

In Memory of In-Memory Detection

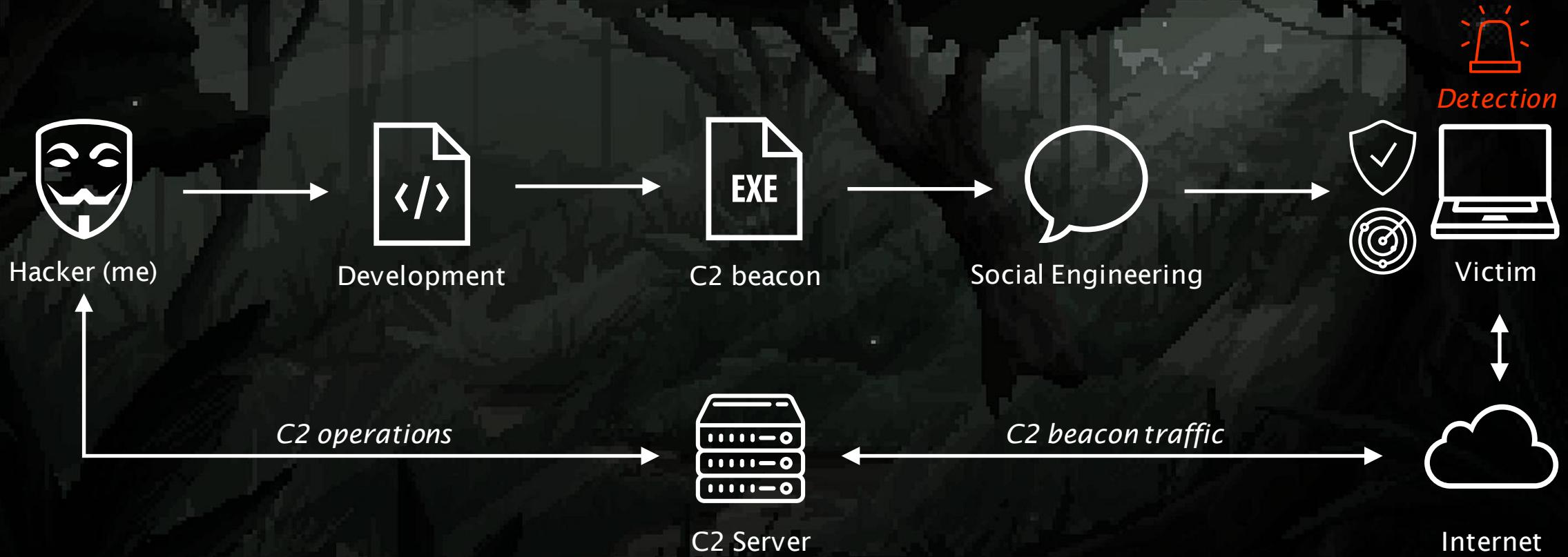
from polymorphic to metamorphic



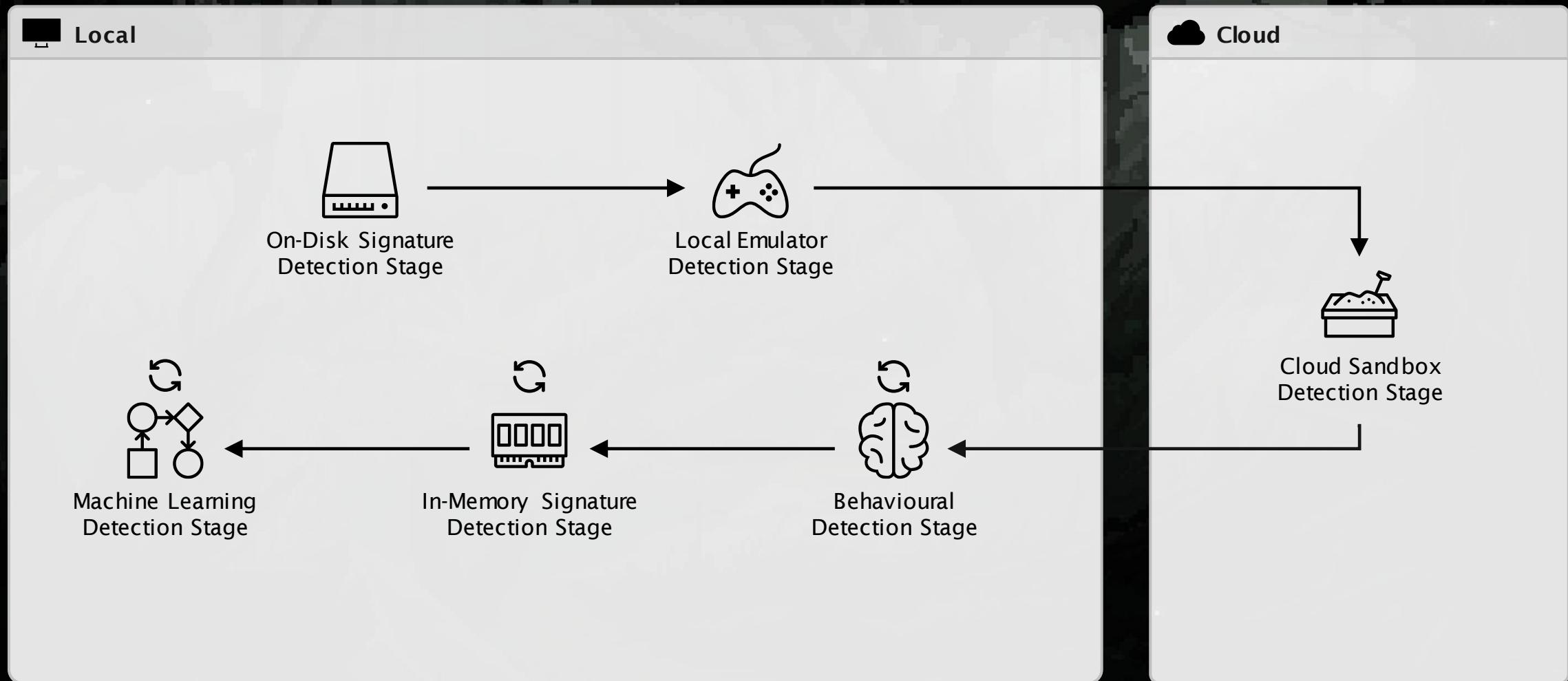
About Tijme (me)

- Offensive Cyber @ ABN AMRO Bank (Netherlands)
- Previously Red Teaming @ Northwave (Netherlands)
- Author of exploits & malwarez
- Socials username is @tijme
[Bluesky](#) - [GitHub](#) - [LinkedIn](#) - [Twitter](#)

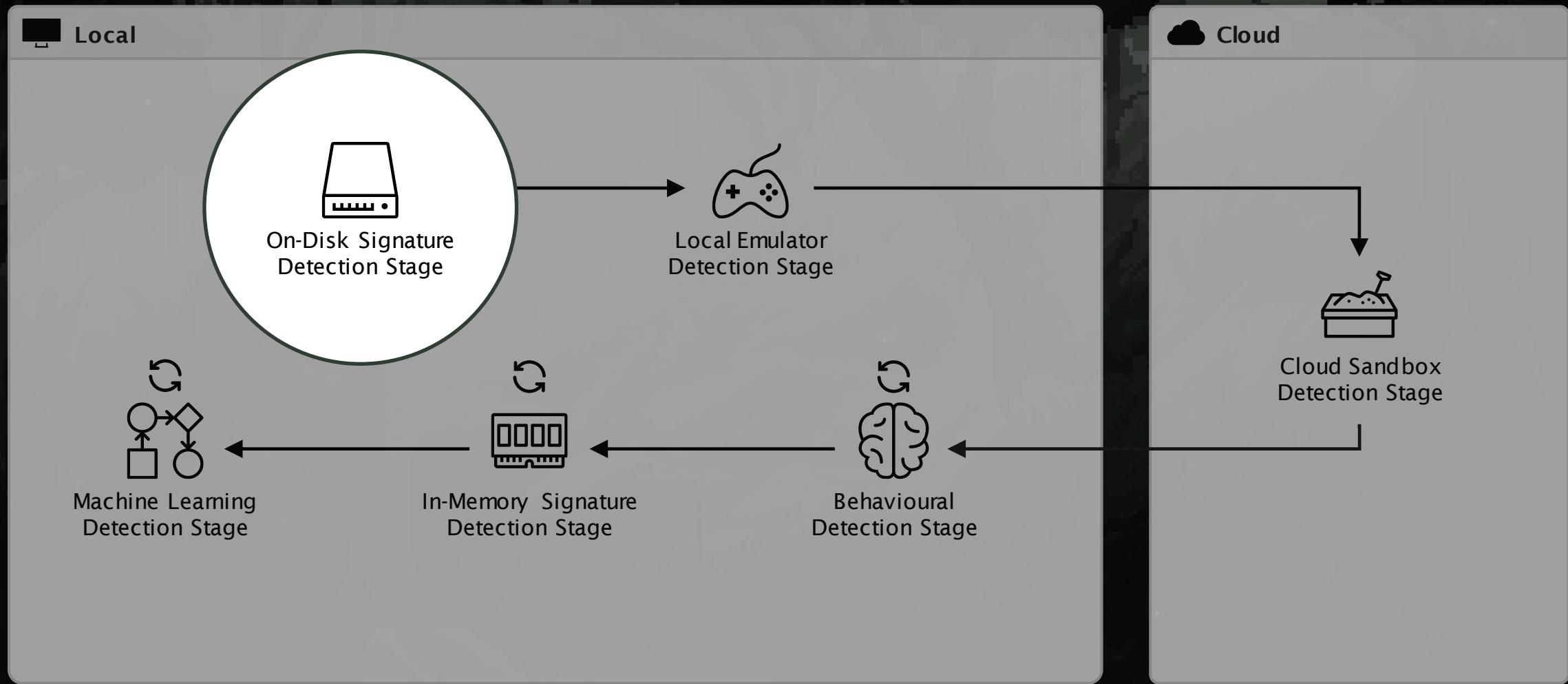
Malware



Detection stages



Detection stages



Talk outline

1. Polymorphic malware

Often found in malware antique stores

2. Metamorphic malware

Often seen on malware buzzword bingo cards

3. Dittobytes project

A true metamorphic cross-compiler

❖ Demo

Packer

The screenshot shows a GitHub repository page for 'elf-packer' by telepath9000. The repository is public and has 6 forks and 32 stars. The 'Code' tab is selected, showing the master branch with 1 commit. The commit details are as follows:

File	Change	Date
include	renamed fill... to inject_payload	4 months ago
src	changed encryption iteration to 64 bit	4 months ago
.gitignore	added compiledb.json to gitignore	6 years ago
LICENSE	complete rewrite	7 years ago
Makefile	fixed segfault in prepare_elf	4 months ago
README.md	Update README.md	6 years ago

The repository also contains a README and an MIT license file. The 'About' section describes the tool as encrypting 64-bit elf files that decrypt at runtime. It includes tags for linux, encryption, binary-analysis, elf-parser, self-modifying-code, injection-attacks, elf64, elf-binaries, and elf-format. The repository has 32 stars, 2 watchers, and 6 forks.

Packer

Your original code

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char** argv) {
    printf("Hello world");

    WinExec("whoami");

    return 0;
}
```

packed.exe

Packer

Your original code

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char** argv) {
    printf("Hello world");

    WinExec("whoami");

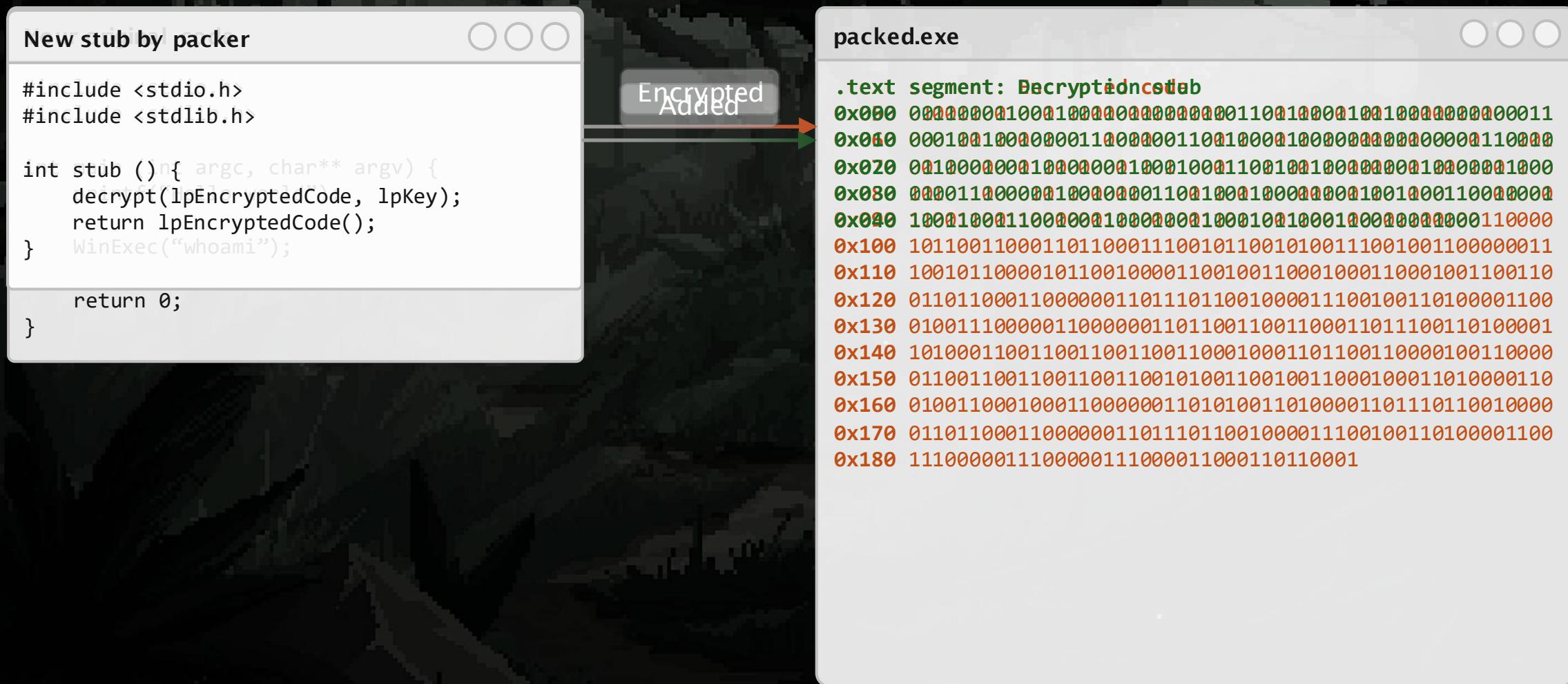
    return 0;
}
```

Encrypted

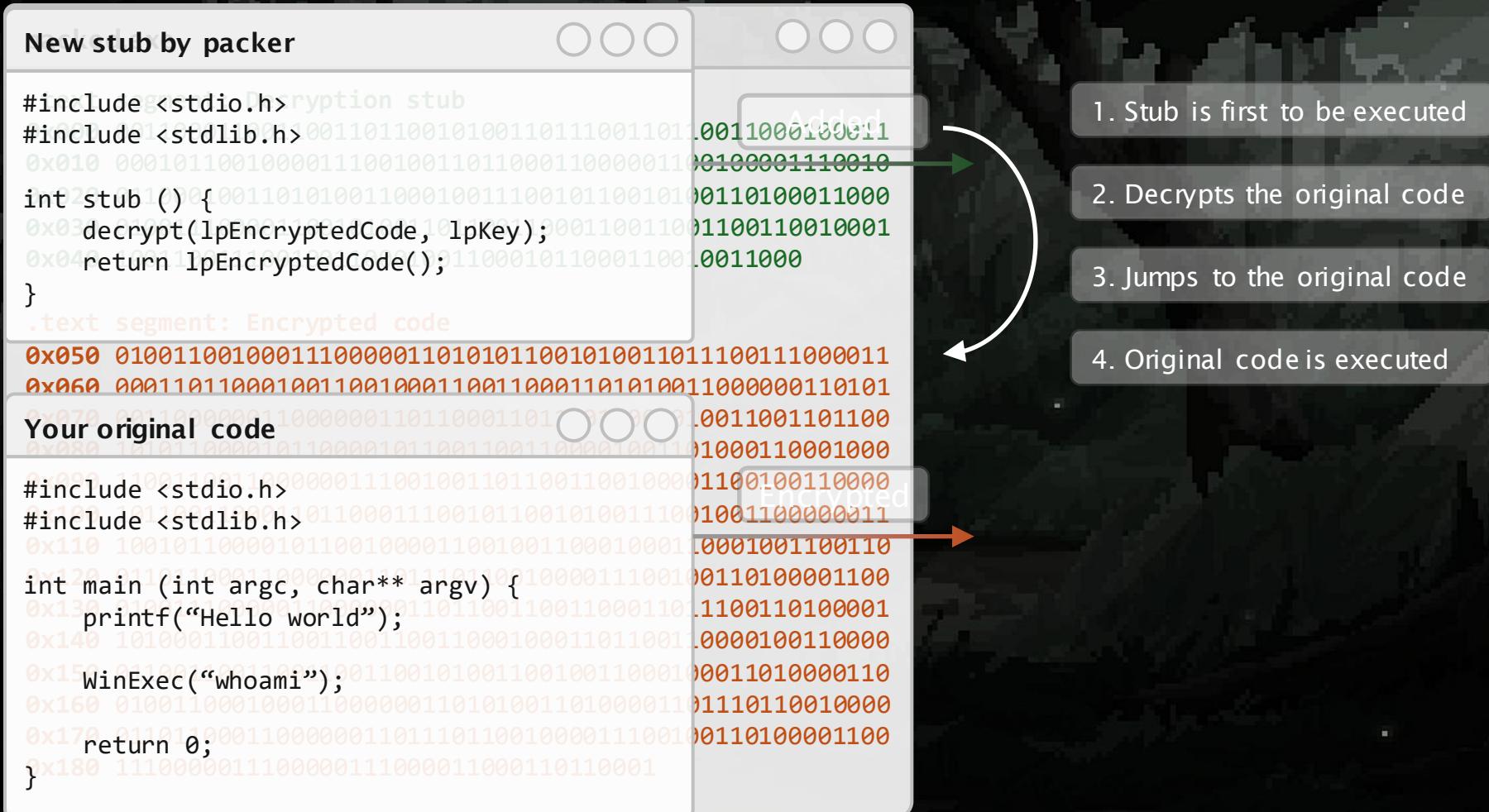
packed.exe

```
.text segment: Encrypted code
0x050 0100110010001110000011010101100101001101100111000011
0x060 000110110001001100100011000110001101010011000000110101
0x070 00110000001100000011011000110111011001010011000000110101
0x080 1010110000101100001011001100110000100110100011000100
0x090 110011001100000011100100110110011001000110010011000
0x0A0 10110011000110110001110010110010110010011100100110000011
0x0B0 10010110000101100100001100100110001000110010010011001100
0x0C0 0110110001100000011011101100100001110010011011001101000
0x0D0 01001110000011000000110110011001100110011011100110100001
0x0E0 101000110011001100110001000110110011000010011000
0x0F0 011001100110011001100101001100100011000100011010000110
0x100 010011000100011000000110101001010011001000110111011001000
0x110 01101100011000000110111011001000011100100110110011000
0x120 01101100011000000110111011001000011100100110110001100
0x130 0100111000001100000011011001100110011011100110100001
0x140 101000110011001100110001000110110011000010011000
0x150 011001100110011001100101001100100011000100011010000110
0x160 01001100010001100000011010100101000110111011001000
0x170 0110110001100000011011101100100001110010011011001000
0x180 11100000111000001110000110001100011011001100011000110001
```

Packer



Packer



Polymorphic stub

Sample 1

```
.text segment: Decryption stub  
0x000 00110001100110011011001010011011100110110011000100011  
0x010 000101100100001110010011011000110000001100100001110010  
0x020 011000100110100110001001110010110010100110100011000  
0x030 010011100001100100110110011000110011001100110010001  
0x040 100110011100100110001011000110010011000
```

Sample 2

```
.text segment: Decryption stub  
0x000 00110001100110011011001010011011100110110011000100011  
0x010 000101100100001110010011011000110000001100100001110010  
0x020 0110001001101001001100010011100101100101001100100011000  
0x030 0100111000011001001101100110001100110011001100110010001  
0x040 100110011100100110001011000110010011000110010011000
```

In both examples, the stub is static (signatureable) each compilation.

```
0x050 01001100100011100000110101011001010011011100111000011  
0x060 000110110001001100100011001100011010011000000110101  
0x070 00110000001100000011011000110111011001010011001101100  
0x080 1010110000101100001011001100001001101000110001000  
0x090 1100110011000000110010011011001100100001100100110000  
0x100 10110011000011011000111001011001010011100100110000011  
0x110 1001011000010110010000110010011000100011001001100110  
0x120 01101100011000000110111011001000011100100110100001100  
0x130 0100111000001100000011011001100011011100110100001  
0x140 1010001100110011001100010001101100110000100110000  
0x150 0110011001100110011001010011000100011010000110  
0x160 01001100010001100000011010100110100001101110110010000  
0x170 01101100011000000110111011001000011100100110100001100  
0x180 111000001110000011000110001101100010001
```

```
0x050 011000110011100100010001011000000101000100000001000000  
0x060 00100000010000001000000010000000100000001000000010000000100  
0x070 000001000000010000000010000000100010011010110110010101  
0x080 11100101011111011011110111100001110011001000100011101  
0x090 000100000010110110000101000100000010000000100000001000000010  
0x100 00000010000000100000001000000010000000100000001000000010000000  
0x110 0100000001000000010000000100000001000000010000000100000001000  
0x120 10011101100110010101110010011010010110011001111001001  
0x130 00010000010100010000000100000001000000010000000100000001000000  
0x140 001000000010000001000000010000000100000001000000010000000100  
0x150 00001011101001011000001010001000000100000001000000010000000  
0x160 100000001000000010000000100000001000000010000000100000001000000  
0x170 0001000000010000000100000001000100110101101001011001000010  
0x180 00100011101000100000001000100101010100001110101011000100
```

Ultimate polymorphic form

The screenshot shows a GitHub repository page for "Simple Polymorphic Engine — SPE32". The README section contains the following text:

Simple Polymorphic Engine (SPE32) is a simple polymorphic engine for encrypting code and data.

SPE32 allows you to encrypt any data and generate a unique decryption code for this data.
The encryption algorithm uses randomly selected instructions and encryption keys.

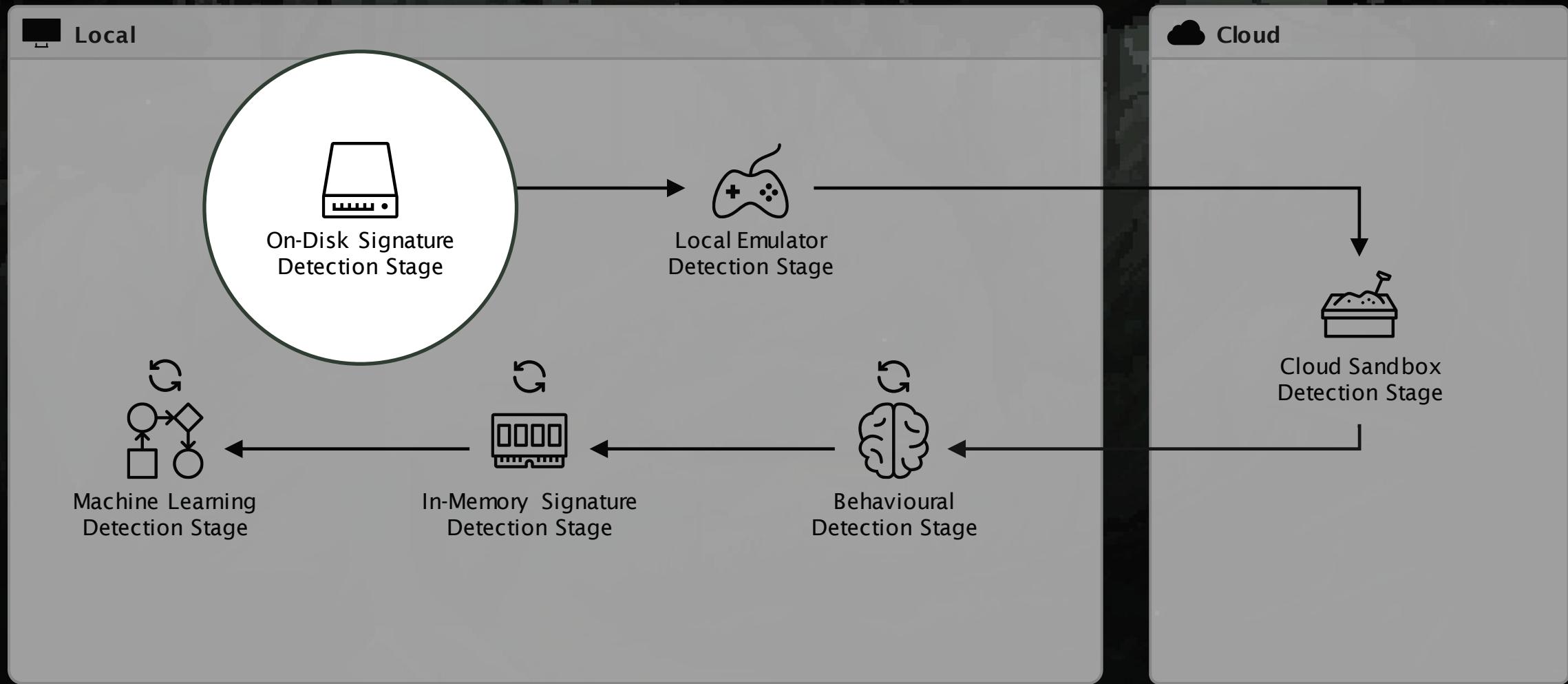
The generated decryption code will be different each time.

Polymorphic decryption code as viewed in x86dbg debugger

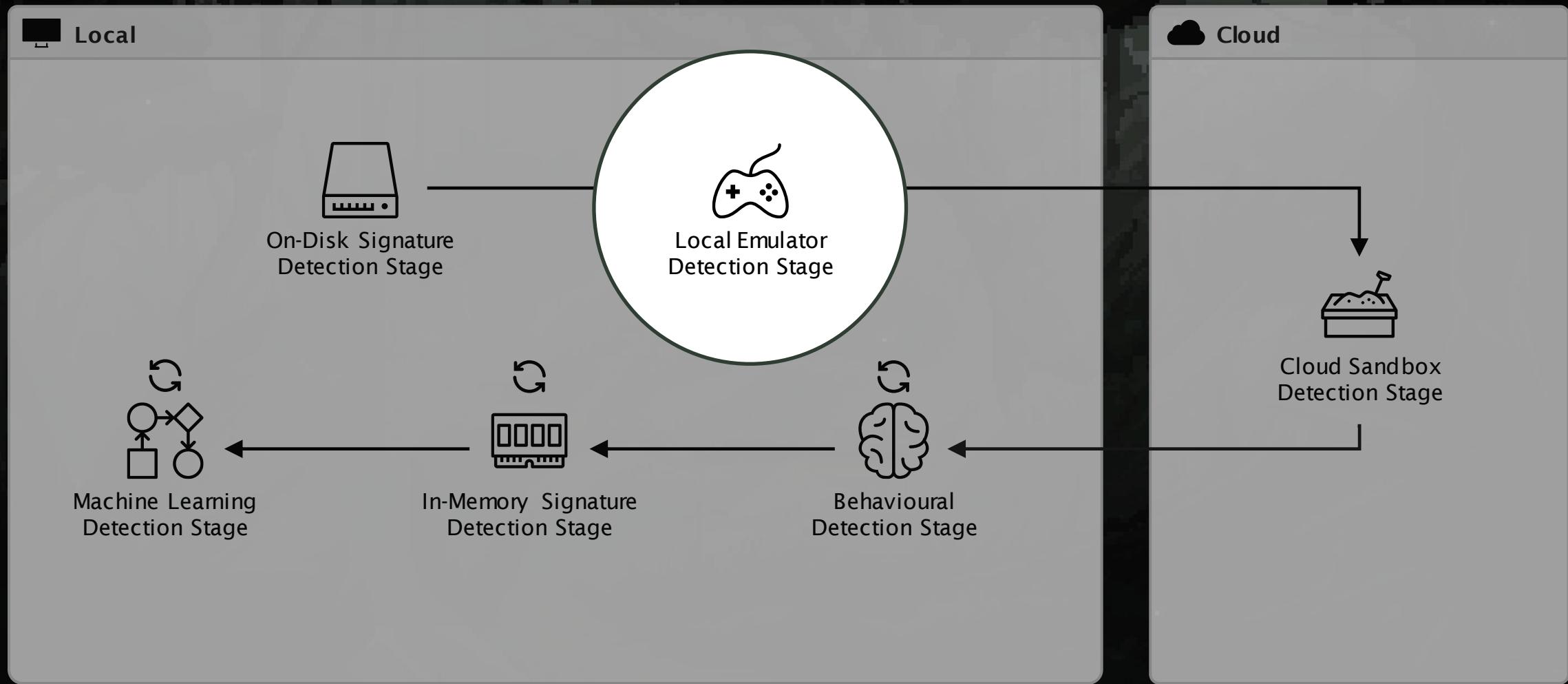
A screenshot of the x86dbg debugger interface is shown, displaying assembly code for the polymorphic decryption routine. The assembly code includes various instructions like JMP, POP, PUSH, CALL, NEG, ADD, SUB, XOR, and MOV, along with random-looking opcodes and addresses. The debugger's CPU tab is active, showing the assembly listing and registers (EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI) and stack dump panes.

Ultimate polymorphic form

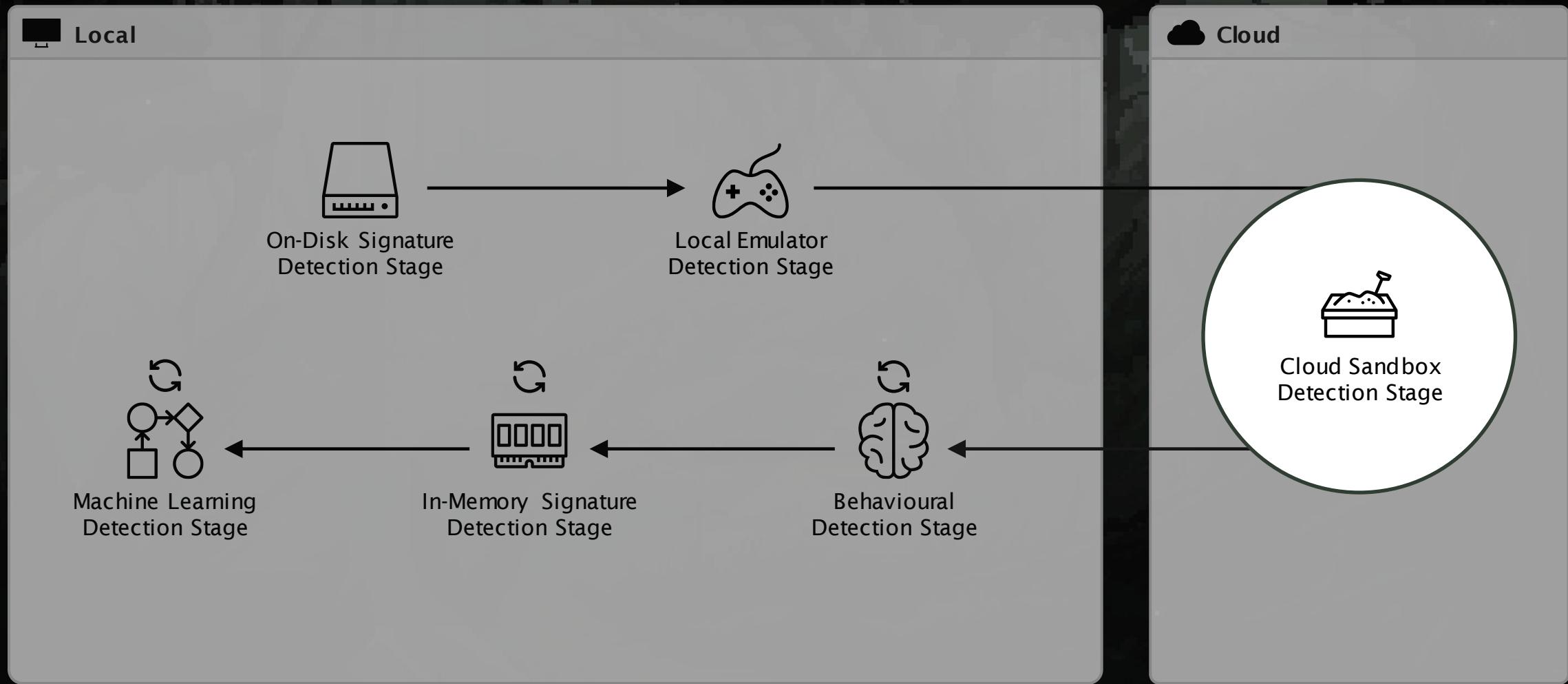
Detection stages



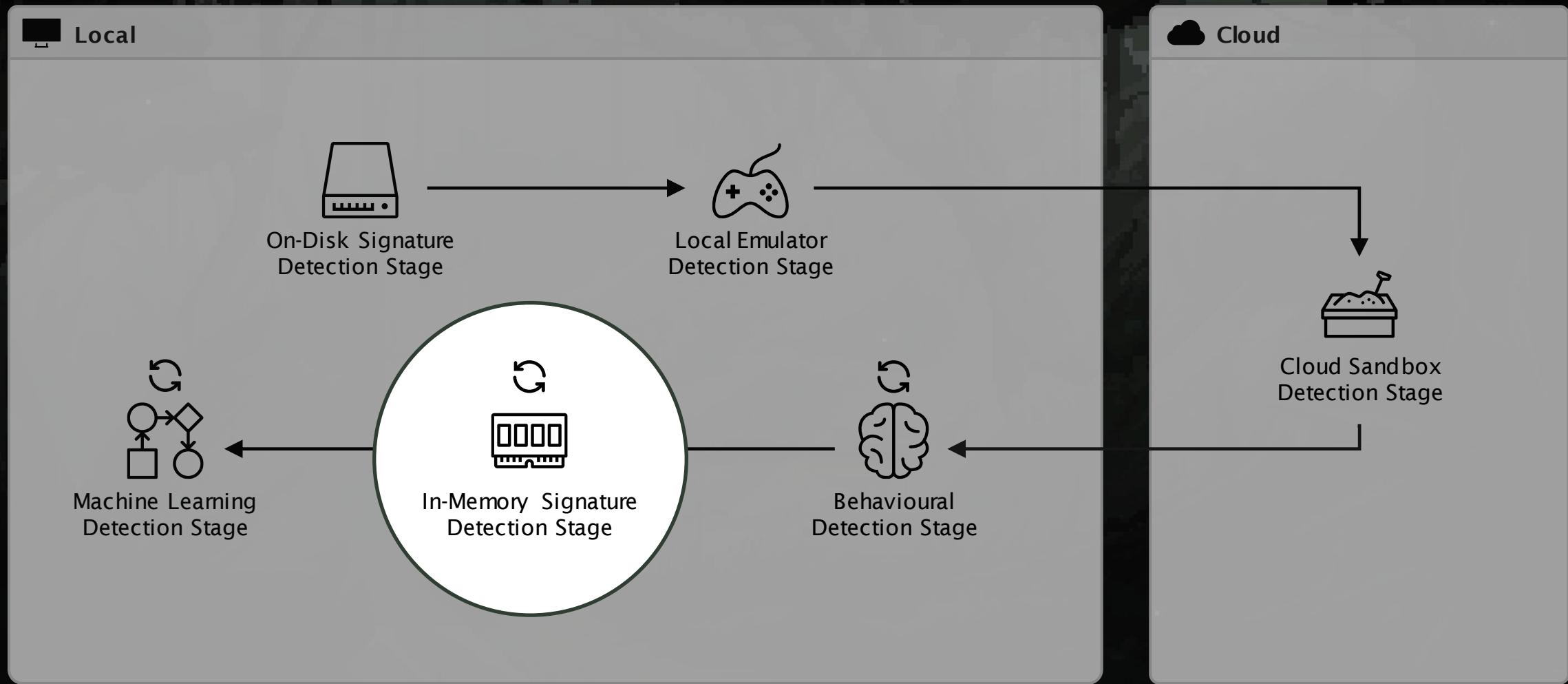
Detection stages



Detection stages



Detection stages



On-disk vs in-memory

SSD packed.exe (on disk)

```
.text segment: Decryption stub (random)
0x000 00110001100110011011001010011011100110110011000100011
0x010 00010110010000111001001101100011000001100100001110010
0x020 01100010011010100110001001110010110010100110100011000
0x030 01001110000110010100110110011000110011001100110010001
0x040 10011001110010011000100110001011000110010011001001100
```

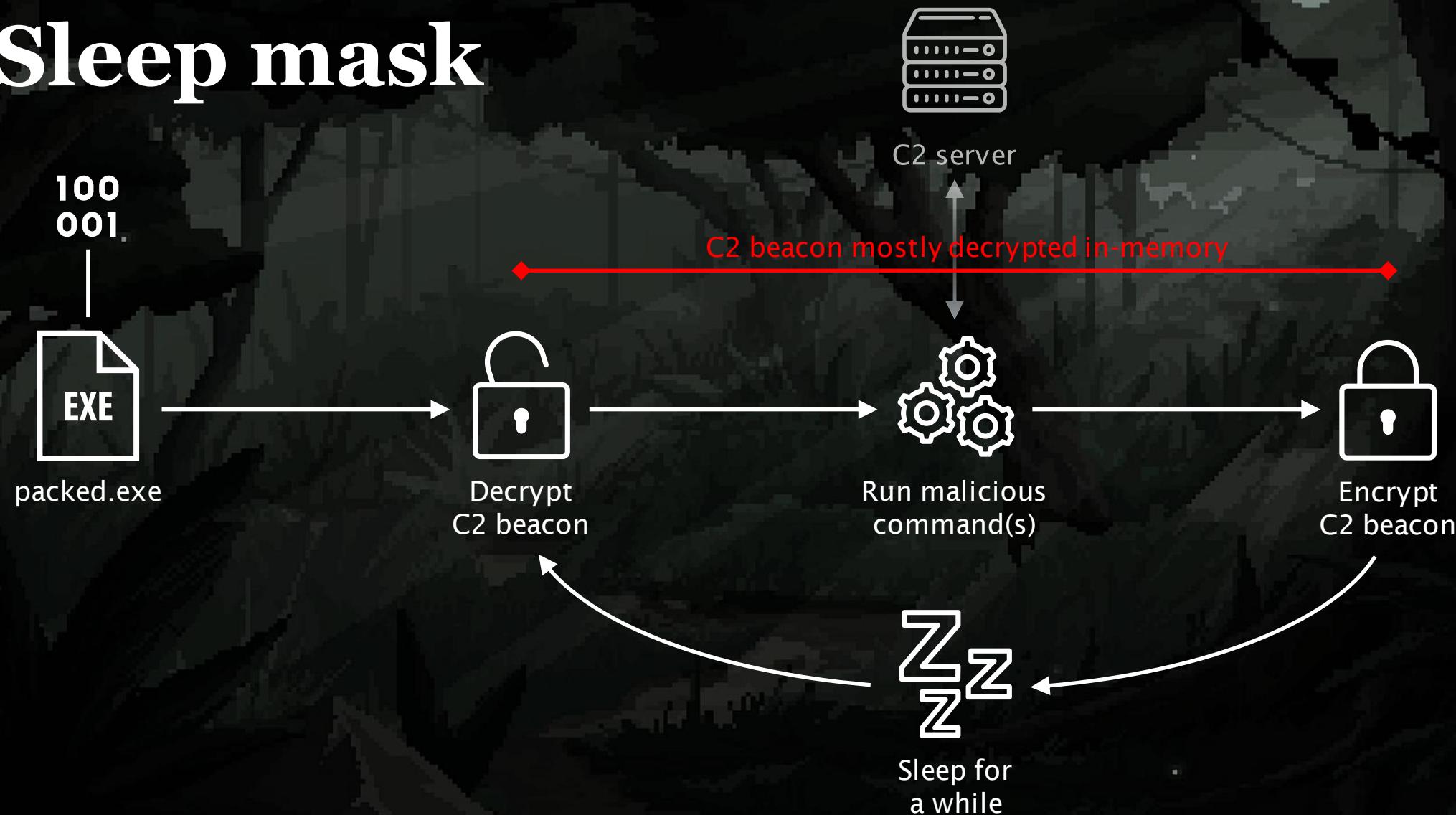
```
.text segment: Encrypted code (random)
0x050 0100110010001110000011010101011001010011011100111000011
0x060 00011011000100110010001100110001101010011000000110101
0x070 001100000011000000110110001101110110010100110011001101100
0x080 10101100001011000010110011001100001001101000110001000
0x090 1100110011000000111001001101100110010000110010011000
0x100 10110011000110110001110010110010100111001001100000011
0x110 100101100001011001000011001001100010001100010011001100
0x120 011011000110000001101110110010000111001001101000011000
0x130 01001110000011000000011011001100110001101110011010000
0x140 1010001100110011001100110001000110110011000010011000
0x150 01100110011001100110010100110010011000100011010000110
0x160 01001100010001100000001101010100110100001101110110010000
0x170 0110110001100000001101110110010000111001001101000011000
0x180 1110000011100000011100001100011000110110001
```

00 00 packed.exe (in memory)

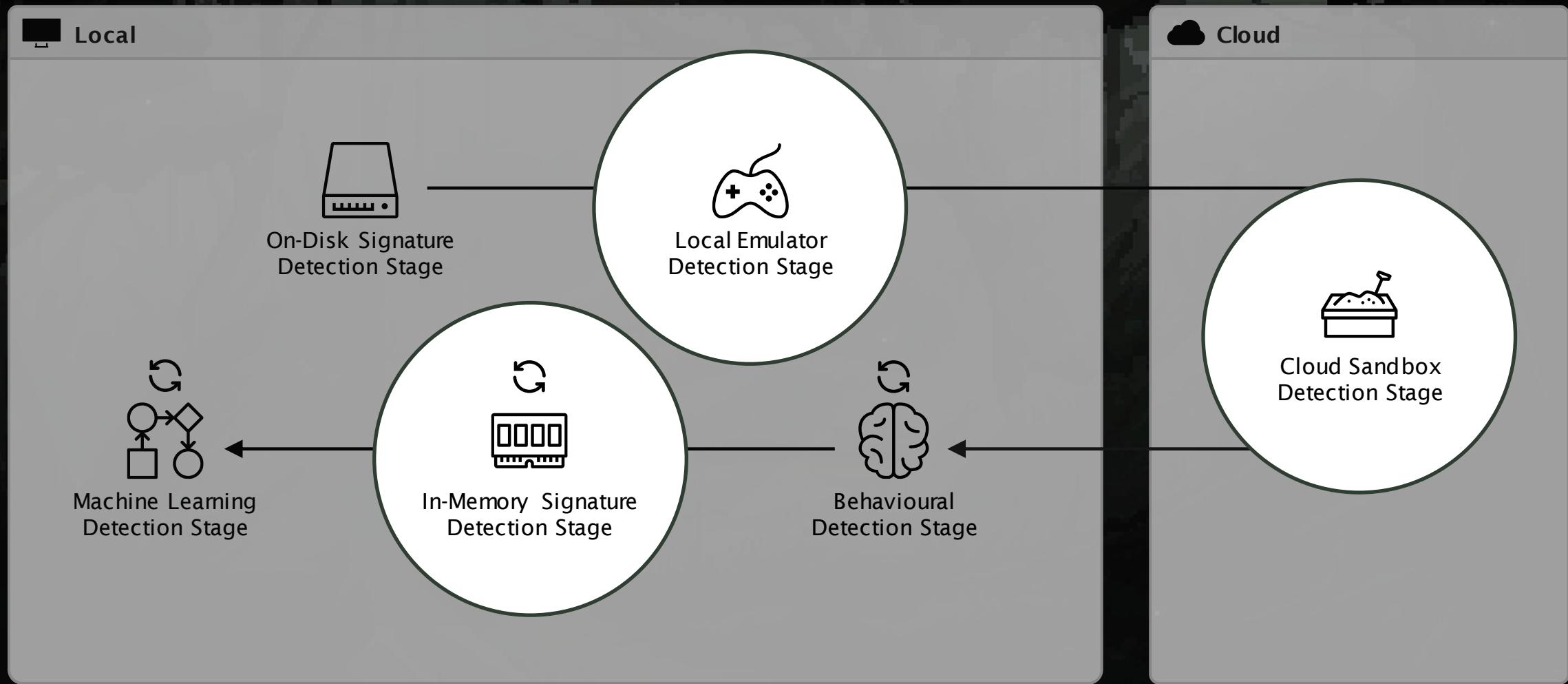
```
.text segment: Decryption stub (random)
0x000 00110001100110011011001010011011100110110011000100011
0x010 00010110010000111001001101100011000001100100001110010
0x020 01100010011010100110001001110010110010100110100011000
0x030 01001110000110010100110110011000110011001100110010001
0x040 10011001110010011000100110001011000110010011001001100
```

```
.text segment: Decrypted code (running)
0x050 010011001000111000001011 push rbp
0x060 000110110001001100100011 mov rbp, rsp
0x070 001100000011000000110110 mov DWORD PTR [rbp-0xC],0x1
0x080 101011000010110000101100 mov DWORD PTR [rbp-0x8],0x2
0x090 110011001100000011100100 mov eax, DWORD PTR [rbp-0xC]
0x100 101100110001101100011100 cdq
0x110 100101100001011001000011 idiv DWORD PTR [rbp-0x8]
0x120 011011000110000001101111 mov DWORD PTR [rbp-0x4], eax
0x130 010011100000110000001101 mov r15, [r11] # Hello
0x140 101000110011001100110011 mov r14, [r11+0x40] # World
0x150 011001100110011001100101 mov r13, [r11+0x50] # !
0x160 010011000100011000000110 mov eax, 0x0
0x170 011011000110000001101110 pop rbp
0x180 1110000011100000011100001 ret
```

Sleep mask



Detection stages



Talk outline

1. Polymorphic malware

Often found in malware antique stores

2. Metamorphic malware

Often seen on malware buzzword bingo cards

3. Dittobytes project

A true metamorphic cross-compiler

❖ Demo

Metamorphic malware



Results in *exact same* assembly,
every time you compile it.

Code

```
int test_function(int arg1) {
    int result;

    if (arg1 == 0xA) {
        result = 0x0A;
        result = result - 0x05;
    } else {
        result = 0x05;
    }

    return result;
}
```

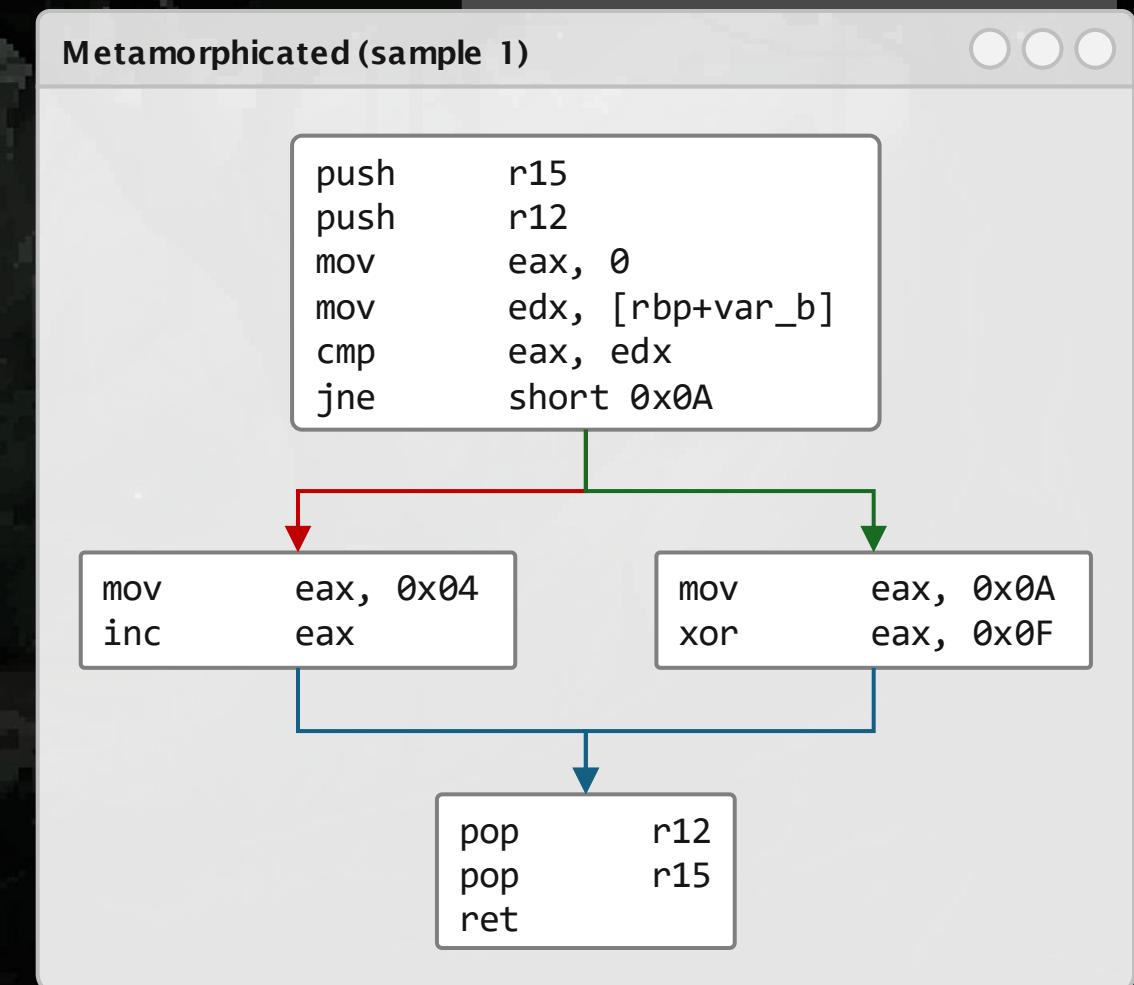
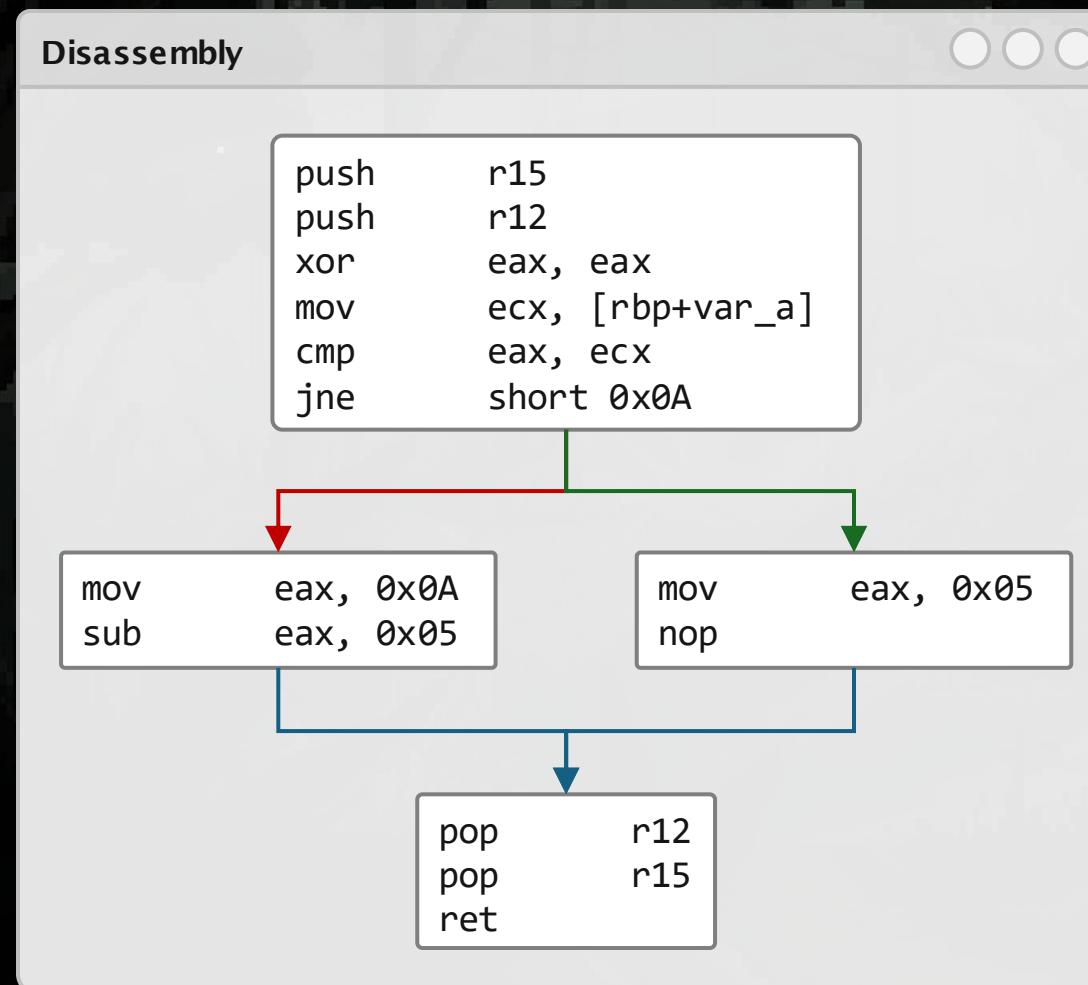
Disassembly

```
graph TD
    A["push r15  
push r12  
xor eax, eax  
mov ecx, [rbp+var_a]  
cmp eax, ecx  
jne short 0x0A"] --> B["mov eax, 0x0A  
sub eax, 0x05"]
    A --> C["mov eax, 0x05  
nop"]
    B --> D["pop r12  
pop r15  
ret"]
```

The diagram illustrates the control flow of the assembly code. It starts with a conditional block (labeled A) that branches to either a subtraction node (labeled B) or a mov/nop node (labeled C). Both B and C then converge at a final cleanup node (labeled D).

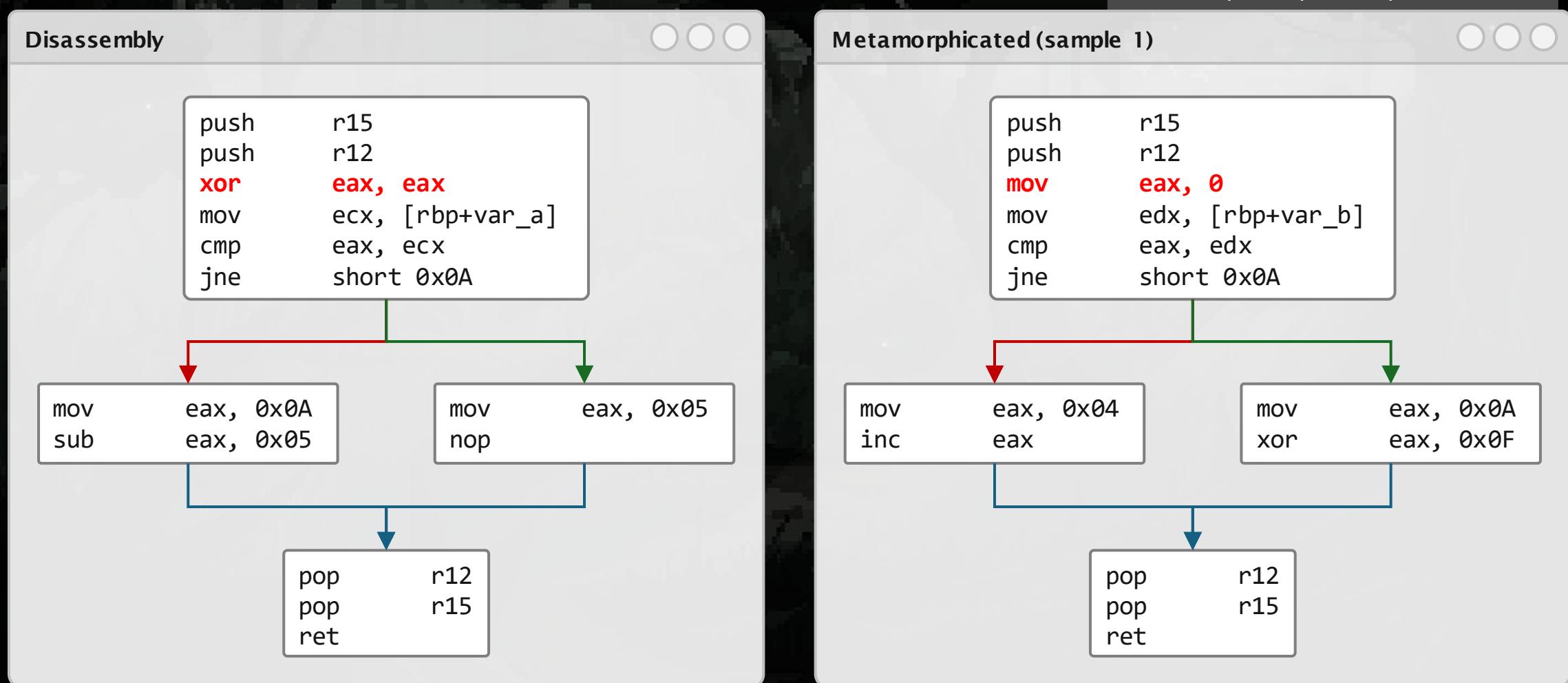
Metamorphic malware

↻ Results in *different* assembly,
every time you compile it.



Metamorphic malware

↻ Results in *different* assembly,
every time you compile it.



Metamorphic malware



Results in *different* assembly,
every time you compile it.

Disassembly

```
push    r15  
push    r12  
xor    eax, eax  
mov     ecx, [rbp+var_a]  
cmp     eax, ecx  
jne     short 0x0A
```

```
mov     eax, 0x0A  
sub    eax, 0x05
```

```
mov     eax, 0x05  
nop
```

```
pop    r12  
pop    r15  
ret
```

Metamorphicated (sample 2)

```
push    r15  
push    r12  
mov    eax, 0  
mov     edx, [rbp+var_b]  
cmp     eax, edx  
jne     short 0x0A
```

```
pop    r9  
push    r9  
mov    rdi, 0x11  
sub    rdi, eax  
shr    rdi, 0x01  
jmp    short 0xB
```

```
mov     xor    eax, 0xA  
eax, 0xF
```

Metamorphic malware

metamorphic-engine - GitHub Topics

https://github.com/topics/metamorphic-engine

Product Solutions Resources Open Source Enterprise Pricing

Search or jump to... / Sign In Help

Explore Topics Trending Collections Events GitHub Sponsors

metamorphic-engine Star

Here is 1 public repository matching this topic...

jakydibe / Zone Star 16

Code Issues Pull requests

keystone capstone obfuscator crypter metamorphism metamorphic-engine pe-obfuscator

Updated on May 11 Python

Improve this page Add a description, image, and links to the [metamorphic-engine](#) topic page so that developers can more easily learn about it.

Create this topic

Add this topic to your repo To associate your repository with the [metamorphic-engine](#) topic, visit your repository's landing page and select "Manage topics."

Last updated

Metamorphic malware

The image shows a screenshot of a GitHub repository page for a project named 'Zone'. The URL in the address bar is highlighted with a green box. The page content includes a README section with a quote about creating a metamorphic engine/PDF malware, followed by a section titled 'Code Morphing Techniques' which is also highlighted with a green box. This section contains two numbered items: '1. Equal Code Substitution' and '2. Random NOPs Insertion'. The first item has a list of code substitutions, and the second item has a list of random instructions. The entire screenshot is set against a dark background with a large, semi-transparent image of a hooded figure in the upper right corner.

GitHub - jakydibe/Zone

<https://github.com/jakydibe/Zone>

README

Creating a metamorphic engine/PDF malware is no small feat. This project has been the most challenging endeavor of my life, largely due to the lack of existing code to reference. The name 'Zone' is inspired by the film "Stalker" by Andrei Tarkovsky, reflecting a place where every path appears different but leads to the same destination.

Code Morphing Techniques

1. Equal Code Substitution

This technique involves replacing certain instructions with other instructions that achieve the same result but in a different way. For example:

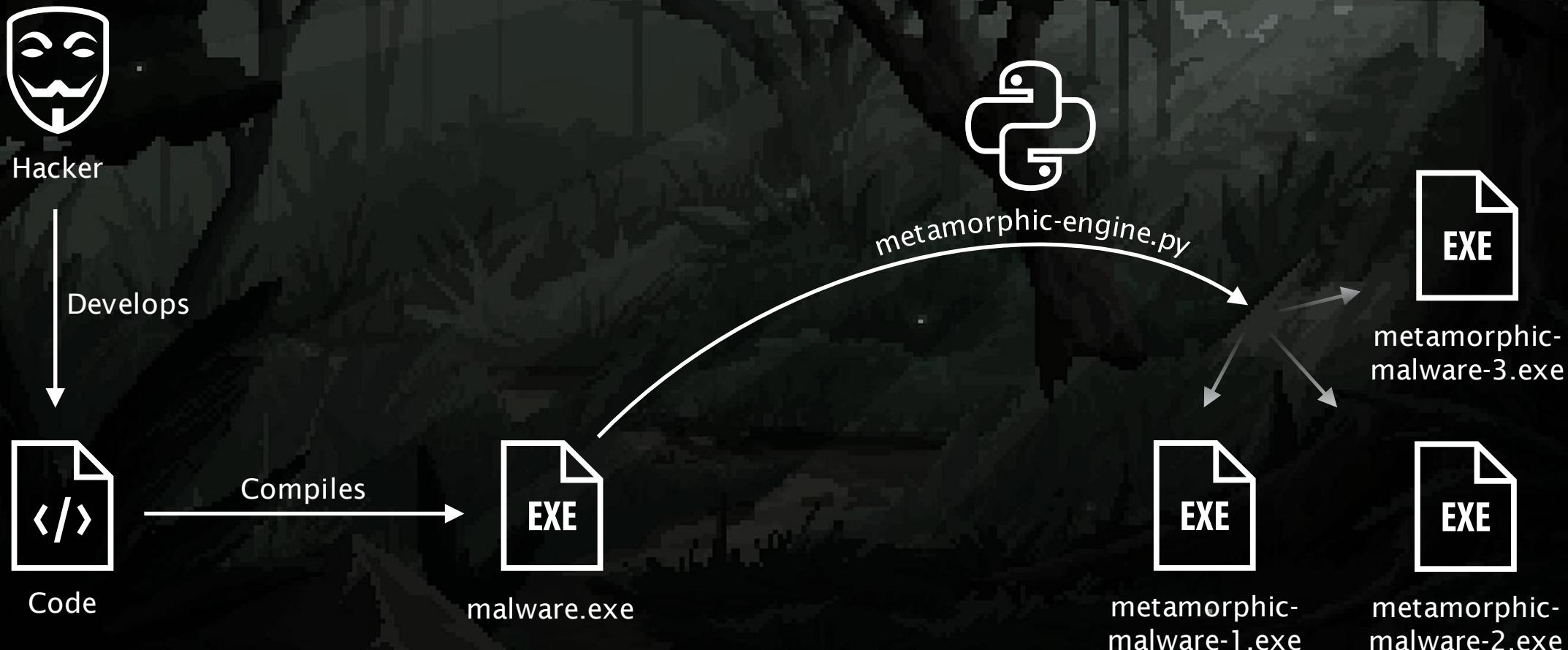
- `xor reg, reg → mov reg, 0`
- `sub reg, 0xVALUE → add reg, -0xVALUE`
- `mov REG, 0xVALUE → push VALUE; pop REG`

2. Random NOPs Insertion

Random instructions that do not affect the functionality of the program are inserted into the code. These include:

- `nop`
- `push add, sub`
- `jmp add, sub`

Post-compile metamorphifications



Post-compile metamorphifications

malware.exe (original)

```
// Syscall 'print' reference  
0x0000 mov rax, 1  
// Write to 'stdout'  
0x0004 mov rdi, 1  
// Reference to the bytes to print  
0x0008 lea rsi, [rip+0x0]  
// Amount of bytes to print  
0x000E mov rdx,  
// Call the 'print' function and print H1À  
0x0012 syscall  
// The three bytes (inlined) (ascii: H1À)  
0x0014 0x48, 0x31, 0xC0
```

malware.exe (metamorphicated)

```
// Syscall 'print' reference  
0x0000 mov rax, 1  
// Write to 'stdout'  
0x0004 mov rdi, 1  
// Reference to the bytes to print  
0x0008 lea rsi, [rip+0x0]  
// Amount of bytes to print  
0x000E mov rdx,  
// Call the 'print' function and print H1À  
0x0012 syscall  
// The three bytes (inlined) (ascii: H1À)  
0x0014 xor rax, rax
```

cmd.exe

```
tijme@vm $ .\malware.exe  
H1À
```

Post-compile metamorphifications

The image shows four windows arranged in a 2x2 grid, illustrating post-compile metamorphifications.

- malware.exe (original)**: Shows assembly code for printing "Hello". The instruction at address 0x0014 is `0x0014 0x48, 0x31, 0xC0`.
- malware.exe (metamorphicated)**: Shows assembly code for printing "Hello". The instruction at address 0x0014 has been changed to `0x0014 mov rax, 0`, which is highlighted in red.
- cmd.exe**: Shows the command `tijme@vm $.\malware.exe`. The output is "Hello".
- cmd.exe**: Shows the command `tijme@vm $.\malware.exe`. The output is "Hello" (note the extra space).



And this is why...

We need pre-compile metamorphic engines!

Pre-compile metamorphifications

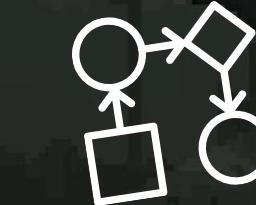


Hacker

Develops



Code



Compile directly with
metamorphic engine



metamorphic-
malware-1.exe



metamorphic-
malware-2.exe



metamorphic-
malware-3.exe

Talk outline

1. Polymorphic malware

Often found in malware antique stores

2. Metamorphic malware

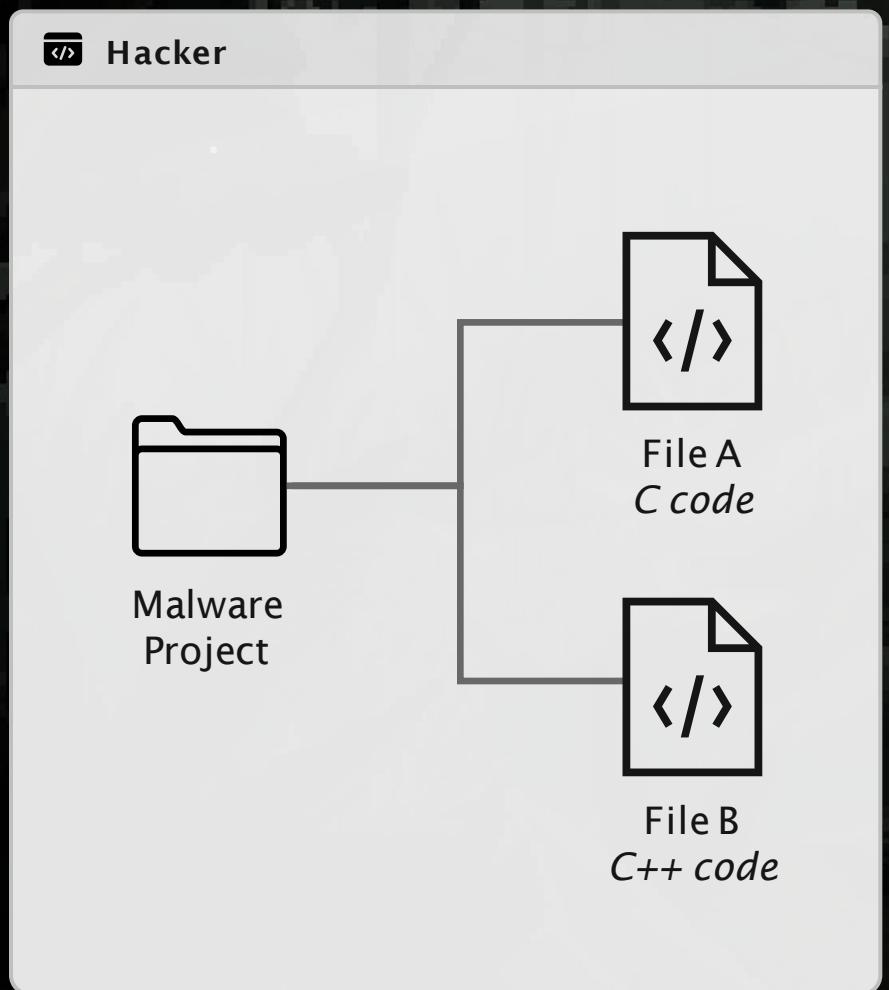
Often seen on malware buzzword bingo cards

3. Dittobytes project

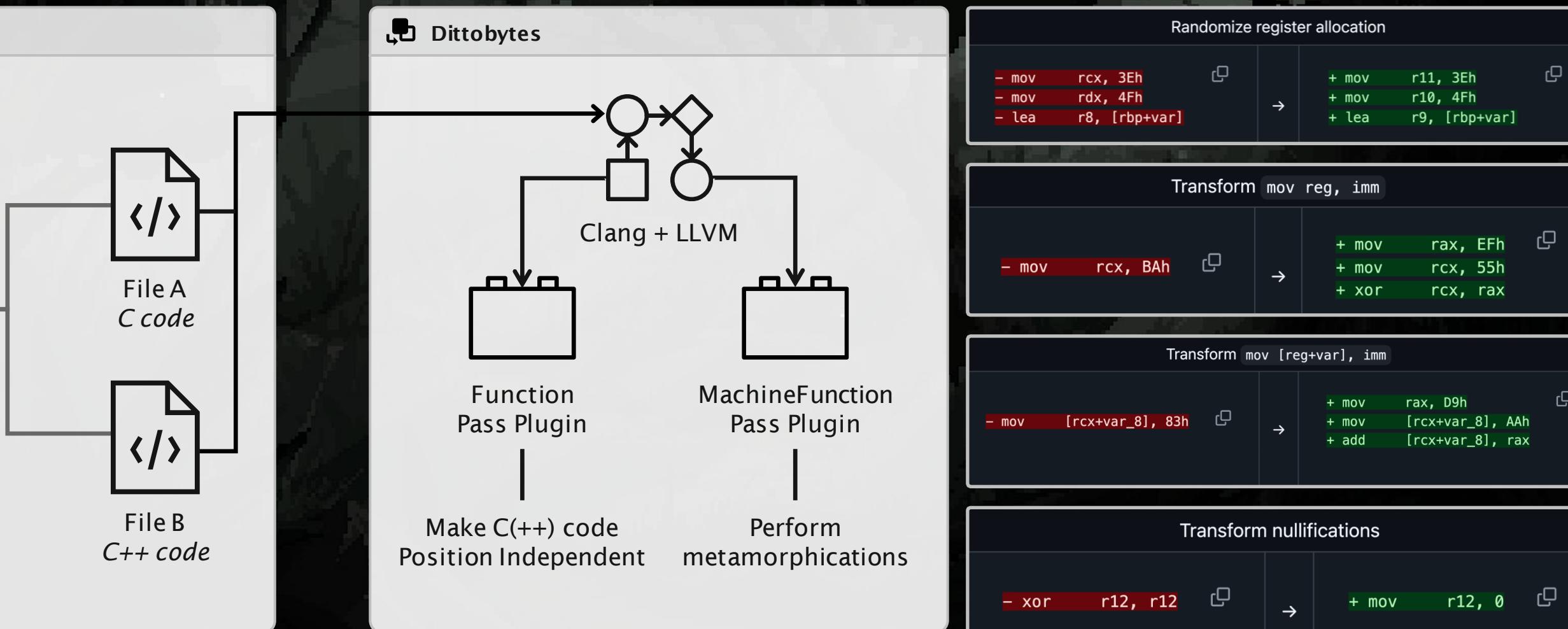
A true metamorphic cross-compiler

❖ Demo

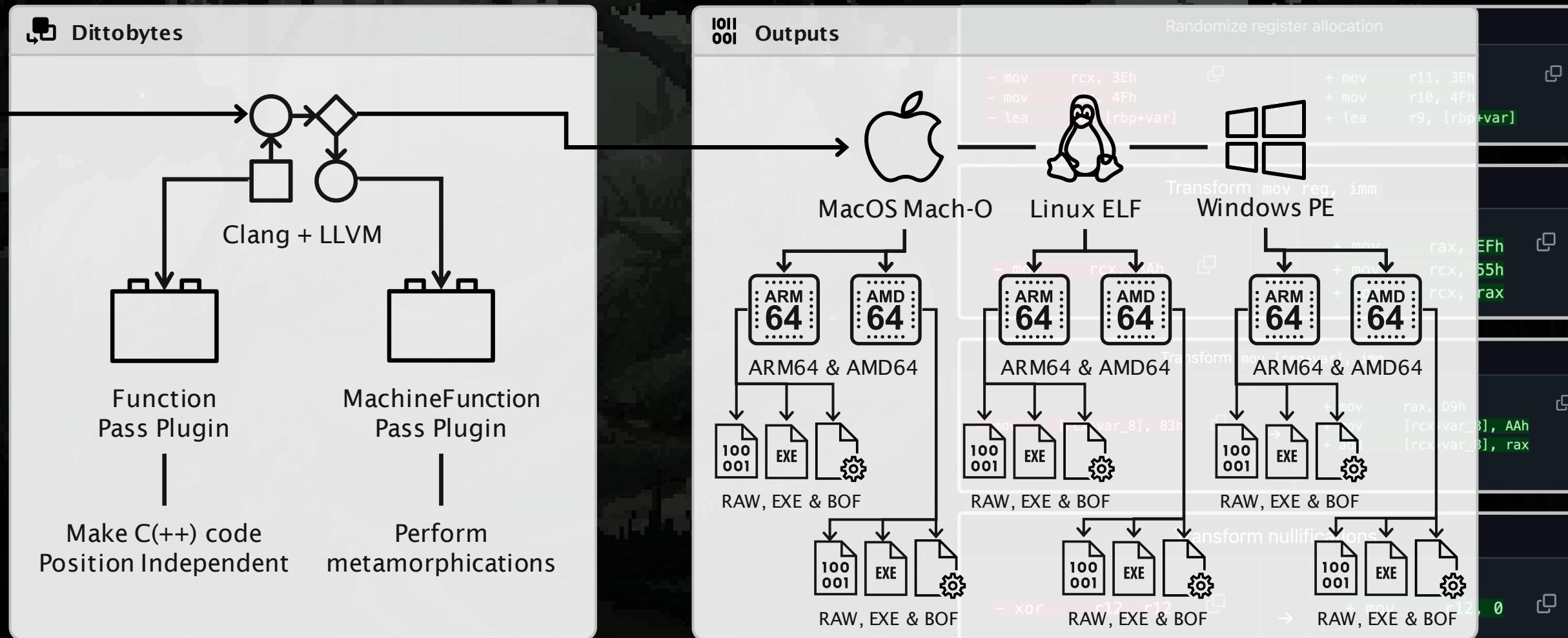
Dittobytes



Dittobytes



Dittobytes



Dittobytes

Dittobytes GitHub project

A screenshot of a web browser window displaying the GitHub repository for "Dittobytes". The URL in the address bar is <https://github.com/tijme/dittobytes>. The page content includes a "Getting started" section with sections for "Overview", "Preparing", and "Developing", each with associated links and difficulty levels.

tijme/dittobytes: Metamorphic c x +

https://github.com/tijme/dittobytes

README MPL-2.0 license

Getting started

Overview

- ▶ Directory structure

Preparing

- ▶ Cloning the repository
- ▶ Building the build tools in a Docker container
Difficulty: **easy**
- ▶ Installing the build tools on Windows Subsystem for Linux instead
Difficulty: **intermediate**

Developing

- ▶ The basics
- ▶ A hello world

Dittobytes

Creating the Dittobytes build env (Docker)

⌚ Apple M3 Pro:
20 minutes

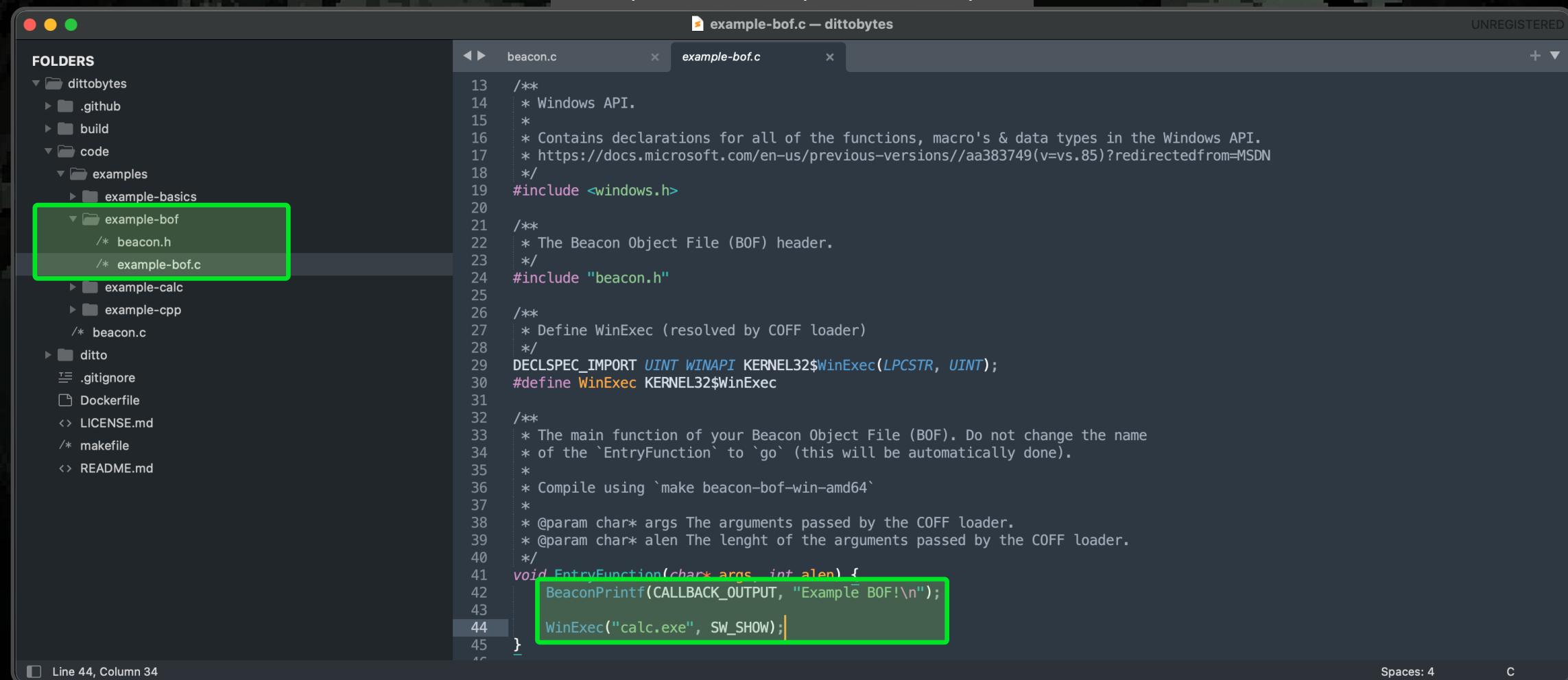
⌚ X64 laptop:
2,5 hours

```
docker buildx build -t dittobytes .

=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 3.37kB
=> [internal] load metadata for docker.io/library/debian:bookworm-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 2.16kB
=> [ 1/27] FROM docker.io/library/debian:bookworm-slim@sha256:2424c1850714a4d94666ec928e24d86de958646737b1d113f5b2207be44d37d8
=> CACHED [ 2/27] RUN apt update -y
=> CACHED [ 3/27] RUN apt install -y --no-install-recommends gnupg2 wget ca-certificates apt-transport-https autoconf automake cmake dpkg-dev file make patch libc6-dev mingw-w64 nano python
=> [ 4/27] COPY your-root-ca.crt /usr/local/share/ca-certificates/tia.crt
=> [ 5/27] RUN update-ca-certificates
=> [ 6/27] COPY ditto/scripts/requirements.txt /tmp/requirements.txt
=> [ 7/27] RUN python3 -m pip install -r /tmp/requirements.txt --break-system-packages
=> [ 8/27] WORKDIR /opt
=> [ 9/27] RUN git clone --depth 1 https://github.com/tijme/forked-dittobytes-macos-sdk.git macos-sdk
=> [10/27] WORKDIR /opt
=> [11/27] RUN wget https://github.com/mstorsjo/llvm-mingw/releases/download/20240619/llvm-mingw-20240619-msvcrt-ubuntu-20.04-x86_64.tar.xz
=> [12/27] RUN tar -xf llvm-mingw-20240619-msvcrt-ubuntu-20.04-x86_64.tar.xz
=> [13/27] RUN mv llvm-mingw-20240619-msvcrt-ubuntu-20.04-x86_64 llvm-winlin
=> [14/27] RUN rm llvm-mingw-20240619-msvcrt-ubuntu-20.04-x86_64.tar.xz
=> [15/27] WORKDIR /opt
=> [16/27] RUN git clone --depth 1 --branch release/18.x https://github.com/tijme/forked-dittobytes-llvm-project.git llvm-source
=> [17/27] WORKDIR /opt/llvm-source/build
=> [18/27] RUN cmake -G Ninja .. -DLLVM_ENABLE_PROJECTS="clang;lld" -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD="X86;AArch64" -DCMAKE_INSTALL_PREFIX=/opt/llvm -DBUILD_ 7.6s
=> [19/27] RUN ninja
=> => # [3097/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/ToolChains/RISCVToolchain.cpp.o
=> => # [3098/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/ToolChains/PPCLinux.cpp.o
=> => # [3099/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/ToolChains/InterfaceStubs.cpp.o
=> => # [3100/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/Types.cpp.o
=> => # [3101/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/ToolChains/WebAssembly.cpp.o
=> => # [3102/4186] Building CXX object tools/clang/lib/Driver/CMakeFiles/obj clangDriver.dir/ToolChains/ZOS.cpp.o
```

Dittobytes

Example code to compile with Dittobytes



```
beacon.c      example-bof.c
13  /**
14   * Windows API.
15   *
16   * Contains declarations for all of the functions, macro's & data types in the Windows API.
17   * https://docs.microsoft.com/en-us/previous-versions/aa383749(v=vs.85)?redirectedfrom=MSDN
18   */
19 #include <windows.h>
20
21 /**
22  * The Beacon Object File (BOF) header.
23  */
24 #include "beacon.h"
25
26 /**
27  * Define WinExec (resolved by COFF loader)
28  */
29 DECLSPEC_IMPORT UINT WINAPI KERNEL32$WinExec(LPCSTR, UINT);
30 #define WinExec KERNEL32$WinExec
31
32 /**
33  * The main function of your Beacon Object File (BOF). Do not change the name
34  * of the `EntryFunction` to `go` (this will be automatically done).
35  *
36  * Compile using `make beacon-bof-win-amd64`
37  *
38  * @param char* args The arguments passed by the COFF loader.
39  * @param char* alen The lenght of the arguments passed by the COFF loader.
40  */
41 void EntryFunction(char* args, int alen) {
42     BeaconPrintf(CALLBACK_OUTPUT, "Example BOF!\n");
43
44     WinExec("calc.exe", SW_SHOW);
45 }
```

Line 44, Column 34 Spaces: 4 C

Dittobytes

Example code to compile with Dittobytes

The screenshot shows a terminal window titled "example-calc.c — dittobytes". The left pane displays a file tree for a project named "dittobytes". The "example-calc" folder is selected and highlighted with a green box. Inside this folder, the file "example-calc.c" is also highlighted with a green box. The right pane contains the source code for "example-calc.c". A specific block of code is highlighted with a green box, which includes the following lines:

```
// Run WinExec and return its return value
char CalculatorBinary[] = "calc.exe";
return (uint32_t) context.functions.WinExec(CalculatorBinary, SW_SHOW) > 31;
```

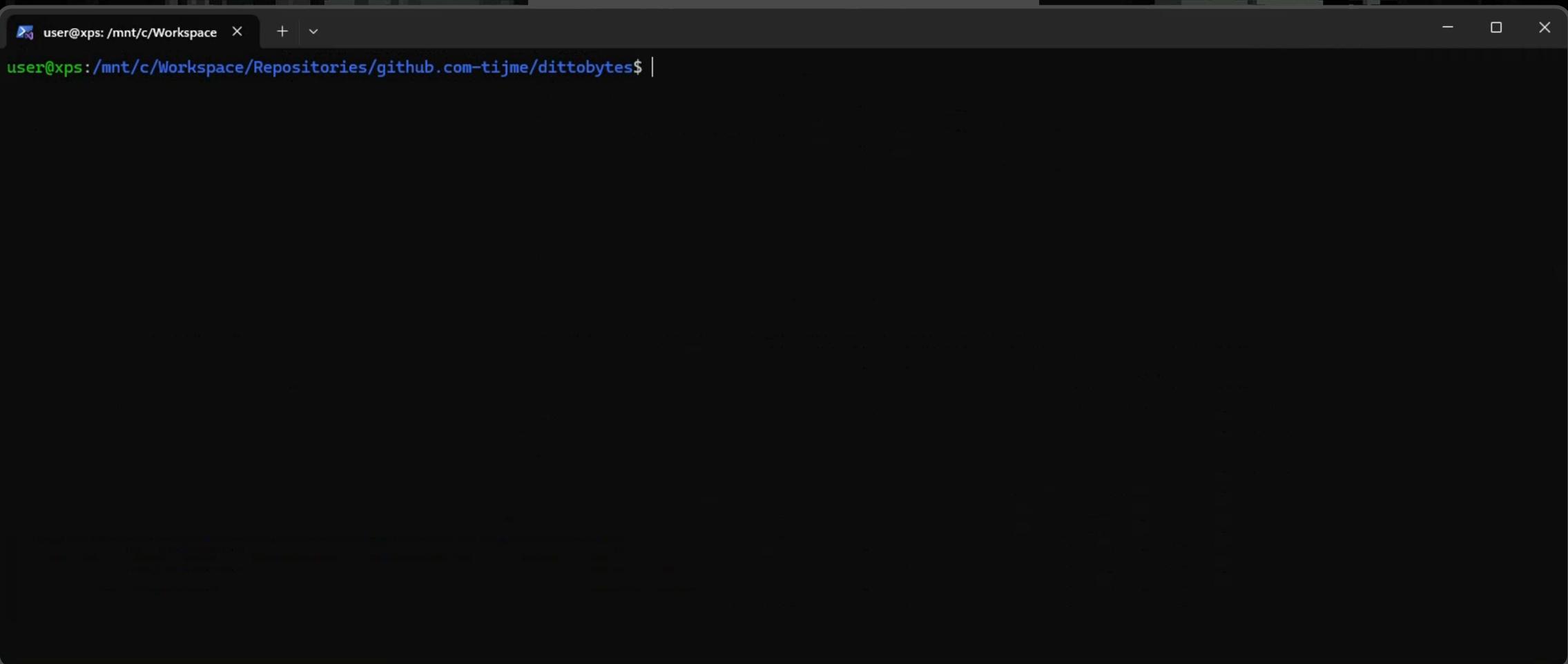
The code implements a function "EntryFunction" that performs the following steps:

- Initializes a "Relocatable" context.
- Populates module and function tables.
- Runs WinExec with the binary "calc.exe" and returns its result.
- Gets the current Process Environment Block (PEB).
- Returns a pointer to the PEB.

At the bottom of the terminal window, the status bar shows "Line 180, Column 18" and "Spaces: 4".

Dittobytes

Building the example code using Dittobytes



```
user@xps: /mnt/c/Workspace$
```

Dittobytes

Building the example code using Dittobytes

```
user@xps:/mnt/c/Workspace$ make help
[+] Building your code:
- make beacon-[format]-[platform]-[arch]                                // (Re-)compile your code
  ↳ Available formats: exe,raw,bof
  ↳ Available platforms: win,lin,mac
  ↳ Available architectures: amd64,arm64
- make beacon-bof-win-amd64                                         // (Re-)compile your code to Windows AMD64 BOF/COFF
- make beacon-raw-mac-arm64                                         // (Re-)compile your code to MacOS ARM64 raw shellcode
- make beacon-raw-lin-all                                           // (Re-)compile your shellcode to raw shellcode for Linux and any architecture
- make beacon-raw-all-all                                         // (Re-)compile your shellcode to raw shellcode for any platform and architecture
- make beacon-all-all-all                                         // (Re-)compile your shellcode to executable, BOF/COFF and raw shellcode for any platform and architecture
[+] Dittobytes internals:
- make ditto-loaders                                              // (Re-)compile all pre-shipped Ditto shellcode loaders
- make ditto-transpilers                                         // (Re-)compile the pre-shipped Ditto LLVM transpilers/passes
[+] Test suite:
- make test-suite-build                                         // (Re-)compile all feature tests
- make test-suite-test                                           // Run all feature tests (for the current architecture)
[+] Cleanup:
- make clean                                                       // Remove all your code builds ('beacon-*') from the build folder
- make clean-ditto-loaders                                       // Remove all loader builds ('loaders-*') from the build folder
- make clean-ditto-transpilers                                    // Remove all transpiler builds from the transpiler build folders
[+] Help:
- make help                                                       // Show this help message
user@xps:/mnt/c/Workspace$ |
```

Dittobytes

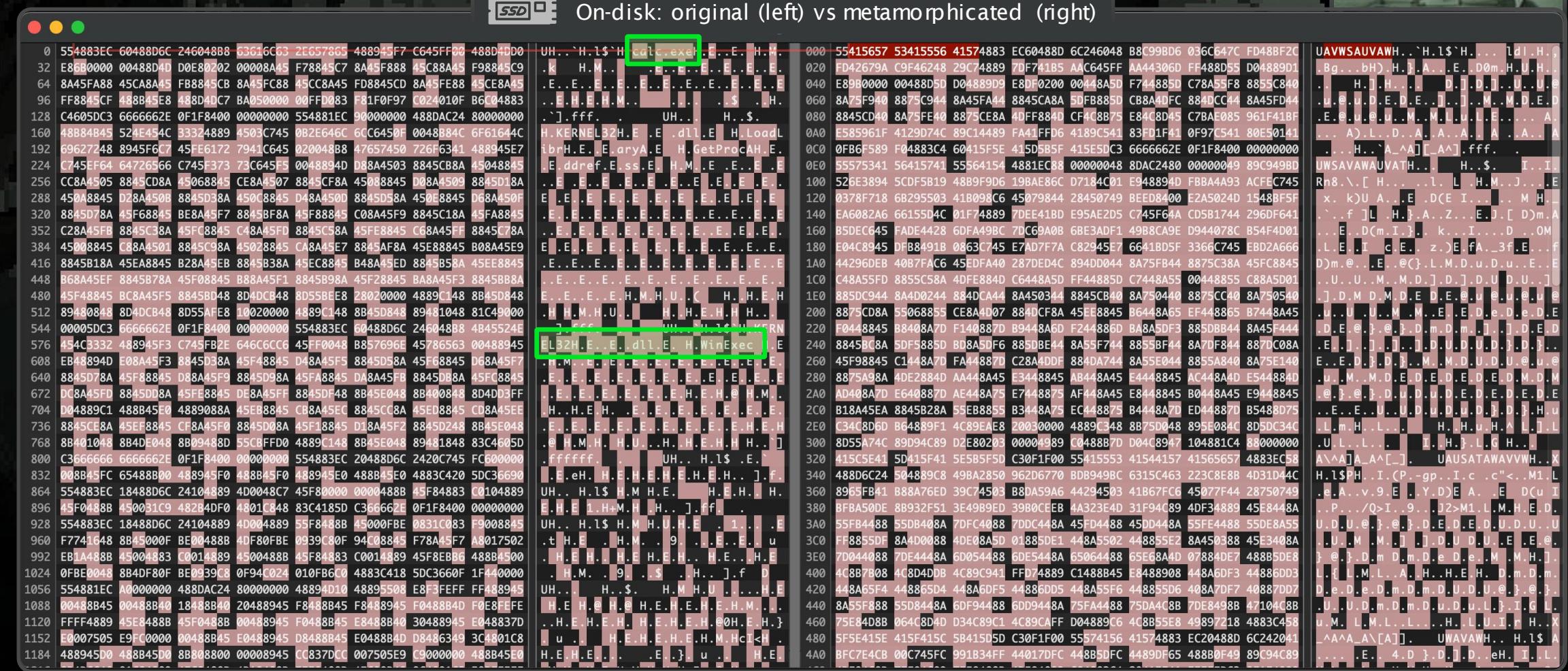
Running the example code using Dittobytes

```
user@xps: /mnt/c/Workspace × + ▾
✓ Modified immediate value using random option 'XOR'.
↳ MachineTranspiler passing function 'RelocatablePreliminaryGetProcAddress( ... )' for step '1'.
    ↳ Running ARM64 module: TransformRegMovImmediates(option=XOR, modifyAll=1).
- Intermediate compile of build/beacon-win-arm64.meta2.mir.
- Intermediate compile of build/beacon-win-arm64.meta3.mir.
    ↳ MachineTranspiler passing function 'go( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'InitializeRelocatable( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'PopulateTables( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'RelocatableNtGetPeb( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'RelocatableGetDataTableEntry( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'RelocatableStrCmp( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
    ↳ MachineTranspiler passing function 'RelocatablePreliminaryGetProcAddress( ... )' for step '2'.
        ↳ Running ARM64 module: TransformNullifications(modifyAll=0).
- Intermediate compile of build/beacon-win-arm64.obj.
- Intermediate cleanup of build files.
- Done building BOF beacon-bof-win-arm64.

user@xps:/mnt/c/Workspace/Repositories/github.com-tijme/dittobytes$ ls -lah build/beacon*
-rwxrwxrwx 1 user user 4.0K Aug  3 18:53 build/beacon-win-amd64.exe
-rwxrwxrwx 1 user user 3.1K Aug  3 18:53 build/beacon-win-amd64.obj
-rwxrwxrwx 1 user user 2.1K Aug  3 18:53 build/beacon-win-amd64.raw
-rwxrwxrwx 1 user user 4.0K Aug  3 18:53 build/beacon-win-arm64.exe
-rwxrwxrwx 1 user user 3.1K Aug  3 18:53 build/beacon-win-arm64.obj
-rwxrwxrwx 1 user user 2.2K Aug  3 18:53 build/beacon-win-arm64.raw
user@xps:/mnt/c/Workspace/Repositories/github.com-tijme/dittobytes$
```

Dittobytes

Dittobytes



September 5, 2025

Tijme @ OrangeCon

46

Dittobytes

Memory: In-memory: original (left) vs metamorphicated (right)

RAX	.text:00007FF6FAFC1000 push rbp	RAX	.text:00007FF7D4581000 push rbp
RDX	.text:00007FF6FAFC1001 sub rbp, 60h	RDX	.text:00007FF7D4581001 push r13
RIP	.text:00007FF6FAFC1005 lea rbp, [rsp+60h]	RIP	.text:00007FF7D4581003 push r15
R9	.text:00007FF6FAFC100A mov rax, 6578652E636C6163h	R9	.text:00007FF7D4581005 push r14
	.text:00007FF6FAFC1014 mov [rbp+var_9], rax		.text:00007FF7D4581007 push rbx
	.text:00007FF6FAFC1018 mov [rbp+var_1], 0		.text:00007FF7D4581008 push r12
	.text:00007FF6FAFC101C lea rcx, [rbp+var_30]		.text:00007FF7D458100A sub rsp, 68h
	.text:00007FF6FAFC1020 call sub_7FF6FAFC1090		.text:00007FF7D458100E lea rbp, [rsp+60h]
	.text:00007FF6FAFC1025 lea rcx, [rbp+var_30]		.text:00007FF7D4581013 mov r14, 0FD1EE24A962E9956h
	.text:00007FF6FAFC1029 call sub_7FF6FAFC1230		.text:00007FF7D458101D mov rdx, 98668764F542F835h
	.text:00007FF6FAFC102E mov al, byte ptr [rbp+var_9]		.text:00007FF7D4581027 xor rdx, r14
	.text:00007FF6FAFC1031 mov [rbp+var_39], al		.text:00007FF7D458102A mov [rbp+30h+var_31], rdx
	.text:00007FF6FAFC1034 mov al, byte ptr [rbp+var_9+1]		.text:00007FF7D458102E mov cl, 56h ; 'V'
	.text:00007FF6FAFC1037 mov [rbp+var_38], al		.text:00007FF7D4581030 mov [rbp+30h+var_29], 56h ; 'V'
	.text:00007FF6FAFC103A mov al, byte ptr [rbp+var_9+2]		.text:00007FF7D4581034 xor [rbp+30h+var_29], cl
	.text:00007FF6FAFC103D mov [rbp+var_37], al		.text:00007FF7D4581037 lea r15, [rbp+30h+var_58]
	.text:00007FF6FAFC1040 mov al, byte ptr [rbp+var_9+3]		.text:00007FF7D458103B mov rcx, r15
	.text:00007FF6FAFC1043 mov [rbp+var_36], al		.text:00007FF7D458103E call sub_7FF7D45810E0
	.text:00007FF6FAFC1046 mov al, byte ptr [rbp+var_9+4]		.text:00007FF7D4581043 lea r13, [rbp+30h+var_58]
	.text:00007FF6FAFC1049 mov [rbp+var_35], al		.text:00007FF7D4581047 mov rcx, r13
	.text:00007FF6FAFC104C mov al, byte ptr [rbp+var_9+5]		.text:00007FF7D458104A call sub_7FF7D4581340
	.text:00007FF6FAFC104F mov [rbp+var_34], al		.text:00007FF7D458104F mov r11b, byte ptr [rbp+30h+var_31]

Dittobytes



In-memory: original (left) vs metamorphicated (right)

```
1 .text:00007FF6FAFC1090 push rbp
2 .text:00007FF6FAFC1091 sub rsp, 90h
3 .text:00007FF6FAFC1098 lea rbp, [rsp+80h]
4 .text:00007FF6FAFC10A0 mov rax, 32334C454E52454Bh
5 .text:00007FF6FAFC10AA mov [rbp+10h+var_D], rax
6 .text:00007FF6FAFC10AE mov [rbp+10h+var_5], 6C6C642Eh
7 .text:00007FF6FAFC10B5 mov [rbp+10h+var_1], 0
8 .text:00007FF6FAFC10B9 mov rax, 7262694C64616F4Ch
9 .text:00007FF6FAFC10C3 mov [rbp+10h+var_1A], rax
10 .text:00007FF6FAFC10C7 mov [rbp+10h+var_12], 41797261h
11 .text:00007FF6FAFC10CE mov [rbp+10h+var_E], 0
12 .text:00007FF6FAFC10D2 mov rax, 41636F7250746547h
13 .text:00007FF6FAFC10DC mov [rbp+10h+var_29], rax
14 .text:00007FF6FAFC10E0 mov [rbp+10h+var_21], 65726464h
15 .text:00007FF6FAFC10E7 mov [rbp+10h+var_1D], 7373h
16 .text:00007FF6FAFC10ED mov [rbp+10h+var_1B], 0
17 .text:00007FF6FAFC10F1 mov [rbp+10h+var_38], rcx
18 .text:00007FF6FAFC10F5 mov al, byte ptr [rbp+10h+var_D]
19 .text:00007FF6FAFC10F8 mov [rbp+10h+var_45], al
20 .text:00007FF6FAFC10FB mov al, byte ptr [rbp+10h+var_D+1]
21 .text:00007FF6FAFC10FE mov [rbp+10h+var_44], al
22 .text:00007FF6FAFC1101 mov al, byte ptr [rbp+10h+var_D+2]
```

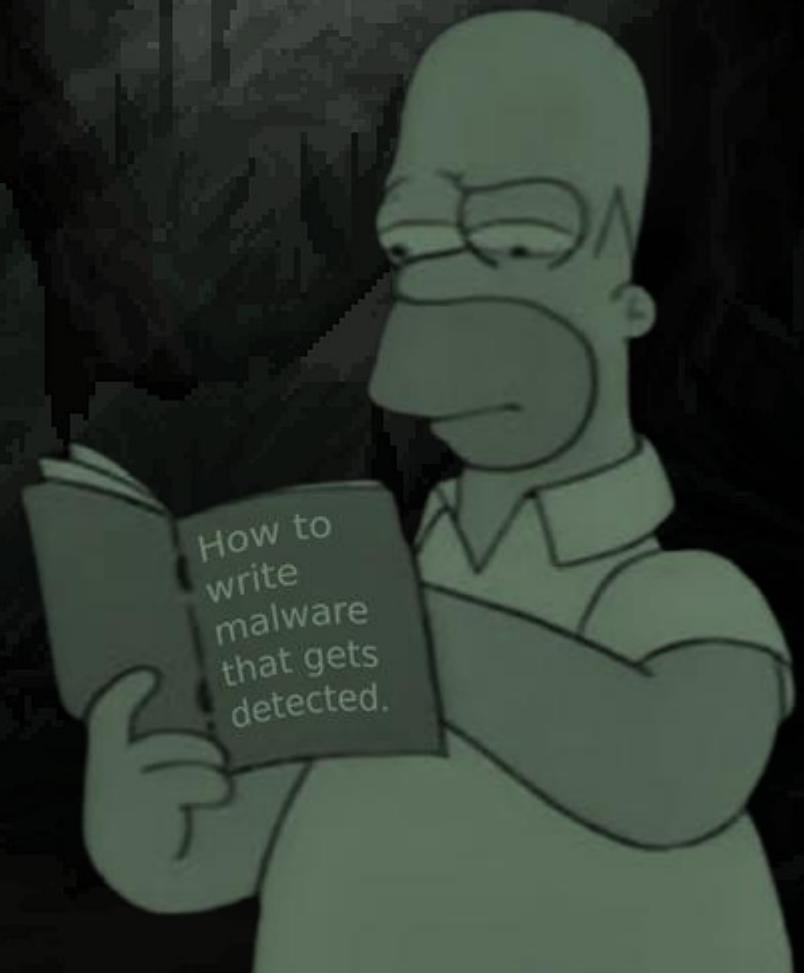
```
1 .text:00007FF6D5471000 push rbp
2 .text:00007FF6D5471001 push r14
3 .text:00007FF6D5471003 push r12
4 .text:00007FF6D5471005 push rdi
5 .text:00007FF6D5471006 push rsi
6 .text:00007FF6D5471007 push r15
7 .text:00007FF6D5471009 push r13
8 .text:00007FF6D547100B sub rsp, 60h
9 .text:00007FF6D547100F lea rbp, [rsp+60h]
10 .text:00007FF6D5471014 mov r13, 1246845D81FC9E55h
11 .text:00007FF6D547101E mov rax, 773EE173E290FF36h
12 .text:00007FF6D5471028 xor rax, r13
13 .text:00007FF6D547102B mov [rbp+30h+var_39], rax
14 .text:00007FF6D547102F mov r9b, 2Fh ; '/'
15 .text:00007FF6D5471032 mov [rbp+30h+var_31], 2Fh ; '/'
16 .text:00007FF6D5471036 sub [rbp+30h+var_31], r9b
17 .text:00007FF6D547103A lea r9, [rbp+30h+var_60]
18 .text:00007FF6D547103E mov rcx, r9
19 .text:00007FF6D5471041 call sub_7FF6D54710E0
20 .text:00007FF6D5471046 lea r12, [rbp+30h+var_60]
21 .text:00007FF6D547104A mov rcx, r12
22 .text:00007FF6D547104D call sub_7FF6D5471330
```





But...

Does it now properly bypass
in-memory detections?



Evasion



Automatic Yara rule generator

By Florian Roth

GitHub - Neo23x0/yarGen: yarGen · GitHub

https://github.com/Neo23x0/yarGen

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 Branch 14 Tags

Go to file Code

Neo23x0 Merge pull request #54 from ShobhitMishra-bot/patch-1 c16faff · 4 months ago 206 Commits

3rdparty v0.18.0 8 years ago

screens v0.14.2 10 years ago

tools Cleanup 10 years ago

.gitignore Update .gitignore 2 years ago

LICENSE License 11 years ago

README.md v0.24.0 - AI output support 2 years ago

prepare-release.sh chore: prepare release script 5 years ago

requirements.txt replace pefile with lief & add support of opcode extractio... 2 years ago

yarGen.py fix-invalid-escape-sequence 4 months ago

About

yarGen is a generator for YARA rules

python malware malwareanalysis
malware-analysis malware-research
yara

Readme View license Activity 1.7k stars 90 watching 296 forks Report repository

Releases 7

Evasion



Automatic Yara rule generator

Automatic Yara rule generator

yargen_rules.yar — yarGen UNREGISTERED

FOLDERS

- yarGen
- .dbs
- 3rdparty
- dbs
- dropzone
- screens
- tools
- .gitignore
- LICENSE
- /* prepare-release.sh
- <> README.md
- requirements.txt
- /* yarGen.py
- yargen_rules.yar

rule beacon_win_amd64 {
meta:
 description = "memsamples - file memsample-beacon-win-amd64-1.exe"
 author = "Tijme Gommers"
 reference = "https://github.com/Neo23x0/yarGen"
 date = "2025-08-19"
 hash1 = "5dd006e88b691ba22d6b4507ac9036e3aaaa8efb1f13552590709521a7d46b74"
strings:
 \$op0 = { 55 41 57 57 41 54 41 55 48 83 ec 70 48 8d 6c 24 }
 \$op1 = { 4c 8d 45 c0 4c 89 c1 e8 fe 02 00 00 8a 4d ef 88 }
 \$op2 = { 55 57 53 41 55 56 41 56 41 57 41 54 48 81 ec 98 }
condition:
 uint16(0) == 0x5a4d and filesize < 20KB and
 all of (\$op*)
}

Line 15, Column 2 Spaces: 3 Plain Text

Evasion



Automatic Yara rule generator

```
-zsh  
[09:17:22] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-1.exe  
1 (DETECTED)  
[09:17:24] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-2.exe  
0 (undetected)  
[09:17:26] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-3.exe  
0 (undetected)  
[09:17:28] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-4.exe  
0 (undetected)  
[09:17:31] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-5.exe  
0 (undetected)  
[09:17:34] ~/D/yarGen >>> yara -c -r yargen_rules.yar ~/Downloads/memsample-beacon-win-amd64-6.exe  
0 (undetected)  
[09:17:34] ~/D/yarGen >>> [REDACTED]
```



Sample used to generate signature (DETECTED)



Samples 2 ... 6 (undetected)



Dittobytes compiled all samples based on the same code.

Evasion



Automatic Yara rule generator

UNREGISTERED

FOLDERS

- yarGen
- .dbs
- 3rdparty
- dbs
- dropzone
- screens
- tools
- .gitignore
- LICENSE
- /* prepare-release.sh
- <> README.md
- requirements.txt
- /* yarGen.py
- yargen_rules.yar

yargen_rules.yar — yarGen

```
rule beacon_win_amd64 {
    meta:
        description = "memsamples - file memsample-beacon-win-amd64-1.exe"
        author = "Tijme Gommers"
        reference = "https://github.com/Neo23x0/yarGen"
        date = "2025-08-19"
        hash1 = "5dd006e88b691ba22d6b4507ac9036e3aaaa8efb1f13552590709521a7d46b74"
    strings:
        $op0 = { 55 41 57 57 41 54 41 55 48 83 ec 70 48 8d 6c 24 }
        $op1 = { 55 41 57 57 41 54 41 55 48 83 ec 70 48 8d 6c 24 }
        $op2 = { 55 41 57 57 41 54 41 55 48 83 EC 70 48 8D 6C 24 70 }

    code:
        push    rbp
        push    r15
        push    rdi
        push    r12
        push    r13
        sub     rsp, 70h
        lea     rbp, [rsp+70h]
```

Line 15, Column 2 Spaces: 3 Plain Text

Evasion

Tijme's Signature Generator



```
1 .text:00007FF7D4581000 push rbp
2 .text:00007FF7D4581001 push r13
3 .text:00007FF7D4581003 push r15
4 .text:00007FF7D4581005 push r14
5 .text:00007FF7D4581007 push rbx
6 .text:00007FF7D4581008 push r12
7 .text:00007FF7D458100A sub rsp, 68h
8 .text:00007FF7D458100E lea rbp, [rsp+60h]
9 .text:00007FF7D4581013 mov r14, 0FD1EE24A962E9956h
10 .text:00007FF7D458101D mov rdx, 98668764F542F835h
11 .text:00007FF7D4581027 xor rdx, r14
12 .text:00007FF7D458102A mov [rbp+30h+var_31], rdx
13 .text:00007FF7D458102E mov cl, 56h ; 'V'
14 .text:00007FF7D4581030 mov [rbp+30h+var_29], 56h ; 'V'
15 .text:00007FF7D4581034 xor [rbp+30h+var_29], cl
16 .text:00007FF7D4581037 lea r15, [rbp+30h+var_58]
17 .text:00007FF7D458103B mov rcx, r15
18 .text:00007FF7D458103E call sub_7FF7D45810E0
19 .text:00007FF7D4581043 lea r13, [rbp+30h+var_58]
20 .text:00007FF7D4581047 mov rcx, r13
21 .text:00007FF7D458104A call sub_7FF7D4581340
22 .text:00007FF7D458104F mov r11b, byte ptr [rbp+30h+var_31]
```

```
1 .text:00007FF6D5471000 push rbp
2 .text:00007FF6D5471001 push r14
3 .text:00007FF6D5471003 push r12
4 .text:00007FF6D5471005 push rdi
5 .text:00007FF6D5471006 push rsi
6 .text:00007FF6D5471007 push r15
7 .text:00007FF6D5471009 push r13
8 .text:00007FF6D547100B sub rsp, 60h
9 .text:00007FF6D547100F lea rbp, [rsp+60h]
10 .text:00007FF6D5471014 mov r13, 1246845D81FC9E55h
11 .text:00007FF6D547101E mov rax, 773EE173E290FF36h
12 .text:00007FF6D5471028 xor rax, r13
13 .text:00007FF6D547102B mov [rbp+30h+var_39], rax
14 .text:00007FF6D547102F mov r9b, 2Fh ; '/'
15 .text:00007FF6D5471032 mov [rbp+30h+var_31], 2Fh ; '/'
16 .text:00007FF6D5471036 sub [rbp+30h+var_31], r9b
17 .text:00007FF6D547103A lea r9, [rbp+30h+var_60]
18 .text:00007FF6D547103E mov rcx, r9
19 .text:00007FF6D5471041 call sub_7FF6D54710E0
20 .text:00007FF6D5471046 lea r12, [rbp+30h+var_60]
21 .text:00007FF6D547104A mov rcx, r12
22 .text:00007FF6D547104D call sub_7FF6D5471330
```



Evasion

Tijme's Signature Generator



```
● ● ● ✎ 2 -zsh
[20:41:39] ~/Downloads >>> python3 sigmakeatron.py memsample-beacon-win-amd64-1.exe memsample-beacon-win-amd64-2.exe
Found 2 potential signatures:
- Byte sequence at offset 12 also found in other metamorphic sample:
  48 81 ec e8 03 00 00 48 8d ac 24 80 00 00 00
- Byte sequence at offset 22830 also found in other metamorphic sample:
  48 8b 85 00 04 00 00 4c
[20:41:39] ~/Downloads >>>
```

 Find similarities in two samples that are based on the same code.

Evasion

Tijme's Signature Generator



```
-zsh
[09:48:29] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-1.exe
1 (DETECTED)
[09:48:31] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-2.exe
1 (DETECTED)
[09:48:32] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-3.exe
0 (undetected)
[09:48:34] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-4.exe
1 (DETECTED)
[09:48:36] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-5.exe
0 (undetected)
[09:48:38] ~/Downloads >>> yara -c -r sigmakeatron.yar ~/Downloads/memsample-beacon-win-amd64-6.exe
0 (undetected)
[09:48:38] ~/Downloads >>>
```

Samples used to generate signature (DETECTED)

Another metamorphic sample is also DETECTED.

Dittobytes compiled all samples based on the same code.

Evasion

Tijme's Signature Generator



```
[20:41:39] ~/Downloads >>> python3 sigmakeatron.py memsample-beacon-win-amd64-1.exe memsample-beacon-win-amd64-2.exe
Found 2 potential signatures:
- Byte sequence at offset 12 also found in other metamorphic sample:
  48 81 ec e8 03 00 00 48 8d ac 24 80 00 00 00
- Byte sequence at offset 22830 also found in other metamorphic sample:
  48 8b 85 00 04 00 00 4c
[20:41:39] ~/Downloads >>>
```

48 81 EC E8 03 00 00	sub	rsp, 3E8h
48 8D AC 24 80 00 00 00	lea	rbp, [rsp+80h]
48 8B 85 00 04 00 00	mov	rax, [rbp+400h]
4C 8B 58 50	mov	r11, [rax+50h]



Dittobytes compiled all samples based on the same code.

Evasion

Tijme's Signature Generator



Google TI - Search - p:0 type:e X +

https://www.virustotal.com/gui/search/p%253A0%2520type%253Aexecutable%2520s%253A100%252B%2520tag%253Asigned%...

Smart search | Tijme Gommers

Seen in the last month

Search for all executables that:

- Have a valid code signature.
- Have been submitted 100+ times.
- Are not marked as malicious.
- Contain the specific byte sequence.

File Hash	Submitted By	Scan Status	Last Scan	First Scan	Size	
1b727cd6e9b6277488dcdaab444	Setup - Bloxshade.exe	1 / 100	0 / 72	2025-06-29 01:01:43	2025-08-10 14:00:16	1.3 K 11.78 MB
cc550dff570e0b7a8fc36e7146...	Logitech G HUB	1 / 100	0 / 53	2025-06-29 14:59:45	2025-08-08 02:16:40	300 32.08 MB
5121084f0561cadc856bd2350e...	OfficeClickToRun.exe	0 / 100	0 / 73	2025-07-01 21:09:47	2025-08-11 16:07:47	100 66.20 MB
4be79c358bfff313f206b29fd2a...	C:\Program Files\WizTr...	1 / 100	0 / 73	2025-07-02 01:53:23	2025-08-11 12:05:04	243 12.44 MB

Evasion



Tijme's Signature Generator

Google TI - Search - p:0 type:e X Google TI - File - 5121084f0561 X +

https://www.virustotal.com/gui/file/5121084f0561cadc856bd2350e04f5eb155d37f71cbaaf4ec9e3aadb7d002319/details

Smart search | 🔍 | ⬆️ | ⬇️ | ? | Tijme Gommers | 🚙

5121084f0561cadc856bd2350e04f5eb155d37f71cbaaf4ec9e3aadb7d002319

Signers

- Microsoft Corporation
 - Name Microsoft Corporation
 - Status Valid
 - Issuer Microsoft Windows Code Signing PCA 2024
 - Valid From 06:08 PM 04/17/2025
 - Valid To 06:08 PM 04/15/2026
 - Valid Usage Code Signing, 1.3.6.1.4.1.311.61.6.1, 1.3.6.1.4.311.76.22.6.1
 - Algorithm sha384RSA
 - Thumbprint 883B405B08B7039DEECA731F53EB236E3CF198A3
 - Serial Number 33 00 00 00 54 53 E5 67 39 FB 6E 57 BB 00 00 00 00 54
- + Microsoft Windows Code Signing PCA 2024
- + Microsoft Root Certificate Authority 2010

Counter Signers

- + Microsoft Time-Stamp Service
- + Microsoft Time-Stamp PCA 2010
- + Microsoft Root Certificate Authority 2010

OfficeClickToRun.exe

Evasion



Tijme's Signature Generator

Google TI - Search - p:0 type:e X Google TI - File - 0b7c0f0bebb X +

https://www.virustotal.com/gui/file/0b7c0f0bebb593ecfd4a2486454b41bb89aaf323ac8ed91c528107356f2d6442/details

Smart search | Tijme Gommers

0b7c0f0bebb593ecfd4a2486454b41bb89aaf323ac8ed91c528107356f2d6442

Signers

- Rockstar Games, Inc.
 - Name Rockstar Games, Inc.
 - Status Valid
 - Issuer DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1
 - Valid From 12:00 AM 01/18/2023
 - Valid To 11:59 PM 01/20/2026
 - Valid Usage Code Signing
 - Algorithm sha256RSA
 - Thumbprint 565932392989B3616F2968E1B1D6F974561B1F32
 - Serial Number 0D 88 C0 8F 56 6D 2B 1F 0C 19 4D B1 F8 CA C9 A9
- + DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1
- + DigiCert Trusted Root G4
- + DigiCert

Counter Signers

- + DigiCert Timestamp 2024
- + DigiCert Trusted G4 RSA4096 SHA256 TimeStamping CA

Rockstar-Games-Launcher.exe

So how do we detect it?

Entropy-based detection

No, metamorphifications do not encrypt or compress (parts of the code/binary).

RWX-memory detection

No, metamorphic malware does not need to allocate writable and executable memory regions.

Stack-based detection

No, stack-based strings/sequences not present if you use existing string encryption techniques.

API-import detection

No, Dittobytes produces Position Independent Code (PIC), which does not use the import table.

API-call sequence detection

Detection possible if your malware produces a suspicious sequence of API-calls.

Network traffic detection

Detection possible if your malware produces a suspicious network traffic pattern.

In Memory of In-Memory Detection

Thank you

Questions? I'm around all day at OrangeCon!

