


Workshop: Payload Obfuscation for Red Teams



Duncan Ogilvie

Setup: workshop.ogilvie.pl

1. Login to your GitHub account
2. [Fork the repository](#)
3. Click the green `<> Code` button
4. Press `...` and then `New with options...`
5. Change `Machine type` to `4-core`
6. Then `Create codespace`
7. Wait a ~5 minutes while the image is loading 
 - Press `Show log` to see progress

Introduction

- Technical Requirements:
 - C programming (**absolutely required**)
 - Basic Reverse Engineering
- Hands-on workshop format
- Interactive: Ask questions anytime!

Who am I?

- Creator of `x64dbg`
- Worked in DRM for 5 years
- Currently working in mobile security R&D
 - Never worked on a red/blue team

Outline

- Basics of VM-based obfuscation
- Learn about RISC-V
- Compile your own obfuscated payloads
- VM hardening techniques
- Future work discussion

Quick demo!

Training

- Available for on-site training/consulting
- Reach out!
 - training@ogilvie.pl
 - Socials: [Discord](#) / [LinkedIn](#) / [X](#)

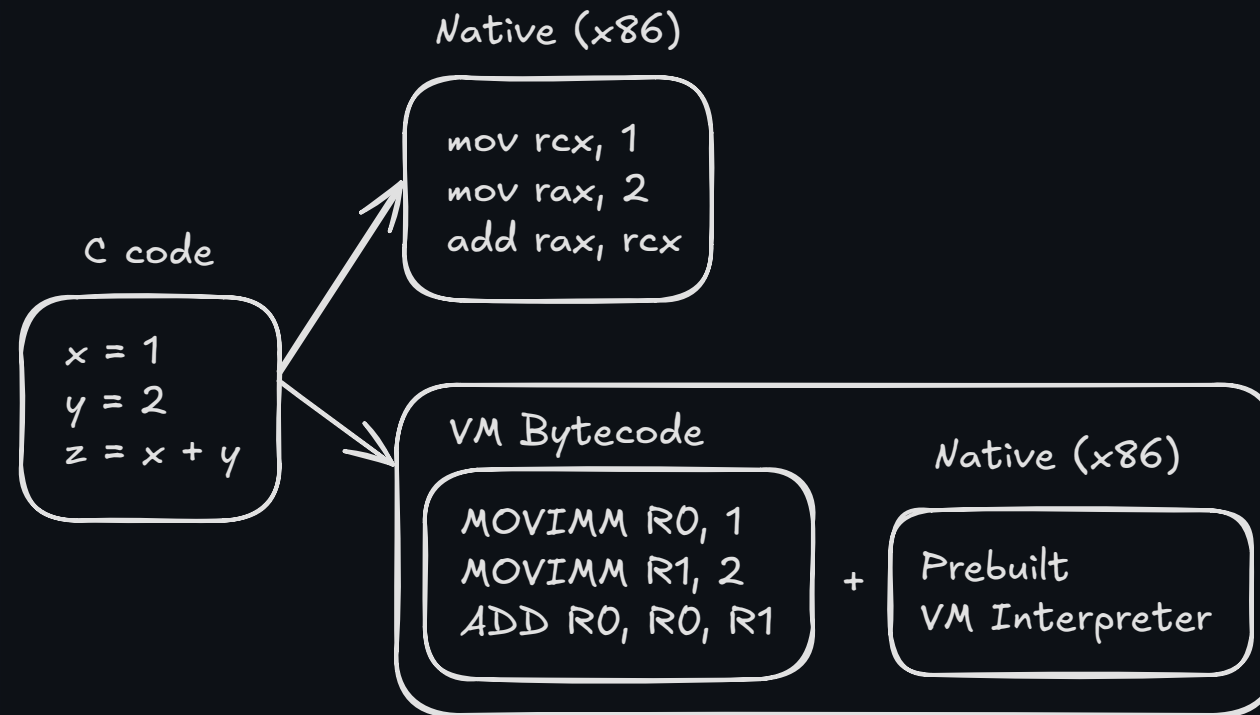
Motivation

- Democratize obfuscation techniques
- Raise reverse engineering cost
- Hinder generic detection: makes Yara rules harder
- Learn & have fun!

Non-goals

- Not a "turn-key" advanced obfuscator
- No deep dives into compiler/code analysis theory
- Focus: Introduce foundational concepts, inspire further development.

VM Obfuscation: introduction



VM Obfuscation: interpreter

```
def vm_interpreter(bytecode):  
    vm = Context(pc=0)  
    while True:  
        op = fetch_opcode(bytecode, vm.pc) # read opcode  
        switch op:  
            case add: handle_add(vm)  
            case sub: handle_sub(vm)  
            case ...: handle_xxx(vm)  
  
        vm.pc = next_pc(vm)  
        # loop for the next instruction
```

Exercise 1: `exercise_1/vm_basics.md`

VM Obfuscation: discussion

- Tradeoff
 - Easy to write a custom VM
 - Difficult to translate code into it
- Solution
 - Use LLVM to compile to a well-supported architecture
 - Implement a VM based on that architecture -> RISC-V

RISC-V: introduction

- Open-source architecture, well-documented
- Core set: `rv64im` (base integer instructions)
 - ~69 instructions, relatively simple to implement
- Extensions (F,D,V) are optional; Clang targets `rv64im`
- True RISC: Simple instructions, no complex flags/side-effects

RISC-V: registers

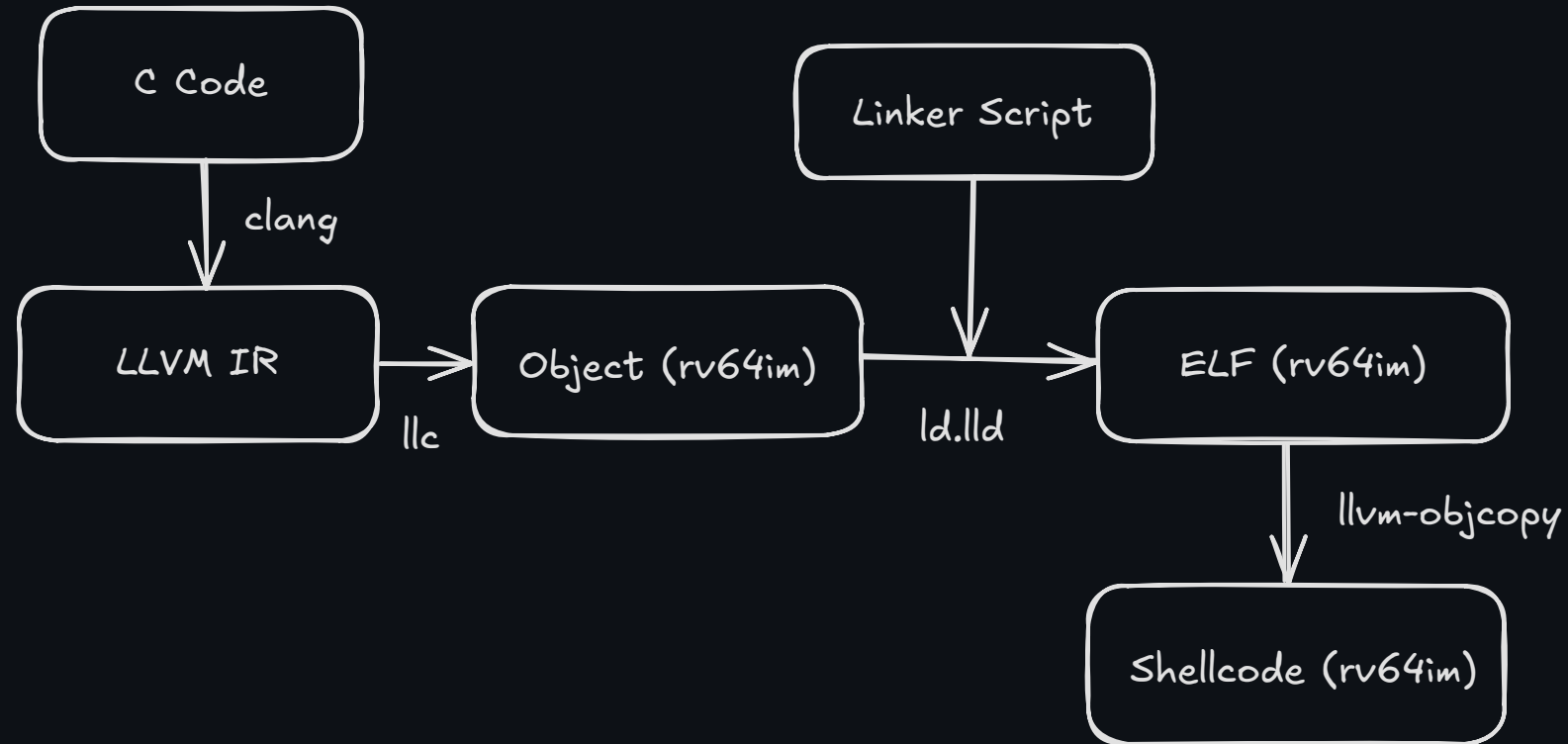
Name	ABI Mnemonic	Meaning	Preserved across calls?
x0	zero	Zero	-- (Immutable)
x1	ra	Return address	No
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	-- (Unallocatable)
x4	tp	Thread pointer	-- (Unallocatable)
x5 - x7	t0 - t2	Temporary registers	No
x8 - x9	s0 - s1	Callee-saved registers	Yes
x10 - x17	a0 - a7	Argument registers	No
x18 - x27	s2 - s11	Callee-saved registers	Yes
x28 - x31	t3 - t6	Temporary registers	No

RISC-V: common instructions

- `addi` : add immediate (32-bit)
- `mv` : mov register
- `sw` : store 32-bit
- `li` : load immediate
- `lw` : load 32-bit
- `add` : add registers
- `blt` : branch if less than
- `addiw` : add immediate (64-bit)
- `ret` : return from subroutine (`jalr x0, x1, 0`)

[RISC-V Instruction Set Specifications](#), [Official reference](#), [Pseudo Instructions](#)

RISC-V: compiling shellcode



Demo!

RISC-V: disassembling

- Load the ELF file in Ghidra (it has symbols)
- Go one instruction at a time and add comments

RISC-V: tracing

- The `riscvm` project has a `--trace` flag
- Prints every executed instruction
- Prints register values
- Used for debugging payloads

RISC-V: exercises

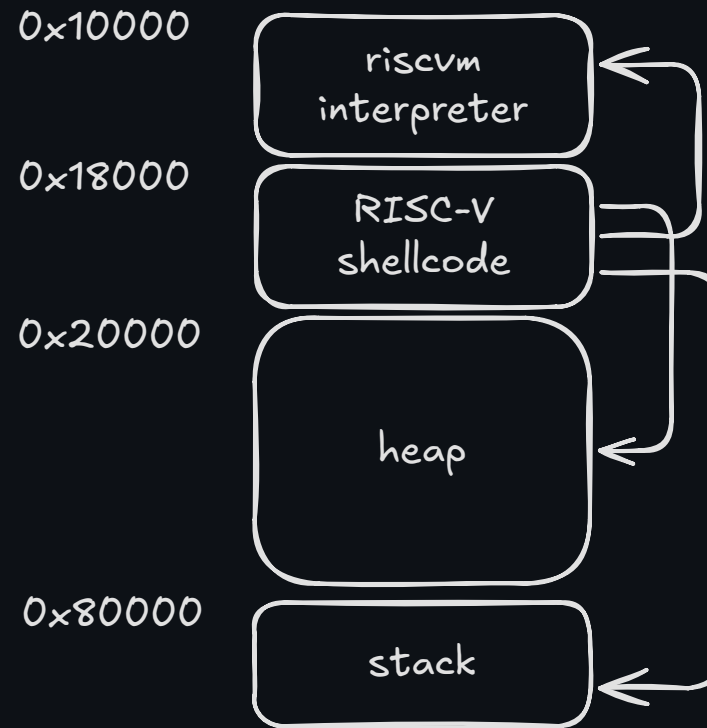
- Compile & run simple RISC-V programs
- Disassemble & trace the executed code

Exercise 2: `exercise_2/shellcode.md`

Host Interaction: memory model

- Host (x86) and guest (RISC-V) share memory space
- Avoids memory translation logic
- Limitation: Guest cannot directly execute host code

Host Interaction: memory model



Host Interaction: primitives

- Implemented via `scall` instruction
- `resolve_import` : Resolves host function addresses.
 - e.g., `MessageBoxA` in `user32.dll`
- `host_call` : Executes a resolved host function
 - Sequence: VM Exit -> Stub -> Host Call -> Return to Stub -> VM Enter

Host Interaction: exercises

- Call `puts` from RISC-V code.
- Read a file, get its length, and display it.

Exercise 3: `exercise_3/host_interaction.md`

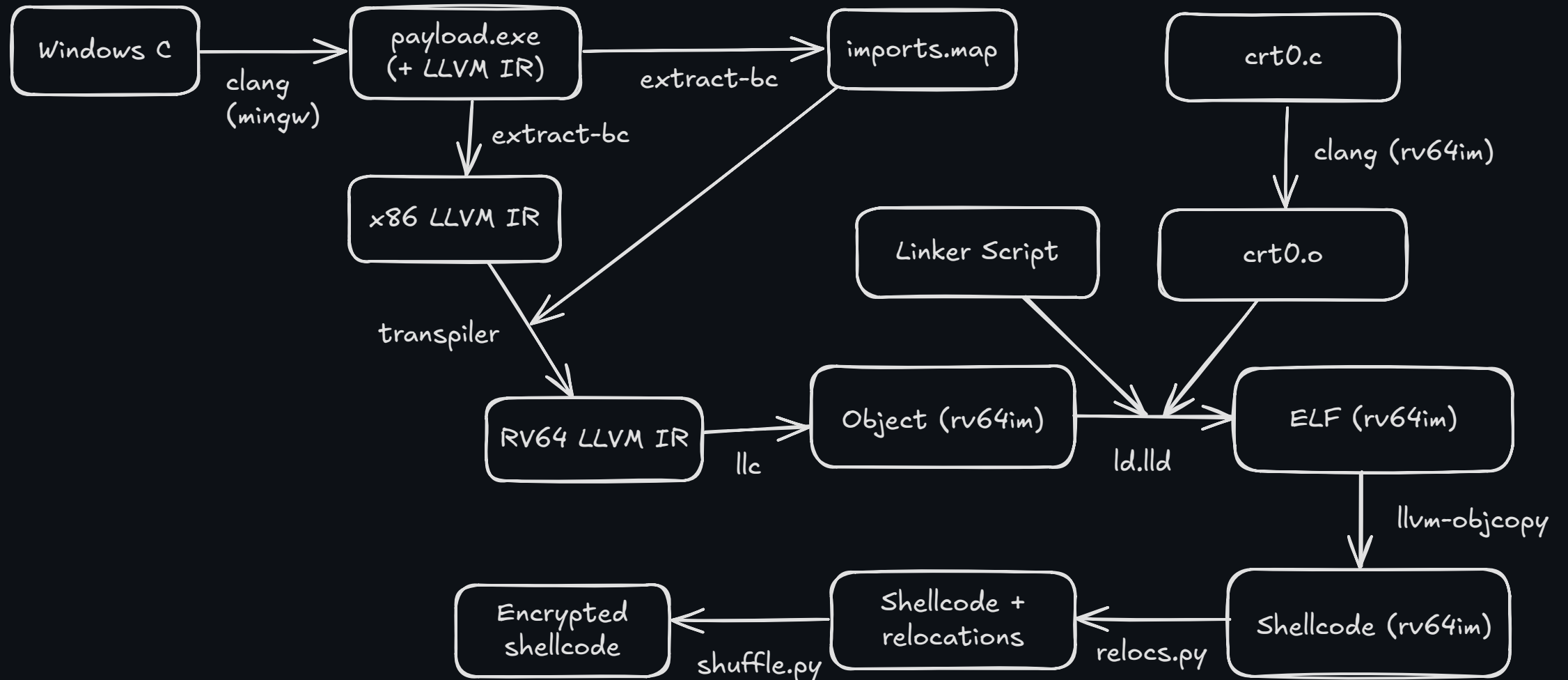
Transpiler: introduction

- Tedious to write code for `riscvm` by hand
- Solution
 - Compile regular code to LLVM IR
 - Translate import calls to `host_call`
 - Use `llc` to emit `rv64im` shellcode
- Implementation: LLVM C++ API (homework)

Transpiler: crt0

- Role of `crt0` :
 - Apply relocations
 - Resolve imports
 - Initialize C runtime (`initterm`)
 - Call main
 - Exit
- Linker script: controls memory layout

Transpiler: diagram



Transpiler: limitations

- Incomplete C++ support (no exceptions)
- No callbacks from host -> guest
- No translation of existing x64 shellcode

Transpiler: exercises

- Build the `riscvm` project for Windows
- Build the `payload` project
- Run the example payloads in `wine` (or on your host)

Exercise 4: `exercise_4/transpiler.md`

VM Hardening: overview

- Goal: Increase RE effort with lightweight tricks
- Layer these on top of existing VM obfuscation

VM Hardening: simple encryption

- Method 1: XOR instructions at decode-time
 - Key can be static or derived (e.g., from PC)
- Method 2: Decrypt entire bytecode in memory before execution
 - e.g., from resource section to dynamically allocated page
- Generally easier to reverse than opcode shuffling

VM Hardening: opcode shuffling

- Remap opcodes/sub-opcodes (e.g., `add` from index 0 to 5)
- Breaks standard disassemblers (Ghidra)
- Can be randomized per sample
- Favorable attacker vs. defender effort trade-off

VM Hardening: direct threading

- Alternative to traditional `switch`-based dispatcher
- Can improve performance & obfuscate control flow
- Implemented via Clang attributes/macros

VM Hardening: exercises

- Build `riscvm` with hardening features enabled
- Play around!

Exercise 5: `exercise_5/hardening.md`

Future Work

- Obfuscate VM instruction handlers
 - Goal: Resist pattern matching by RE tools
 - Methods: O-LLVM, x86 rewriters (different tradeoffs)
- Environment keying

Closing Remarks

- Remember: **Shut down the Codespace!**
- Continue at home!
- Available for training/consulting!
 - Privately: `training@ogilvie.pl`
 - [GitHub Discussions](#)