Bière sécu Bordeaux

1st event
Combining static and dynamic binary analysis

*ret-sync*
Context

- 2 approaches in reverse-engineering (RE):
  - static (disass/decompile) → IDA, Ghidra, etc.
  - dynamic (debug) → x64dbg, WinDbg, LLDB, etc.

- Possible to combine both worlds in the same tool...

- ... but often painful to use (eg. IDA dbg)

- Annoying to switch between multiple tools
Context

- **Classical example:**
  - I’m debugging using WinDbg, I spot a routine or structure which seems interesting
  - I’d like to know if I’ve already documented it within IDA
  - … I need to compute the offset from the load address of my module (ASLR/reloc)
  - … add it to the preferred load address of my module in my idb

- Conclusion: straightforward but painful if I have to do that every 2 minutes
- … even more painful provided that I use x64dbg for usermode and WinDbg for kernelmode
Solutions

- Code a new tool which would combine both worlds...
Solutions

- Code a new tool which would combine both worlds...
Solutions

- Code a new tool which would combine both worlds...
- Set-up a glue which would create an interface between the disass and the debugger(s)...

- ... *ret-sync* by Alexandre Gazet
  
  https://github.com/bootleg/ret-sync
ret-sync: support

- **Static:**
  - IDA
  - Ghidra

- **Dynamic:**
  - WinDbg(-preview)
  - GDB
  - LLDB
  - OllyDbg 1.10
  - OllyDbg v2
  - x64dbg
ret-sync: features

- Permits to “follow” the program workflow in IDA/Ghidra view
  - “step” in the dbg ➔ “step” in the disass static view
- Dynamic switching between multiple idbs
  - trace within `toto.exe` ➔ trace within `toto.idb`
  - `toto.exe` issues a call in `fistouille.dll` ➔ switch to `fistouille.idb`
- Automagical rebase 💖
- Sending commands to the dbg (bp, hbp, lbl, etc.)
- Custom commands¹
- All features are available both in disass AND decompiled views
- etc.

¹ the complete list is documented on the project’s github
ret-sync: design

- Clients/servers scheme
- IDA plugin (focus in this presentation):
  - broker (client)
  - dispatcher (server)
- (Ghidra plugin: uses the built-in ProgramManager)
- debuggers’ plugin: client
ret-sync: design
ret-sync: IDA side

- Creates a window dedicated to the plugin configuration through Qt
- Registers some events:
  - `QtWidgets.QCheckBox`
  - `QtWidgets.QpushButton`
  - etc.
- **which register some callbacks:**
  - `self.cb_sync.stateChanged.connect(self.cb_change_state)`
  - `self.cb_hexrays.stateChanged.connect(self.cb_hexrays_sync_state)`
  - `self.btn.clicked.connect(self.cb_btn_restart)`
- Defines hotkeys
- Defines some command lines options
- Check if the permanent “.sync” configuration file exists
ret-sync: IDA side

- Creates a window dedicated to the plugin configuration through Qt
- Registers some events:
  - `QtWidgets.QCheckBox`
  - `QtWidgets.QpushButton`
  - etc.
- which register some callbacks:
  - `self.cb_sync.stateChanged.connect(self.cb_change_state)`
  - `self.cb_hexrays.stateChanged.connect(self.cb_hexrays_sync_state)`
  - `self.btn.clicked.connect(self.cb_btn_restart)`
- Defines hotkeys
- Defines some command lines options
- Check if the permanent “.sync” configuration file exists
ret-sync: IDA side

self.cb_sync.stateChanged.connect(self.cb_change_state)

- init_broker()
  - Instanciates a “Broker” class creates a worker (“RequestHandler” class)
  - Launches “broker.py” script
ret-sync: design
ret-sync: IDA side

Broker

- Historically coded in order to compensate the lack of `QtNetwork`
- Is a `QtCore.Qprocess`
- Registers some callbacks
  - `self.error.connect(self.cb_on_error)`
  - `self.readyReadStandardOutput.connect(self.cb_broker_on_output)`
  - `self.stateChanged.connect(self.cb_broker_on_state_change)`
- Handles asynchronous messages
ret-sync: IDA side

Broker

- Historically coded in order to compensate the lack of QtNetwork
- Is a QtCore.Qprocess
- Registers some callbacks
  - self.error.connect(self.cb_on_error)
  - self.readyReadStandardOutput.connect(self.cb_broker_on_output)
  - self.stateChanged.connect(self.cb_broker_on_state_change)
- Handles asynchronous messages
ret-sync: IDA side

Broker

def cb_broker_on_out(self):
    # readAllStandardOutput() returns QByteArray
    buffer = self.readAllStandardOutput().data().encode("ascii")
    batch = buffer.split('\n')
    for req in batch:
        self.worker.parse_exec(req.strip())

→ Retrieves everything written to stdout and gives it to the worker to be parsed
ret-sync: design
ret-sync: IDA side

RequestHandler

- "Worker" which addresses all data transmitted to it by the Broker
- Handles all actions related to IDA side:
  - Disass \(\rightarrow\) dbg (go, step, bp, lbl, comment, etc.)
  - Dbg \(\rightarrow\) disass (update view, enable/disable, colors, etc.)
- Custom messages exchanged formatted in JSON
  
  ```json
  {"type": "broker", "subtype": "msg", "msg": "connected to dispatcher"}
  {"type": "broker", "subtype": "notice", "port": "49678"}
  {"type": "dialect", "dialect": "windbg"}
  {"type": "broker", "subtype": "enable_idb"}
  {"type": "loc", "base": 9223363323289862144, "offset": 9223363323290320023}
  ```
ret-sync: design
Instanciates a “BrokerSrv” class:

- server.bind() → binds to localhost
- server.notify() → run_dispatcher()
- server.loop()

A single instance per idb
Instanciates a “BrokerSrv” class:

- server.bind() → binds to localhost
- server.notify() → run_dispatcher() → launches the “dispatcher.py” server, if not already existing, then connects to it while transmitting the corresponding idb name
- server.loop()

A single instance per idb
ret-sync: design
Instanciates “DispatcherSrv” class:
- `bind()` binds to `HOST:PORT` (from “.sync” file or default)
- `loop()`

`loop()` waits for incoming messages:
- Brokers (idbs)
- Debugger(s)

Dedicated methods depending on requests’ types:
- `req_new_client`
- `req_new_dbg`
- `req_dbg_quit`
- `req_sync_mod`
- etc.

Finds the idb matching the module currently debugged (switch_idb())

A single global instance
ret-sync: design
Ret-sync: debuggers views

- Specific to each debugger
- Connects to the dispatcher
- Sends messages (command “step”, command “!sync”, module name, etc.)
- Retrieves messages from the dispatcher (“step”, “bp”, etc.)
Demo time!
Do you have any questions?

THANK YOU FOR YOUR ATTENTION,