# Shellcode

## Summary

In this exercise, we will learn about shellcode.

## Prerequisites

Setup an Ubuntu VM as outlined in the VM setup instructions on Blackboard.

## Details

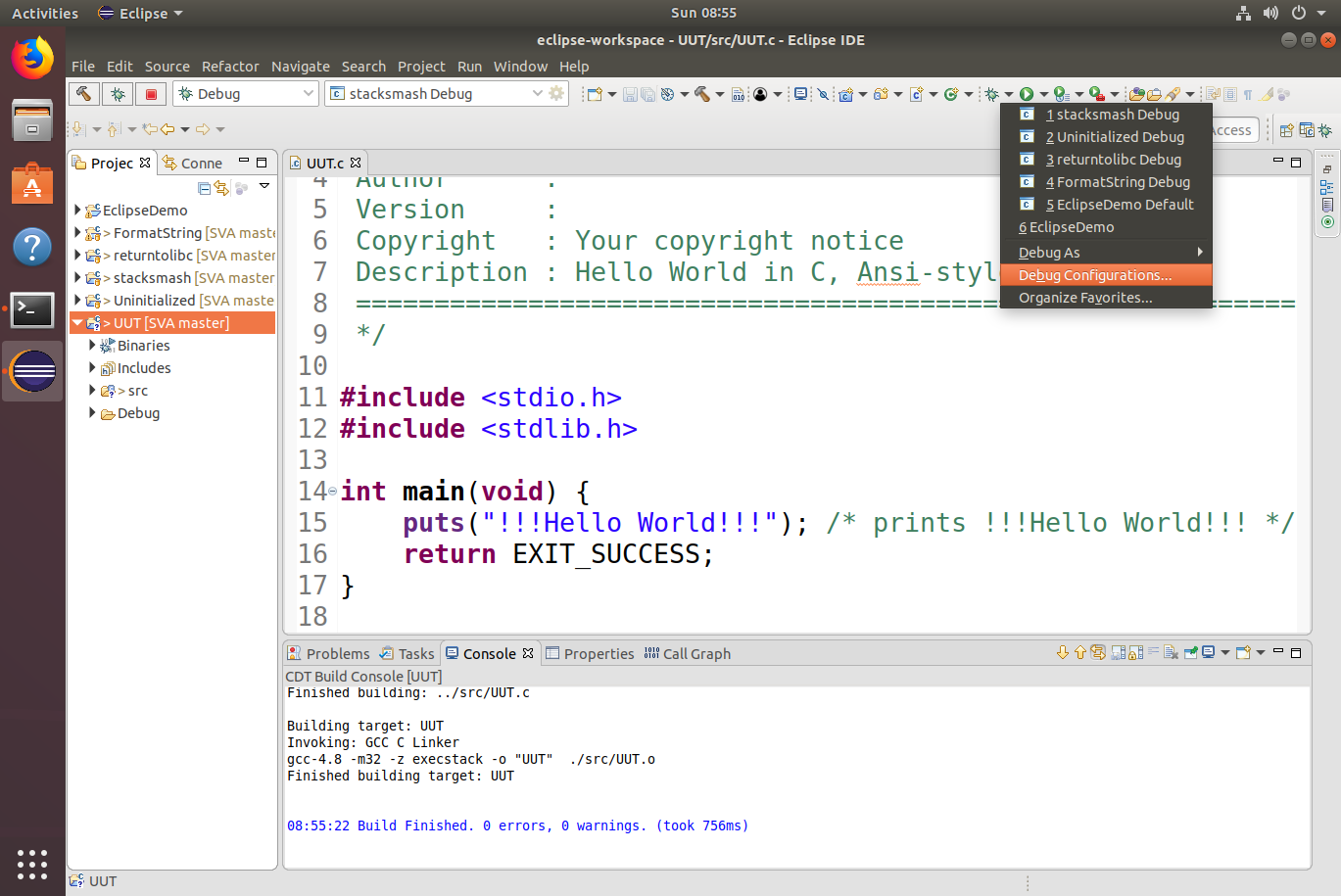
* Download the project into a local sandbox

$ git clone https://gitlab.com/underpantsgnomes/softwaresecurity/64bitlinuxshellcode

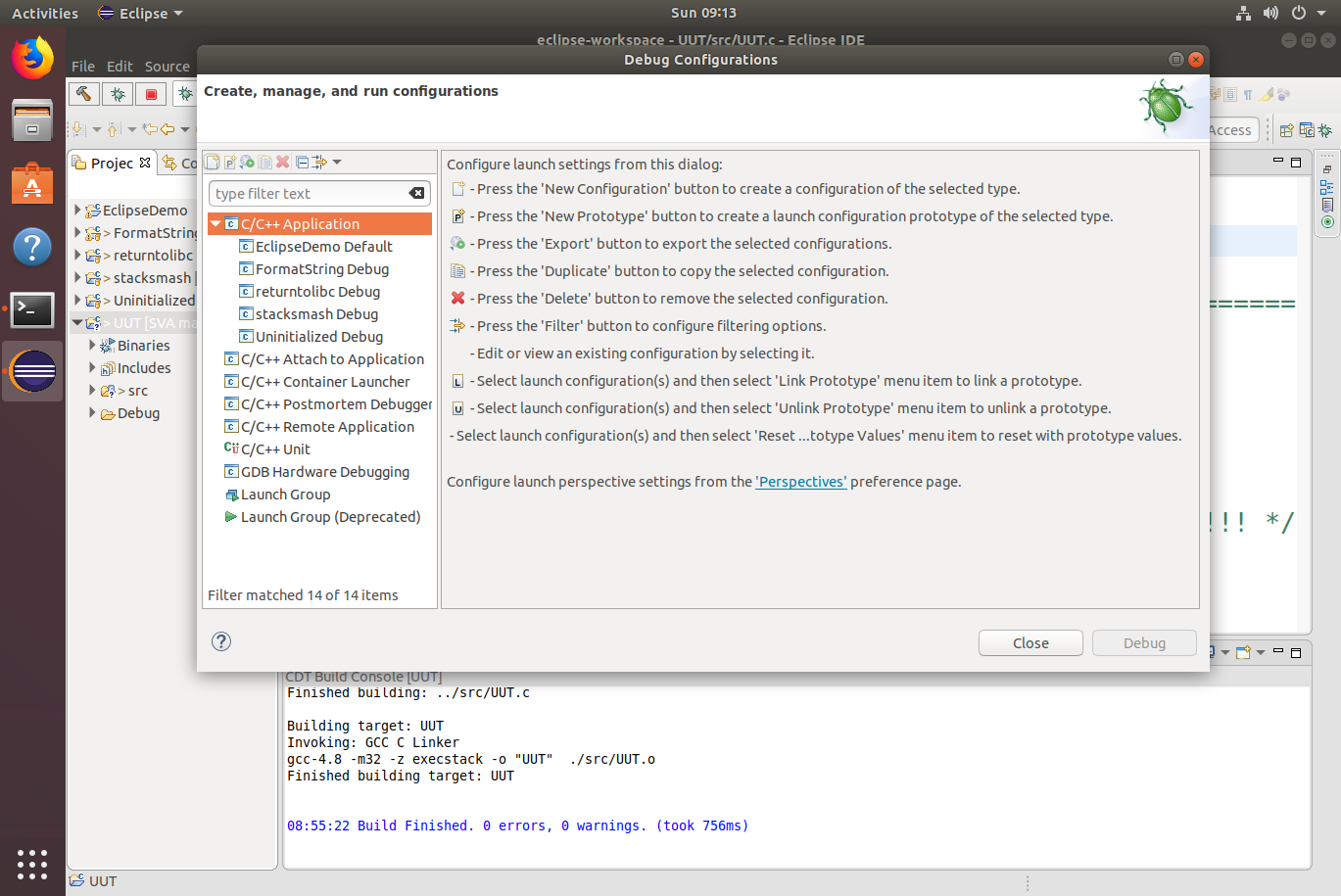
* Create a ~/.gdbinit file with the following contents

|  |
| --- |
| set disassembly-flavor intel  set disable-randomization on |

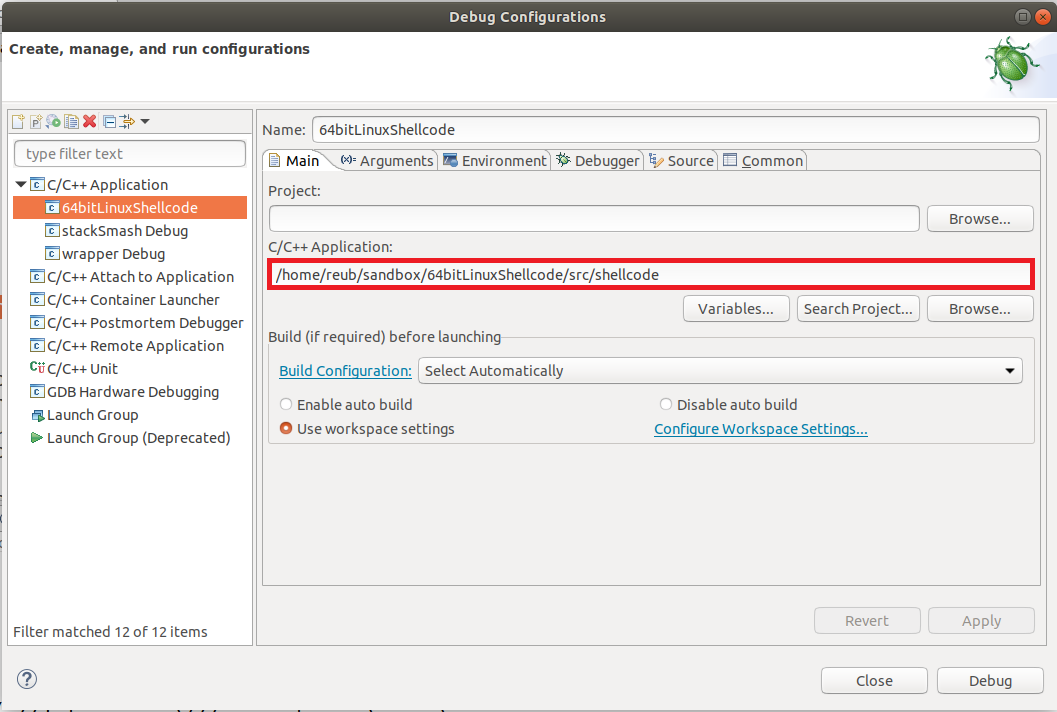
* Import the project in Eclipse
  + File->Open Projects From File System
* From a terminal, build the shellcode from within the src directory
  + $ nasm -f elf64 shellcode.nasm -o shellcode.o
* From a terminal, relocate the compiled binary to a zero-based address space
  + $ ld shellcode.o -o shellcode
* Create the debug target configuration



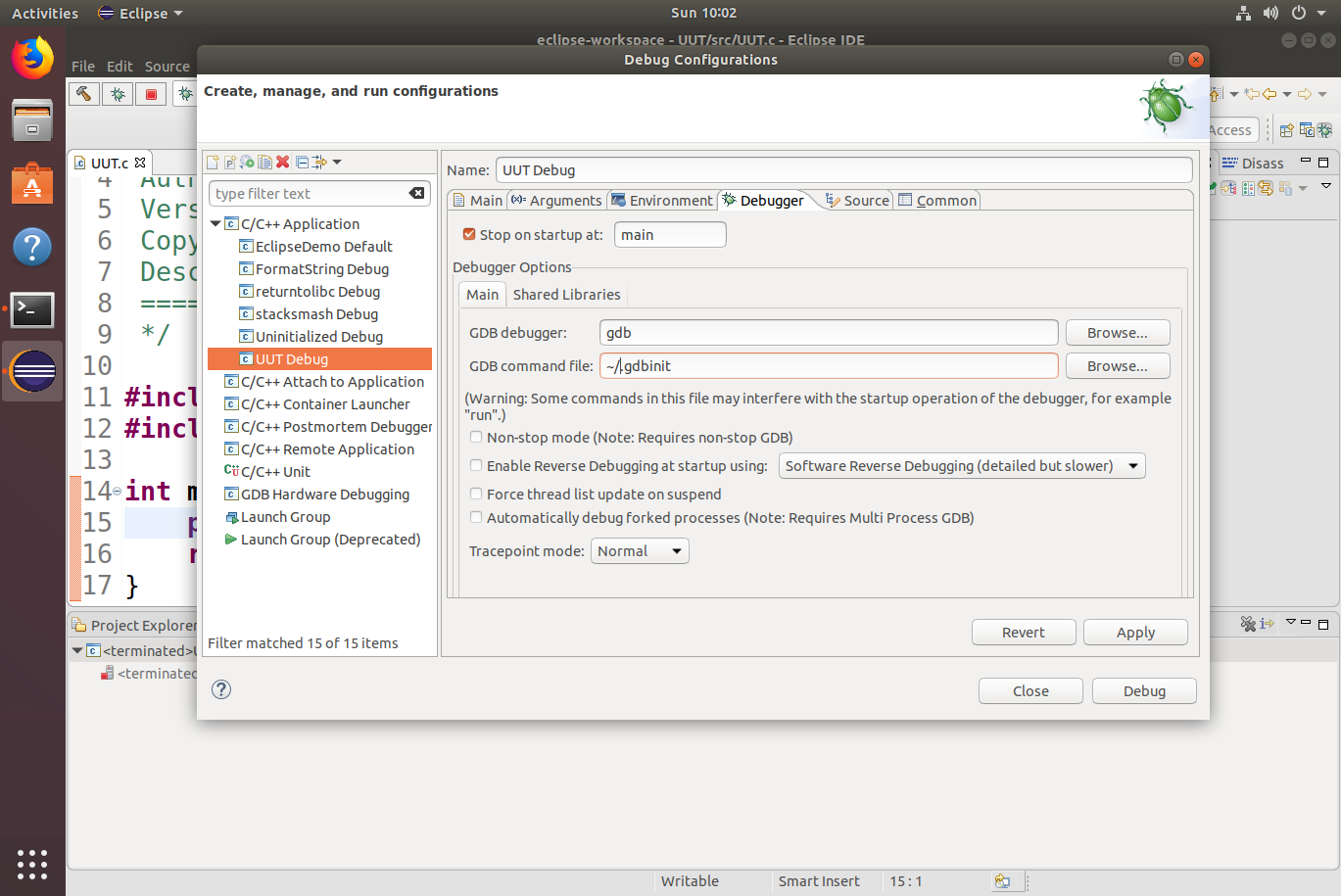
* Select new launch configuration for a C/C++ Application



* Specify the shellcode binary built from the console as the target



* + Go to the Debugger tab and specify an environment that uses ~/.gdbinit



* shellcode.nasm code does the following
  + Traps to the debugger using int3
  + Invokes the system call: execve("/bin/sh", ["/bin/sh"], NULL)
  + Invokes the system call: exit(0)

|  |
| --- |
| global \_start  section .text  \_start:  int3 ; Comment this out when not debugging  ; Put null terminated "/bin/sh" on the stack  mov rbx, 0x68732f6e69622fff ; Load bytes from "/bin/sh" in reverse (i.e., hs/nib/)  shr rbx, 0x8 ; Null terminate it without having a 0x0 byte in the  ; buffer  push rbx ; Push bytes for filename to stack  mov rdi, rsp ; Make rdi point to filename (now located on stack)  xor rax, rax ; Load 0x0 into rax  push rax ; Realigning stack to 16 bytes    push rdi ; Push address of "/bin/sh" (now located on stack), onto  ; the stack  mov rsi, rsp ; Make rsi point to address of "/bin/sh"    xor rdx, rdx ; Load 0x0 into rdx (envp[] is null)  xor rax, rax ; Load 0x0 into rax  mov al, 0x3b ; Load lower byte of rax with code for execve (other  ; bytes already zero from above)    syscall ; Invoke the system call handler    mov rdi, rdx ; Load rdi with exit code 0 (success)  mov al, 0x3c ; Load lower byte of rax with code for exit (other bytes  ; already zero from above)  syscall ; Invoke the system call handler |

* Run the program in Eclipse and step through the assembly instructions of the shellcode when it is called
  + Execute the following in the bash shell when your shell code call works (i.e., when you get the $ bash prompt in your debugging session)

|  |
| --- |
| $ /usr/games/cowsay -f dragon "Grrr!"  < Grrr! >  --------  \ / \ //\  \ |\\_\_\_/| / \// \\  /0 0 \\_\_ / // | \ \  / / \/\_/ // | \ \  @\_^\_@'/ \/\_ // | \ \  //\_^\_/ \/\_ // | \ \  ( //) | \/// | \ \  ( / /) \_|\_ / ) // | \ \_\  ( // /) '/,\_ \_ \_/ ( ; -. | \_ \_\.-~ .-~~~^-.  (( / / )) ,-{ \_ `-.|.-~-. .~ `.  (( // / )) '/\ / ~-. \_ .-~ .-~^-. \  (( /// )) `. { } / \ \  (( / )) .----~-.\ \-' .~ \ `. \^-.  ///.----..> \ \_ -~ `. ^-` ^-\_  ///-.\_ \_ \_ \_ \_ \_ \_}^ - - - - ~ ~-- ,.-~  /.-~ |