Phone: +91 8882618533
Email: info@geekinstitute.org
Website: www.geekinstitute.org

THE GEEK INSTITUTE OF CYBER SECURITY

Regd. By: E-Max India
Centre Code: EMAX/EK80606
(Recognized By Govt. Of India)

(Building Futures Through Digital Knowledge and Innovation)

x86_64 Assembly Language Programming with Debugging and Reverse Engineering Tools (01 Months)

Syllabus

01: Introduction to x86_64 Architecture and Assembly

- ✓ Evolution from x86 to x86_64
- ✓ Instruction sets and CPU architecture overview
- ✓ Registers: general-purpose, special-purpose, SIMD
- ✓ Binary, hex, little endian, memory addressing
- ✓ Setting up the environment: Linux, NASM, GCC, EDB, Ghidra, GDB, Radare2

02: Assembly Language Basics and NASM Syntax

- ✓ Writing your first NASM program
- ✓ Sections: .data, .bss, .text
- ✓ Data definitions: db, dw, dd, dq
- ✓ Labels, comments, and code structure
- ✓ Compiling and linking with NASM and LD/GCC
- ✓ Manual disassembly with ndisasm and objdump

03: Registers, Memory, and Arithmetic

- ✓ General-purpose registers: rax, rbx, rcx, etc.
- ✓ Stack and base pointers: rsp, rbp
- ✓ Addressing modes: immediate, register, direct, indirect
- ✓ Arithmetic operations: add, sub, mul, div
- ✓ Logical and bitwise operations: and, or, xor, shl, shr

04: Control Flow and Conditional Logic

- ✓ Conditional and unconditional jumps: jmp, je, jne, etc.
- ✓ Flags and cmp, test instructions
- ✓ Implementing if, else, and switch logic in assembly
- ✓ Looping constructs: loop, while, for logic
- ✓ Writing clean control flows using labels and blocks

05: The Stack, Procedures, and Calling Conventions

- ✓ Stack operations: push, pop, call, ret
- ✓ System V AMD64 calling convention
- ✓ Function arguments in registers: rdi, rsi, rdx, etc.
- ✓ Stack frames and local variables
- ✓ Recursive functions in assembly
- ✓ Leaf vs non-leaf functions

06: Linux System Calls in Assembly

- ✓ Introduction to the syscall interface (syscall instruction)
- ✓ Using rax to define syscall numbers
- ✓ Parameters in rdi, rsi, rdx, etc.
- ✓ File operations: open, read, write, close
- ✓ Memory: mmap, brk, munmap
- ✓ Exit, fork, exec, and basic process management

07: Debugging with GDB (GNU Debugger)

- ✓ Launching programs in GDB
- ✓ Setting breakpoints, watchpoints, stepping through code
- ✓ Inspecting memory, registers, and stack
- ✓ Using GDB for analyzing syscall behavior
- ✓ Scripting GDB with init files and Python extensions
- ✓ Debugging stripped binaries

08: Disassembly and Static Analysis with Ghidra

- ✓ Installing and configuring Ghidra
- ✓ Importing binaries and analyzing control flow
- ✓ Understanding decompiler output and matching assembly to C
- ✓ Cross-referencing functions, strings, and global data
- ✓ Function signatures, symbol renaming, and scripting with Java/Python
- ✓ Using Ghidra for patching and function reimplementation

09: Binary Analysis with Radare2 and EDB

- ✓ Introduction to Radare2 architecture (r2, r2ghidra, cutter)
- ✓ Disassembling, analyzing functions, and renaming in Radare2
- ✓ Graph view, strings, and syscall tracing
- ✓ Editing assembly in Radare2
- ✓ Using EDB (Evan's Debugger) for GUI-based dynamic analysis
- ✓ Memory, register, and stack inspection via GUI

10: Memory Layout and Buffer Exploits

- ✓ ELF file format and memory segments
- ✓ Stack layout: arguments, return address, base pointer
- ✓ Detecting and understanding stack overflows
- ✓ Buffer overflows and off-by-one vulnerabilities
- ✓ Return-to-libc overview
- ✓ Writing safe functions in assembly

11: Shellcode Development and Injection

- ✓ Understanding position-independent code (PIC)
- ✓ Creating custom shellcode from scratch
- ✓ Avoiding null bytes and bad characters
- ✓ Encoding and decoding shellcode
- ✓ Shellcode execution in C wrappers
- ✓ Writing egghunter and polymorphic shellcode

12: Reverse Engineering Real-world Binaries

- ✓ Reconstructing logic from stripped executables
- ✓ Reversing compiled C programs
- ✓ Identifying obfuscation techniques
- ✓ Binary patching and modification with Ghidra or Radare2
- ✓ Rebuilding C code from disassembled logic
- ✓ Case study: reverse engineering a login cracker

Capstone Projects and Hands-On Practice

- ✓ **Project 1:** Implement a calculator using only syscalls
- ✓ **Project 2:** Reverse engineer and modify a binary to bypass authentication
- ✓ **Project 3:** Create your own shellcode and inject into a vulnerable program
- ✓ **Project 4:** Write a file copy routine using x86_64 Assembly
- ✓ **Project 5:** Use Ghidra and Radare2 to fully reverse engineer a compiled C program