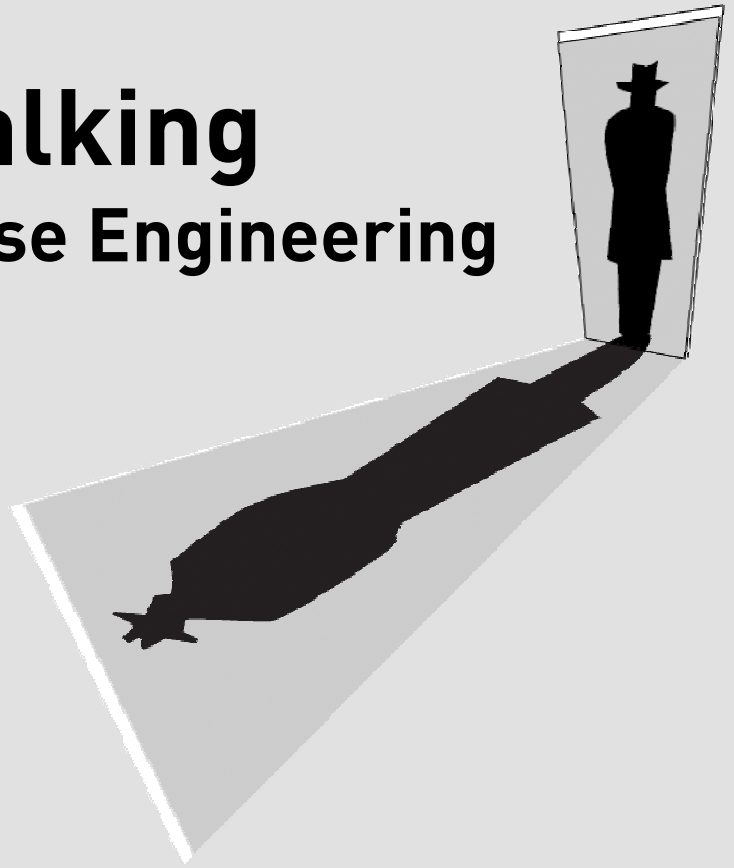
iDEFENSE

Process Stalking

Run-Time Visual Reverse Engineering



Pedram Amini – pamini@idefense.com

Introduction and Agenda

iDEFENSE

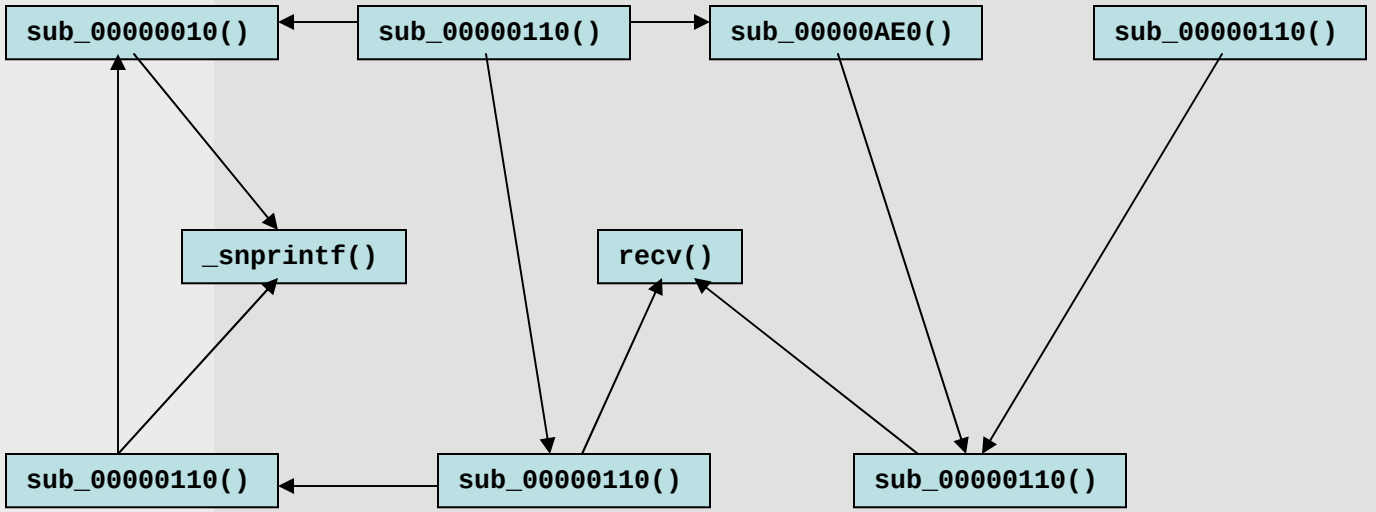
- Pedram Amini
  - Assistant Director, iDEFENSE Labs
  - Security researcher, developer and reverse engineer
  - iDEFENSE Vulnerability Contributor Program

http://labs.idefense.com

- Background information
- Overview and design
- Features and benefits
- Demonstrations
- In development
- Conclusion

Call Graphs

- Disassembled binaries can be visualized as graphs
  - Functions = nodes
  - Calls = edges
- IDA supports this type of visualization
- Useful for viewing the relationships between functions

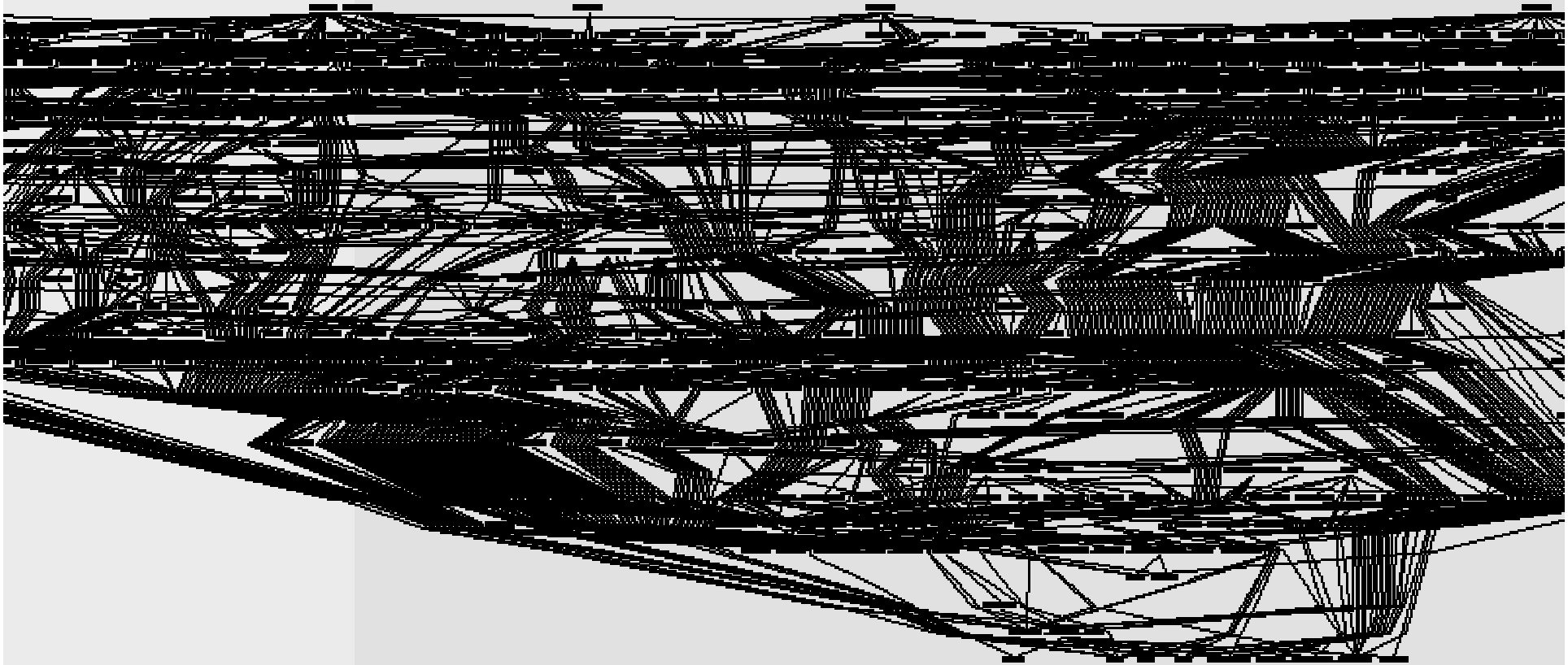


- However…

Call Graphs

iDEFENSE

- In most real-world scenarios, function call graphs can be unmanageable and down right frightening:

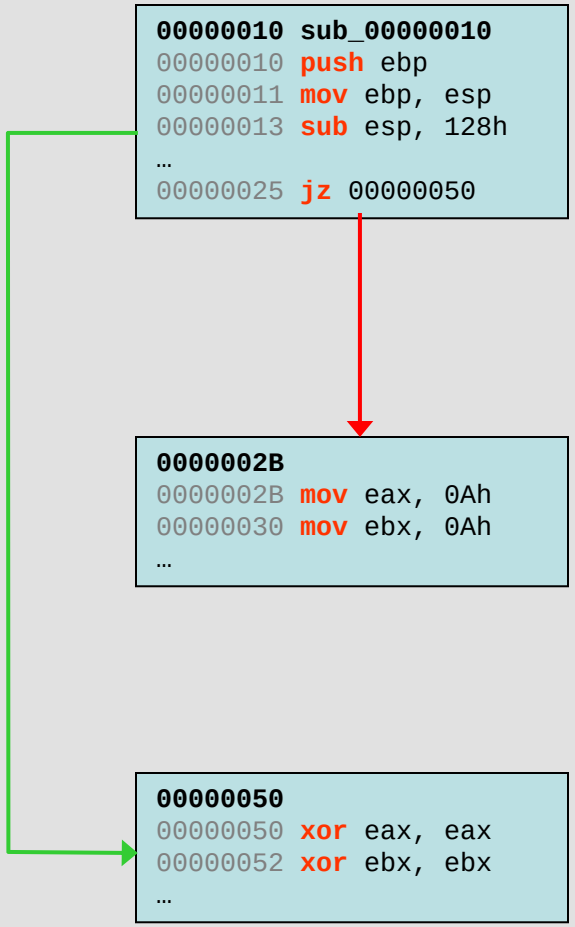


Control Flow Graphs (CFGs)

iDEFENSE

- Functions can also be visualized as graphs
  - Basic blocks = nodes
  - Branches = edges

```
00000010 sub_00000010
00000010 push ebp
00000011 mov ebp, esp
00000013 sub esp, 128h
…
00000025 jz 00000050
0000002B mov eax, 0Ah
00000030 mov ebx, 0Ah
…
00000050 xor eax, eax
00000052 xor ebx, ebx
…
```



- IDA also supports this type of visualization
- Useful for easy viewing of execution paths
- pGRAPH

RE Analysis Challenges

iDEFENSE

- Input tracing
  - What code handles our inputs?
- Code coverage
  - How we can we determine where our fuzzer has gone?
  - How can we get our fuzzer deeper into the process?
- Complexity
  - How can we digest/understand mass volumes of machine code?
- Filtering
  - How can we filter uninteresting trace data? (Example: GUI handling code)
- Trace speed
  - How can we increase the speed of our tracing?

Process Stalker Overview

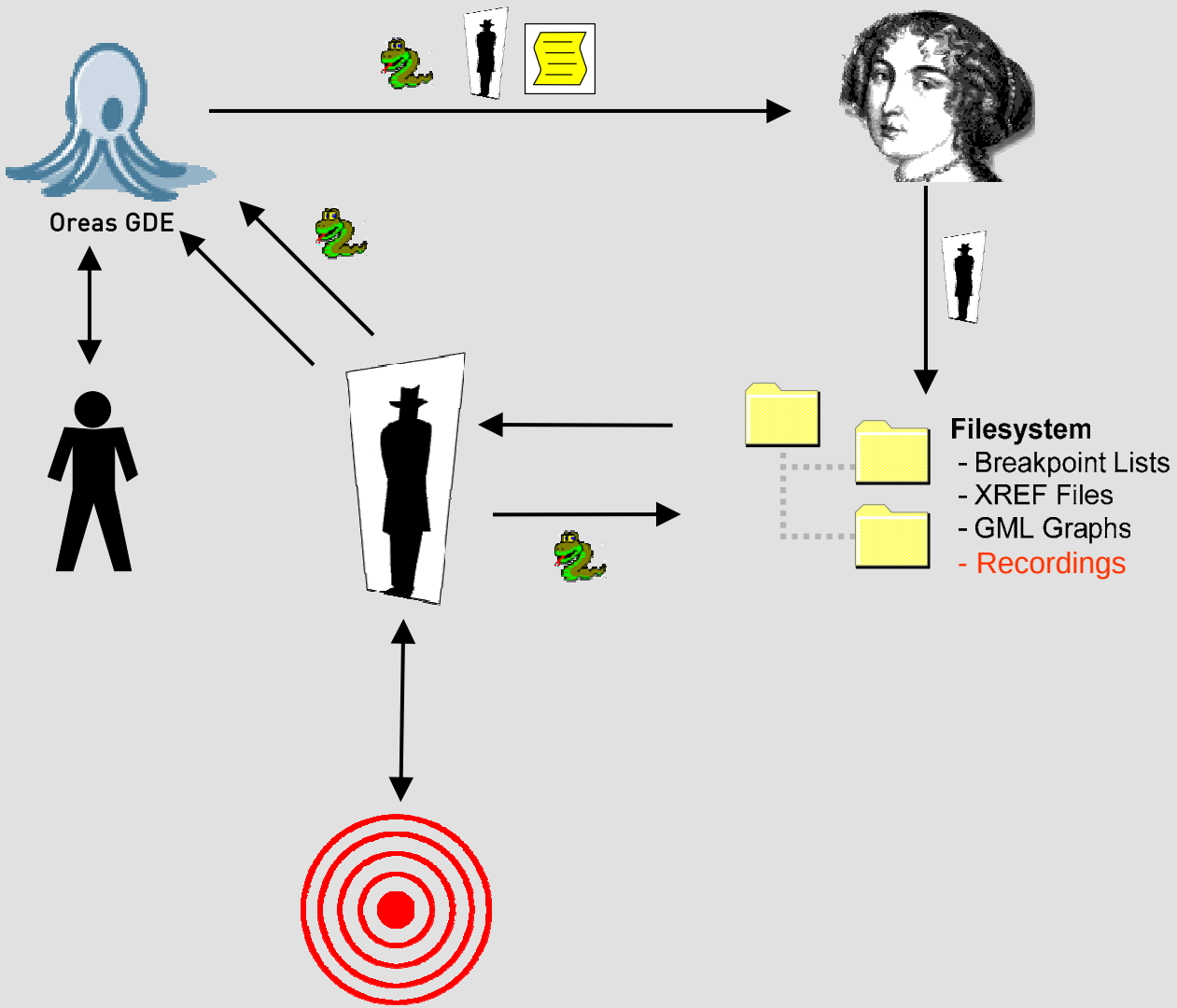
iDEFENSE

- Requirements
  - IDA Pro (commercial)
  - Python (free)
  - Oreas GDE Community Edition (free)
- Components
  - IDA plug-in
  - Standalone tracer
  - Python scripts
- Development
  - C/C++
  - Python + custom API
  - Function Analyzer / Dumbug
- Related work
  - Sabre Security, BinNavi
  - HBGary, Inspector
  - SISecure? (Rootkit.com screenshot)

Data Flow Diagram

iDEFENSE

- Load binary in IDA
- Export to FS
- Stalk process
- Record
- Process results
- View in GDE
- Instrument graphs
- View in GDE again
- Make edits
- Mark locations
- Export back to IDA



Process Stalker IDA Plug-in Internals

iDEFENSE

- Built on top of Function Analyzer
- Analysis routine is applied to each identified function
- Breakpoint entries are generated for every node:
  - ndmpsrvr.dll:0002b1b0:0002b29c
  - Module, function offset, node offset
- Cross reference entries are generated for every call:
  - 0002cbd0:0002cc34:0002bb20
  - Function offset, node offset, called function offset
- Customized .GML graph's are generated for each function:
  - ndmpsrvr.dll-010a1af0.gml
  - ndmpsrvr.dll-010a1b20.gml

Process Stalker Tracer Internals

iDEFENSE

- Built on top of Dumbug
- Attach to or load a target process
- On DLL load events
  - Determine module base address
  - Add loaded module to linked list
  - Automatically import available breakpoints
  - Add function-level breakpoints to self-balanced tree*
- On breakpoint events
  - If recording, write entry to file:
    - 0008c29d:000005cc:IMComms.dll:10001000:0000d25d
    - GetTickCount(), thread ID, module, module base, breakpoint offset
  - Optionally raise breakpoint restore flag and SINGLE_STEP

Process Stalker Script Internals

iDEFENSE

- Written in Python

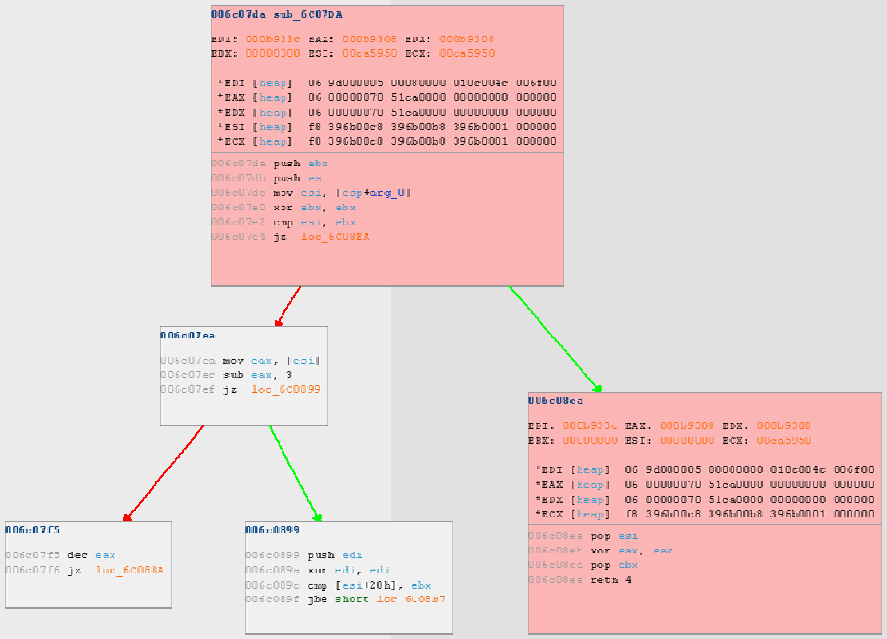
- Process Stalker API: gml, ps_parsers
  - GML: Can parse and manipulate generated .GML files
  - PS_PARSERS: Can parse and manipulate breakpoint lists, recordings, cross-reference lists and register metadata files
  - Fully documented

- Various functionality already implemented:
  - Recording -> list -> breakpoint filter
  - Graph concatenation with optional cross referencing
  - Recursive graph visualization
  - Run trace "folding" for loop visualization
  - And more...

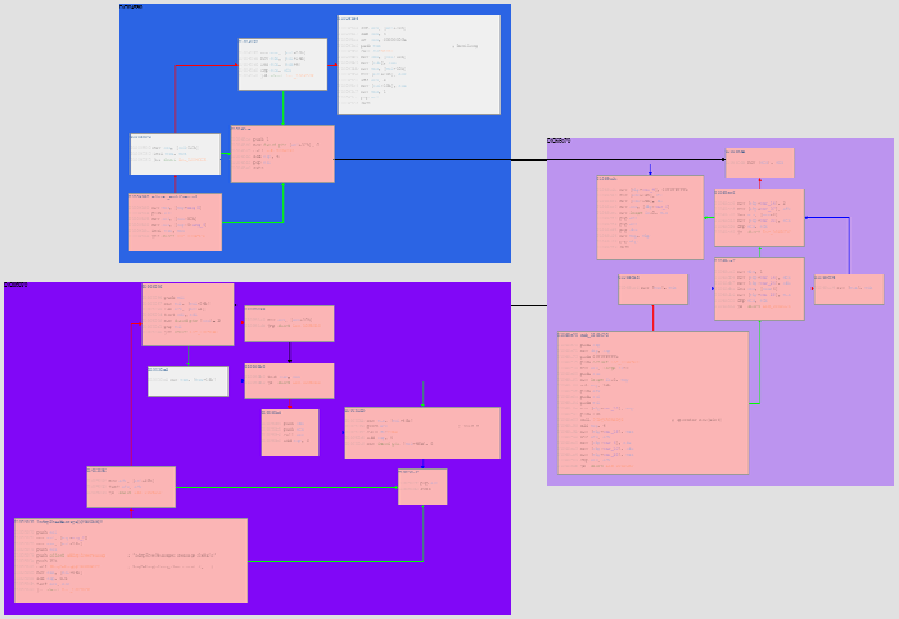
Now the pretty slides...

Visual Run-Time Tracing

- Immediately see which nodes handle your input
- View graphs with different layout algorithms
- View relevant register data



Hierarchical layout

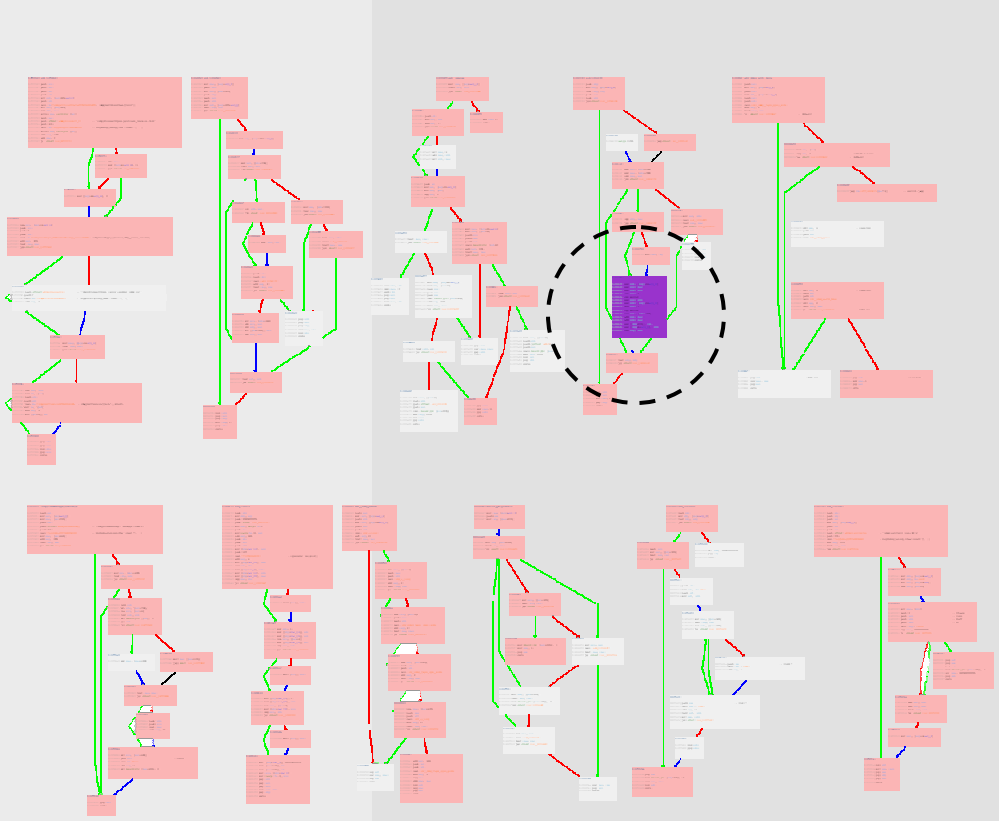


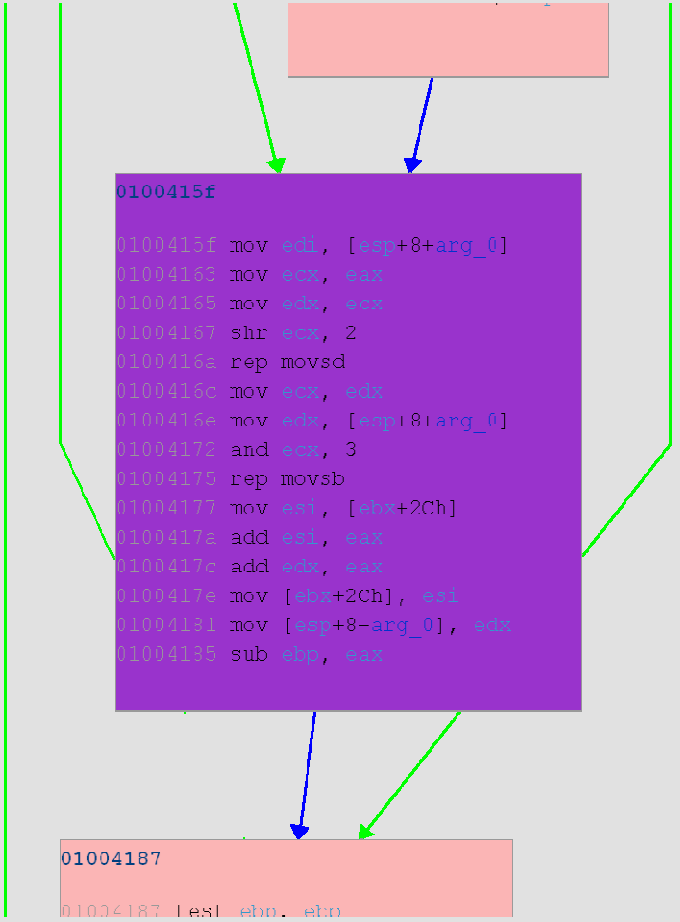
Cluster orthogonal layout

Automated Highlighting

iDEFENSE

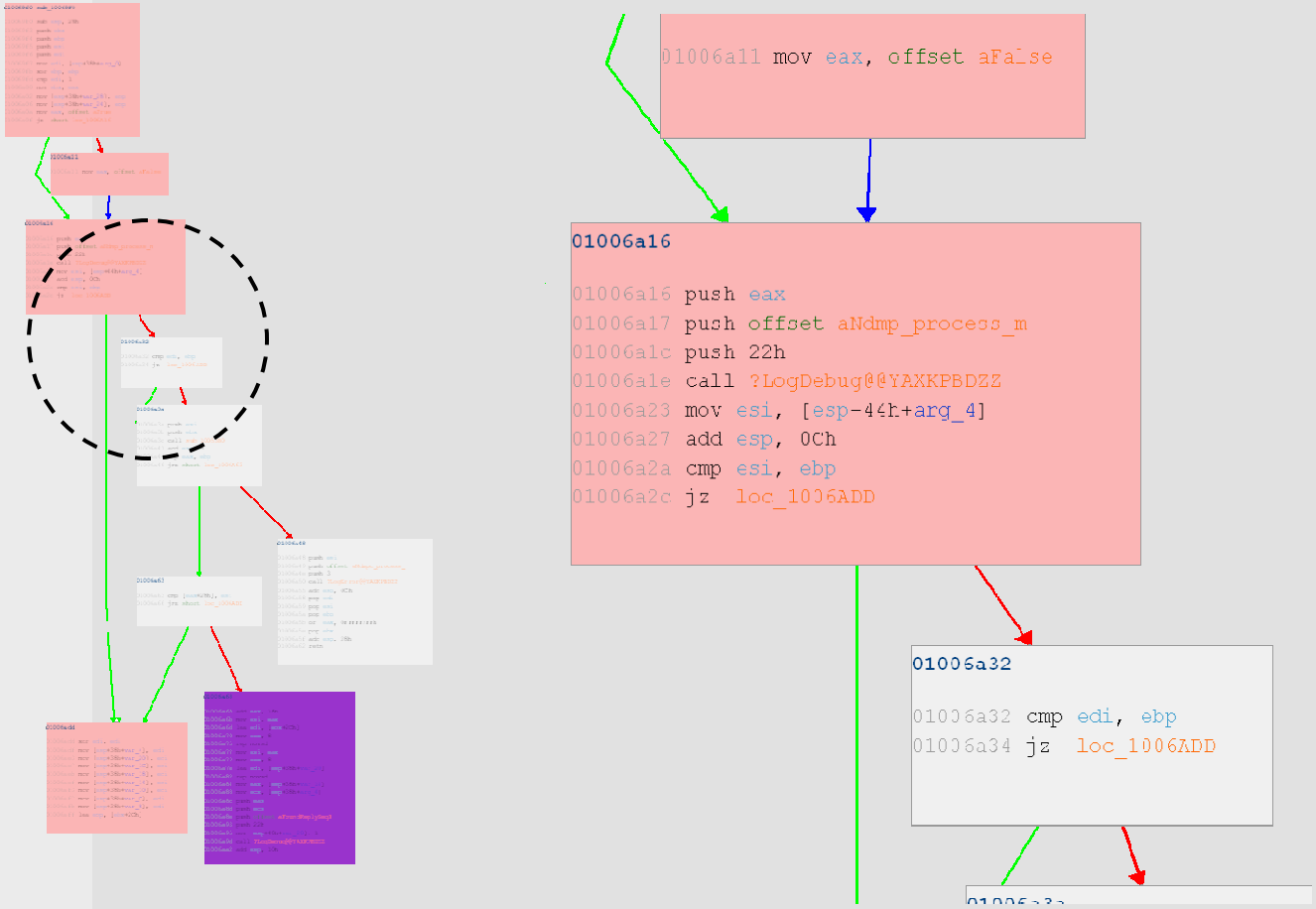
- Potentially interesting nodes are automatically highlighted
- ex: reps, *str*, *wcs*, *alloc*, *mem*





Alternative Paths

- Easily view and examine branch conditions
- Determine changes required to get fuzzer “deeper” into process state



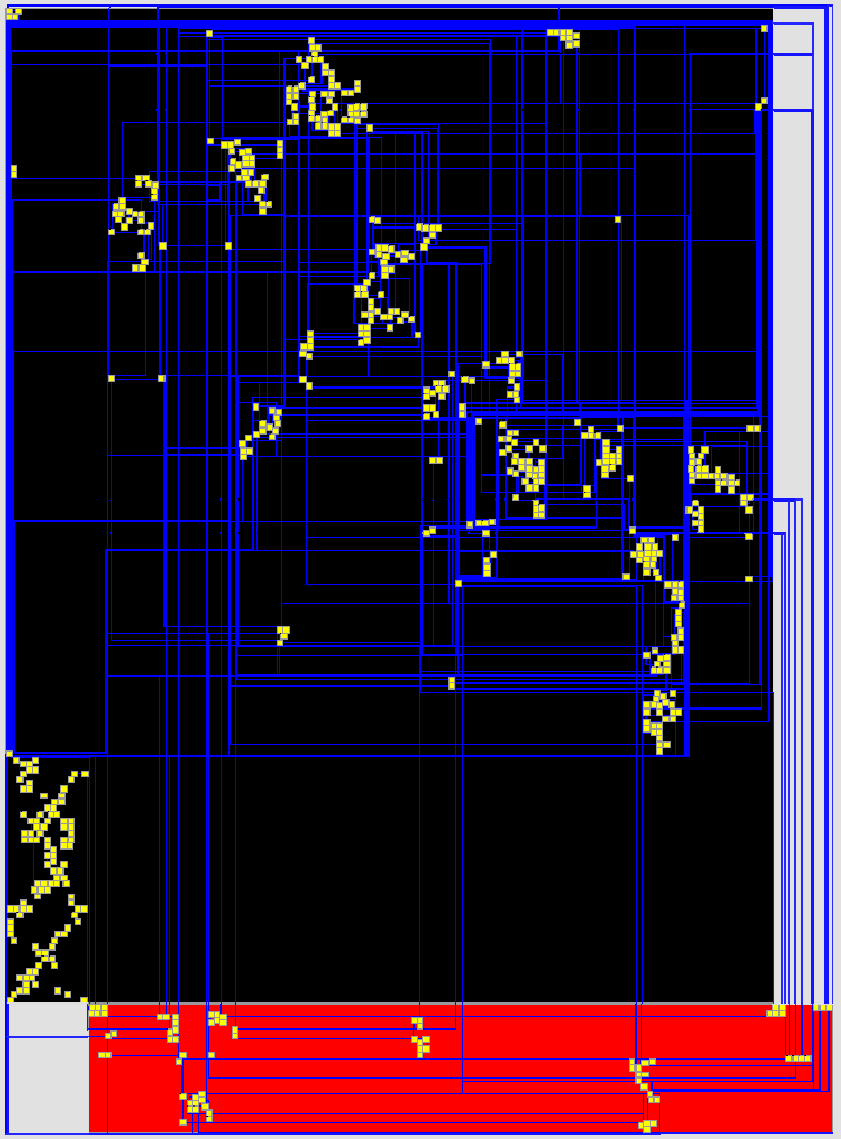
Speed

- Much faster than single-step tracing
- Two modes of operation
  - Breakpoint restore
  - One shot
- Breakpoint filtering can further improve performance
  - Functions only
  - Potentially interesting modules only
  - See next slide

Filtering

iDEFENSE

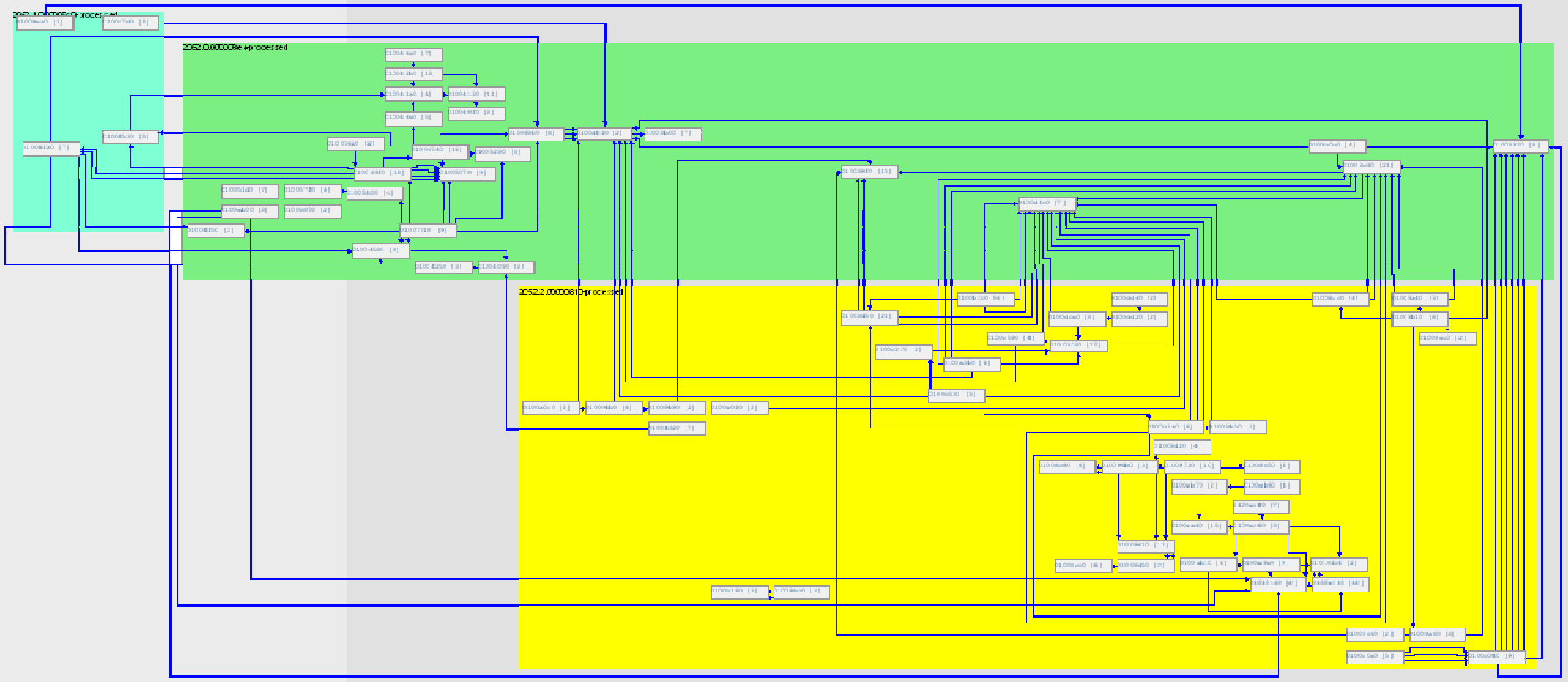
- Recordings can be joined and/or diffed
- Example: GUI handling code can be recorded and diffed out
- MS05-030: MSOE.DLL
  - Black: GUI functions
  - Red: Non-GUI functions



State Mapping

iDEFENSE

- ex: Authenticated vs. non-authenticated code
- ex: What our fuzzer has reached vs. what our fuzzer can reach



Recording Statistics

- Node hit counts
- Node transition times

```
$ ps_view_recording_stats 2284.0.000003d8-processed

function block hit counts for module irc.dll

        46011500    5   46014e81    1   46012510    4
        4600b010    2   460179e0    1   4600ae70    4
        4601559e    1   46006820    4   46006630   24
        ...

function transition times (milliseconds) for module irc.dll

        4600f560   40   460067e0    0   4600f560    0
        46006820    0   46006630    0   4601559e    0
        46006630   21   4600f560   60   460067e0   10
        4600f560    0   46001690    0   4600f560    0
        ...
```

Demonstration

Command Line Arguments

iDEFENSE

- Incase you can't see them during the demo

```
$ process_stalker
process stalker
pedram amini <pedram.amini@gmail.com>
compiled on Jun 14 2005

usage:
  process_stalker <-a pid | -l filename | -la filename args>

options:
  [-b bp list]   specify the breakpoint list for the main module.
  [-r recorder]  enter a recorder (0-9) from trace initiation.
  [--one-time]   disable breakpoint restoration.
  [--no-regs]    disable register enumeration / derefencing.
```

More Commands

```
$ ps_process_recording gui_shit

$ cat gui_shit.* > gui_shit.processed

$ wc –l gui_shit.processed
  4455 gui_shit.processed

$ time ps_bp_filter msoe.dll.bpl msoe.dll.nogui \
  `ps_recording_to_list gui_shit.processed msoe.dll` out
  real    0m28.367s

$ wc –l msoe.dll.bpl msoe.dll.nogui
  58165 msoe.dll.bpl
  50560 msoe.dll.nogui

$ time ps_view_recording_funcs 844.1.processed > hitgraph.gml
  real    0m7.446s

$ time ps_graph_highlight –nodes hit hitgraph.gml > hitgraph_hl.gml
  real    0m5.795s

$ time ps_add_register_metadata 844-regs.1 hitgraph_hl.gml > with_regs.gml
  real 0m7.977s
```

In Development

- Still working on this stuff:
  - Argument dereferencing
    - With automatic detection of ASCII and Unicode strings
  - Smarter highlighting
- Other ideas:
  - Arbitrary data structure visualization
  - Data flow visualization
- Potential design changes:
  - Remove dependency on IDA
  - Switch from debugger to emulation instrumentation (BOCHS)

Questions and Thanks

iDEFENSE

- Thanks to
  - iDEFENSE Labs
  - Gerry Eisenhaur
  - Gaël Delalleau
  - Nicolas RUFF
  - Anyone else I may have forgotten
- And especially … Mike the intern for taping together the graph blanket