Chameleon

Polymorphic Engine

r2con2025 | Bernat – gum3t

Whoami

- Security researcher.
- Exploiting & low level stuff.

Bernat – gum3t

Chameleon | Introduction

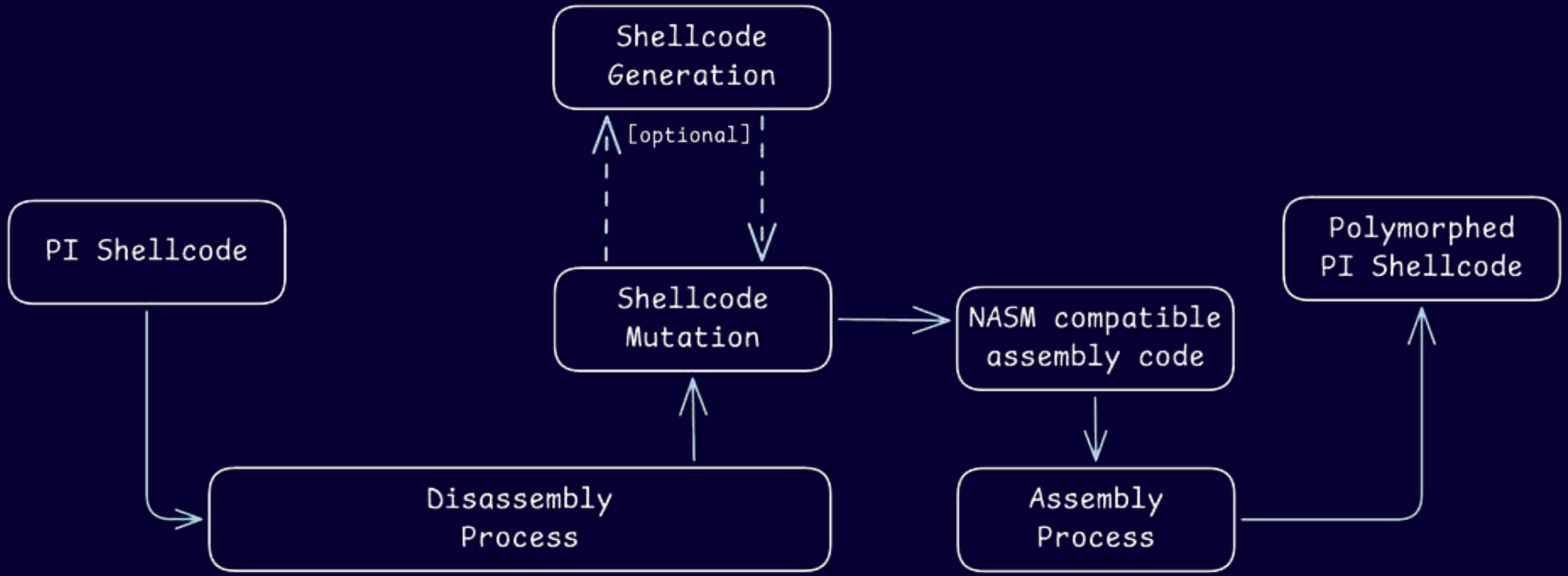
- Chameleon is a polymorphic engine for x86_64 position independent shellcode.

Position Independent Shellcode: Shellcode that executes correctly regardless of its absolute load address.

Chameleon | Requirements

- Evade/avoid signature-based detections in red team engagements.
- Don’t break the given shellcode.

Chameleon | Arquitecture Overview



Chameleon | Disassembly Process

Problems:

- x86_64 instructions have different sizes.
- Linear parsing for disassembly does not work.

Chameleon | Disassembly Process

Example:

- Given the `81 c0 01 00 ff c0 eb fc` sequence of bytes, we can see the following result:

```
┌──[gum3t@ubuntu]-[~/chameleon-polymorphicengine]
└─$ ndisasm -b 64 tmp/code.bin
00000000  81C00100FFC0      add eax,0xc0ff0001
00000006  EBFC              jmp short 0x4
```

Chameleon | Disassembly Process

```
81 c0 01 00 ff c0 eb fc
```

```
81 c0 01 00 ff c0->  add eax, 0xc0ff0001
ff c0            ->  inc eax
eb fc            ->  jmp 0x4
```

Chameleon | Disassembly Process

Solution:

- Get the disassembly data from a basic block graph (control flow graph).

Basic Block: A straight-line sequence of instructions with no branches in except to the entry and no branches out except at the exit.

Chameleon |
Disassembly Process

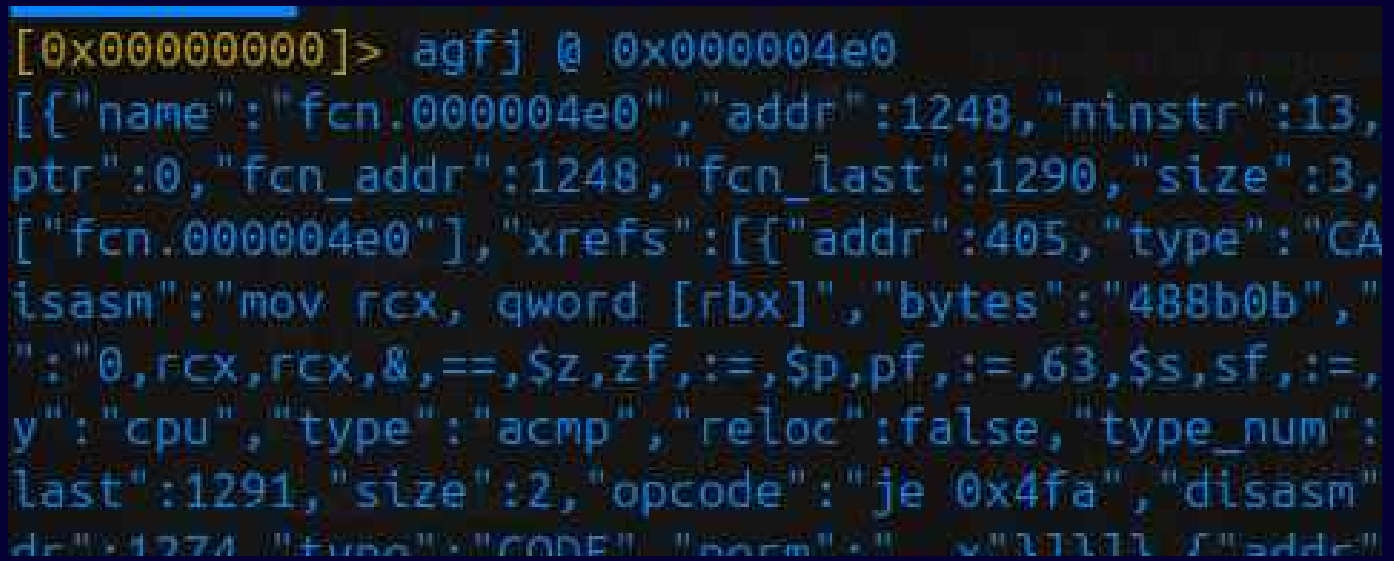
```
[0x00000000]> agf @ 0x0

  0x0
10: fcn.00000000 ();
add eax, 0xc0ff0001

  0x6
jmp 4

  0x4
;[x] CODE XREF from fcn.00000000 @ 0x6(x)
inc eax
```

Chameleon | Disassembly Process

Actual implementation:

- r2pipe – radare2 scripting API
  - aaa to analyze the shellcode.
  - aflq to get all function offsets.
  - agfj @ offset to get all basic blocks graph data in json format.

Chameleon | Disassembly Process



Chameleon | Disassembly Process

How do we turn the radare2 parsed data into valid assembly code?

Chameleon | Disassembly Process

- Each basic block gets assigned a custom asm label:

```
block_0x1000:
        ...
```

- Control flow instructions immediates are replaced by respective labels:

```
jne 0x103f ->      jne block_0x103f
```

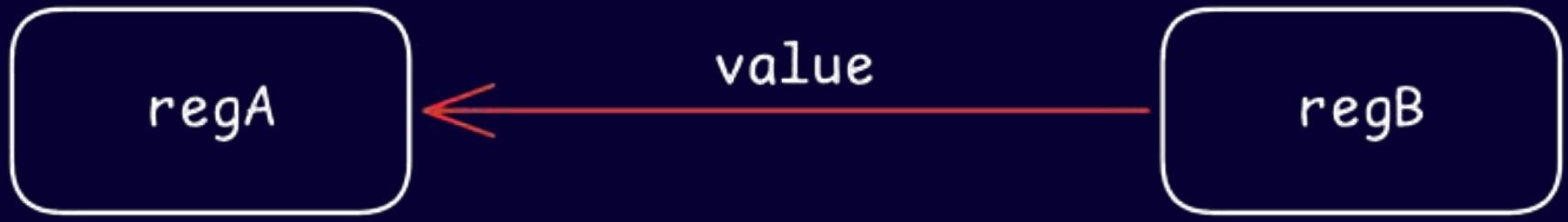
Chameleon | Disassembly Process

- Instructions may be mutated and later hardcoded via db or represented by their mnemonics and operands.

- A final enforced jmp to the fall-through block is placed at the end of the generated asm code of each basic block.

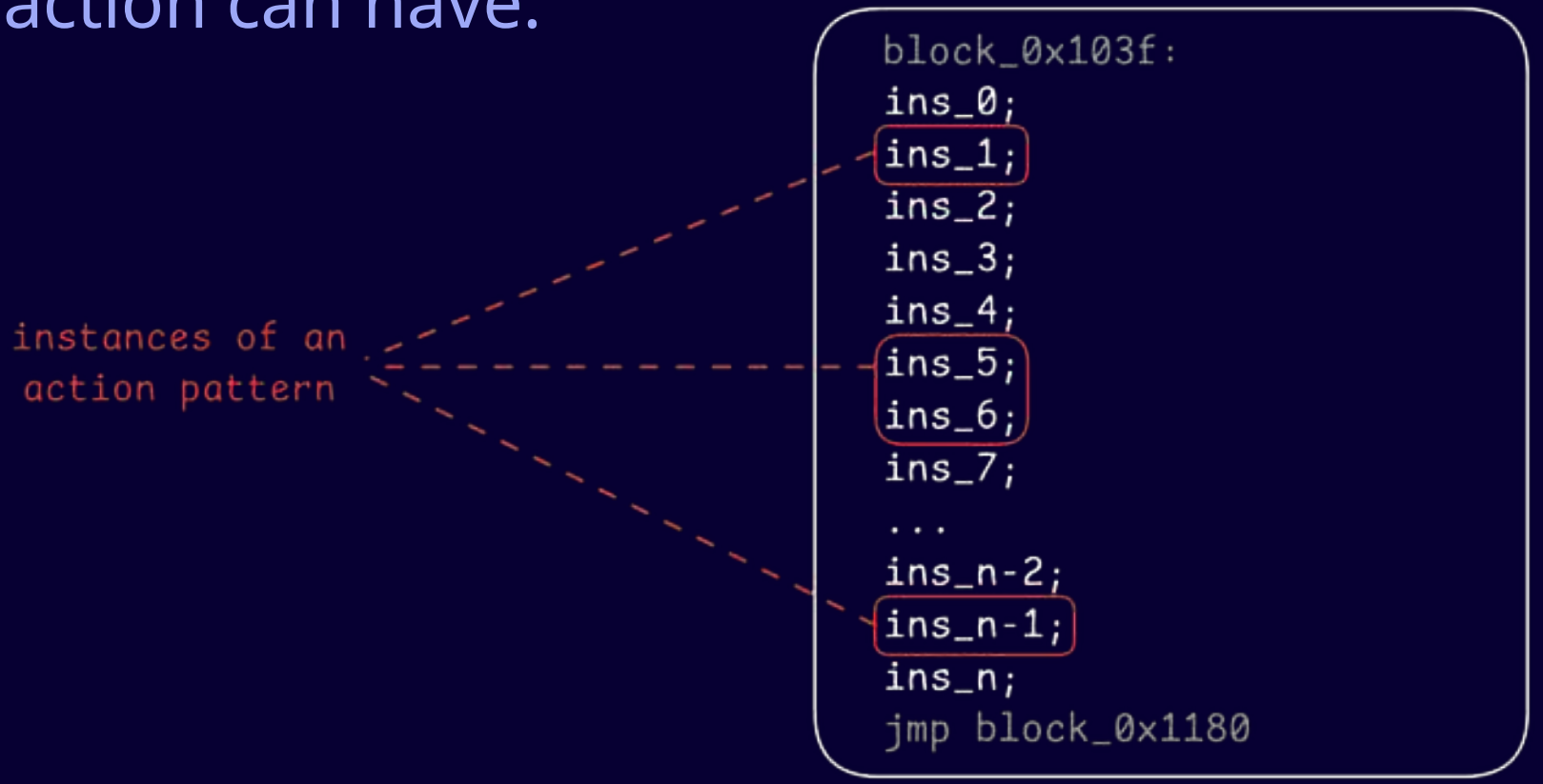
Chameleon | Shellcode Mutation

- MutationPatternPool: Focuses on the mutation of an action.



Chameleon | Shellcode Mutation

- MutationPatternPool -> match_rules:
  - Focus on the detection of the many patterns an action can have.

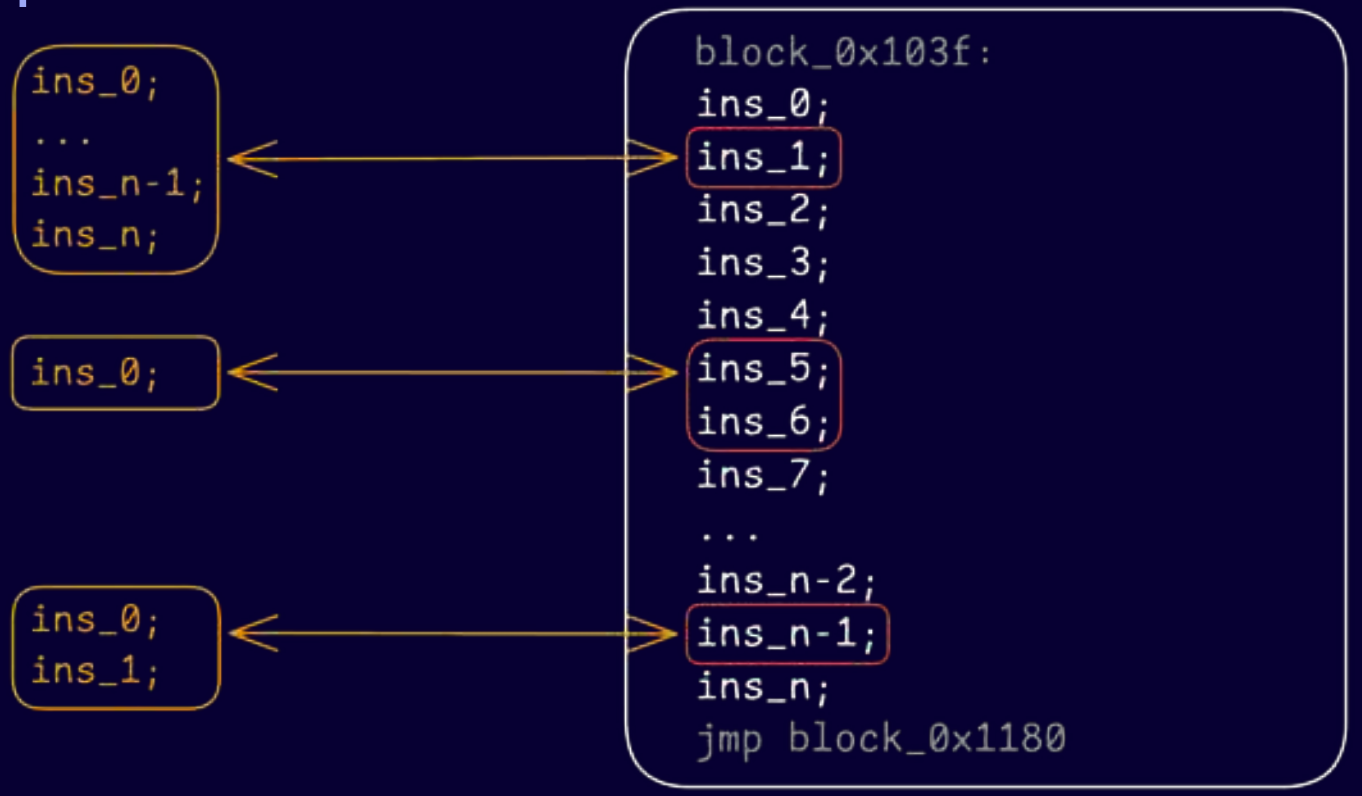


Chameleon | Shellcode Mutation

- MutationPatternPool -> mutation_probability:
  - Probability to apply a mutation once a pattern is found. It is a percentage.

Chameleon | Shellcode Mutation

- MutationPatternPool -> generators:
  - Focus on the generation of code equivalent to the found patterns.



Chameleon | Shellcode Mutation

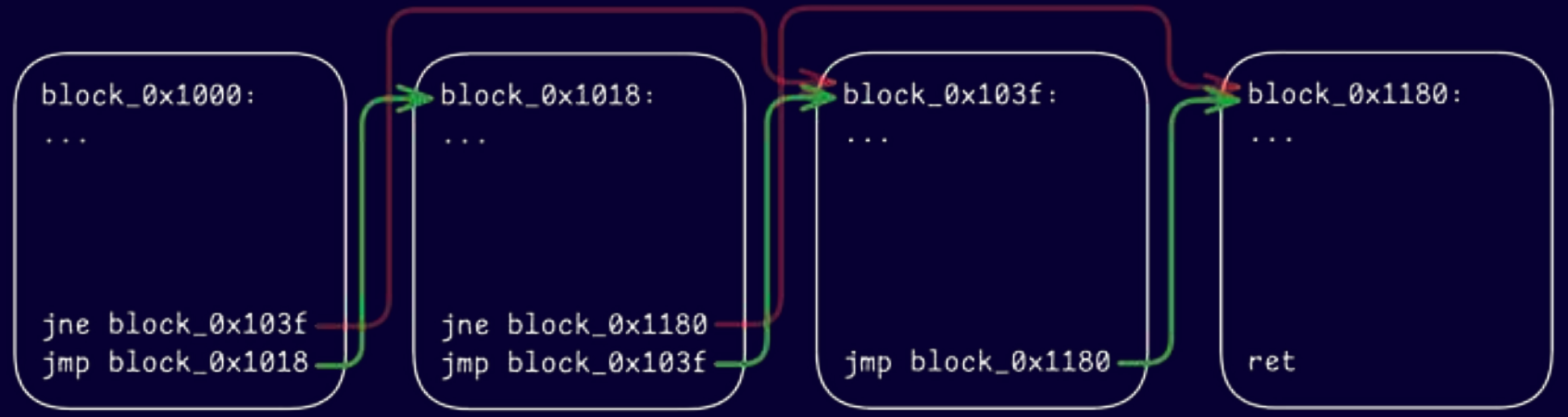
- Cook your own mutations!
  - Detect patterns by opcodes, bytes, ESIL... with your own match rules.
  - Create equivalent code with your own generators.

Chameleon | Shellcode Mutation

- Remember the enforced jmp we add at the end of each basic block?

- Now this allows us to shuffle basic blocks without breaking the control flow!

Chameleon | Shellcode Mutation



Before shuffling blocks

After shuffling blocks

Chameleon | Shellcode Mutation

- After this process, enforced jmp instructions that are redundant can be removed.

Chameleon | Shellcode Generation

- Mutation generators can make use of a code generation feature.

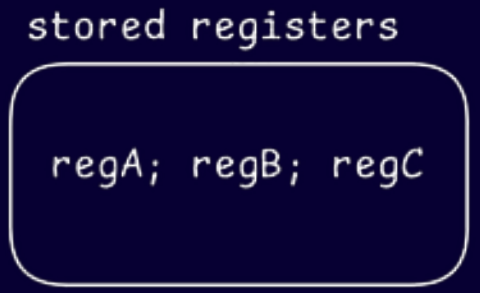
- So, what is it, and how does it work?

Chameleon | Shellcode Generation

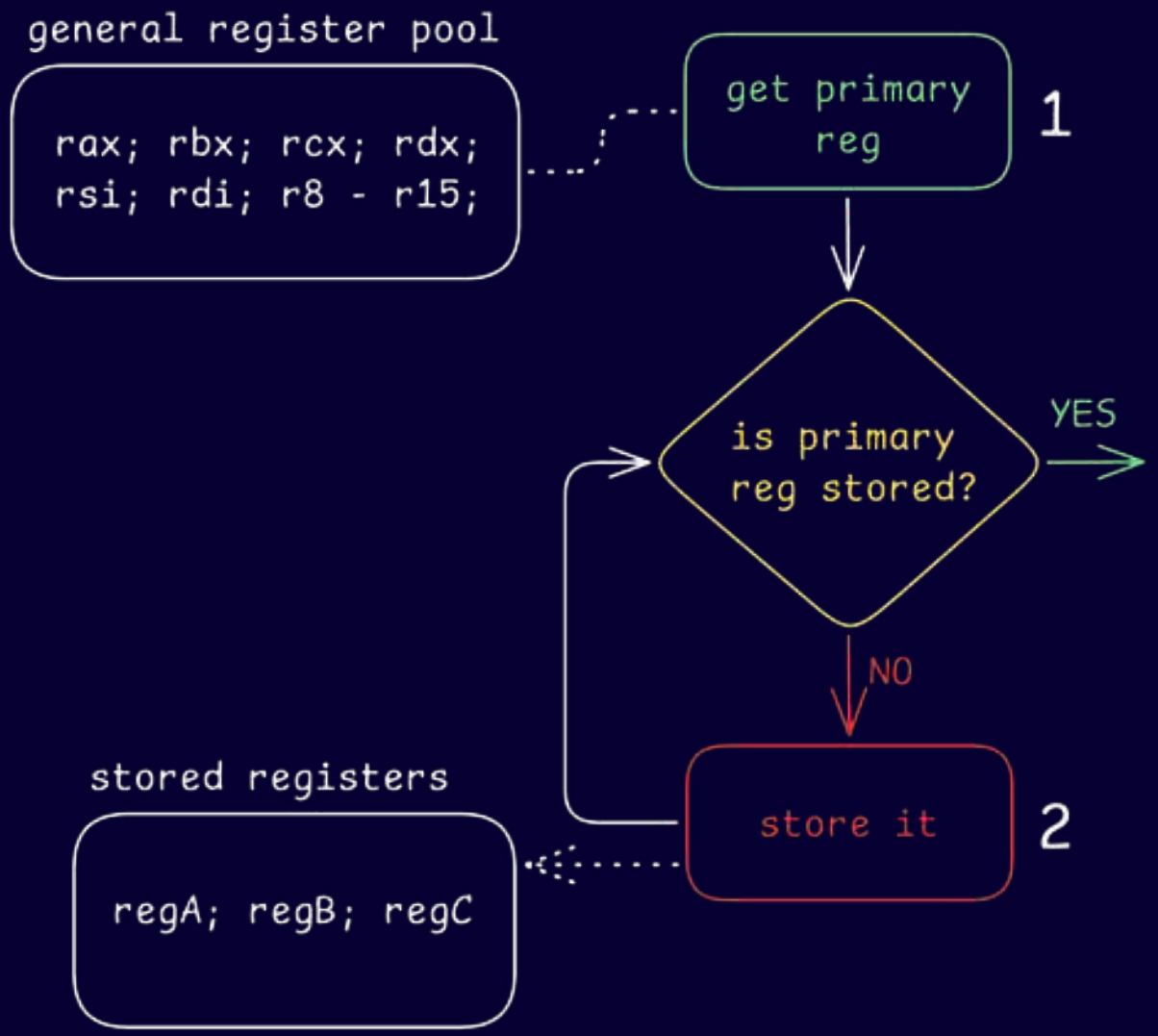
- Chameleon’s code generator is a feature that is capable of creating semantically neutral shellcode blobs that can include nested branches and loops.

Semantically neutral code: Code that changes the program appearance but not its behaviour.

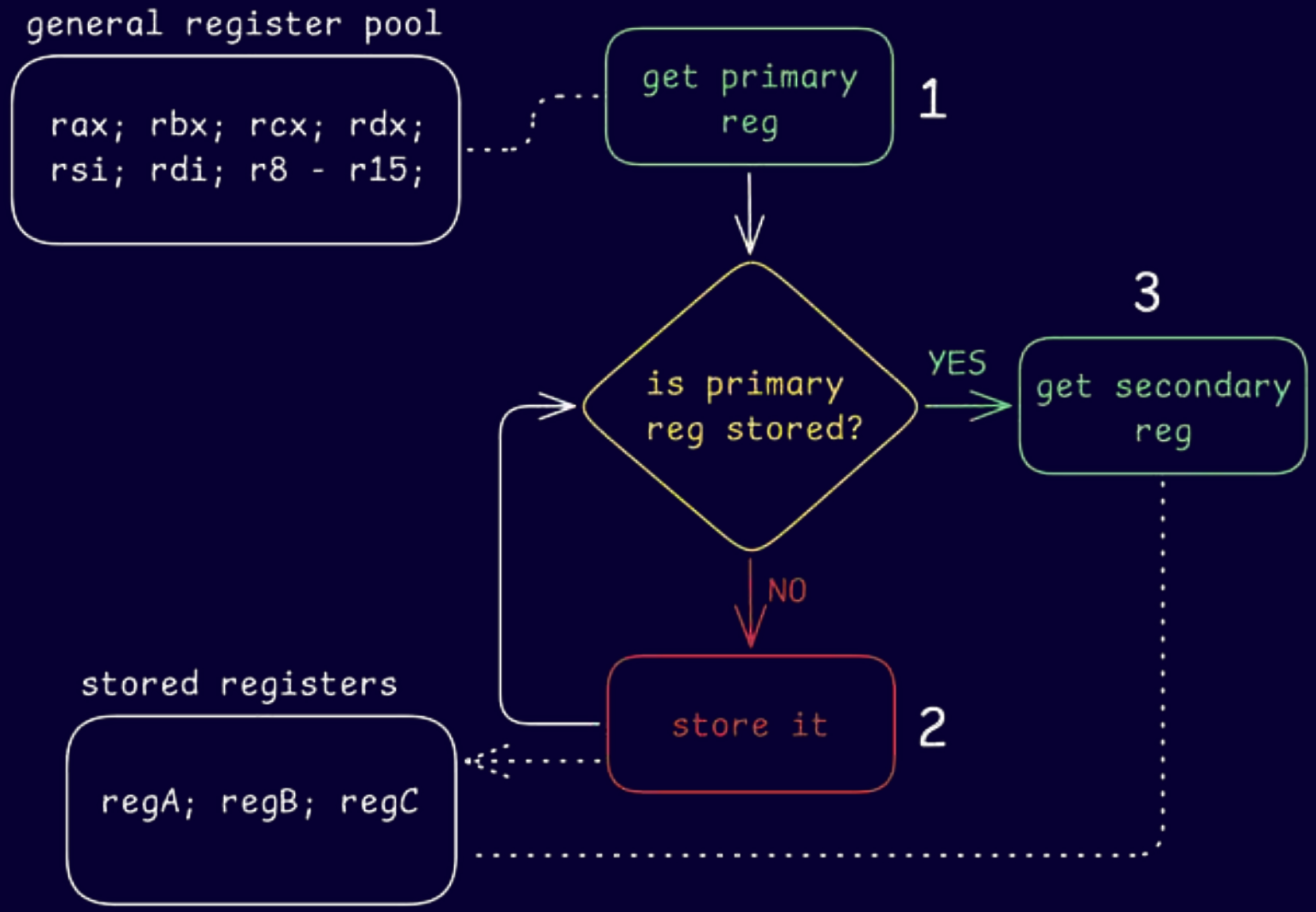
Chameleon | Shellcode Generation



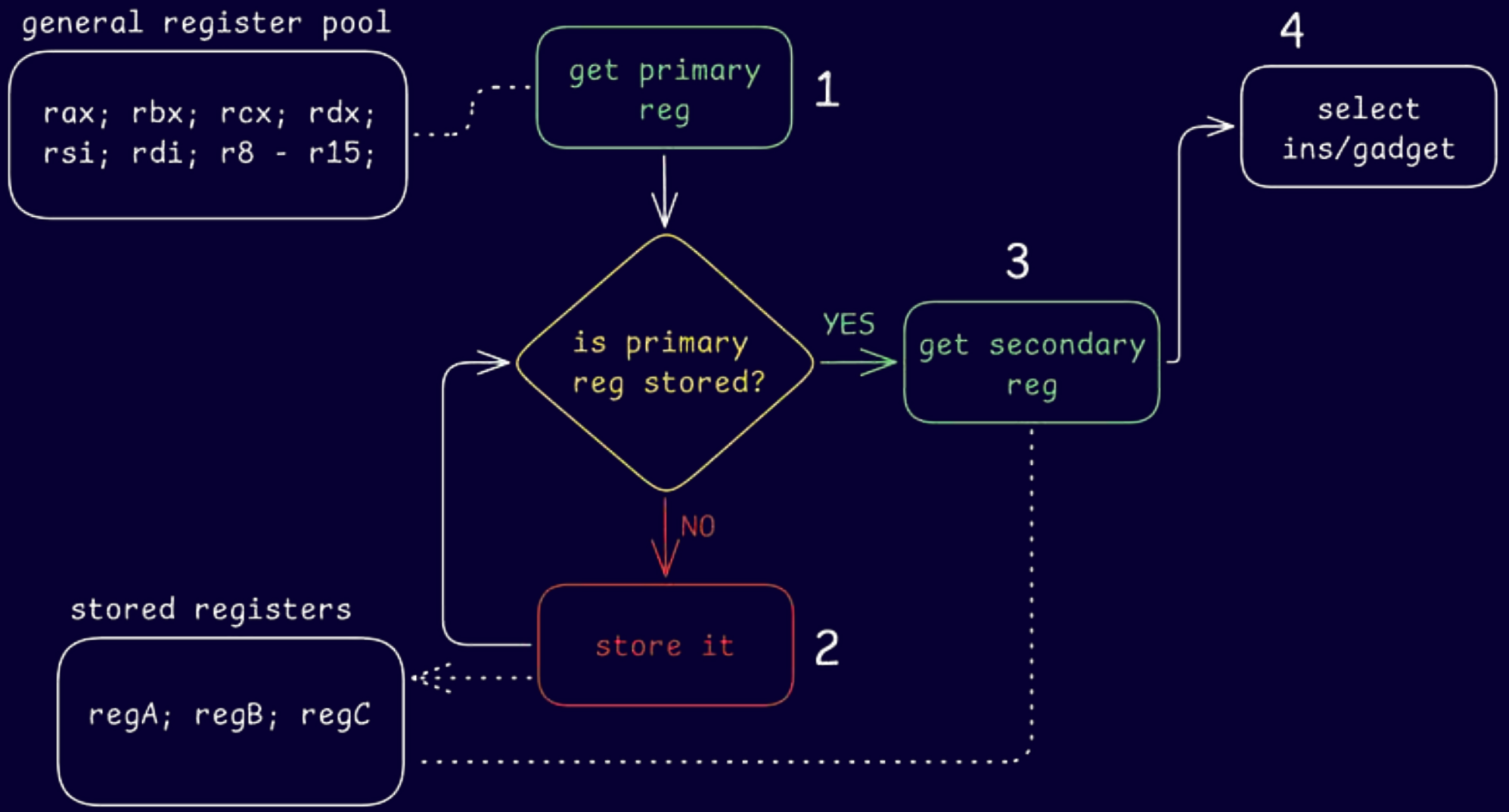
Chameleon | Shellcode Generation



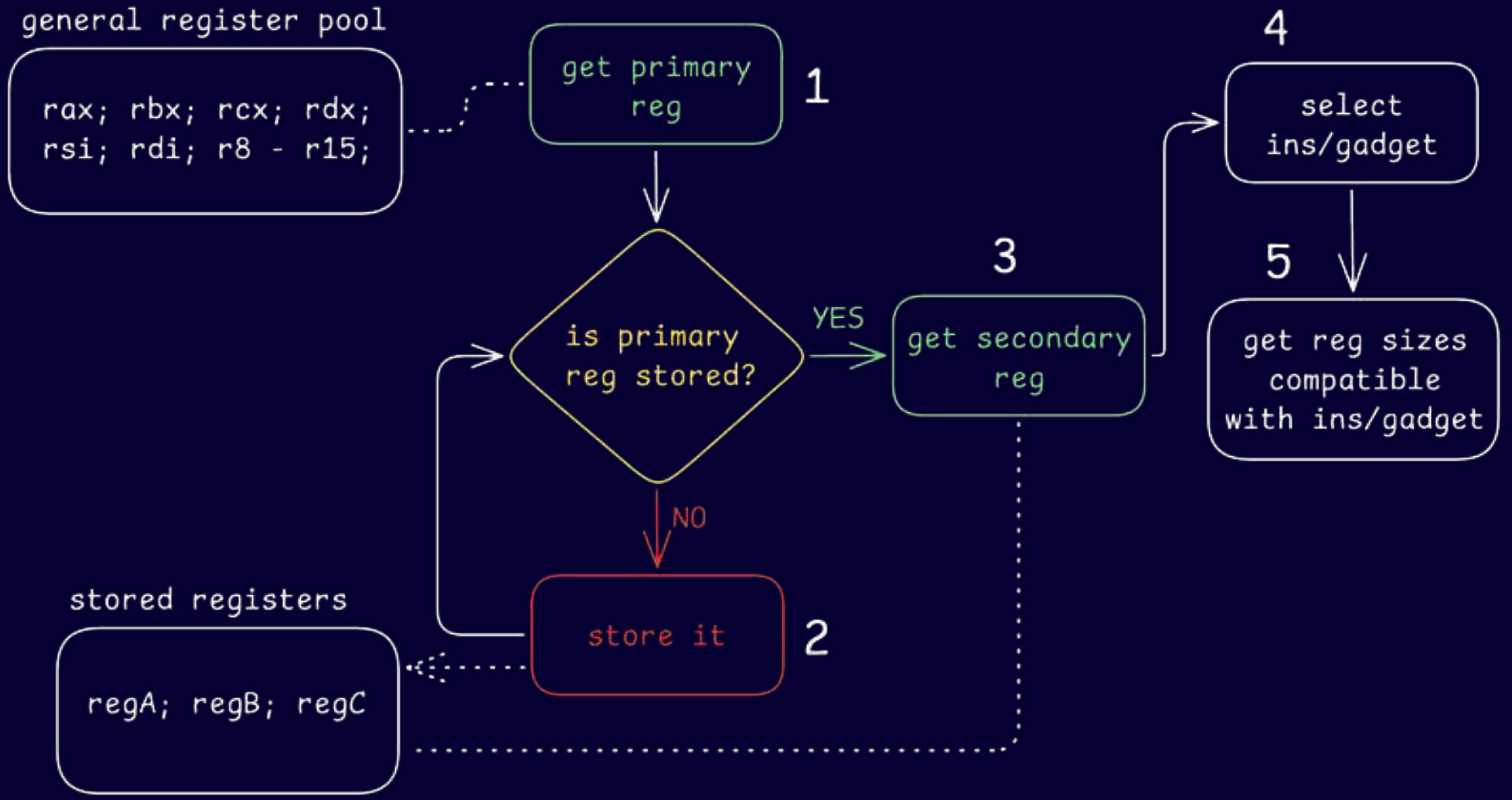
Chameleon | Shellcode Generation



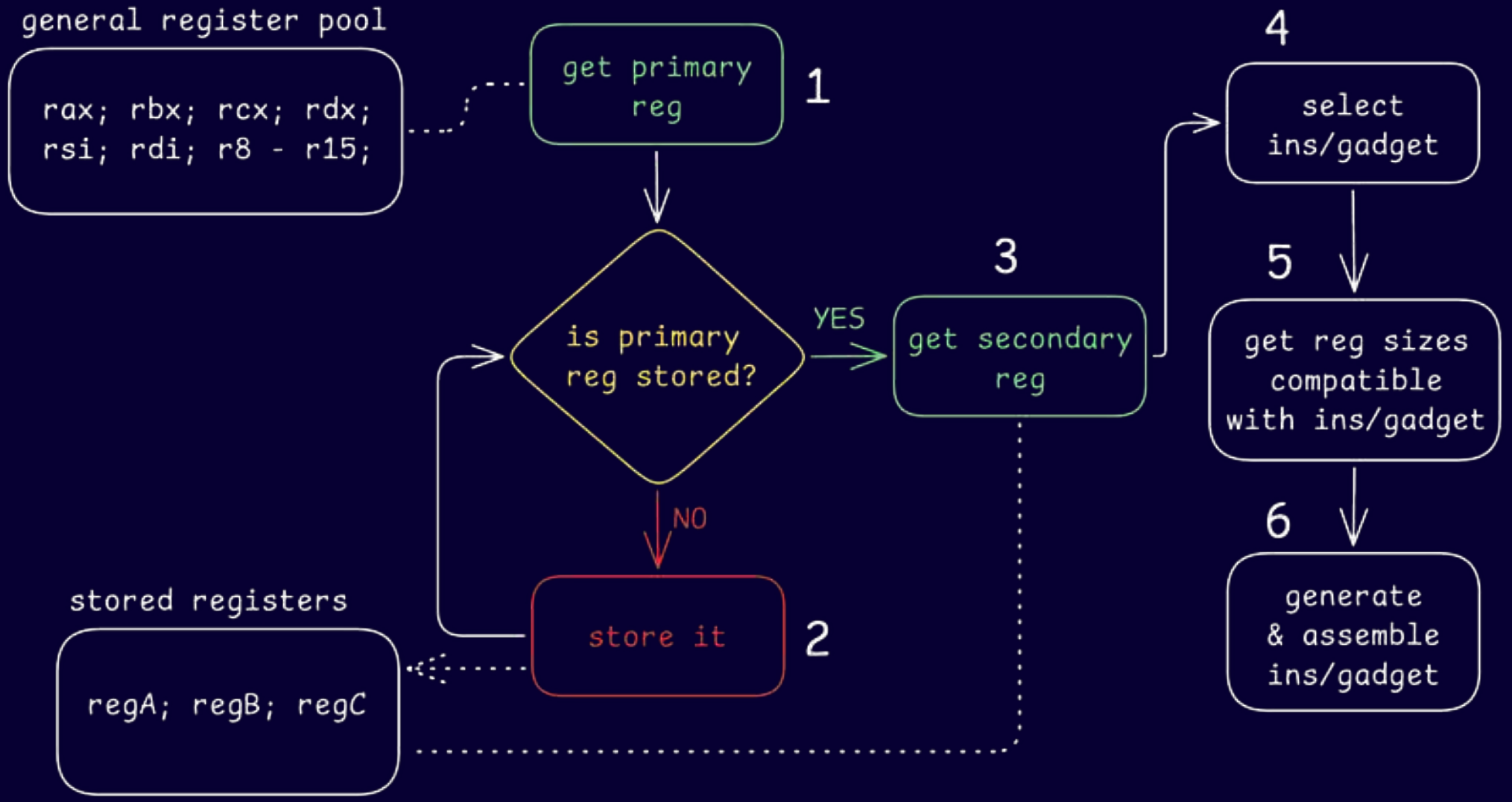
Chameleon | Shellcode Generation



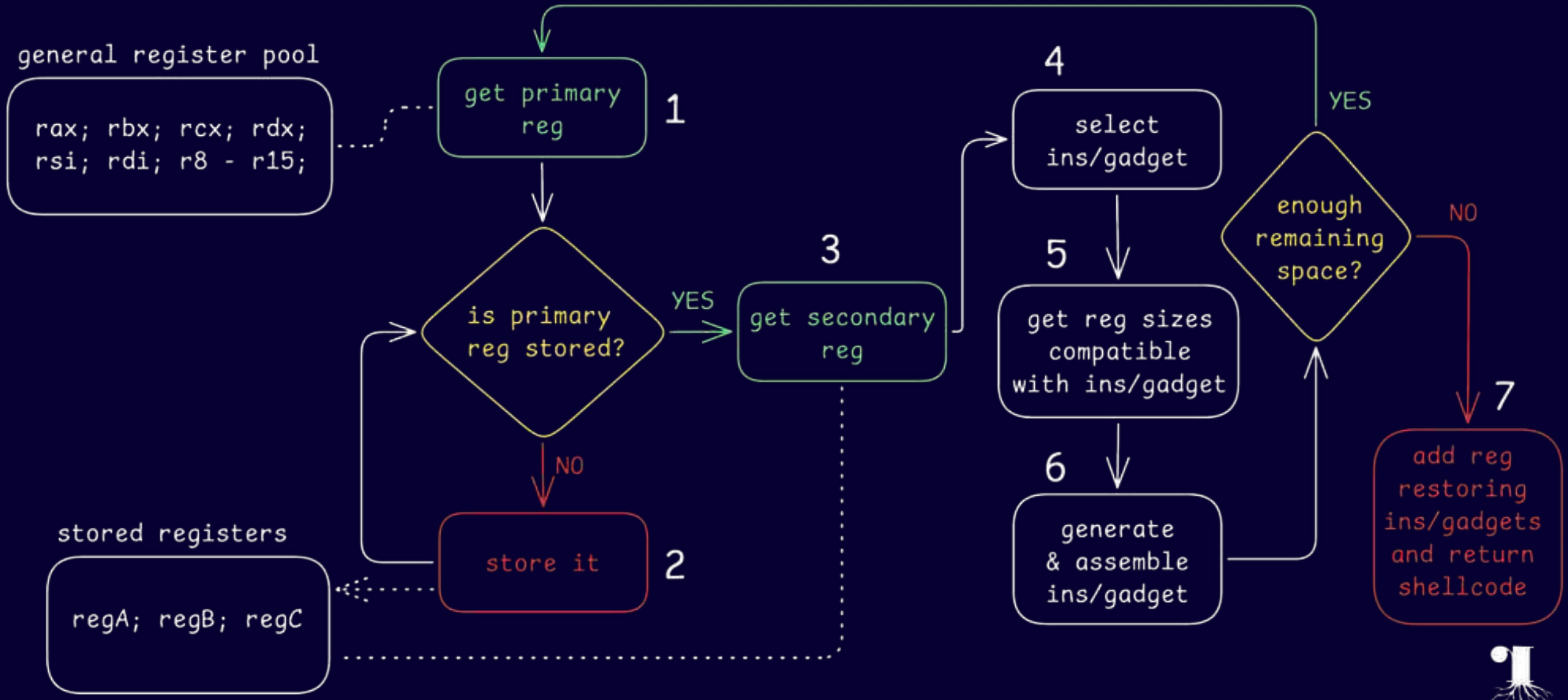
Chameleon | Shellcode Generation



Chameleon | Shellcode Generation



Chameleon | Shellcode Generation



Chameleon | Shellcode Generation

- Instruction implementation example:

```
( lambda reg, sec_reg: f”mov {reg}, rsp;”, REG_64, 1 )
```

Chameleon | Shellcode Generation

- Gadget implementation example:

```
( self.br_check_regs, REG_ALL, 1 )
def br_check_regs(self, reg: str, sec_reg: str) ->
str:
        …
        gadget += f“cmp {reg}, {sec_reg};”
        gadget += pseudo random cjmp gadget to label
        gadget += get_n_junk_ins(reg, sec_reg, n)
        gadget += label
        …
```

Chameleon | Shellcode Generation

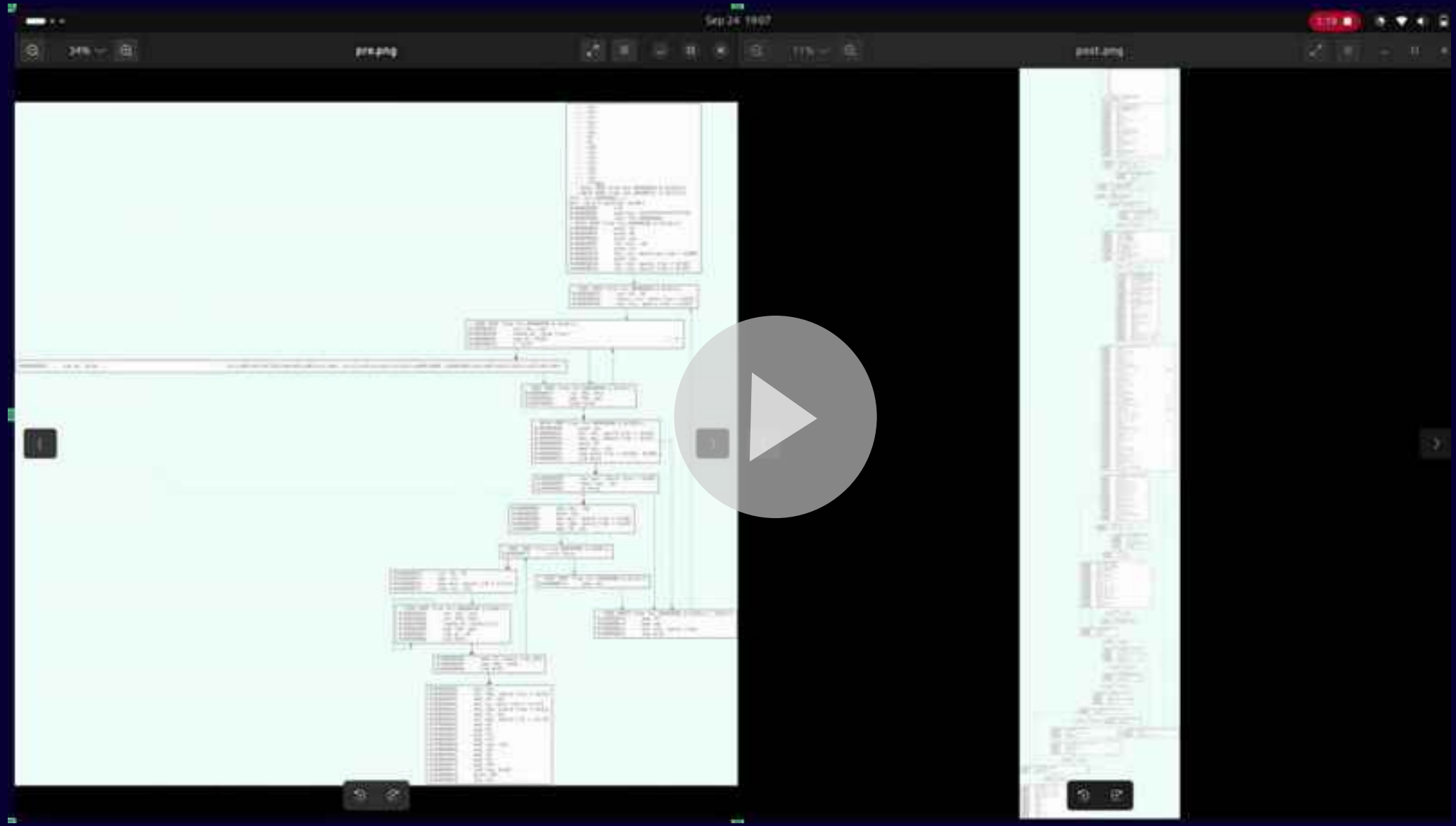
- The generated shellcode is later hardcoded into the block asm using db.

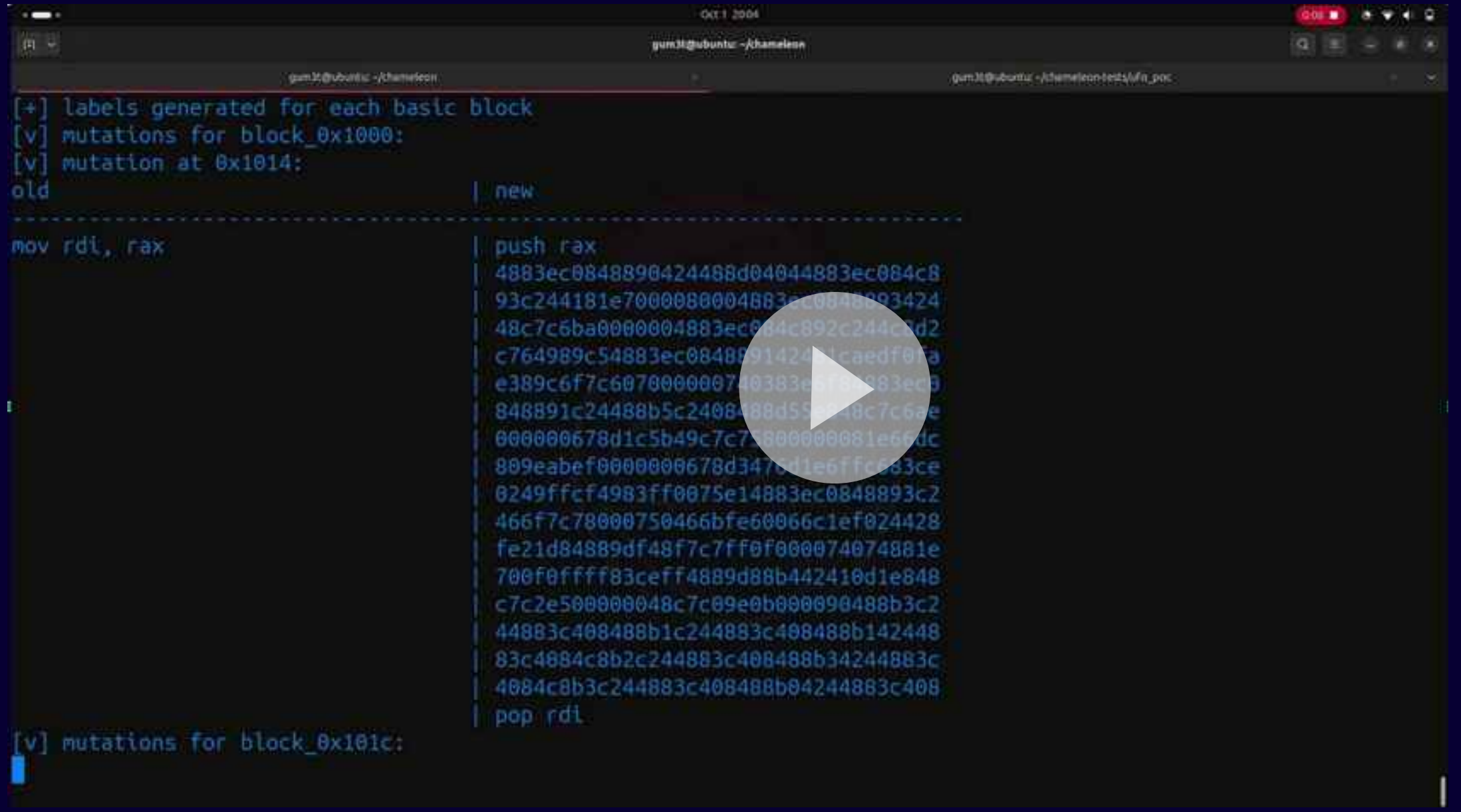
Chameleon | Assembly Process

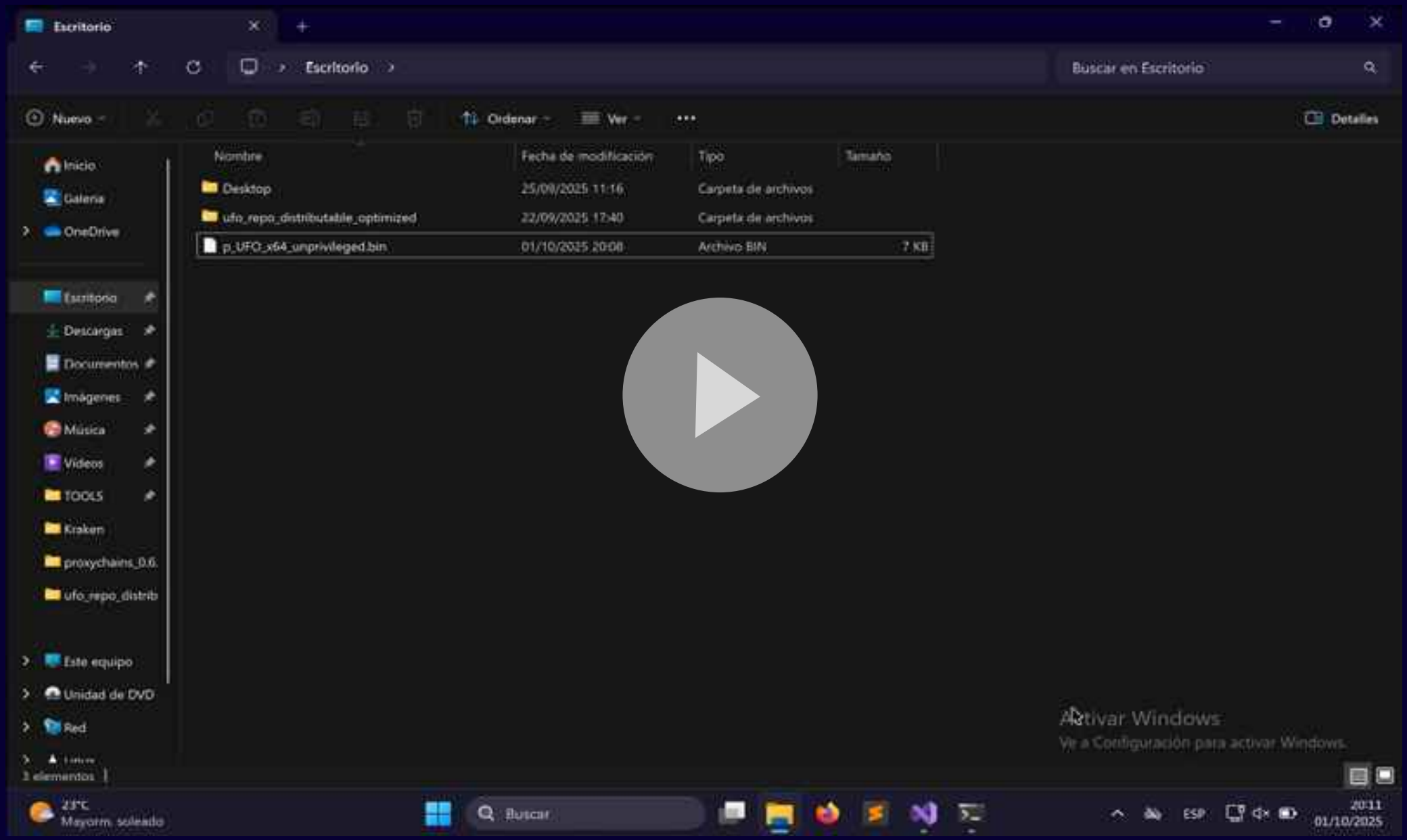
- Nasm compatible asm header:

```
BITS 64
default rel
global _start
section .text
_start:
mutated_sc
```

```
nasm –f bin file.asm –o output.bin
```


Sep 24 19:07
pre.png
post.png


Oct 1 20:04
gum3t@ubuntu: ~/chameleon
gum3t@ubuntu: ~/chameleon
gum3t@ubuntu: ~/chameleon-tests/ufo_poc
[+] labels generated for each basic block
[v] mutations for block_0x1000:
[v] mutation at 0x1014:
old | new
mov rdi, rax | push rax
| 4883ec0848890424488d04044883ec084c8
| 93c244181e70000800048 3ec0848893424
| 48c7c6ba0000004883ec0 4c892c244c8d2
| c764989c54883ec0848891424 caedf0fa
| e389c6f7c607000000 0383e 4883ec0
| 848891c24488b5c2408 88d55 8c7c6ae
| 000000678d1c5b49c7c7 0000081e66dc
| 809eabef0000000678d347 d1e6ffc083ce
| 0249ffcf4983ff0075e14883ec0848893c2
| 466f7c78000750466bfe60066c1ef024428
| fe21d84889df48f7c7ff0f000074074881e
| 700f0ffff83ceff4889d88b442410d1e848
| c7c2e500000048c7c09e0b000090488b3c2
| 44883c408488b1c244883c408488b142448
| 83c4084c8b2c244883c408488b34244883c
| 4084c8b3c244883c408488b04244883c408
| pop rdi
[v] mutations for block_0x101c:


Escritorio
Escritorio
Buscar en Escritorio
Nuevo
Ordenar
Ver
Detalles
Inicio
Galería
OneDrive
Escritorio
Descargas
Documentos
Imágenes
Música
Videos
TOOLS
Kraken
proxychains_0.6.
ufo_repo_distrib
Este equipo
Unidad de DVD
Red
3 elementos
Nombre
Fecha de modificación
Tipo
Tamaño
Desktop
25/09/2025 11:16
Carpeta de archivos
ufo_repo_distributable_optimized
22/09/2025 17:40
Carpeta de archivos
p_UFO_x64_unprivileged.bin
01/10/2025 20:08
Archivo BIN
7 KB
Activar Windows
Ve a Configuración para activar Windows.
23°C
Mayorm. soleado
Buscar
ESP
20:11
01/10/2025

Any questions?

https://github.com/gum3t/chameleon