spin
Static instrumentation for binary reverse-engineering

David Guillen Fandos

Tarragona – Spain

david@davidgf.net
davidgf.net – github.com/davidgfnet

Reverse-engineering

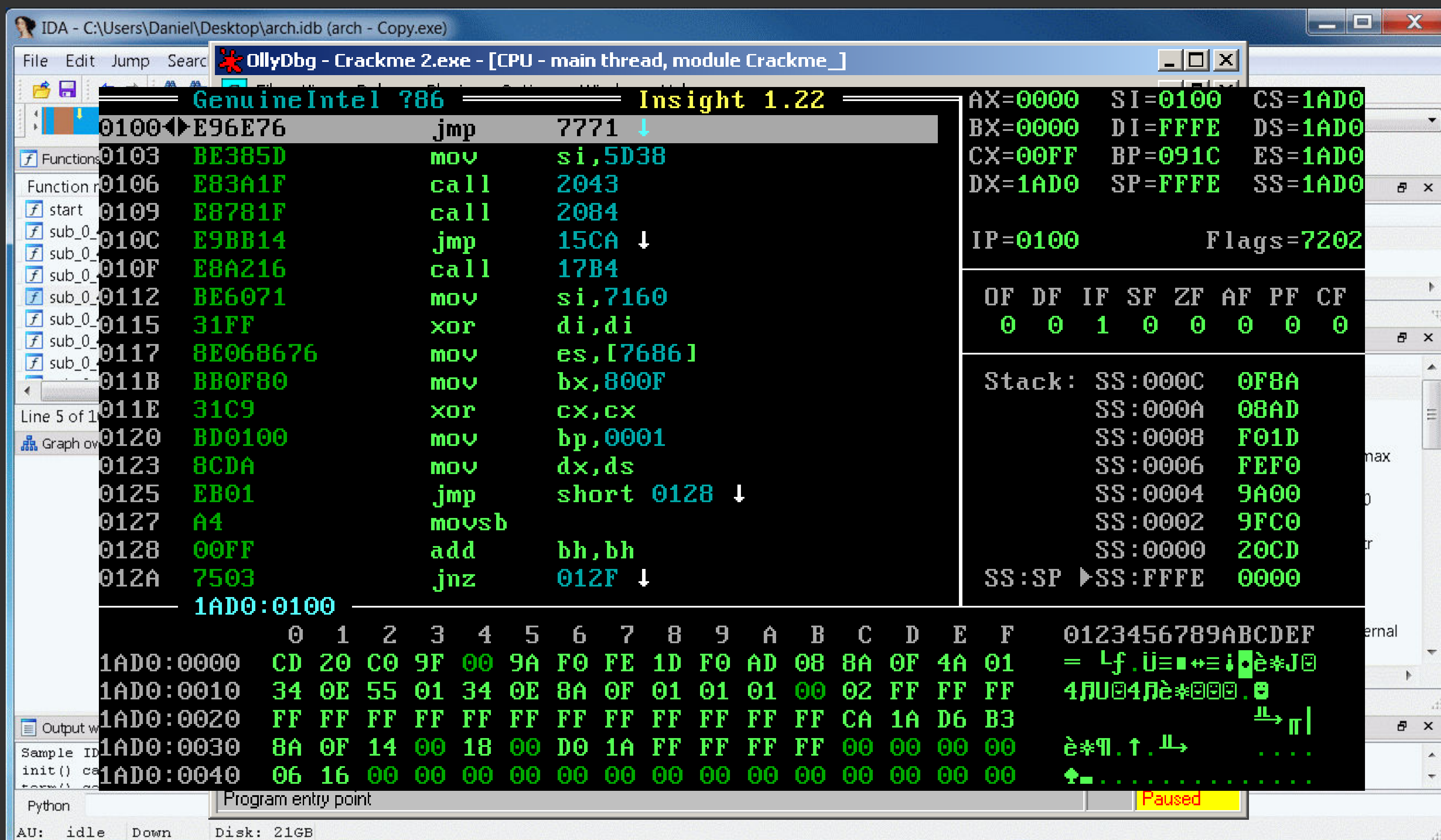
- Discover how a software program works
  - Disassemble binaries
  - Figure out what they do

- Typically done using a **debugger**

Nothing new here!

Reverse-engineering

Sounds easy right?



Reverse-engineering

- Debugging/reading assembly can be tedious
  - In fact it's boring
- In the past assembly was written by humans

Now compilers do all the work!!

- It's difficult to read their machine code but...
- They are predictable, respect call conventions and interfaces...

Reverse-engineering

So... Why don't we take advantage of this to ease our lives?

Could we do automatic-reverse engineering?

Let machines do all work!

Automatic reverse-engineering?

- Is it even possible?
- How much automatic is it?
- Can it replace a 'human'?

Machines, you know...

Automatic reverse-engineering!

- Let's create a tool that does all the dirty job we usually do by hand!

- How?

Let's use **binary instrumentation**

Wait, what da heck is binary instrumentation?

Binary instrumentation 101

- Binary instrumentation is a technique which allows to modify and rewrite existing binaries
  - We can modify their behavior at runtime
  - Typically used in a non-intrusive way: just analyze the program
  - At assembly level: cannot reverse to high level languages
- Many tools available:

  Pin, DynamoRIO, Valgrind ...

Binary instrumentation 101

- Works by injecting instructions in the original code
  - Using something similar to a VM
  - Rewrites code on demand
- It is possible to add user code on instruction basis, basic block, etc.

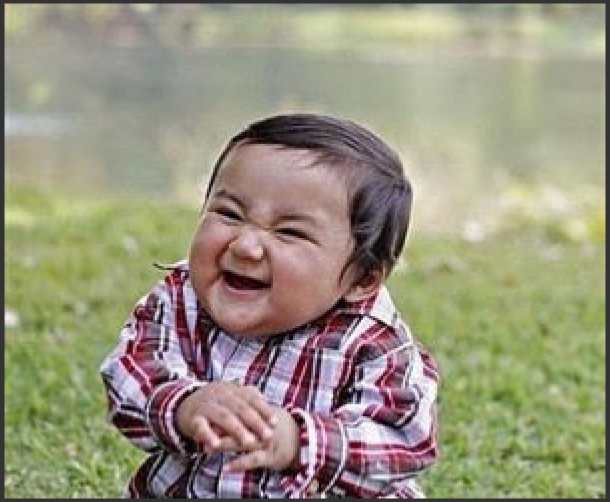
```
...
mov edi, esi
lea (esi,eax,4), ecx
call instrument_func_pre
mov edi, (ecx)
mov (ecx+4), edi
inc edi
call instrument_func_pre
mov edi, (ecx+4)
...
```

x86 example: instrument all memory stores (red are added instructions)

Binary instrumentation 101

- What industry and professionals use binary instrumentation for
  - Application performance evaluation
  - CPU emulation
  - Tracing and profiling
  - Many others...

- What do we use it for...?



Binary instrumentation 4 hackers

- How can we use it for our purposes?
  - Create complex conditional breakpoints
    - Just like debugger does, evaluate something and trigger 'break'
    - This is cool cause debuggers usually only do stateless conditions
  - Create app tracing/logging outputs
    - Dump any interesting info to a file
    - We can also conditionally dump interesting info
  - Modify the application behavior
    - We can modify memory and registers

Binary instrumentation 4 hackers

- Let's try to think like the coder of the App.
- We probably want to work on function basis
  - Look for relevant functions
    - By using complex breakpoints (retaining status across executions) it is possible to characterize functions
    - We can have a look at the stack too!
  - Generate some log with this info
    - We can discard 99% of “boring” functions in the binary

I wrote my own tool to do some of this...

Spin: Static instrumentation

- A tool for instrumenting at function granularity
  - Runs in application memory space
  - Allows us to receive function parameters
  - Optionally we can modify return values

  - We rely on compilers respecting calling conventions (true for C/C++)

Spin: Static instrumentation

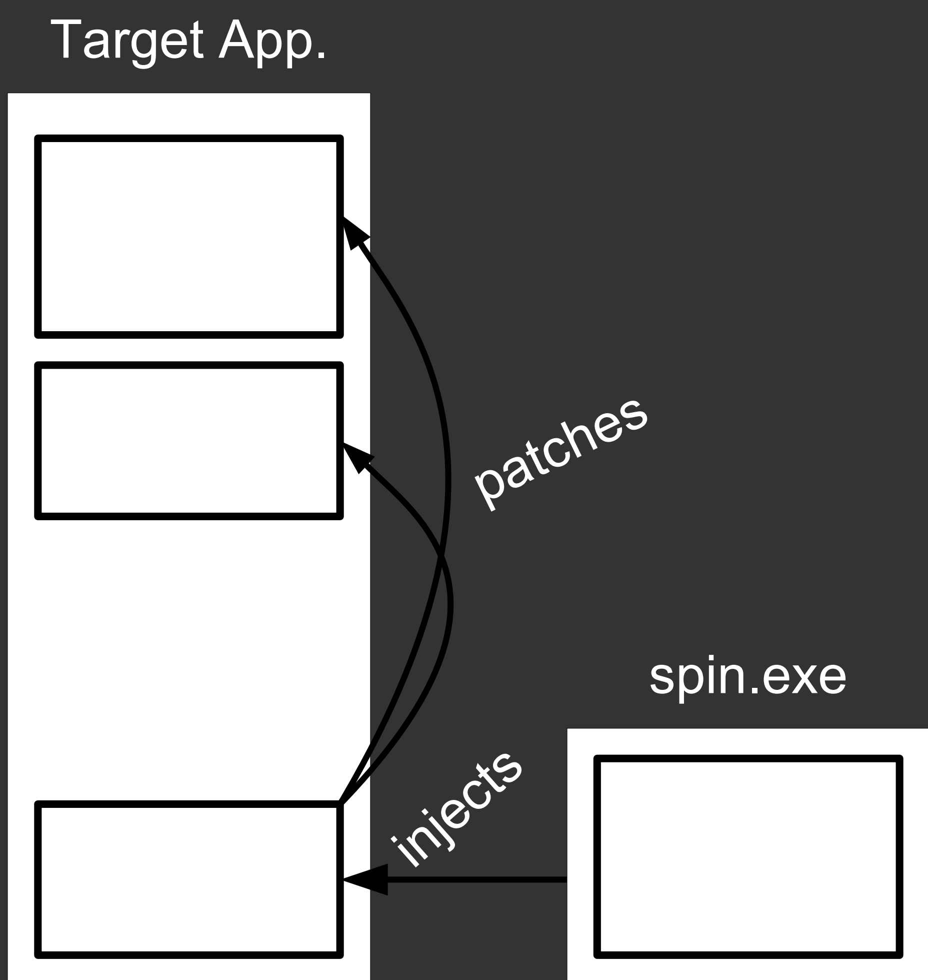
- Works by patching call instructions
  - Only support for immediate encoding
  - This way the instrumentation is *static*
  - It uses the same principle as DLL hooking

```
...
push 0x67
push eax
call 0x4013742
add esp, 8
...
```

↓

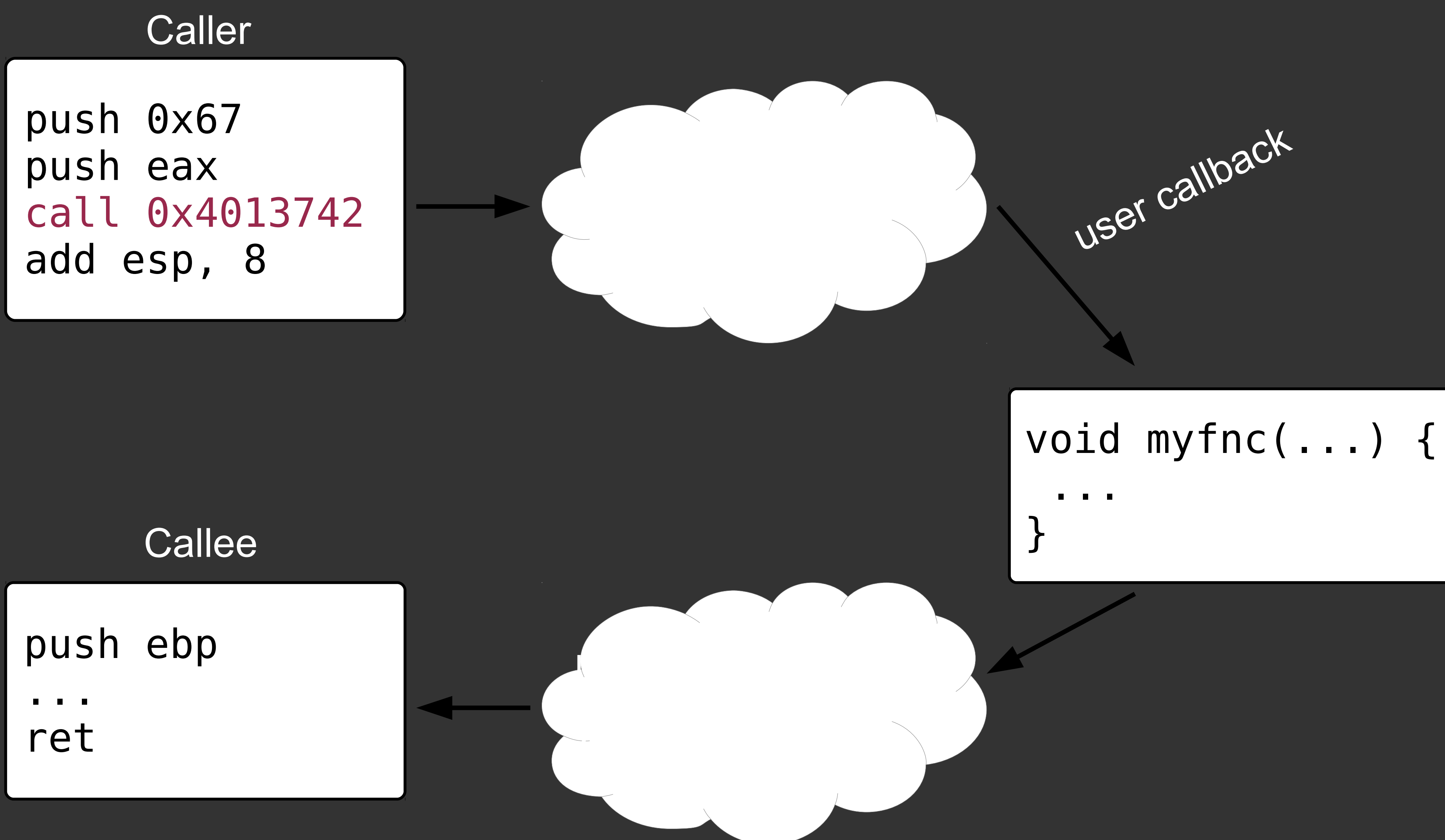
```
...
push 0x67
push eax
call 0xac00de0
add esp, 8
...
```

Spin: Static instrumentation



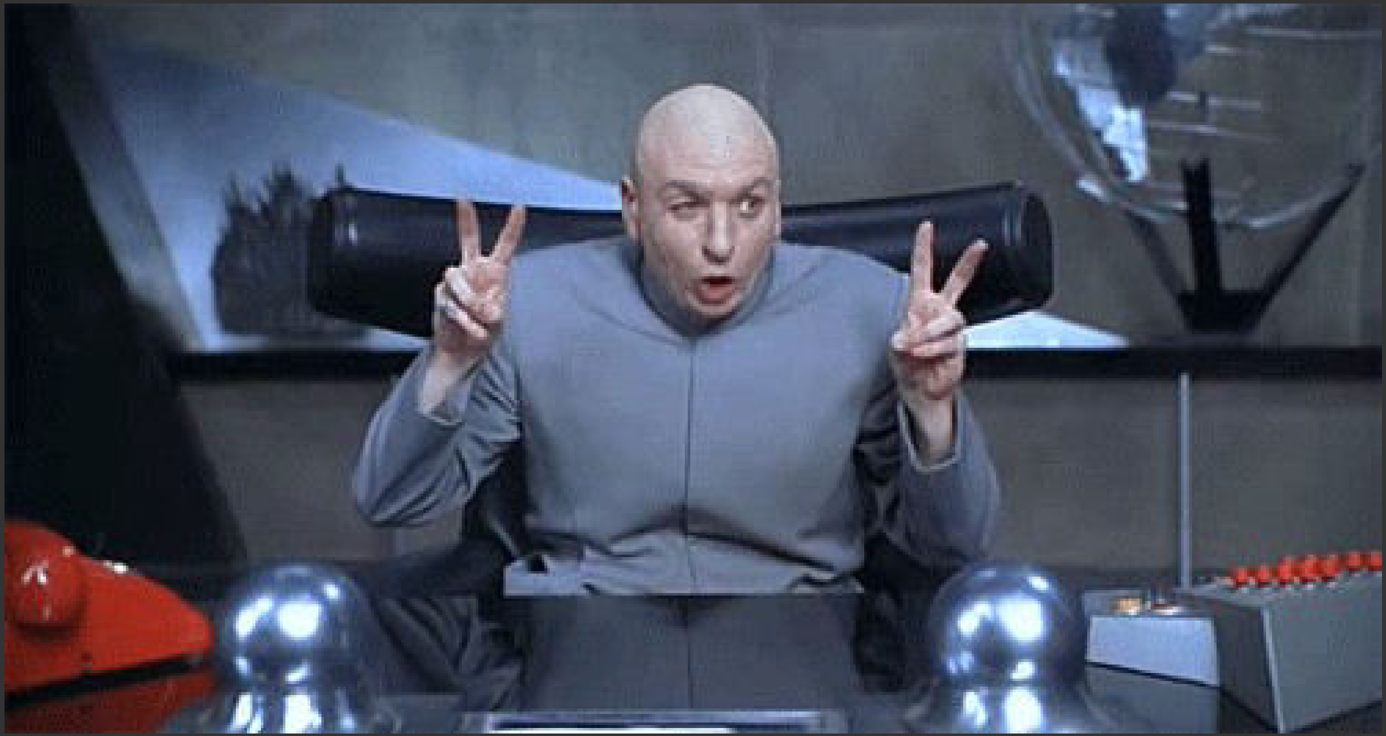
- Calls get redirected to user defined functions
  - DLL injection
  - It is possible to hook/dehook specific instructions or areas
  - Choose modules to patch (avoid patching system/standard libs)

Spin: Static instrumentation



Demo time!

- This demo is just for “educational purposes”



Advanced instrumentation

- Show me more! What else can we do?
  - Advanced object analysis: Dump data from C++ objects and C/C++ structs
  - De-instrument uninteresting functions
    - The overhead is noticeable
    - This can be tricky, we don't want to lose data!
  - Look for patterns across calls
    - Usually is more interesting to locate some functions for later analysis than trying to get the good one
    - I told you! It's not 100% automatic!

Example: std::string

- Analyze function parameters containing std::string objects
  - Important things to know: compiler, libraries ...
  - In our example:
    - MSVC compiler: Uses ECX as 'this' pointer
    - MSVC stdlib: Stores short strings in place, large strings in heap. Pointer at +4 offset.
  - Others: Ability to inject tool at startup

Go demo go!

Example: dynamic dehooking

- Analyzing function calls can be slow.
- Idea: remove hooks from uninteresting functions
  - Simple way to do it: create a criteria and dehook functions matching/not matching it
  - More complex: Retain some status
    - Remove functions which do not match some conditions many times

Another demo?

Conclusions

- It is possible to automate some reverse-engineering methodologies
- But where is the limit?
  - The tool is far from perfect
  - Not suitable for API hooking
  - Protected/obfuscated sources will kick us

Q&A

Thank you!

Questions?