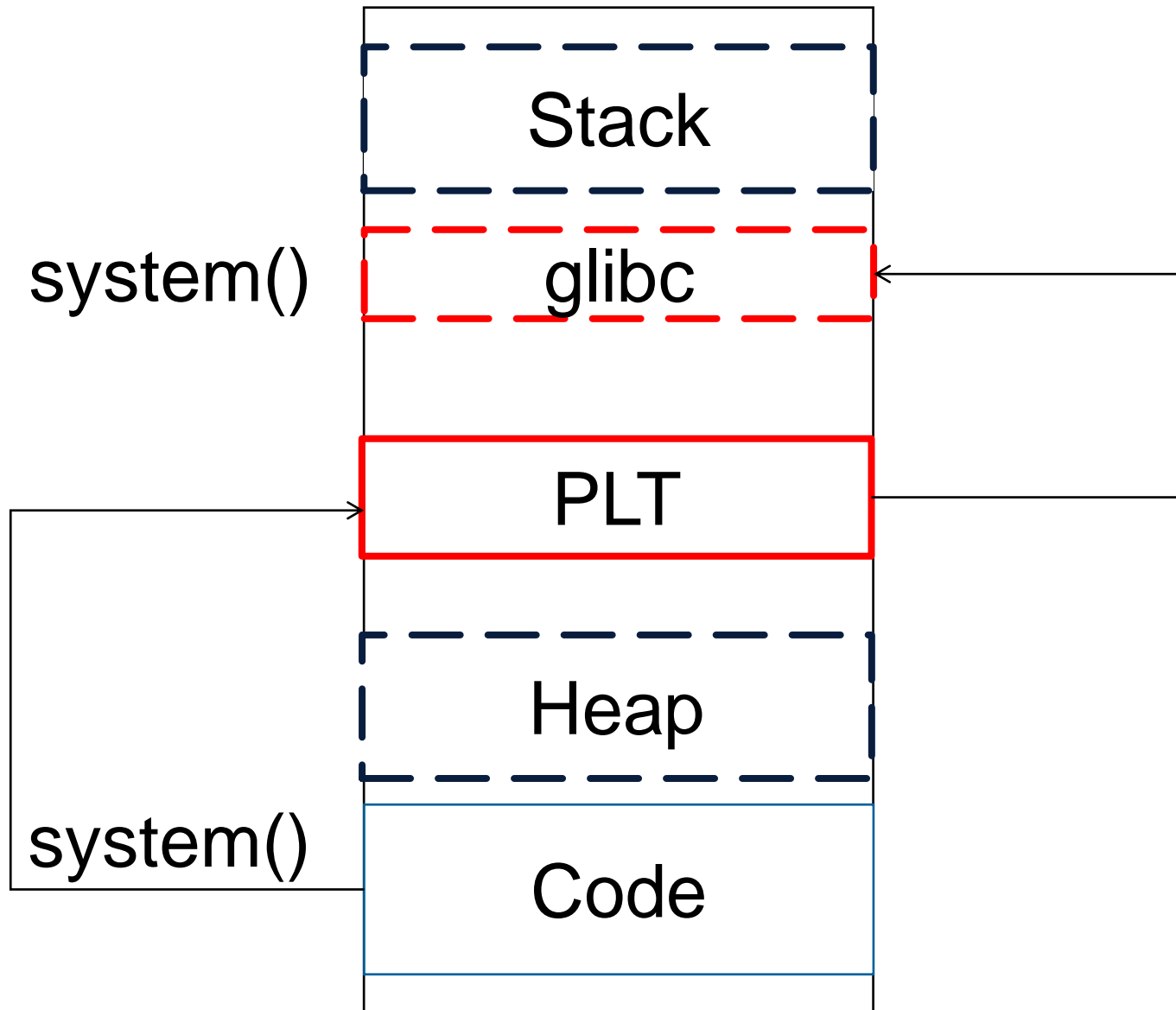


Recap ret2plt

Defeating DEP – Shared Libraries Intro





Return to X

.code:

call <system@plt>

.plt:

jmp *<system@got>

.got:

&system@libc

system@libc:

[Code]

ret2plt

ret2got

ret2libc

Ret2plt exploit in 32 bit

char buffer[64]	SIP
-----------------	-----	-----	-----

rm -rf /	&system@plt	...	&arg
----------	-------------	-----	------



Ret2plt in 32 bit – before ret

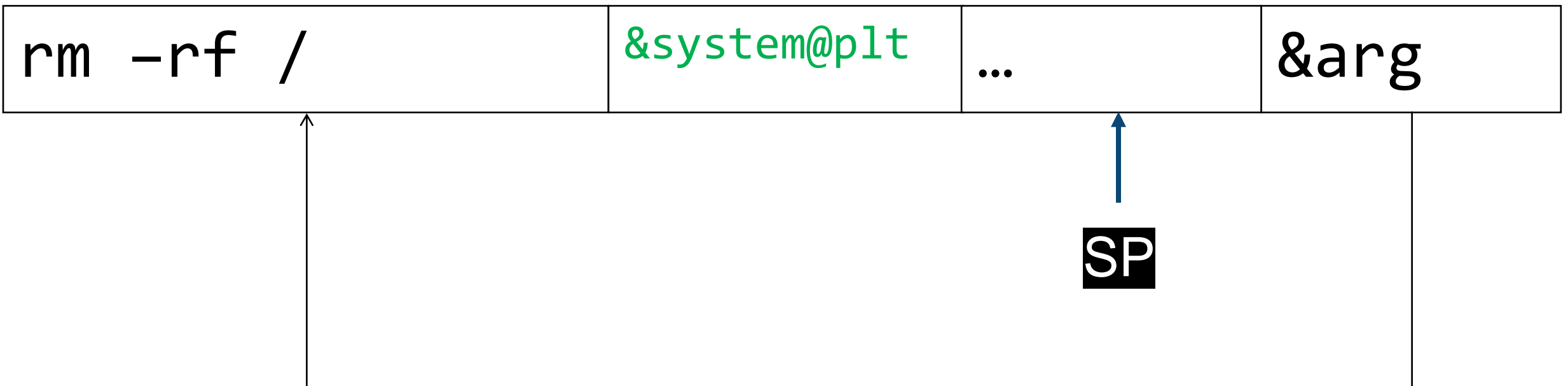
char buffer[64]	SIP
-----------------	-----	-----	-----

rm -rf /	&system@plt	...	&arg
----------	-------------	-----	------

SP

The diagram illustrates the state of the stack before a return instruction. The stack grows downwards. The first row contains 'char buffer[64]', 'SIP' (in green), and two ellipses. The second row contains 'rm -rf /', '&system@plt' (in green), an ellipsis, and '&arg'. A blue arrow points from a black box labeled 'SP' to the first cell of the second row, 'rm -rf /'. A vertical line with an upward-pointing arrow is positioned to the left of the 'rm -rf /' cell, indicating the current stack pointer position.

Ret2plt in 32 bit – after ret



Stack Layout – after call

Saved IP (&__libc_start)

Saved Frame Pointer

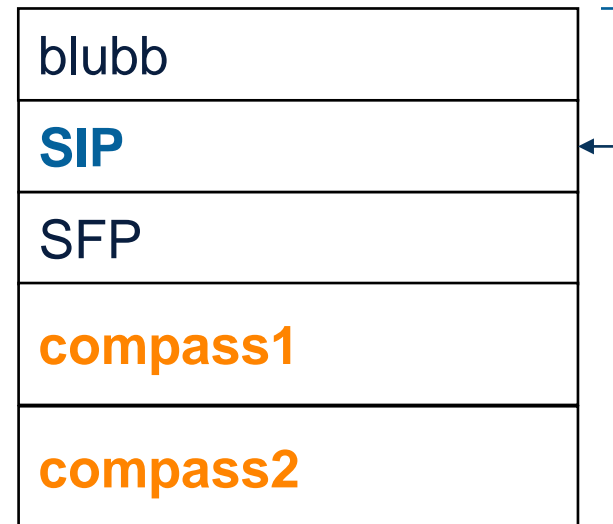
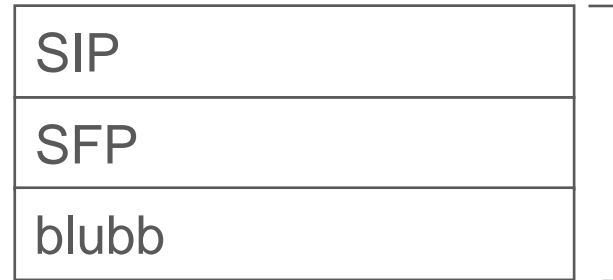
Local Variables <main>

Argument for <foobar>

Saved IP (&return)

Saved Frame Pointer

Local Variables <foobar>



SP

push ↗ ↘ pop

Ret2plt in 64 bit



Function Call Convention Cheat Sheet

x32	Parameter	Syscall nr in
x32 userspace	stack	
x32 syscalls	ebx, ecx, edx, esi, edi, ebp	eax

x64	Parameter	Syscall nr in
x64 userspace	rdi, rsi, rdx, rcx, r8, r9	
x64 syscall	rdi, rsi, rdx, r10, r8, r9	rax

<http://stackoverflow.com/questions/2535989/what-are-the-calling-conventions-for-unix-linux-system-calls-on-x86-64>

Ret2plt exploit in 64 bit

char buffer[64]	SIP
-----------------	-----	-----	-----

rm -rf /	&system@plt
----------	-------------	-----	-----

rdi

