MAKING ROP BETTER WITH CPU EMULATION AND REALTIME ANALYSIS

DYLAN KNOFF (ELBEE_EZ@PROTONMAIL.COM)

HTTPS://FAULTPOINT.COM/POST/2024-06-02-BINJA-PLUGIN-ROPVIEW/

DISTRICTCON YEAR ZERO

Agenda

- What is ROPView?
- Components
- Unicorn Engine
- Gadget Emulation and Analysis
- Data Analysis and Semantic Search
- Demos
- Pitfalls

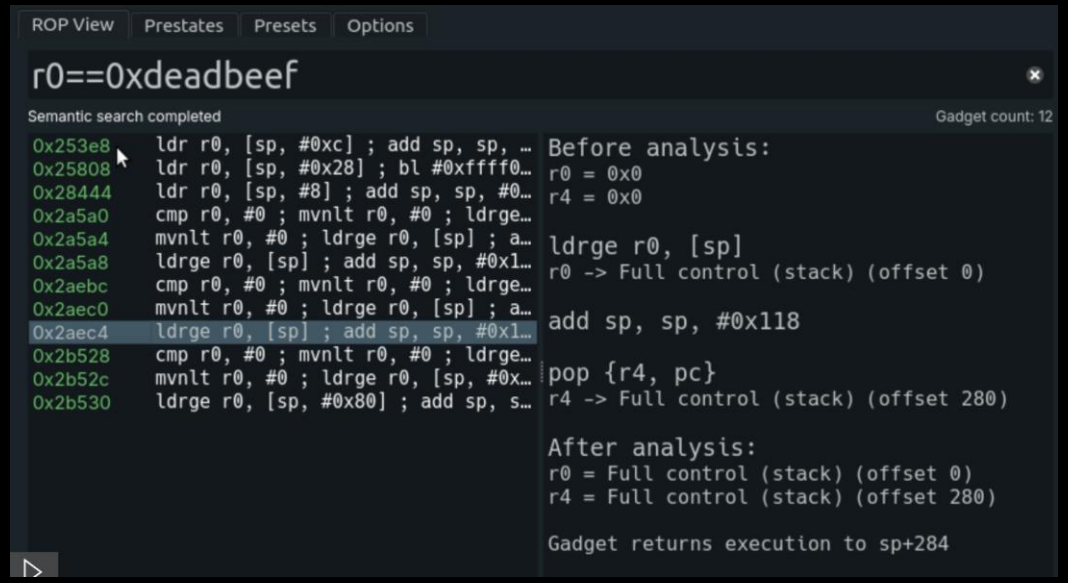
Who am I

Junior at GMU (CompSci)

Experience with RE/VR in embedded devices (but interested in other areas of RE)

Mason Competitive Cyber, U.S. Cyber Team (previously), CTF player, Co-founder @ bhartee.ai

Interested in emulation, automating weaponization, and tooling



Introducing ROPView

Plugin for BinaryNinja, featureful Gadget Searcher/Analyzer

Per-instruction effect analysis (backed by Unicorn)

Dataframe aggregation for pandas-based search queries

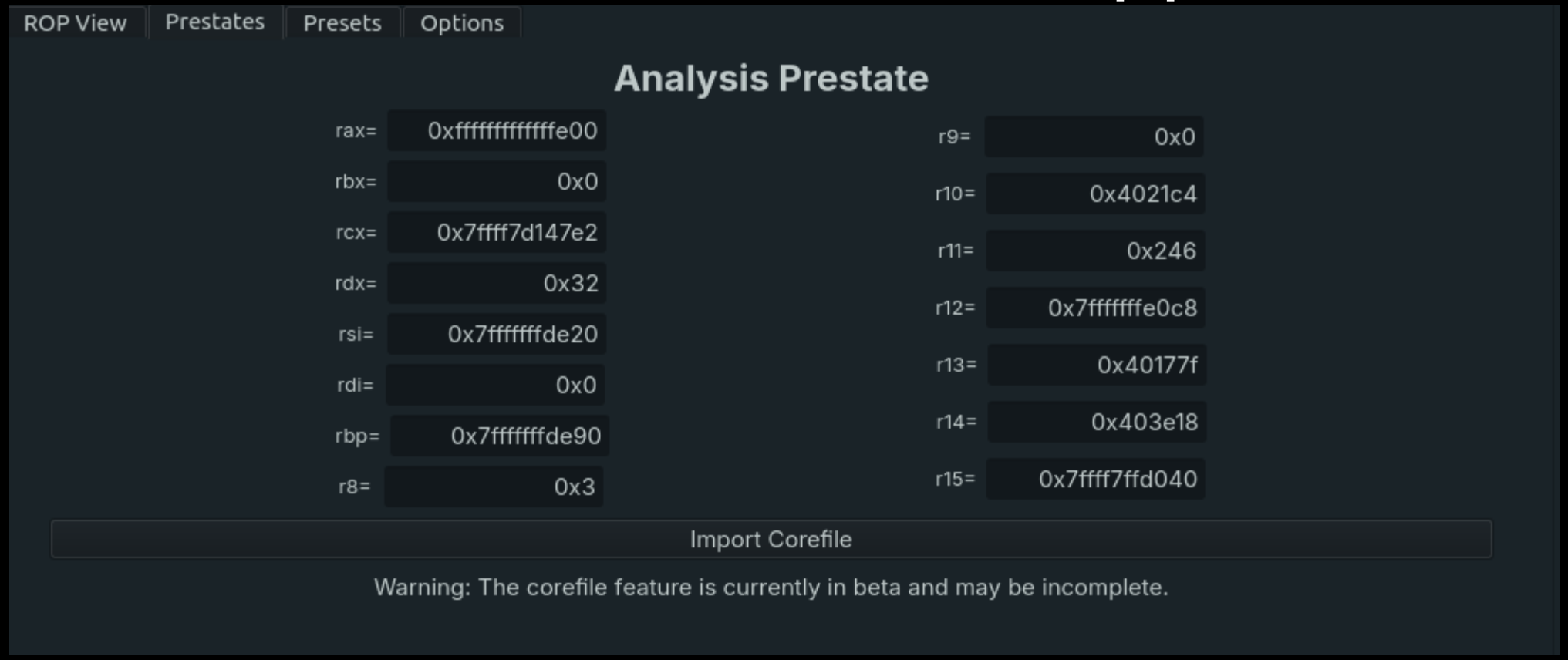
Caching, usual options, search presets, analysis prestates

armv7, thumb2, aarch64, mipsel32+64, mips32+64, i386, and amd64 (extendable constants)

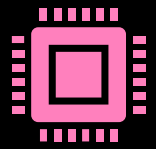
Presets


Create
cet
translates to:
disasm.str.contains("endbr64") and inst_cnt < 10
Add
Current Presets
stackfinder
system
tails
lia0
registers
sleep_a0
$a0 > 0 and $a0 < 600

Prestates (/w Corefile support!)



Components



Discovery

Gadget discovery handled in-house using Binja API (cached)

Analyzer

Handled via Unicorn emulations and hooks (cached)

Aggregator

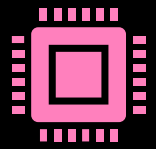
Handled via pandas, sorted by attributes and by register effects (uses Analysis States)

Renderer

Handles gadget repooling events and cache coherency

Components

Coolest!



Discovery

Gadget discovery handled in-house using Binja API (cached)

Analyzer

Handled via Unicorn emulations and hooks (cached)

- Deterministic and self-resolving
- Lightweight, even with analysis steps + hooks!

Aggregator

Handled via pandas, sorted by attributes and by register effects (uses Analysis States)

Renderer

Handles gadget repooling events and cache coherency

Unicorn Engine

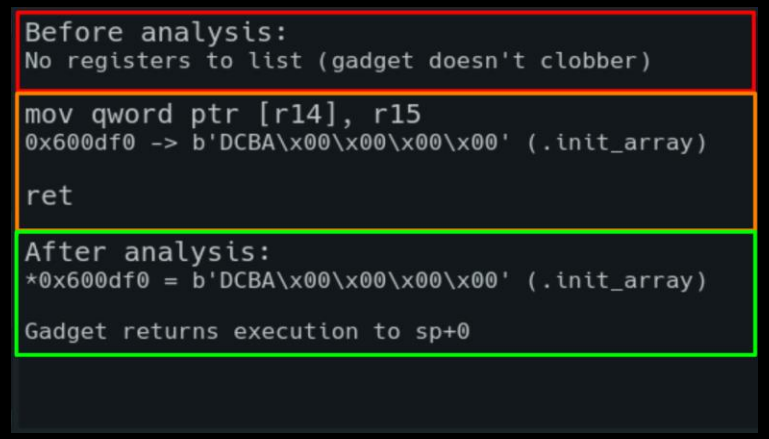
```
# i386
mu = Uc(UC_ARCH_X86, UC_MODE_32)

# write machine code to be emulated to memory
mu.mem_map(CODE_BASE, 0x1000)
mu.mem_write(CODE_BASE, X86_CODE32)
mu.mem_protect(CODE_BASE, 0x1000, (UC_PROT_READ+UC_PROT_EXEC))

# stack
mu.mem_map(STACK_BASE,0x1000)
mu.mem_write(STACK_BASE+0x100,STACK_DATA)
mu.mem_protect(STACK_BASE,0x1000,(UC_PROT_READ+UC_PROT_WRITE))
mu.reg_write(UC_X86_REG_ESP,STACK_BASE+0x100)

# emulate code in infinite time & unlimited instructions
mu.emu_start(CODE_BASE, CODE_BASE + len(X86_CODE32))
```

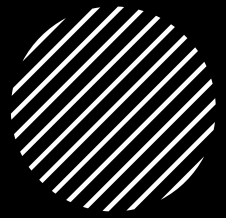
https://www.unicorn-engine.org

- Lightweight and contextless CPU emulation framework backed by Qemu
- Easy to use (mapping, write/read, register access) (+ python bindings)
- Lots of (mostly implemented) hooks including:
  - Before a insn is executed
  - When a memory violation occurs

Drawbacks

- Hooks slow emulation
- Space-complex due to mass instancing (especially during semantic searches)





Analyzer

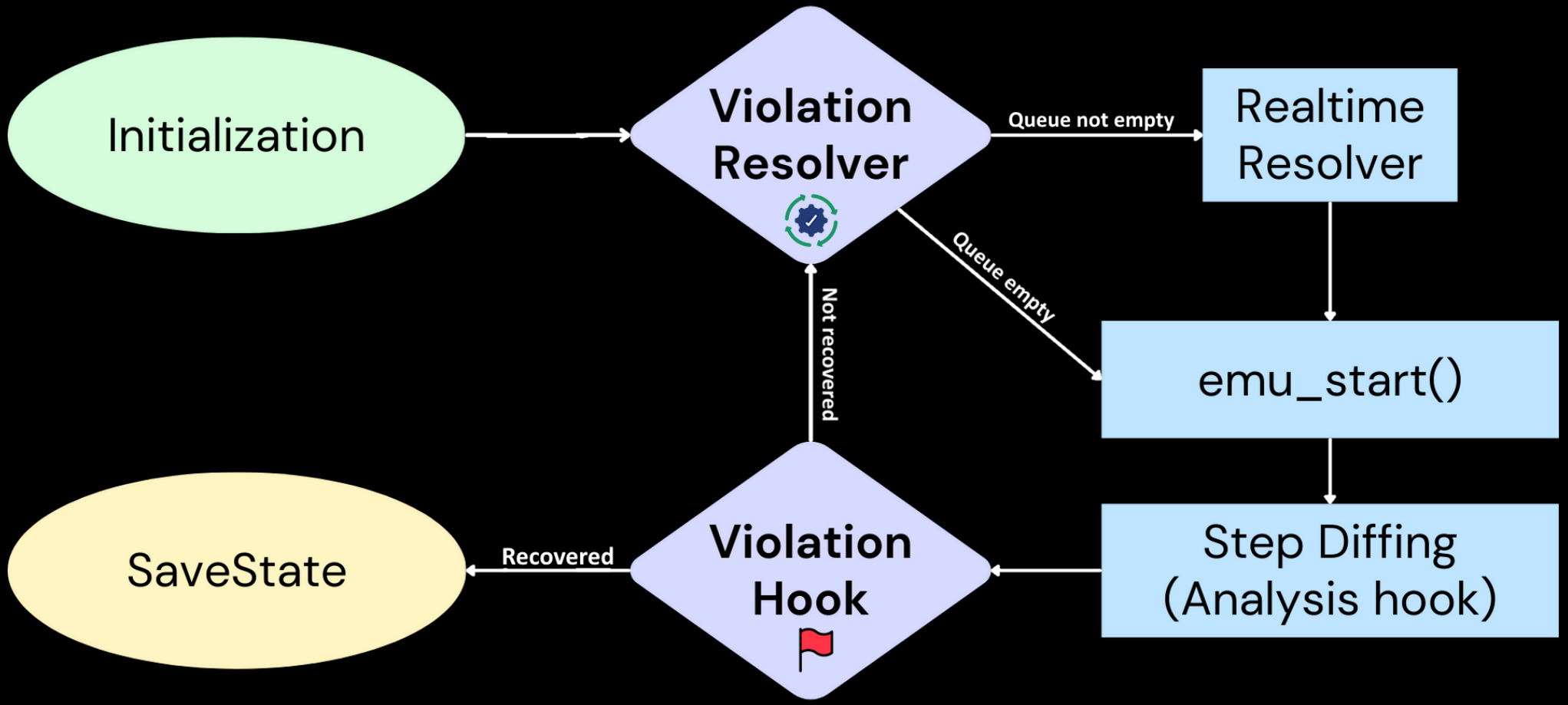
- Emulates and caches effects of selected item.

- The issue: Gadget emulations should be small, but context is large

- Resolver/Analysis Steps
  - Analysis hooks
    - Hook executes in-between fetch and execute and diffs clobbered registers/memory (populates Analysis pane + Dataframe columns).
  - CPU Exception Hooks
    - Hook exceptions to correct emu as needed and track errors. (ie unmapped memory is fetched and marked executable only when an access violation occurs).

Gadget Emulation Framework

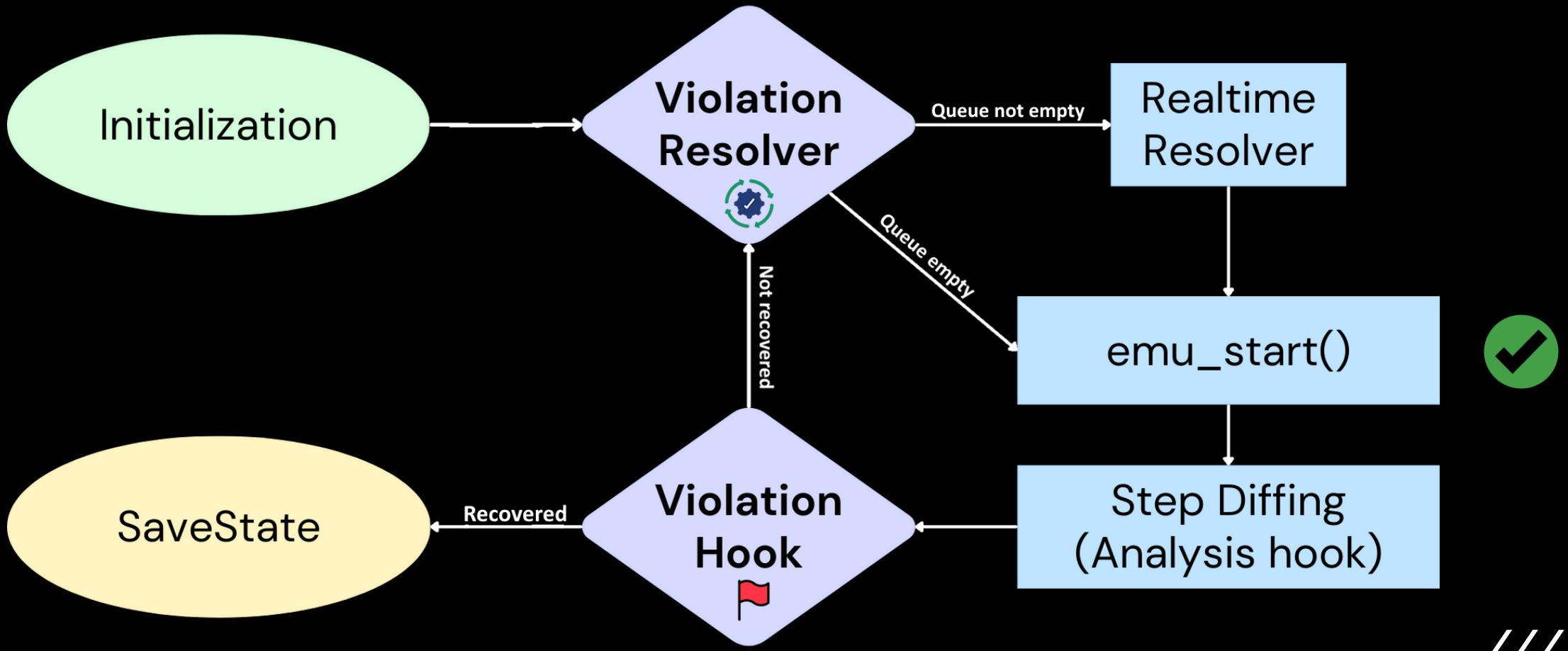
mov [r14], r15; ret;
r14==0x600df0, r15=0x1234



Gadget Emulation Framework

mov [r14], r15; ret;
r14==0x600df0, r15=0x1234

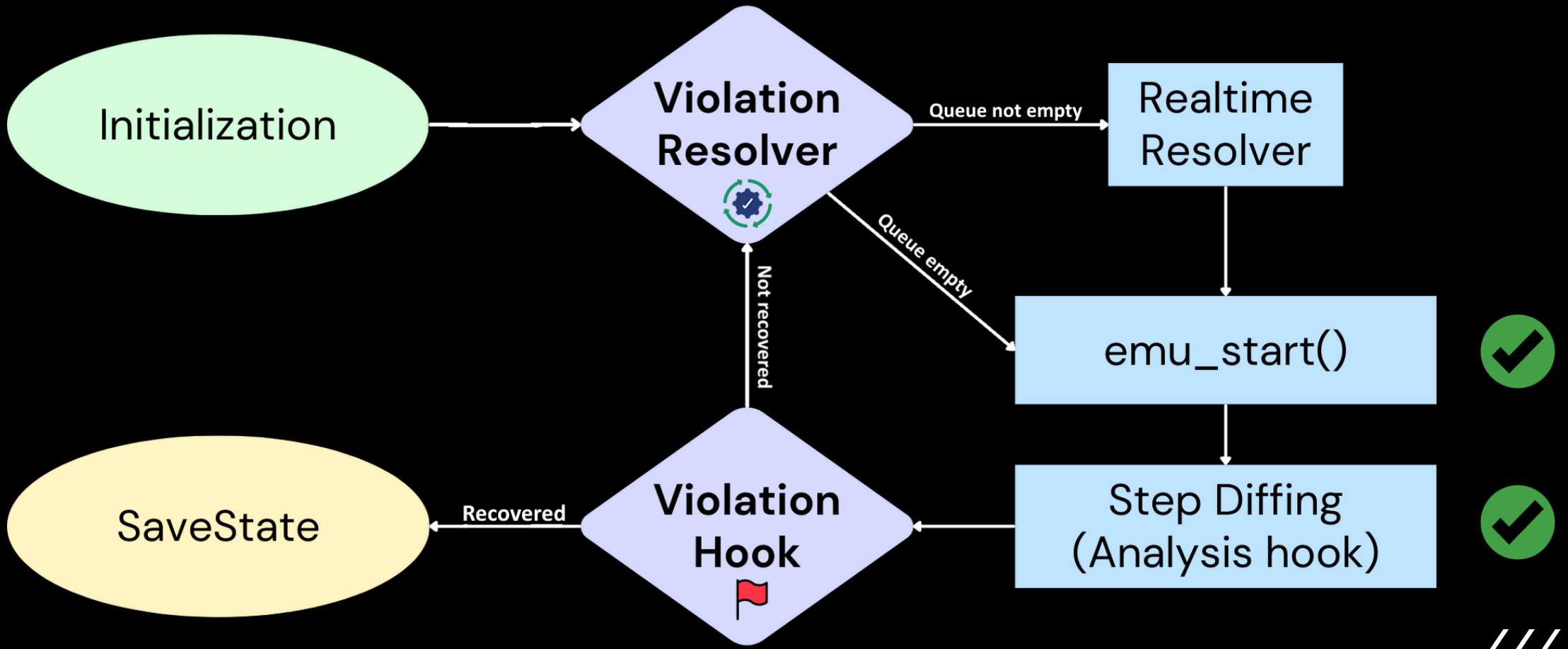
Resolver Queue:
Empty



Gadget Emulation Framework

mov [r14], r15; ret;
r14==0x600df0, r15=0x1234

Resolver Queue:
Empty

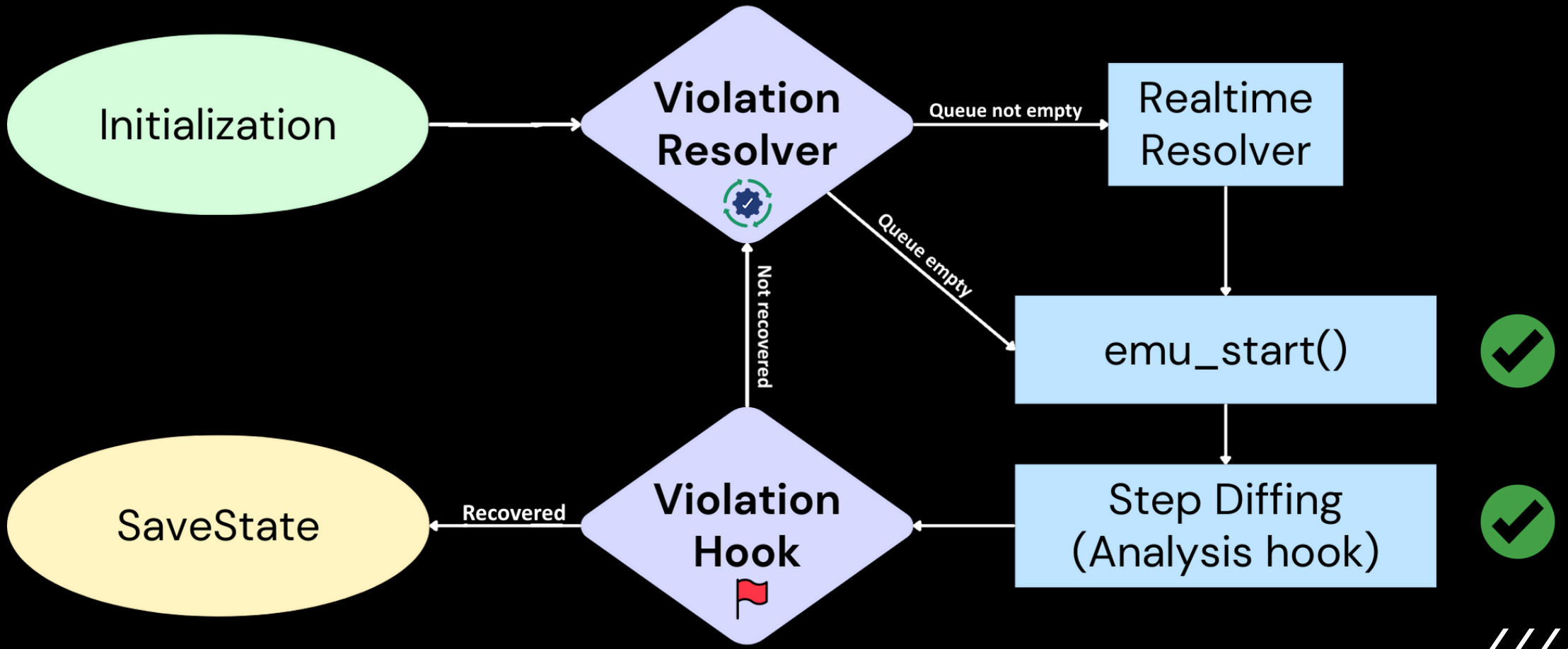


GA_UNMAPPED_WRITE (0x600df0)

Gadget Emulation Framework

mov [r14], r15; ret;
r14==0x600df0, r15=0x1234

Resolver Queue:
(0x600df0)



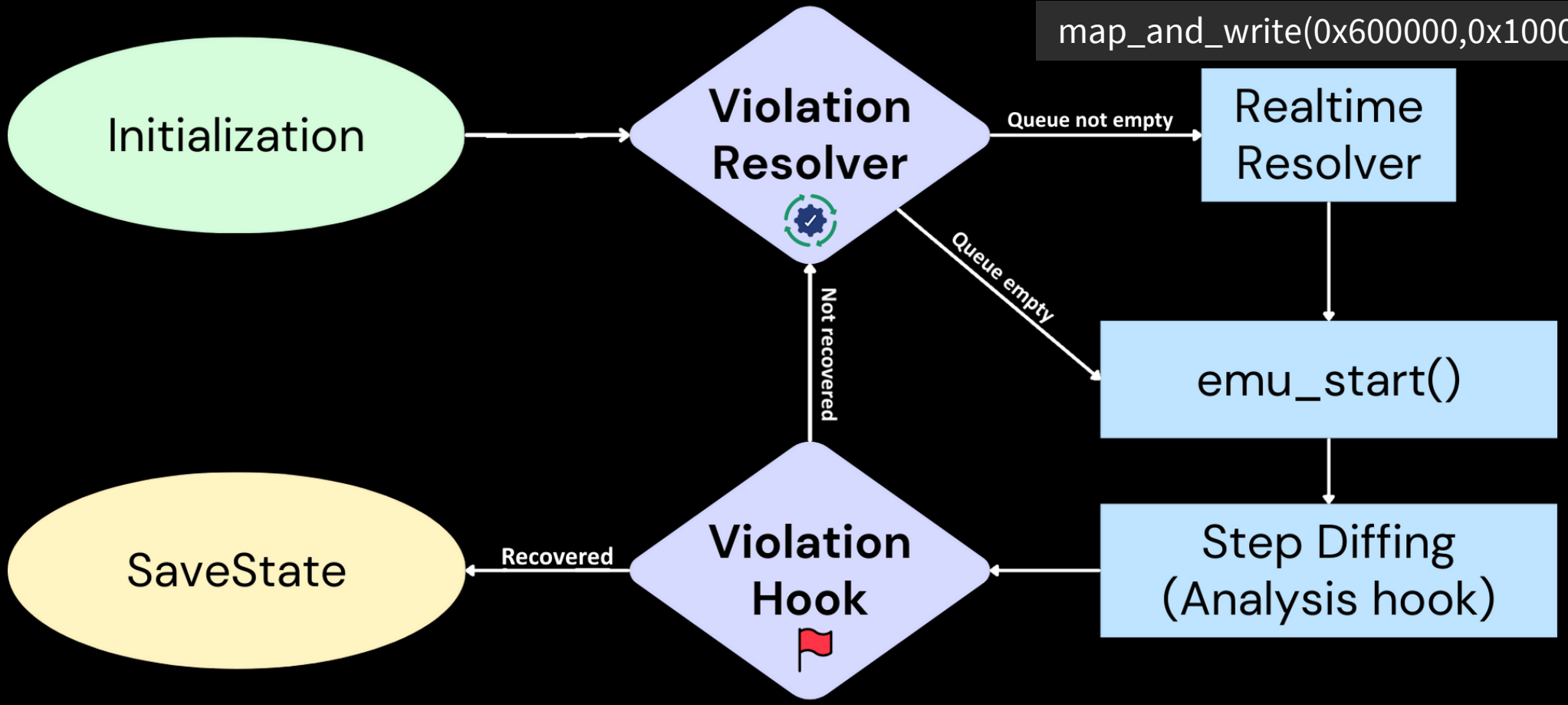
GA_UNMAPPED_WRITE (0x600df0)

Gadget Emulation Framework

mov [r14], r15; ret;
r14==0x600df0, r15=0x1234

Resolver Queue:
Empty

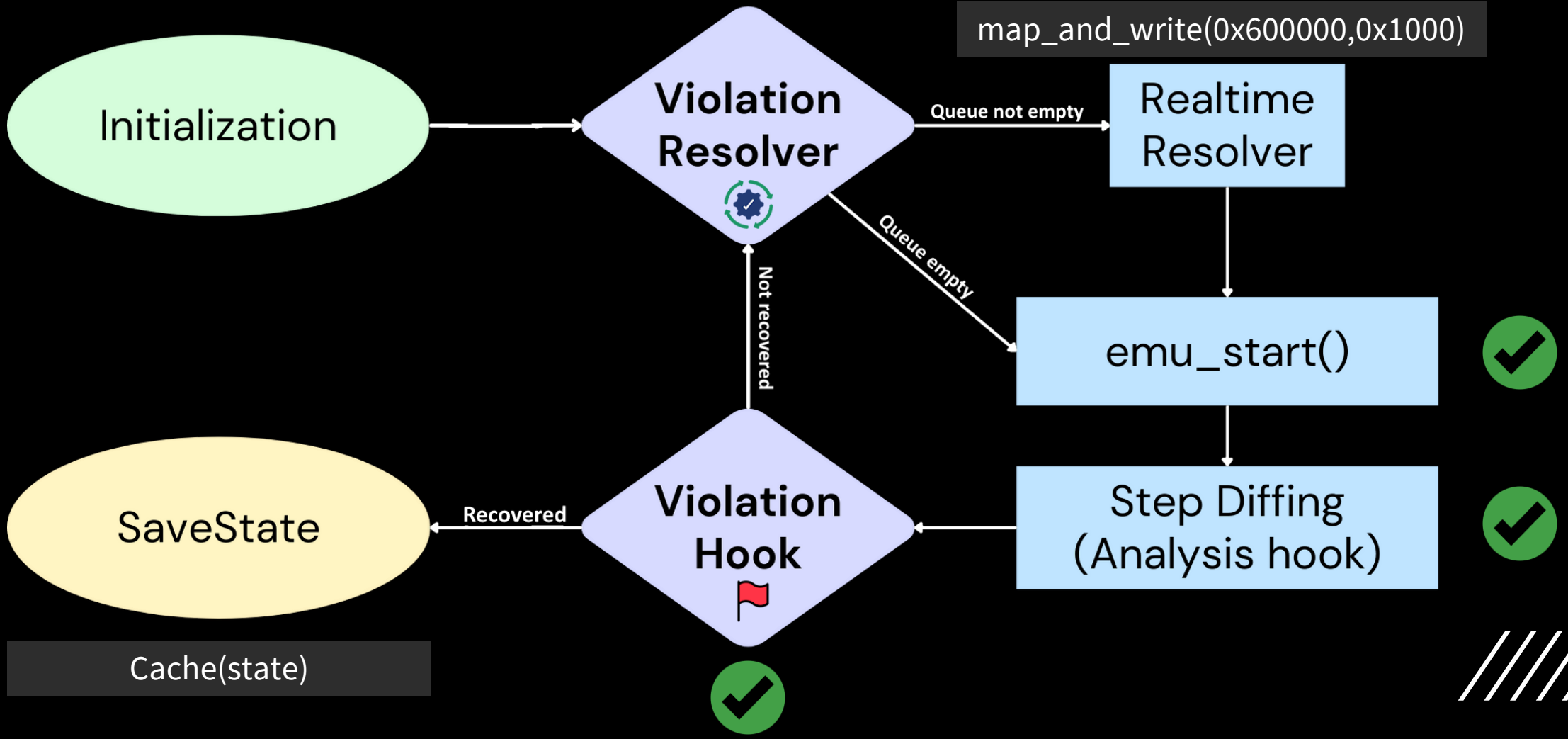
map_and_write(0x600000,0x1000)



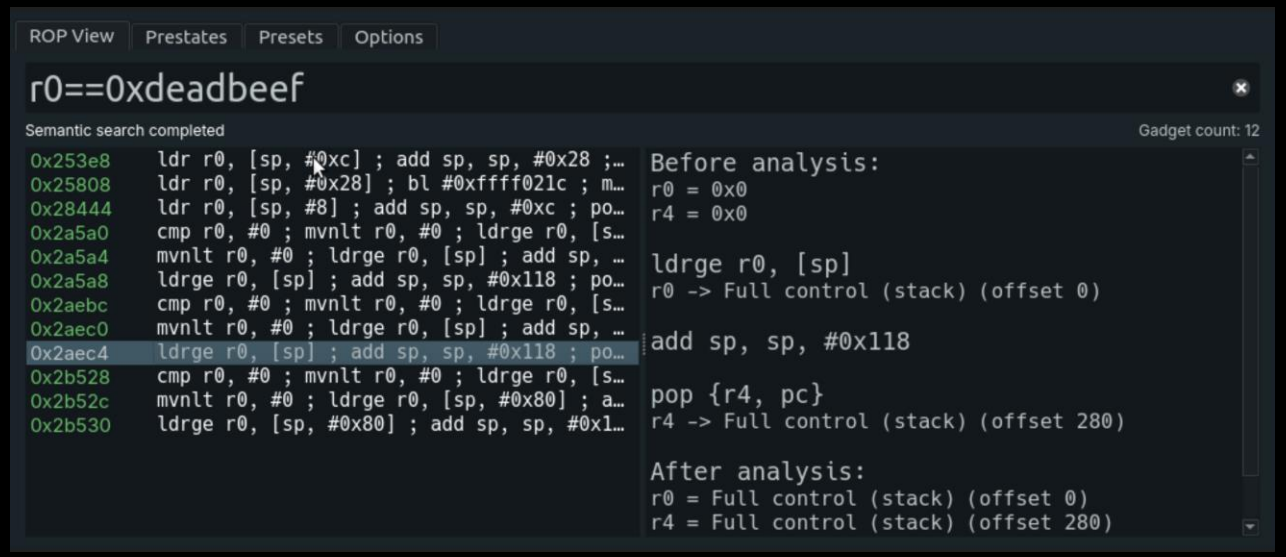
Gadget Emulation Framework

mov [r14], r15; ret;
r14==0x600df0, r15=0x1234

Resolver Queue:
Empty



Cache(state)




```
df.query("((reg==0xdeadbeef or reg==FULL_CONTROL) and not reg==NOT_ANALYZED)")
```

Data Analysis

Dataframe objects includes:

- Addr (int) – Address literal
- Loc (str) – Address as hex string
- Bytes (str) – Raw asm
- Inst_cnt (int) – Instruction count
- Disasm (str) – Gadget disassembly
- Registers (int[]) – Register values (analysis-based)
- Presets

All objects can be queried with the respective pandas functions available for that object.

pandas.Series.str.cat	pandas.Series.str.casefold
pandas.Series.str.center	pandas.Series.str.extractall
pandas.Series.str.contains	pandas.Series.str.find
pandas.Series.str.count	pandas.Series.str.findall
pandas.Series.str.decode	pandas.Series.str.fullmatch
pandas.Series.str.encode	pandas.Series.str.get
pandas.Series.str.endswith	pandas.Series.str.index
pandas.Series.str.extract	pandas.Series.str.join

https://pandas.pydata.org/docs/reference/api/pandas.Series.str.html

SemanticSearch

Search DataFrame

- Contains critical registers
- Dumb sort (by pop)

Populate Analysis

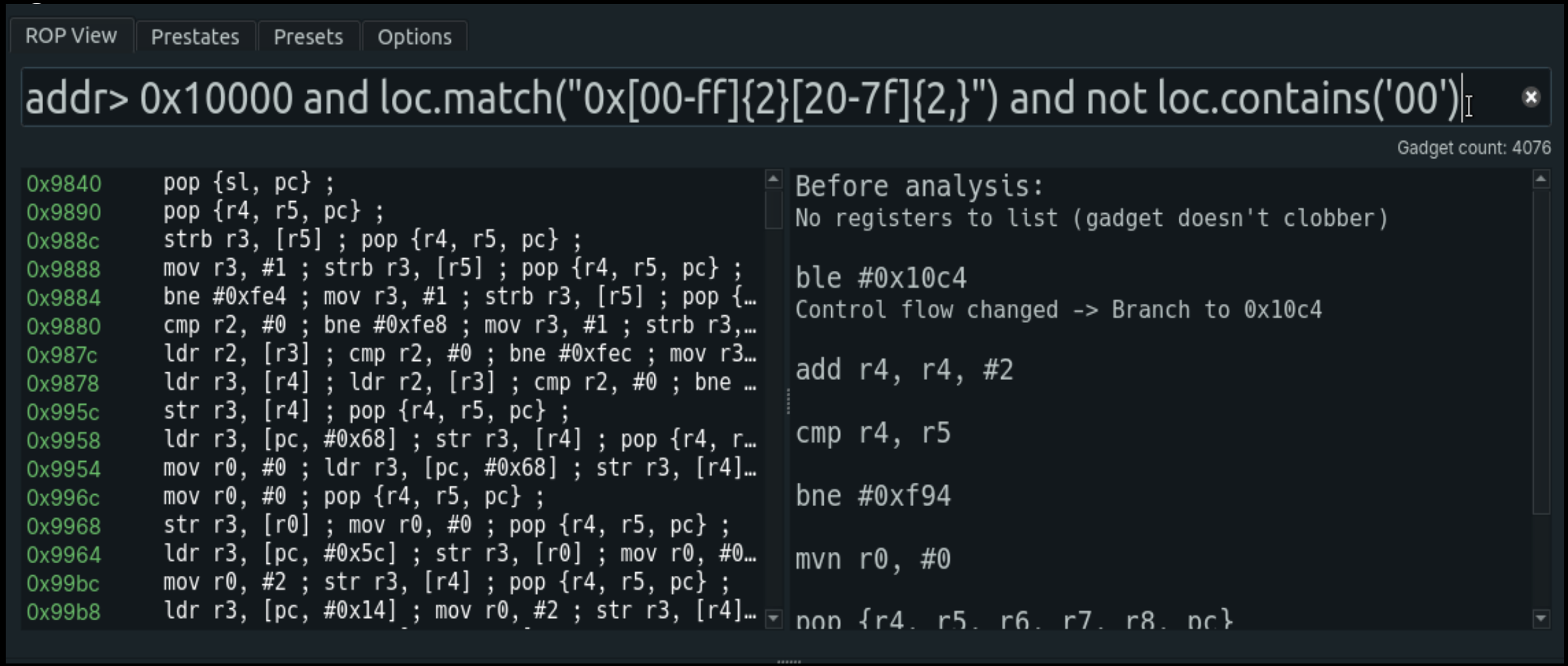
- Based on depth parameter
- Populate Analysis cache
- New + cached analysis saved to Search DF

Query Search DF

- Translate original query
- Query search DF

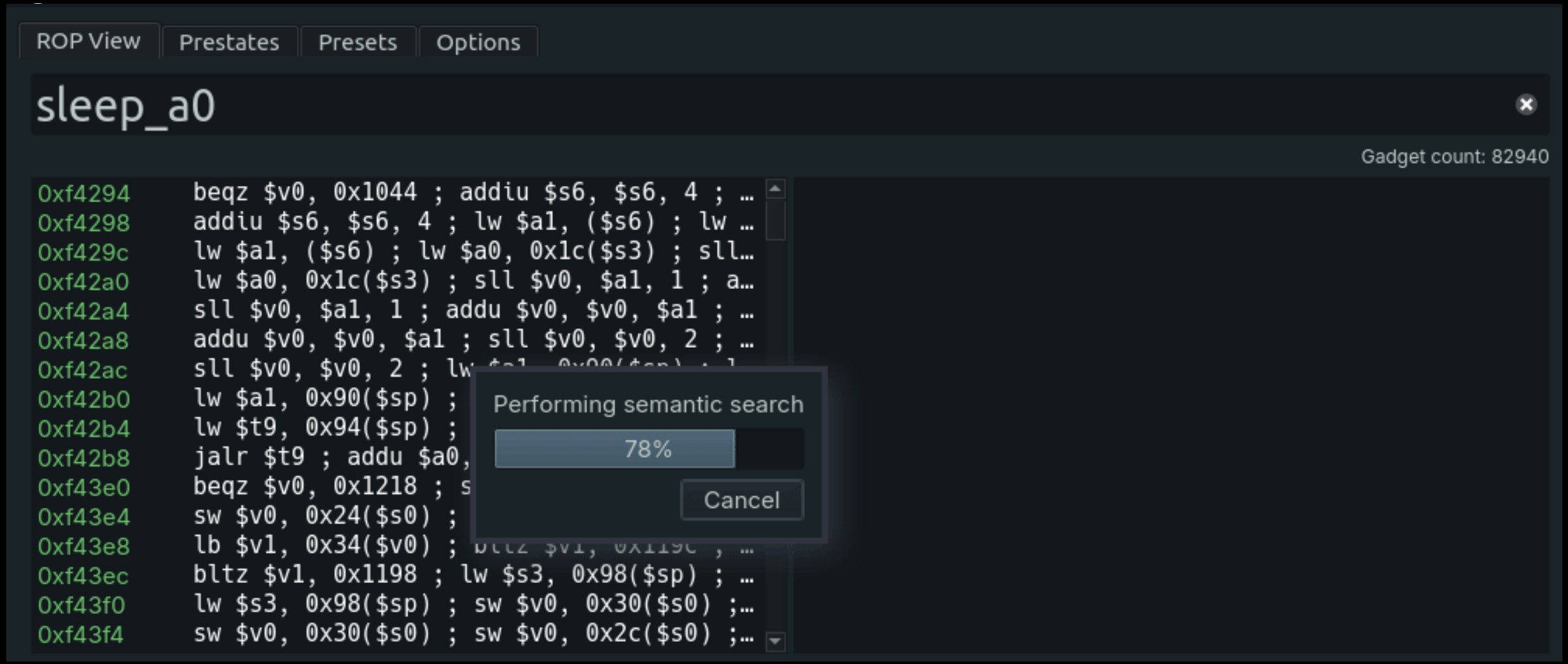
Usage

All bytes except the 2 MSB must be in ASCII range + extra constraints



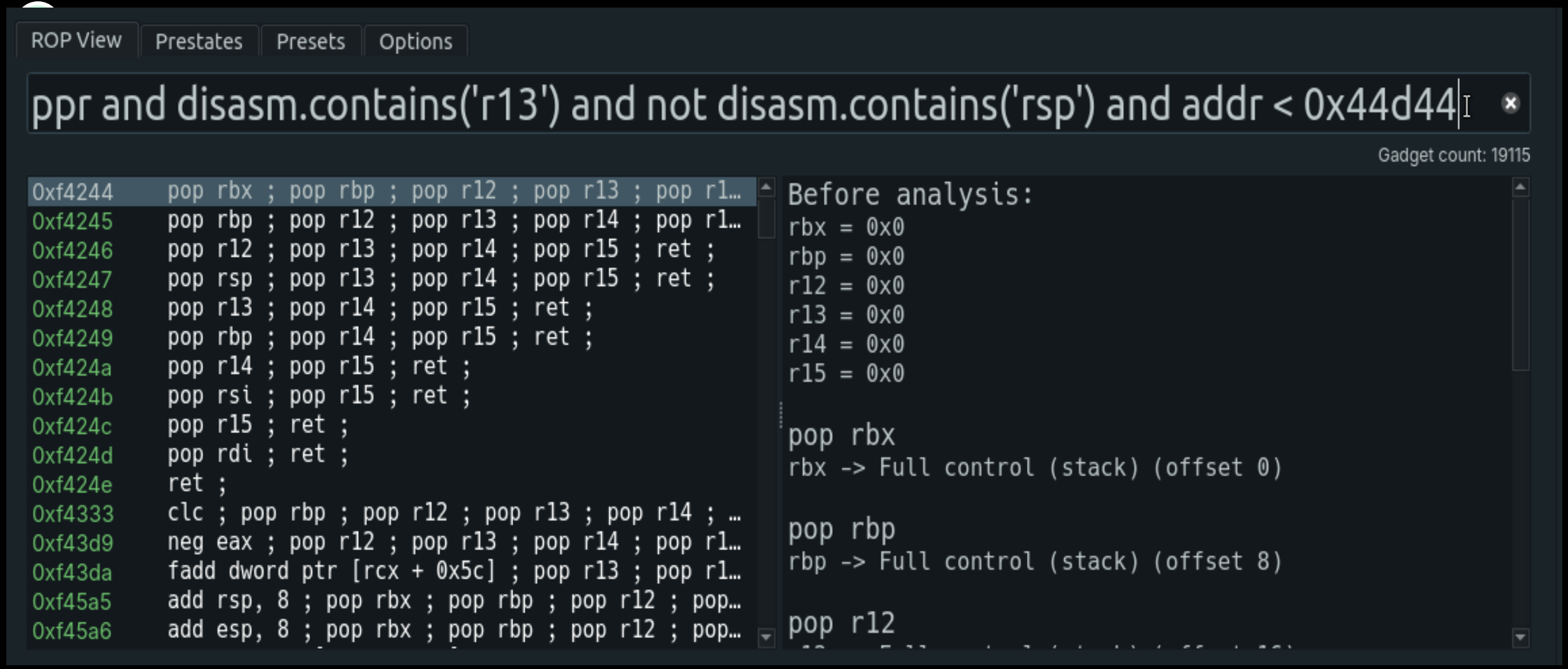
https://faultpoint.com/assets/ropview_demos/ascii.gif

sleep_a0 -> (a0 > 0 and a0 < 600)



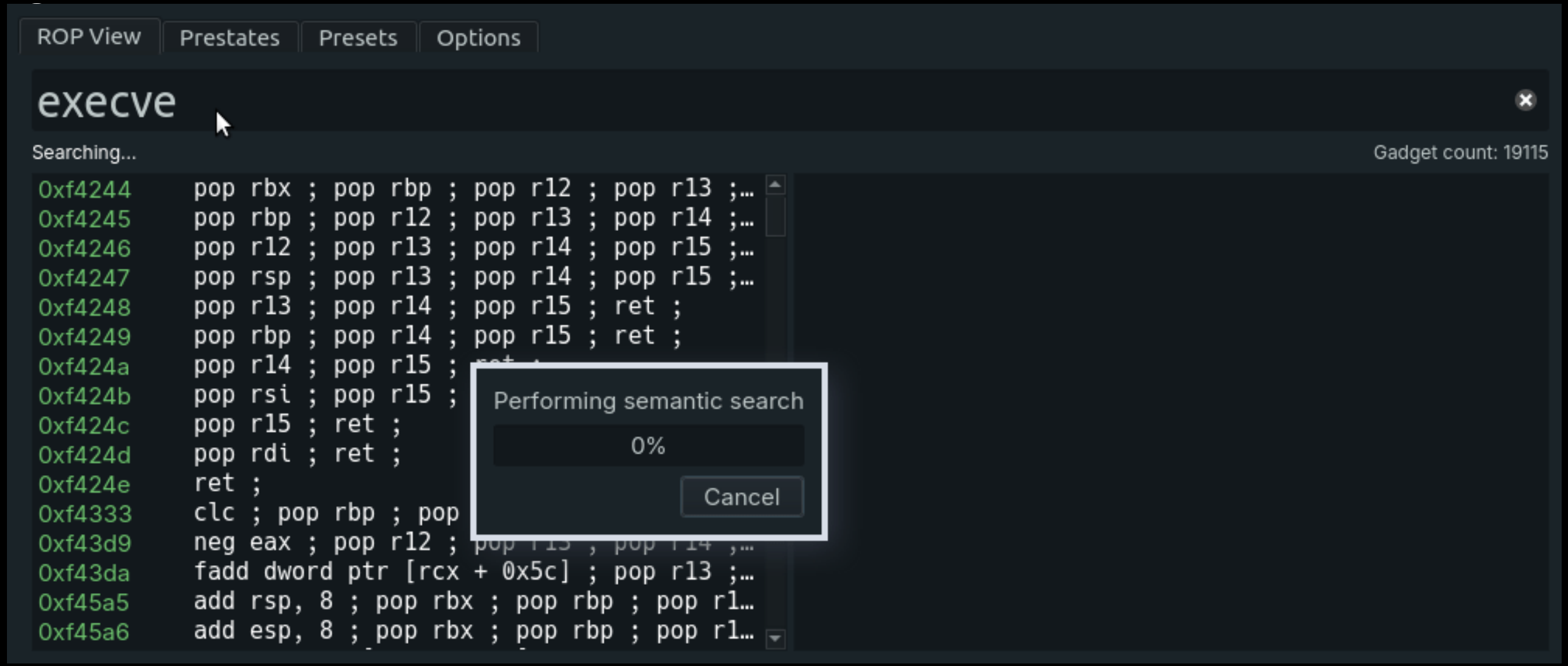
https://faultpoint.com/assets/ropview_demos/sleepa0.gif

Pop-pop-ret, control r13, avoid pivots, constrain range

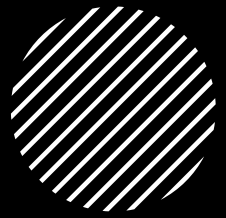


https://faultpoint.com/assets/ropview_demos/constraints.gif

execve -> (rax==0x3b or rdi==CONTROL or rdx==0 or … or sys)



https://faultpoint.com/assets/ropview_demos/execve.gif



Pitfalls

Tuning SemanticSearch parameters	SemanticMatches?
	SemanticDepth
	When do we give up?
Stack accuracy	Emulated stack uses cyclic data, lacks actual stack context
Limited by dependencies	Emulation issues /w unicorn possible

https://github.com/elbee-cyber/RopView/issues

Version 2 released!

https://github.com/elbee-cyber/RopView

Questions?