

# 12 Days of Ghidra

Nathan R

[Twitter](#) | [Mastodon](#)

# Day 4 – Debuggers

# Static and Dynamic Analysis

## Static Analysis

- Look at code without running it

## Dynamic Analysis

- Run the binary and observe its behaviour

So far, we have focus on static analysis, Ghidra can also help us perform dynamic analysis

# Debuggers

Allow us to step through program execution line by line

- View variable values
- Register Values
- Contents in memory

Why might we need to do this?

- Information that is only generated at runtime

# Ghidra Debugger Tool

Connects to a system debugger

- GDB -> Linux
- LLDB -> MacOS
- WinDBG -> Windows

Set breakpoints

- Where you want to stop execution

View variable and memory contents in the GUI

Send input to the program

# Settingup Ghidra Debugger

Open the **program manager** and click on **blue bug**

Open a terminal window – [here](#) for why

- Run “tty” and take note of value e.g. /dev/pts/0
- “sleep 1000000”

Inside Ghidra create a new GDB Target

- “IN-VM GNU gdb local debugger”

In Ghidra GDB Interpreter:

- “set inferior-tty <tty-val>” – e.g. set inferior-tty /dev/pts/0

# Day 4 Binaries

Two binaries

1. Generic binary that should run on all OS's
2. A bonus binary that will only work on Linux and MacOS
  - a. Try using WSL if you are running windows

Suggestions:

1. Set breakpoints at places where dynamic values are being used, e.g. comparison operations