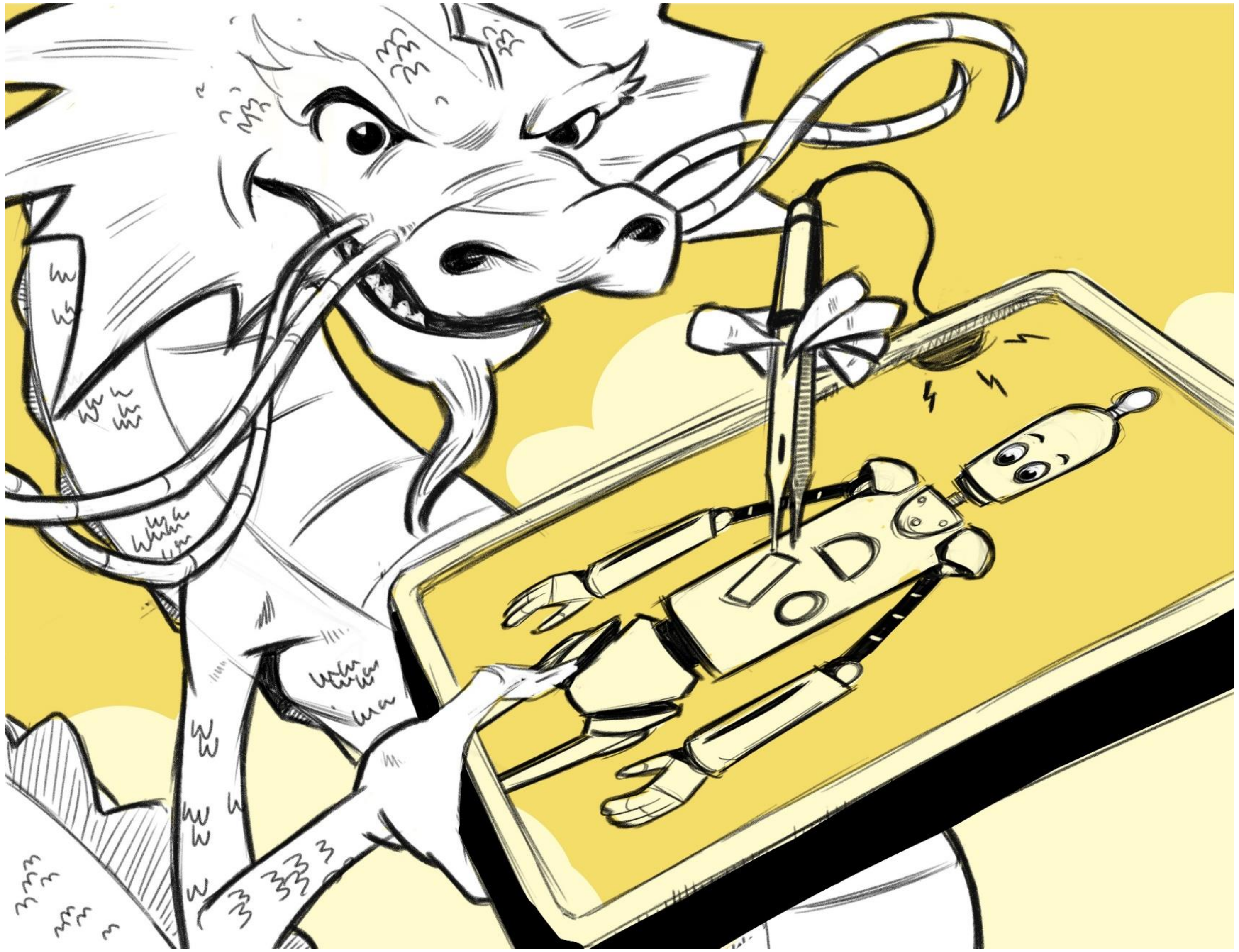
How Ghidra changed my life

Chris Eagle

Kernelcon - 2020

Who am I

- Chris Eagle
    - Reverse Engineer
    - CTF player
    - Long time IDA Pro user
    - Teacher
    - Author
    - Speaker



Why are we here?

- Ghidra released by NSA in early 2019
- What does it mean for IDA users?
  - Mostly it means, they're still using the right tool
  - Maybe it will drive innovation in IDA
    - We have undo now!

What’s Ghidra got that IDA doesn’t?

- Primarily three features
  - Low price tag
    - Out of our control
  - Collaboration server
    - Many efforts to bring collaboration to IDA, but really need Hex-Rays to do this properly
  - Decompilers for all architectures
    - This we can fix

The Ghidra Decompiler

- Ghidra is written in Java
  - Mostly
- The decompiler is written in C++
  - Ghidra launches the decompiler as a child process and communicates with it over pipes
- C++ source for the decompiler is available in the Ghidra source repo
- Why not borrow the decompiler for our own use?

Using Ghidra's Decompiler

- At least three efforts

Talos - https://blog.talosintelligence.com/2019/09/ghida.html

Radare2 - https://github.com/radareorg/r2ghidra-dec

blc - https://github.com/cseagle/blc

The subject of this talk

Talos

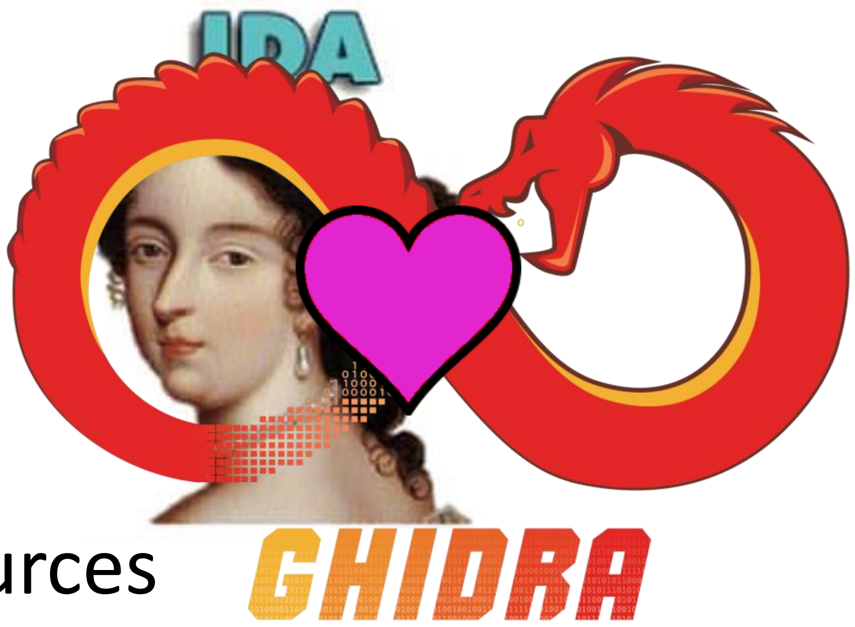
- GhIDA: Ghidra decompiler for IDA Pro
- Requires local Ghidra installation

  OR
- Ghidraaas (Ghidra as a Service)
  - “docker container that exposes the Ghidra decompiler through REST APIs.”
- Uses Ghidra’s idaxml.py to export database to xml
- Shells to headless Ghidra to import xml and decompile
- Renders result in IDA

r2ghidra-dec

- Integrates decompiler’s C++ components directly into radare2
- No need to run Ghidra
- Commands expose different types of decompiler output
  - XML
  - JSON
  - C

BLC



- Officially Binary Lifting Component
- Unofficially Bastard Love Child
- Similar concept to r2ghidra-dec
- Integrate required Ghidra decompiler sources into IDA C++ plugin
  - No change to Ghidra sources
  - Subclass key decompiler classes to bridge to IDA
- Ghidra decompiler can generate xml, json, or C
  - Plugin consumes xml because it's easier to recover the block structure of the code
  - Didn't want to write or integrate a C parser

Installation

- If you intend to build from source
  - Need IDA SDK
  - Clone https://github.com/cseagle/blc.git into `<idasdk>/plugins`
  - Visual Studio – build with blc.sln
  - Linux/Mac – use provided Makefile (may need to adjust paths to your IDA install location)
- If you're courageous you can use the binaries in `blc/bins/<platform>/<idaversion>`
- Copy plugins blc.(dll/so/dylib) and blc64.(dll/so/dylib) to `<idadir>/plugins`

Ghidra Dependency

- No need to run Ghidra, but decompiler needs Ghidra's SLEIGH files
  - These define Ghidra processor modules and how to generate P-code
- Copy entire Ghidra/Processors hierarchy from a Ghidra distro into `<idadir>/plugins` so that you have `<idadir>/plugins/Ghidra/Processors/…`

Usage

- Alt-F3 decompiles current function
- Display and UI strives to mimic Hex-Rays decompiler (but far fewer features)
- Double-click a function name to decompile that function
- ESC to navigate back
- N to rename local variables, parameters, and functions
    - Every attempt made to map Ghidra names to IDA names

Demos

- I laugh at the demo gods! No videos here

TODO

- Integrate IDA type system with Ghidra type system
- Test with less common architectures
  - Decompiler requires valid Ghidra language ID string.
  - Best effort is made to derive one from available IDA information
  - Very few architectures tested so far
    - Lack of test binaries for most architectures
- Add more features similar to Hex-Rays decompiler
  - Currently no context menu actions
- Name demangling
- Structure handling

Conclusion

- Questions
- Feedback and merge requests welcome