Throwing Shurikens: El formato DEX y el análisis estático

Eduardo Blázquez

¿Quién soy?

- Eduardo Blázquez González
- Doctor en Informática por la UC3M
- Java Compiler Engineer en Quarkslab
- Principal desarrollador de Shuriken-Analyzer

¿Qué vamos a ver hoy?

1. Análisis estático de formatos binarios
2. Formato DEX (Dalvik EXecutable)
3. Shuriken-Analyzer
4. Manos a la obra :D

Análisis estático de formatos binarios

Análisis estático de binarios

- Proceso de razonamiento de un binario sin necesidad de ejecutarlo, y con la intención de extraer información sobre este.

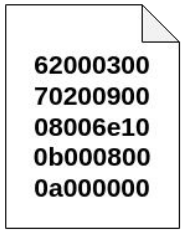
- Necesitamos realizar diferentes procesamientos y transformaciones para poder extraer de información, sin necesidad del código fuente.

- Diferentes procesos a realizar: parsing de las estructuras del binario, desensamblado de las instrucciones, análisis del flujo de control y análisis del flujo de datos.

- Diferentes usos (legítimos y no legítimos):
  - Análisis de Malware
  - Recompilación de binarios a nuevos/otras arquitecturas
  - Análisis de Vulnerabilidades
  - Modificación del Software
  - Extracción de propiedad intelectual
  - …

Ejemplo de pipeline de Análisis

Binario/Bytecode



Parsing - 1

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                        34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

MINI

ELF HEADER

IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001

PROGRAM HEADER TABLE

EXECUTION INFORMATION

p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

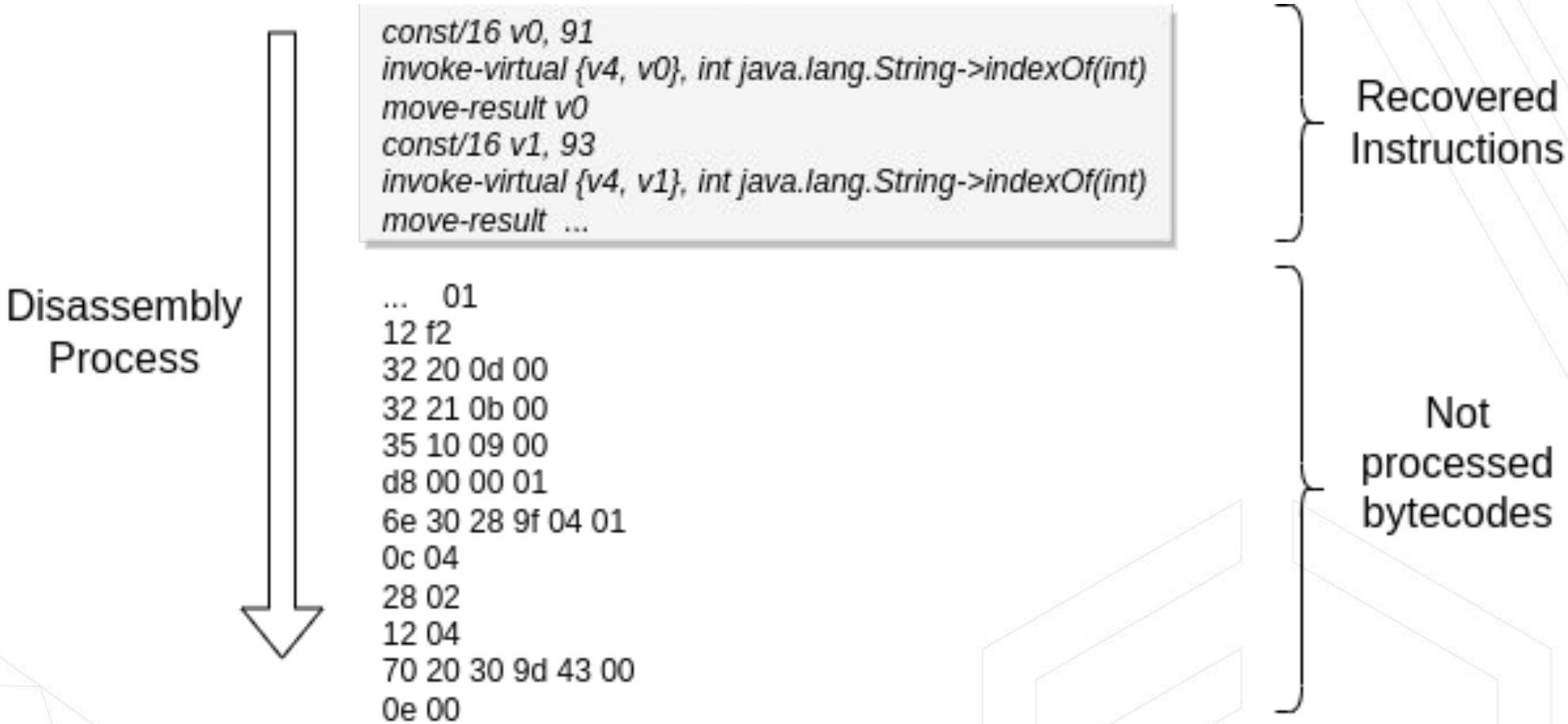
CODE

X86 ASSEMBLY	EQUIVALENT C CODE
mov ebx, 42	return 42;
mov eax, SC_EXIT[1]	
int 80h	

Parsing - 2

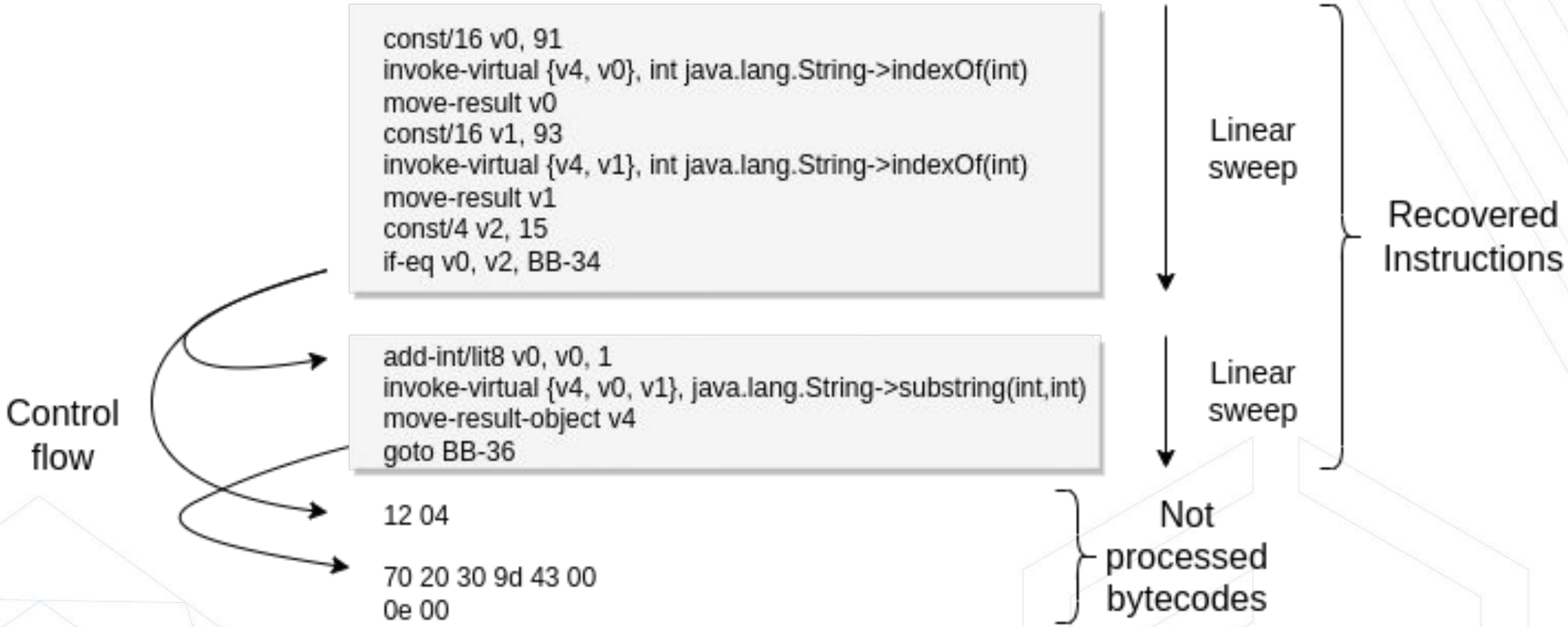
```
Elf Program Header:
TYPE           FLAGS            Offset           V.Addr
           P.Addr           F.Size           M.Size             Align

PHDR           R              0x0000000000000040 0x0000000000000040
         0x0000000000000040 0x00000000000002d8 0x00000000000002d8 0x8

INTERP         R              0x0000000000000318 0x0000000000000318
         0x0000000000000318 0x000000000000001c 0x000000000000001c 0x1
       [Program Interpreter: /lib64/ld-linux-x86-64.so.2]

LOAD           R              0x0000000000000000 0x0000000000000000
         0x0000000000000000 0x00000000000036f8 0x00000000000036f8 0x1000       (DATA)

LOAD           R X            0x0000000000004000 0x0000000000004000
         0x0000000000004000 0x0000000000014db1 0x0000000000014db1 0x1000       (TEXT)

LOAD           R              0x0000000000019000 0x0000000000019000
         0x0000000000019000 0x00000000000071b8 0x00000000000071b8 0x1000       (DATA)

LOAD           RW             0x0000000000020f30 0x0000000000021f30
         0x0000000000021f30 0x0000000000001348 0x00000000000025e8 0x1000       (DATA)

DYNAMIC        RW             0x0000000000021a38 0x0000000000022a38
         0x0000000000022a38 0x0000000000000200 0x0000000000000200 0x8
```

Desensamblado - 1



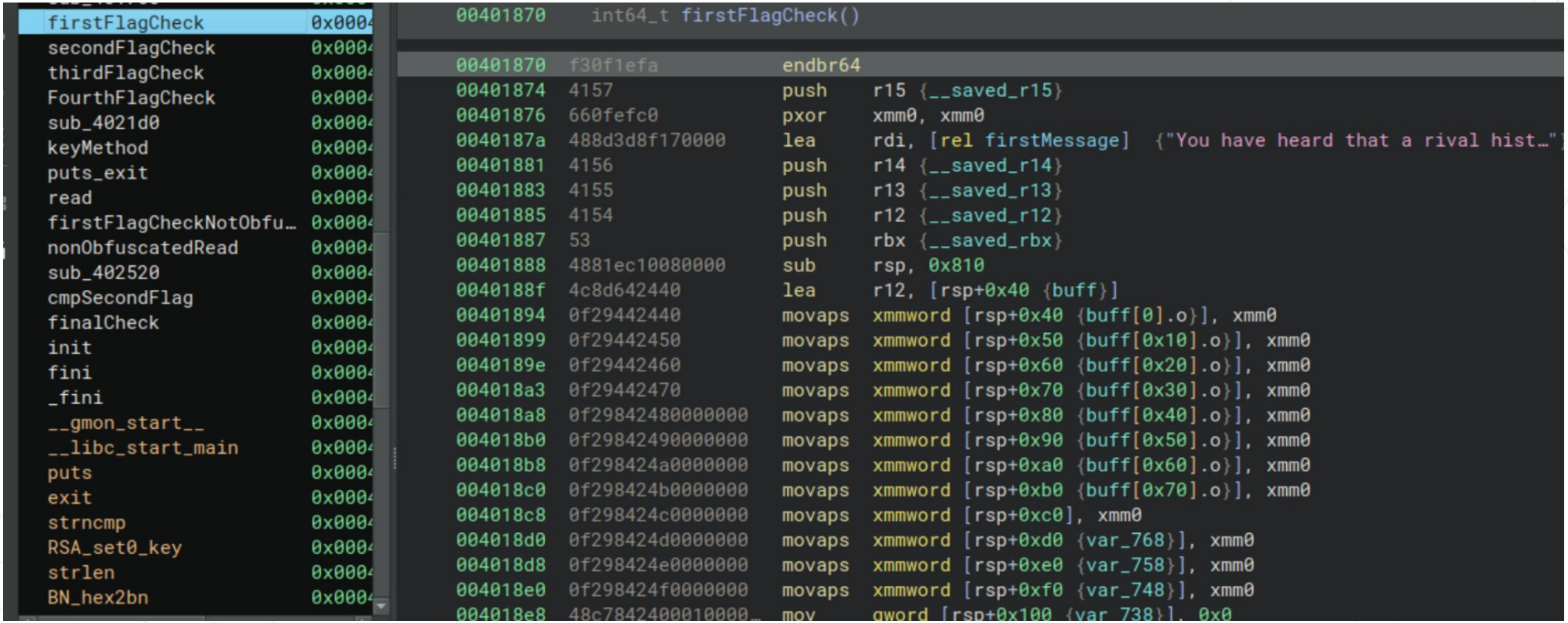
Desensamblado Linear Sweep

Desensamblado - 2



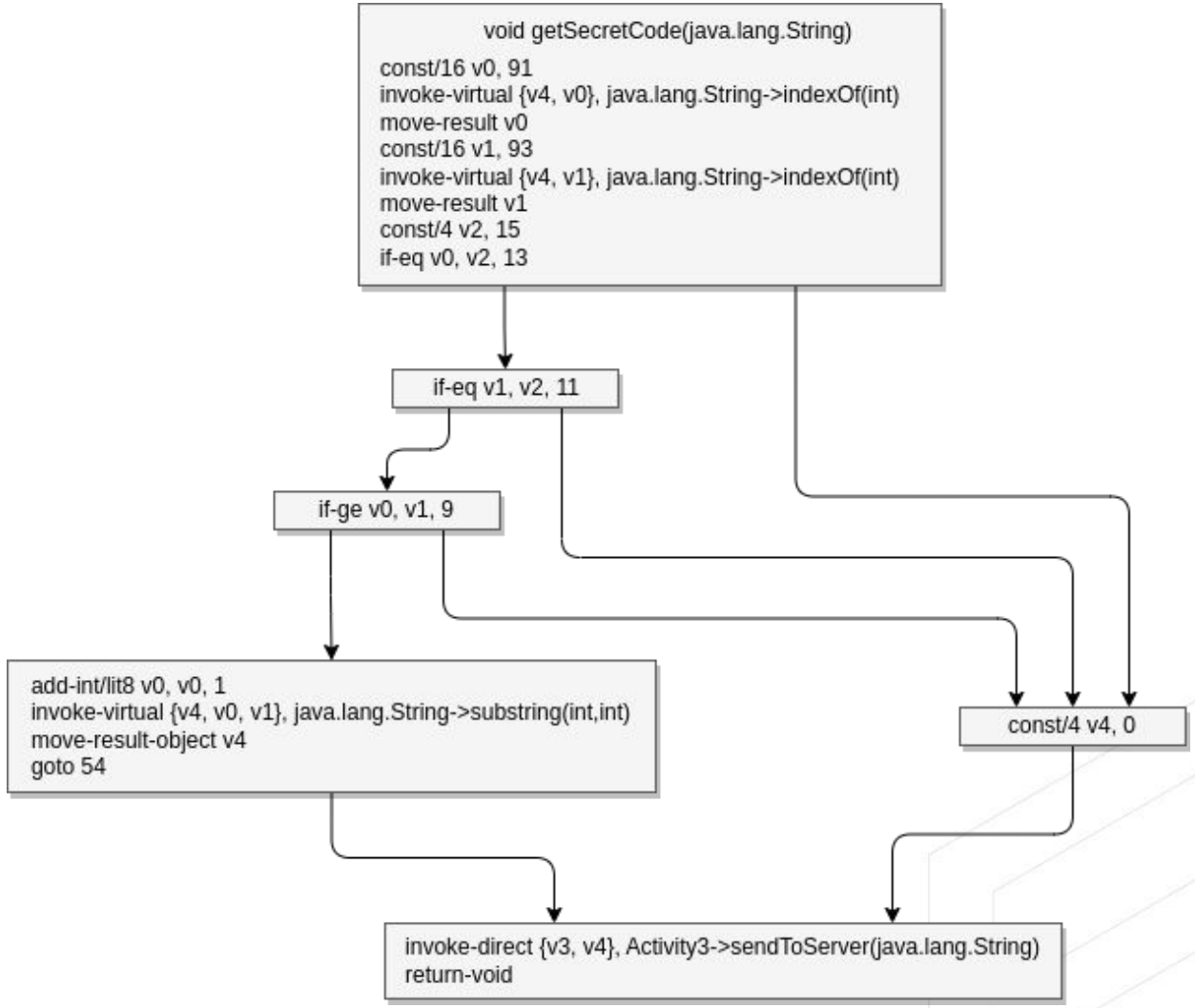
Desensamblado Recursive Traversal

Desensamblado - 3 & Análisis - 1



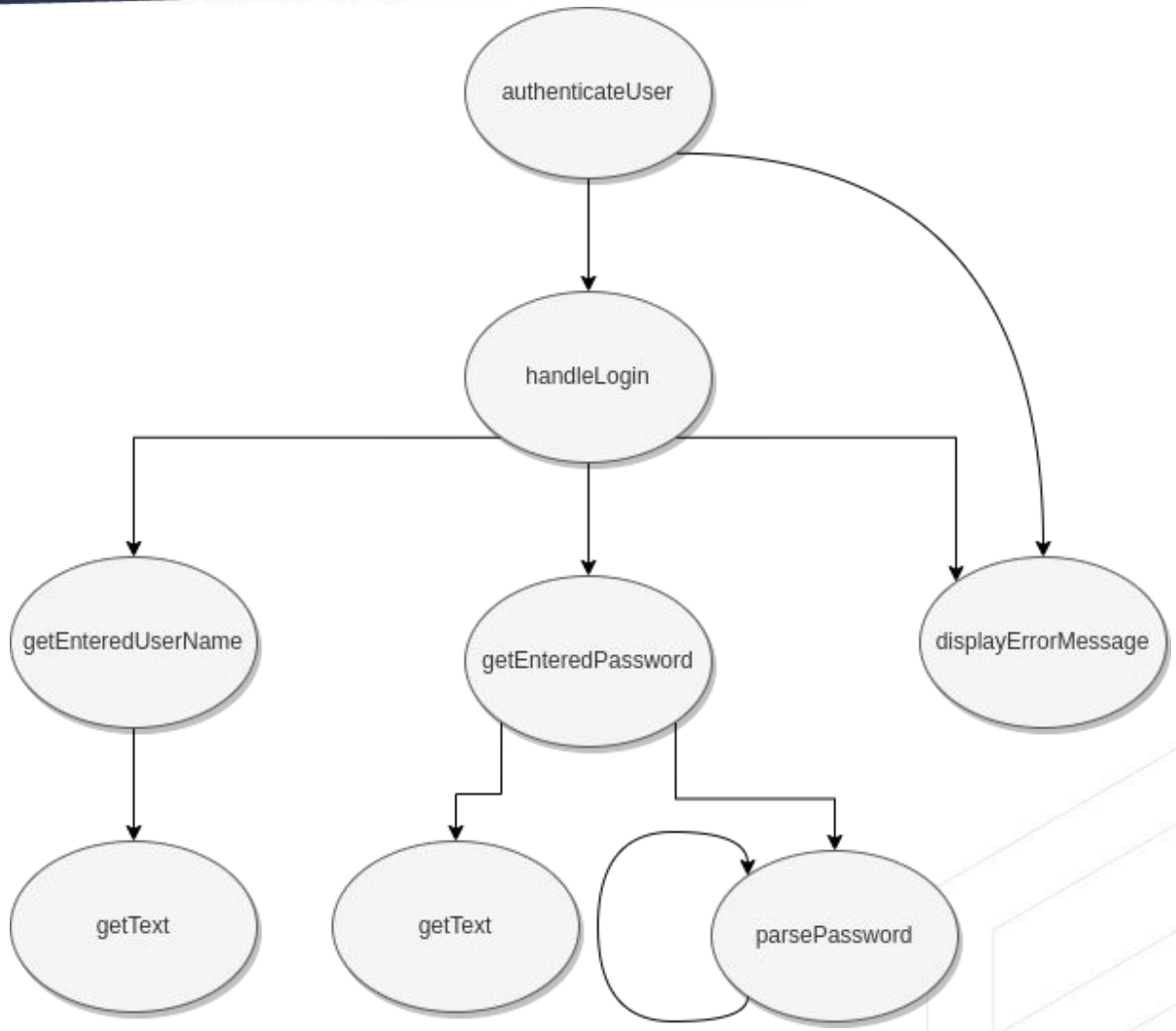
Presentación de Desensamblado y Reconocimiento de Funciones

Análisis - 2



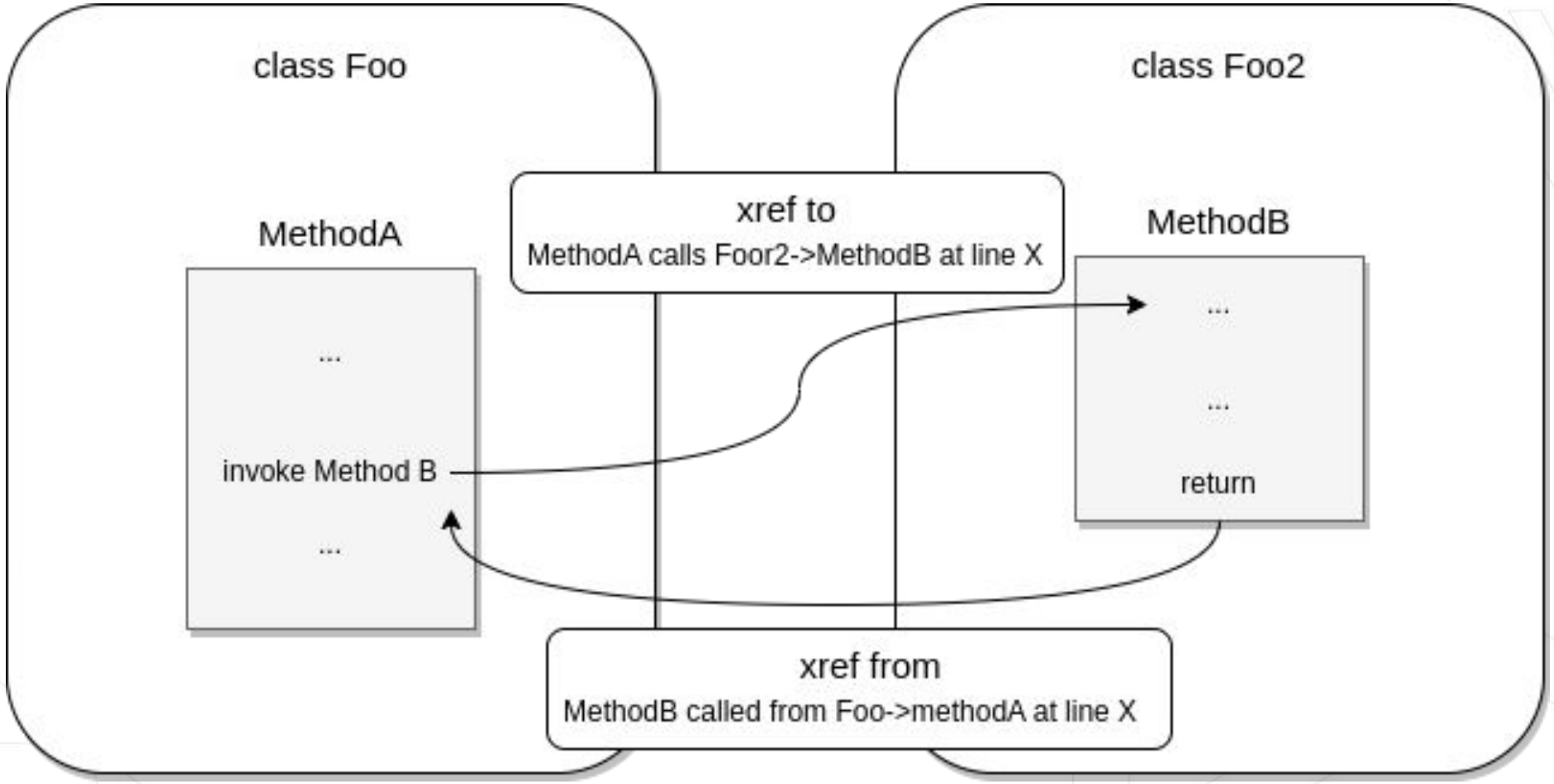
Control-Flow Graph

Análisis - 3



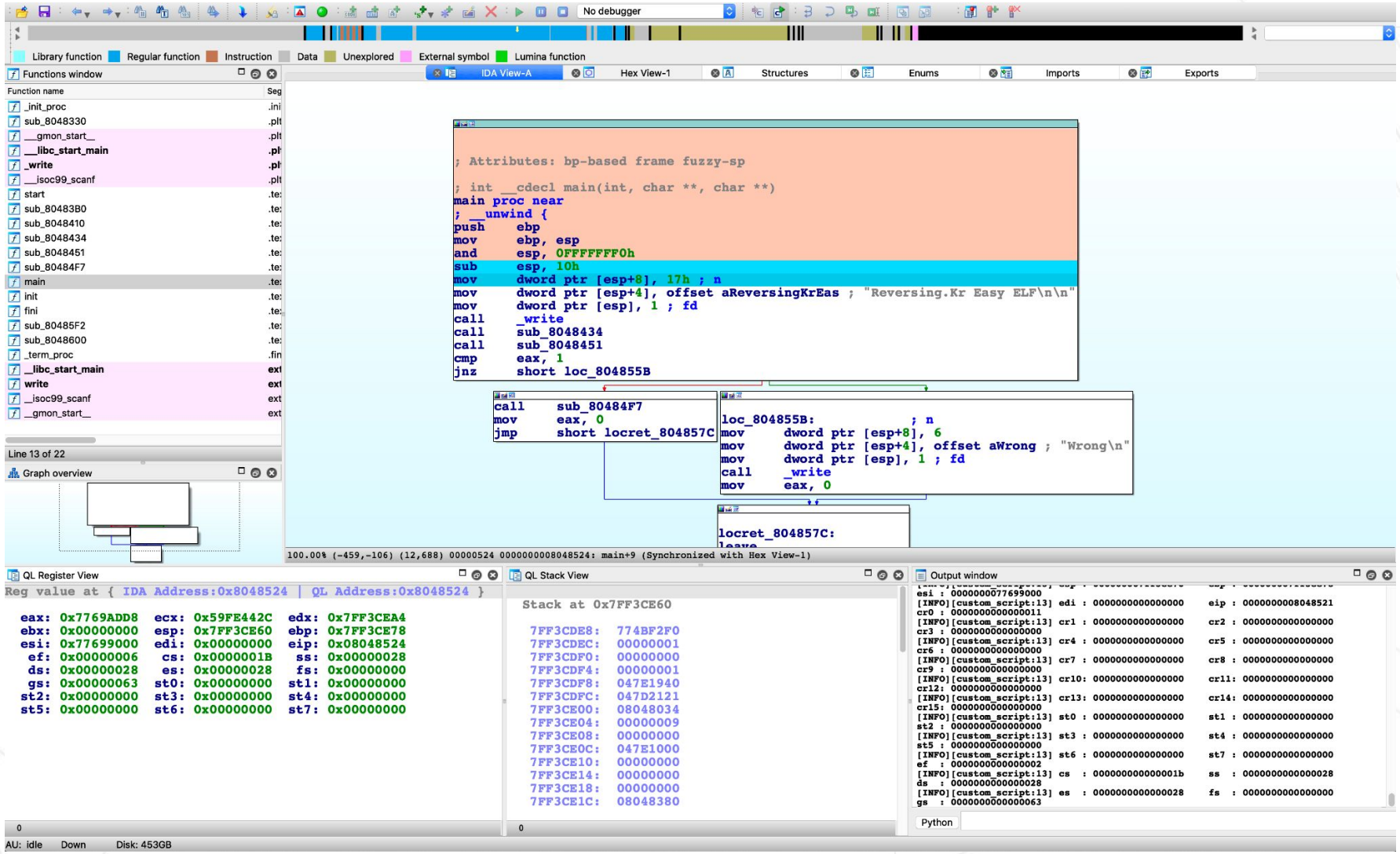
Call Graph

Análisis - 4

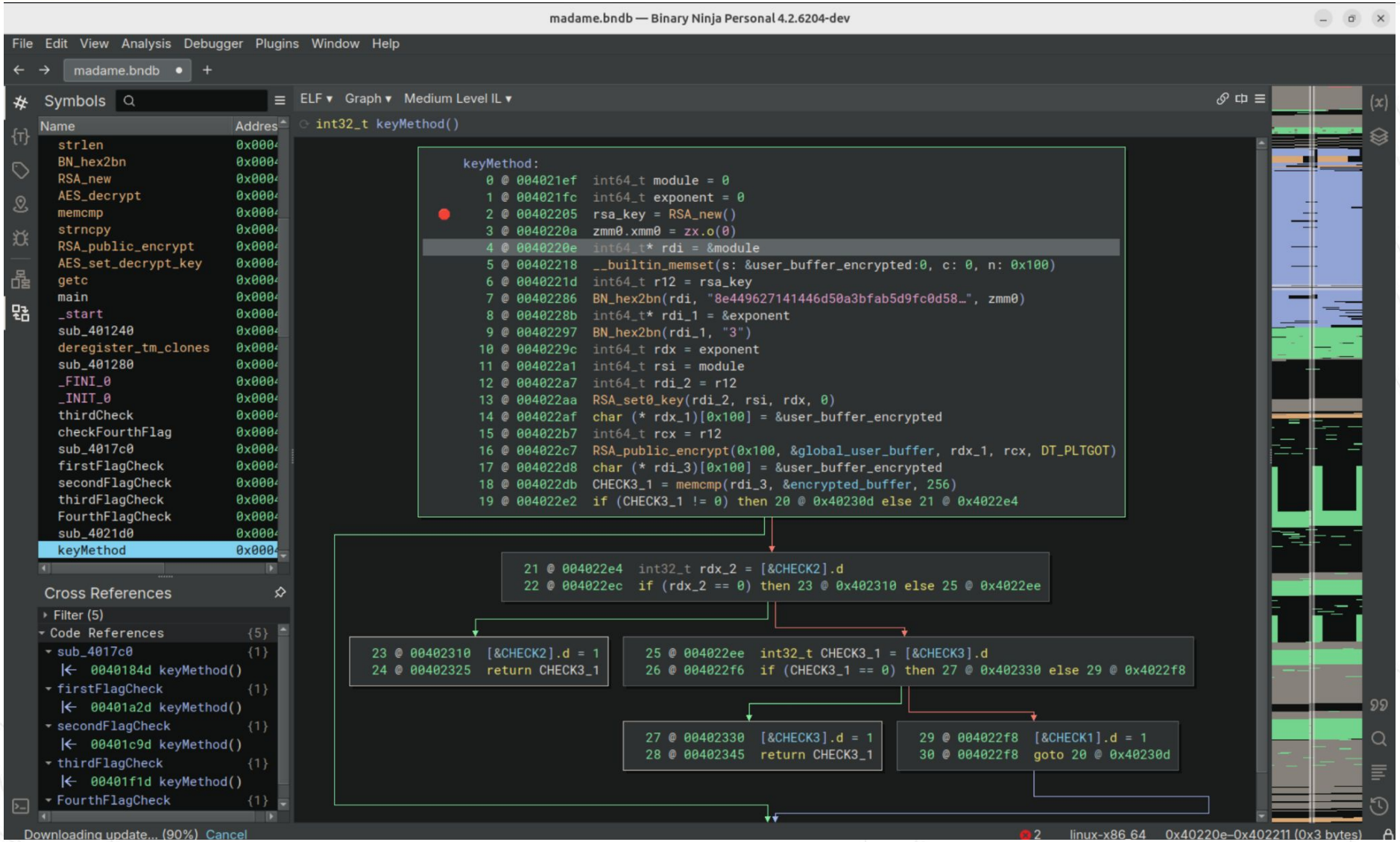


Cross-References

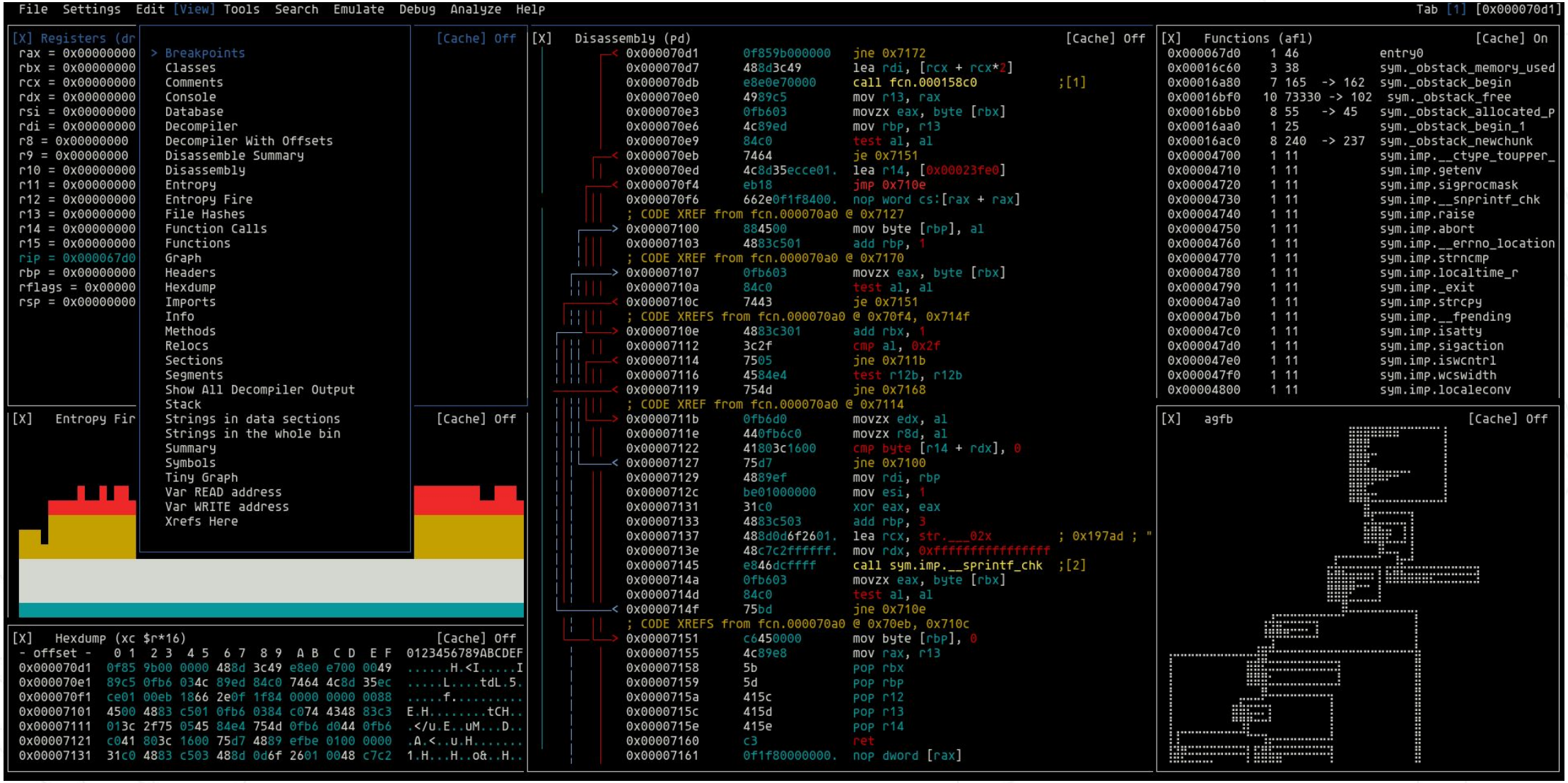
Herramientas de Análisis Estático - IDA Pro



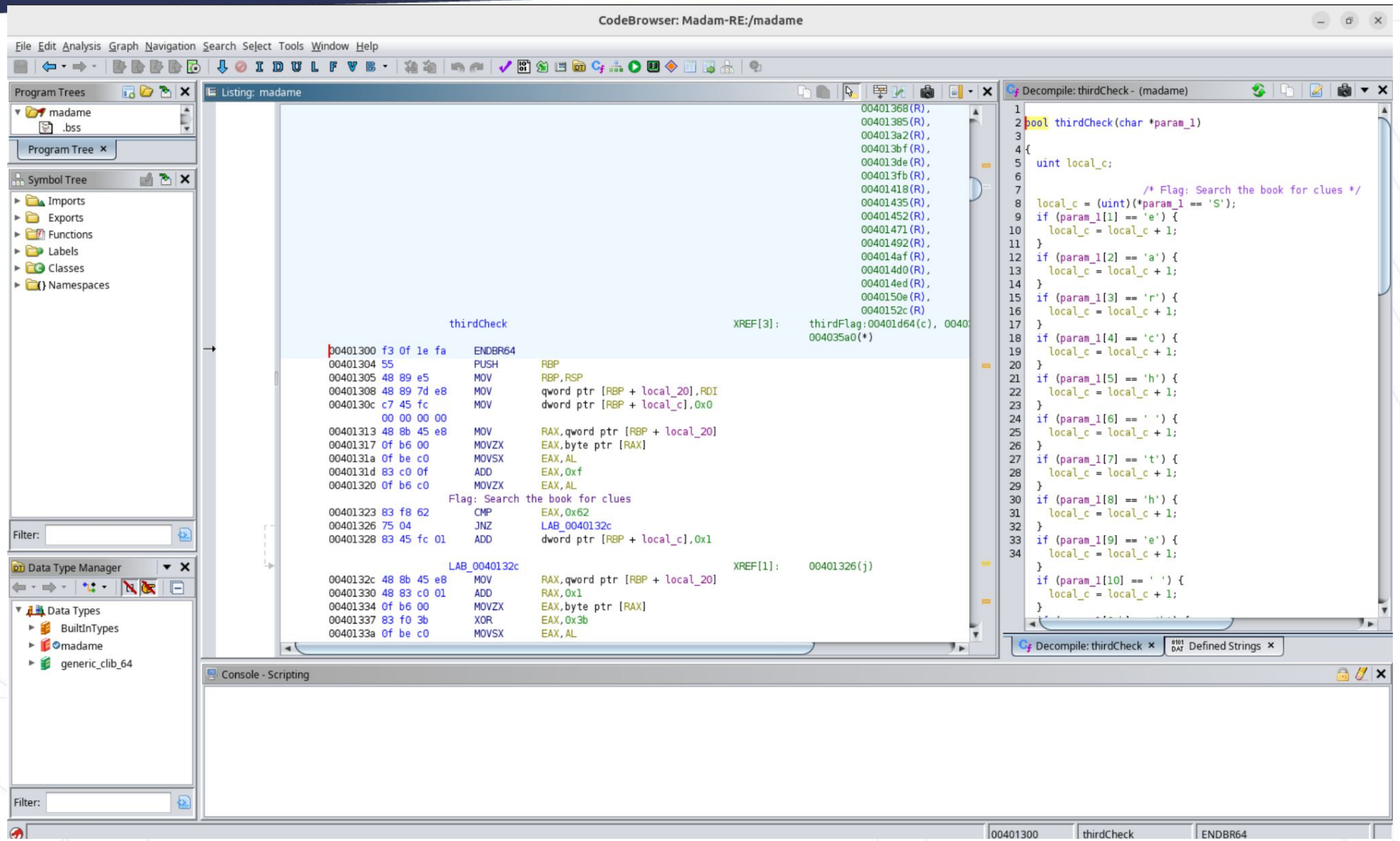
Herramientas de Análisis Estático - Binary Ninja



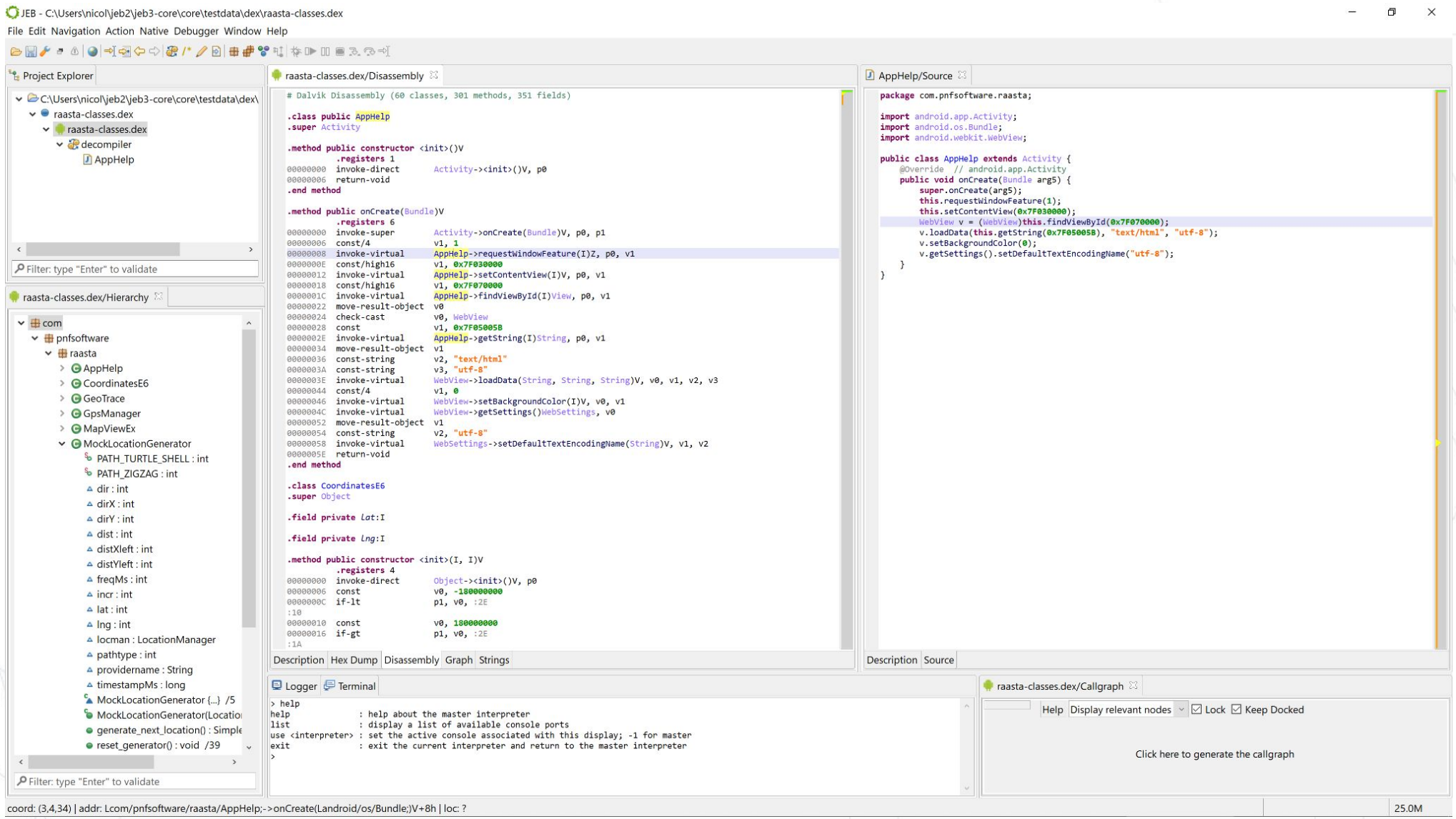
Herramientas de Análisis Estático - Radare2



Herramientas de Análisis Estático - Ghidra

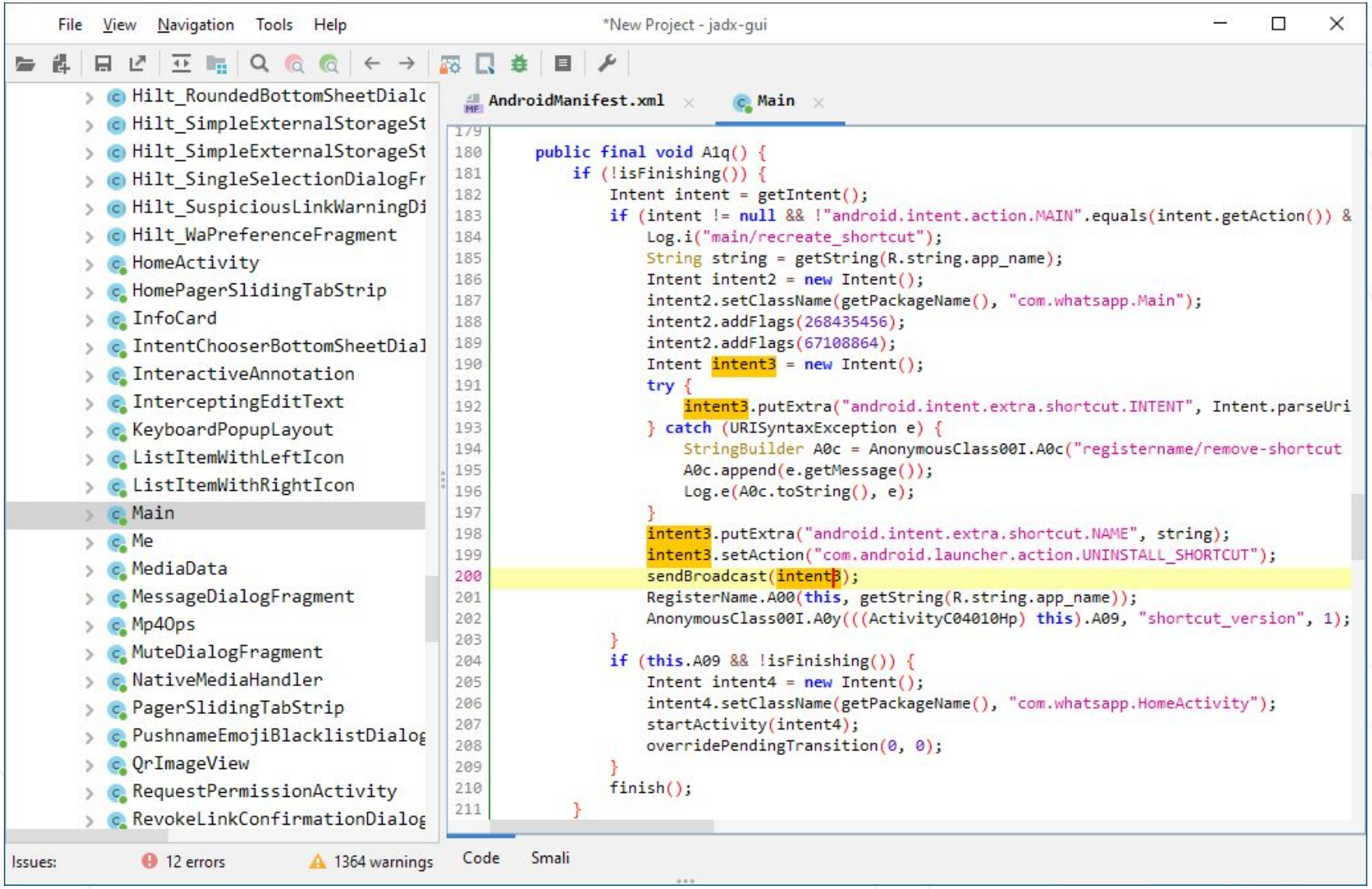


Herramientas de Análisis Estático - JEB



Quarkslab

Herramientas de Análisis Estático - Jadx



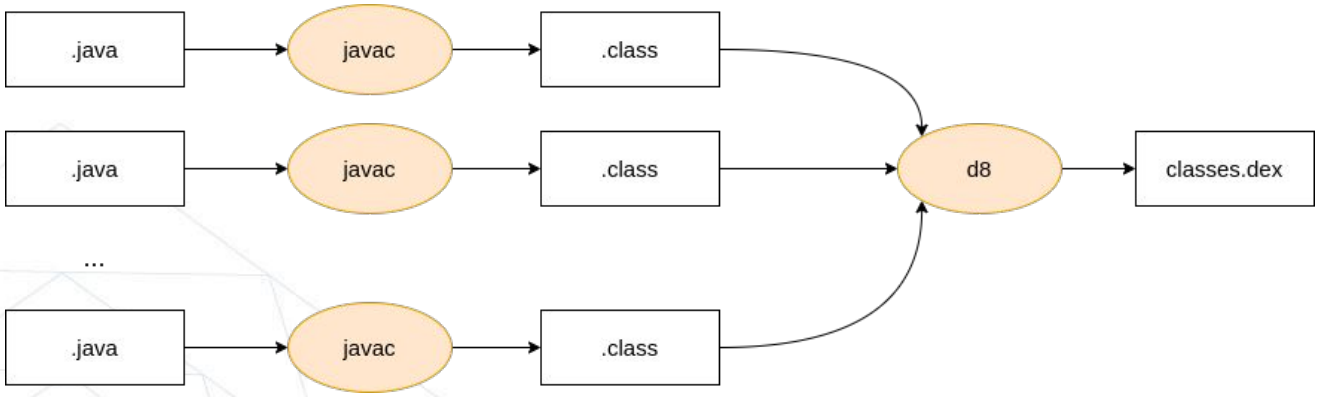
Herramientas de Análisis Estático - …

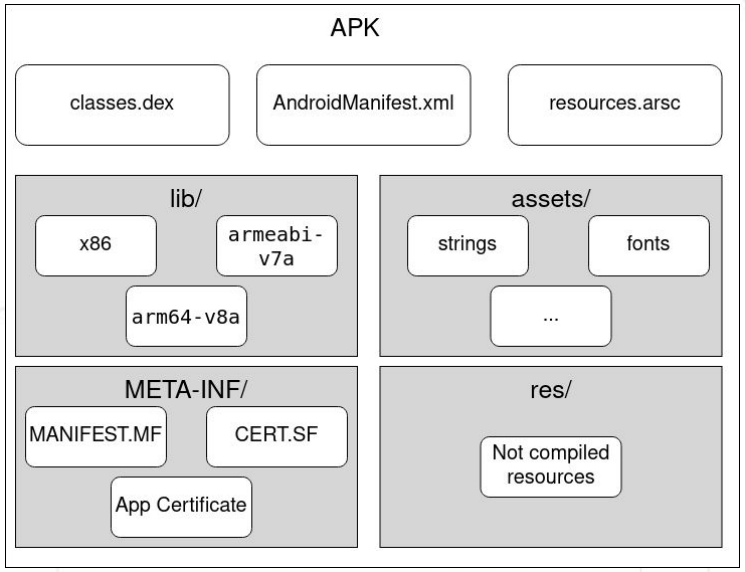
- Tenemos también librerías con las que nosotros podemos crear nuestras propias herramientas de análisis estático:
  - Lief: parser creado por Romain Thomas (ex-Quarkslab) para analizar las cabeceras de distintos tipos de archivo (ELF de Linux, PE de Windows, DEX, Mach-O…)
  - Capstone/Keystone: Desensamblador y Ensamblador creados por Nguyen Anh Quynh en principio basado en el código de LLVM.
  - Zydis: Desensamblador para x86 y x86-64, parecido a Capstone, pero sin soporte para otras arquitecturas.
  - Nyxtone: Desensamblador y Ensamblador creado por Emproof, basado en LLVM, escrito en C++.
  - ASM: Librería para manipular bytecode de Java, escrita en Java.
  - …
- Por mi parte, en mi github podéis encontrar:
  - elfparser_e: un simple parser para ELF, escrito en C ofrece las estructuras básicas y tiene un binding en Python.
  - Shuriken-Library: librería escrita por Shuriken-Group para el análisis de DEX y APK (de momento), escrita en C++ con una API en C y un binding en Python.

Formato DEX

Archivos Dalvik EXecutable

- Dentro de un APK, la forma de distribuir el código de las aplicaciones es a través de archivos Dalvik EXecutable (DEX).
- Antiguamente el código de un archivo DEX era ejecutado por la Dalvik VirtualMachine (DVM), actualmente es compilado a nativo y ejecutado por el Android RunTime (ART).
- Los archivos DEX tienen ciertas limitaciones, por ejemplo, el máximo número de clases dentro de un DEX son 65535, si una aplicación tiene más clases, un APK contendrá más de un DEX.
- Los archivos DEX son generados (normalmente) a partir de código Java/Kotlin, que se compila a archivos “class” y finalmente se juntan en un archivo DEX.

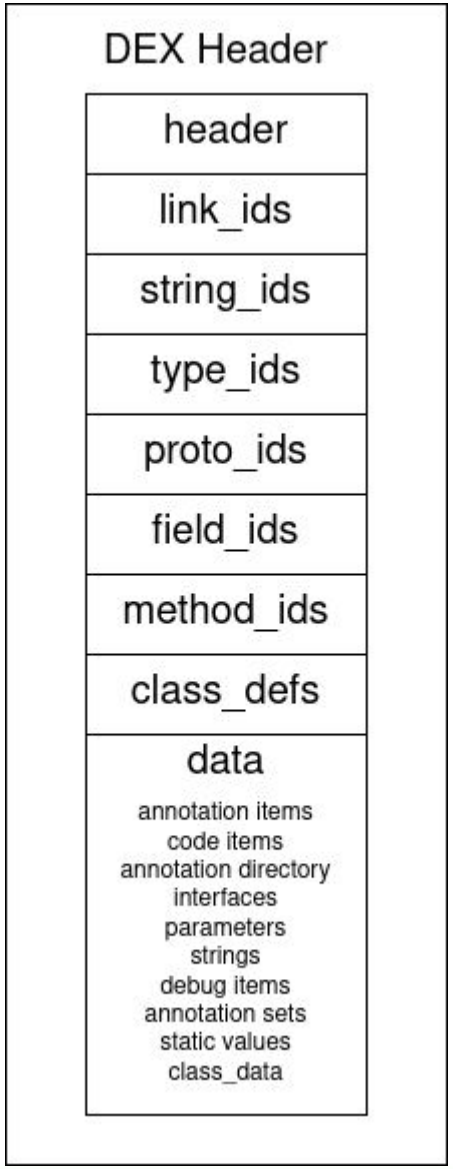




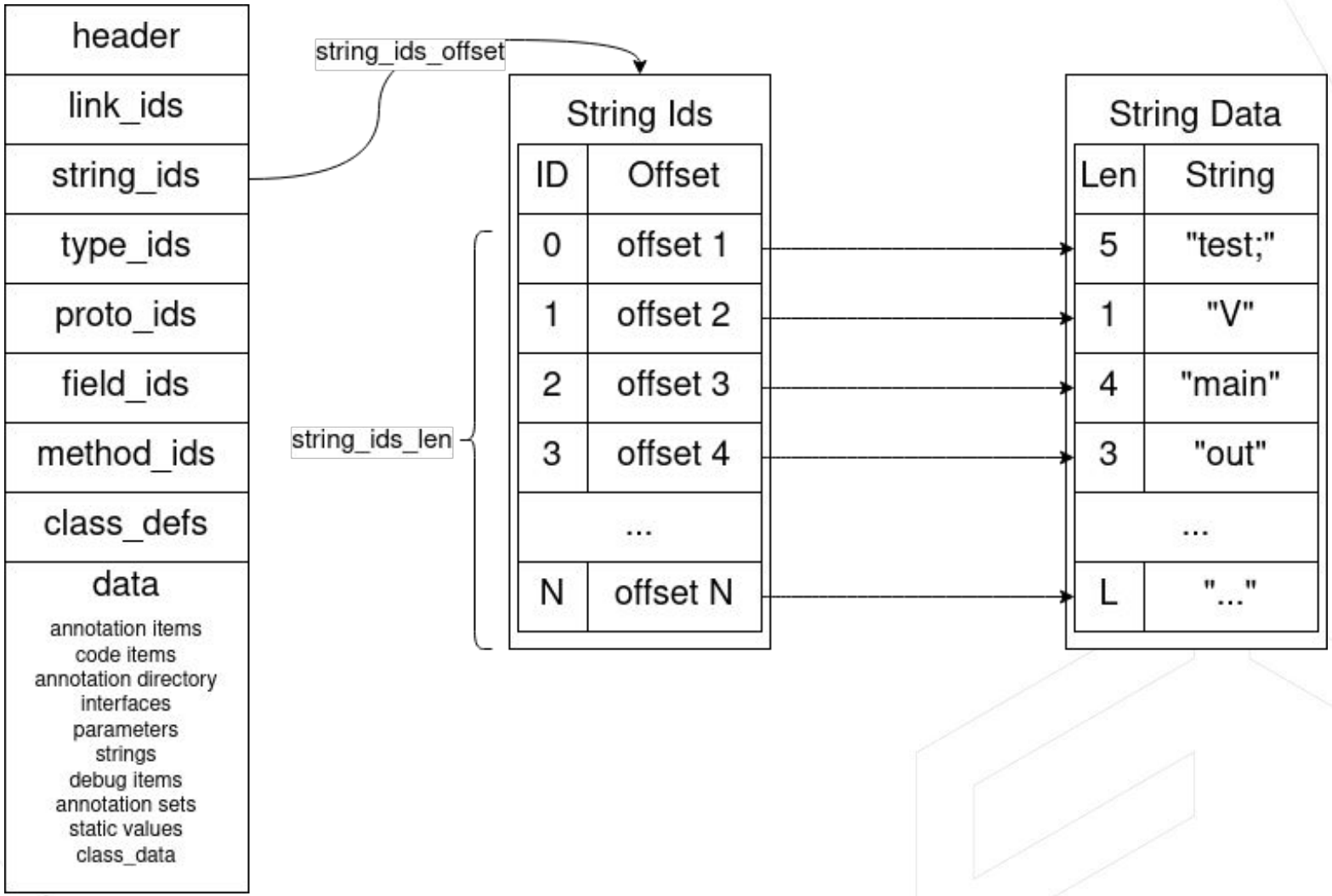
Formato DEX

- El formato DEX está principalmente formado por tablas, estas tablas tienen contenido o índices a filas de otras tablas.
  - Una **cabecera DEX** al principio del archivo que indica la estructura del archivo, esta cabecera contiene offsets y tamaños de otras partes de la cabecera. Podemos encontrar otros datos como el magic number, etc.
  - **String IDs**, esta tabla contiene todas las strings usadas en el DEX representadas en un formato conocido como *uleb128*. Las demás cabeceras apuntarán a esta para nombres de clases, métodos, tipos, etc.
  - **Type IDs**, todos los tipos usados en el código Java. Tenemos los tipos fundamentales especificados con una sola letra (e.g. Z para booleanos, B para bytes, D para double, V void, etc), así como las clases usadas de las librerías y las creadas por el programador.
  - **Proto IDs**, tabla que contiene la estructura de los métodos, especificando el tipo de retorno y los parámetros.
  - **Field IDs**, tabla con información de todos los *fields* del código, tipo y clase a la que pertenece.
  - **Method IDs**, tabla con todos los métodos de la aplicación, contiene información de la clase a la que pertenece, también apunta a una entrada a la tabla Proto ID con el prototipo del método.
  - **Class Defs**, tabla con información sobre las clases de una aplicación. Esta estructura apunta a otras estructuras más complejas como **AnnotationDirectoryItem** con información sobre *fields, métodos y parámetros.* **ClassDataItem** con más información sobre *fields,* y *métodos*.

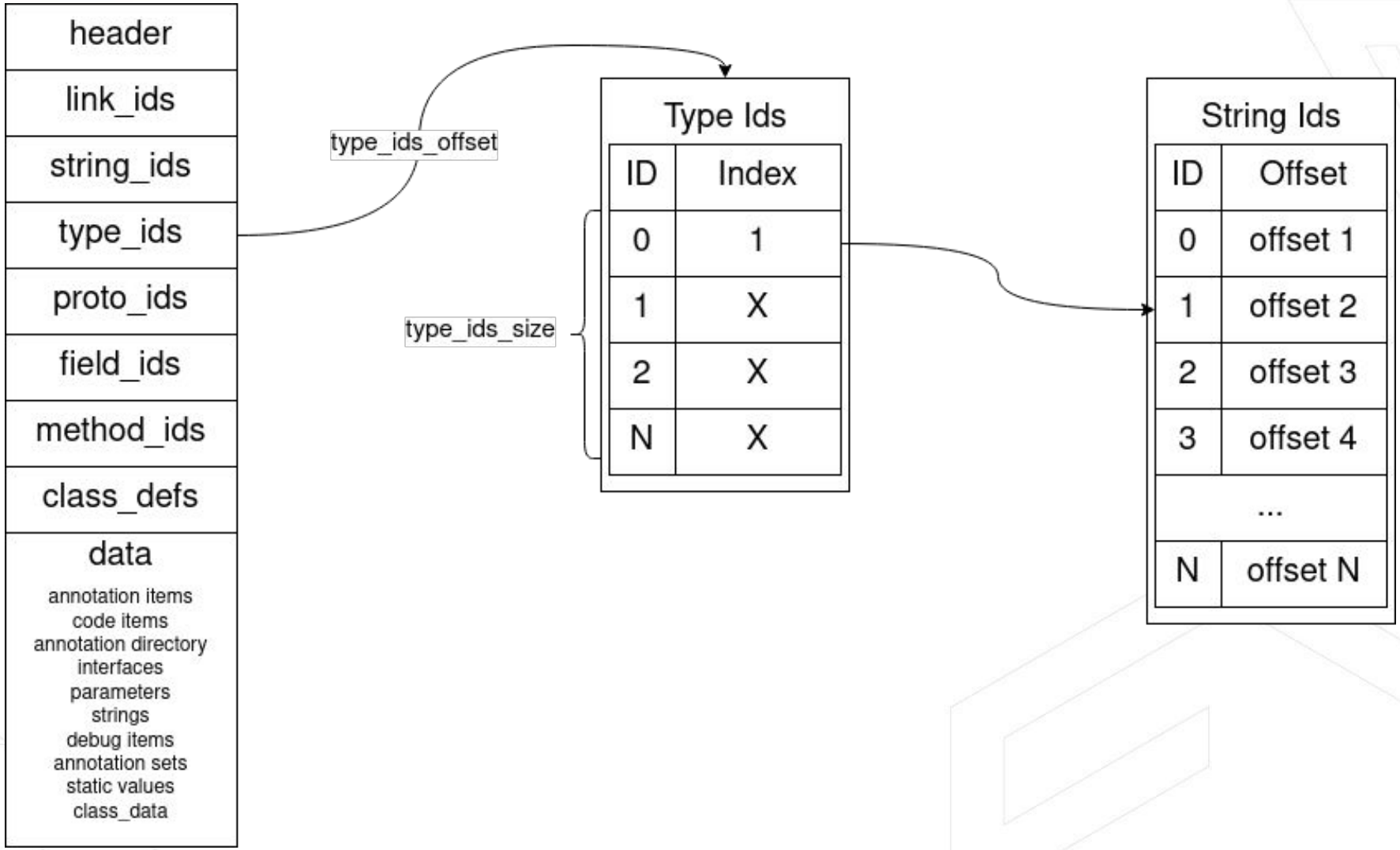
Formato DEX - Cabecera



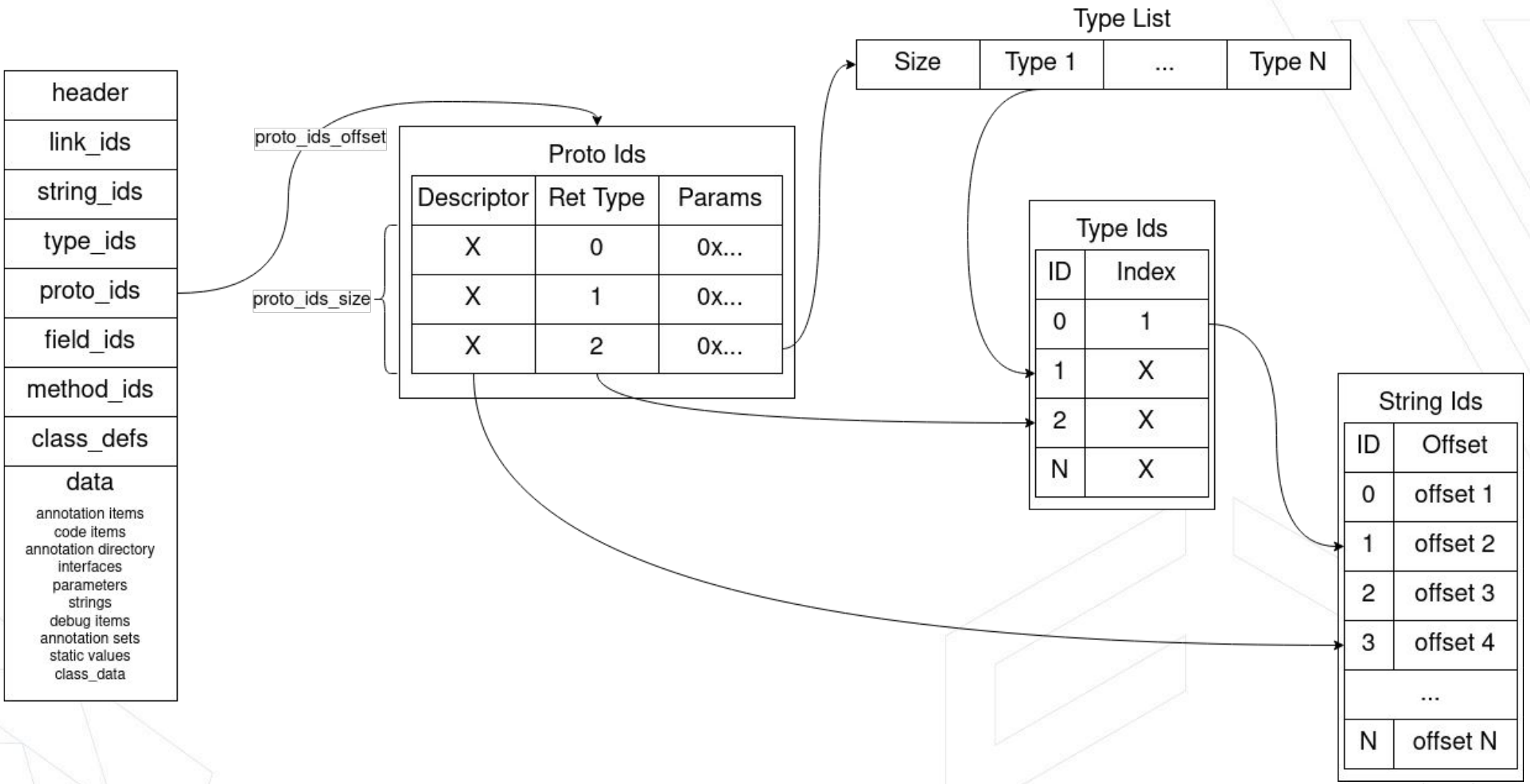
Formato DEX - String IDs



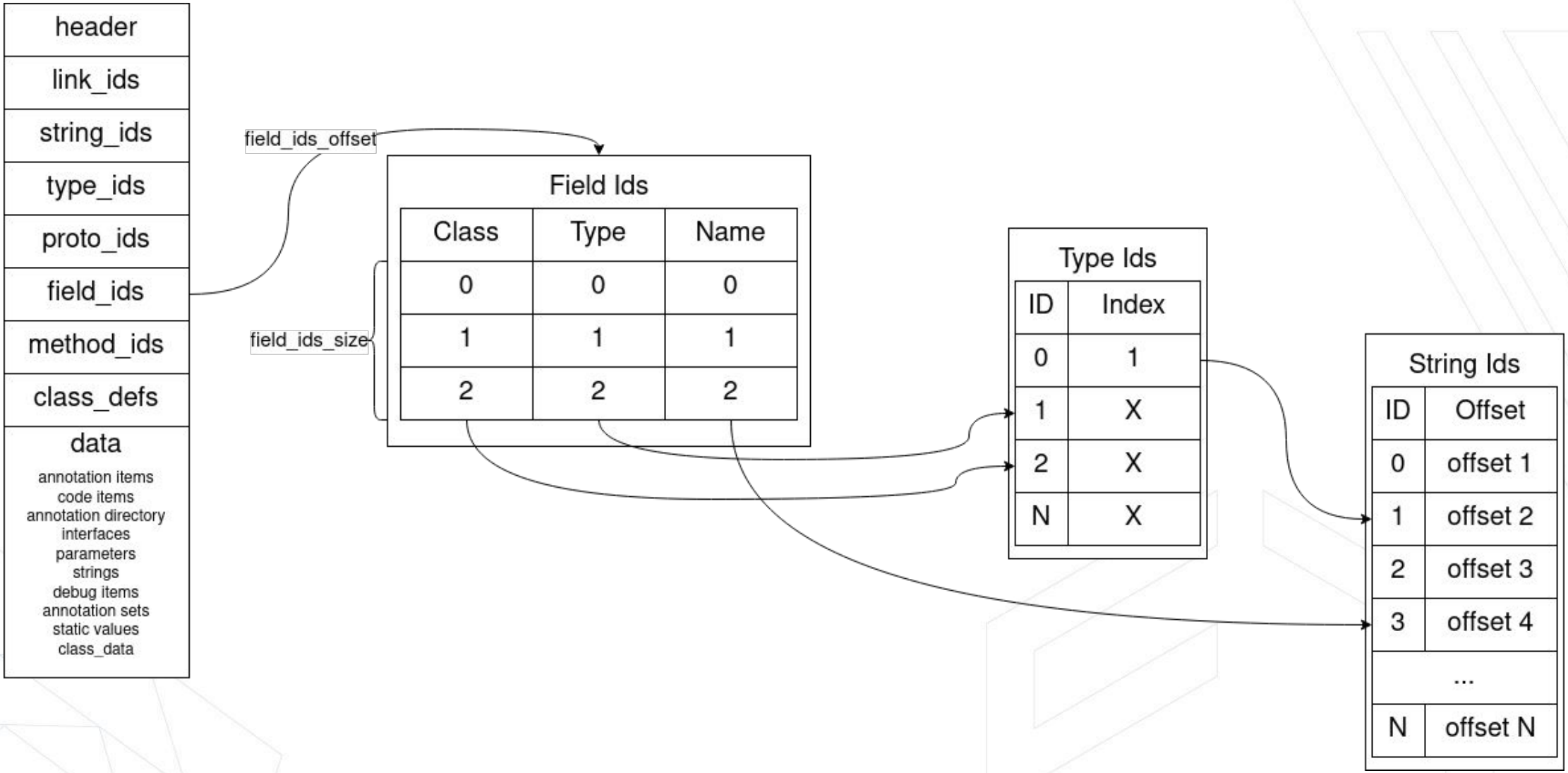
Formato DEX - Type IDs



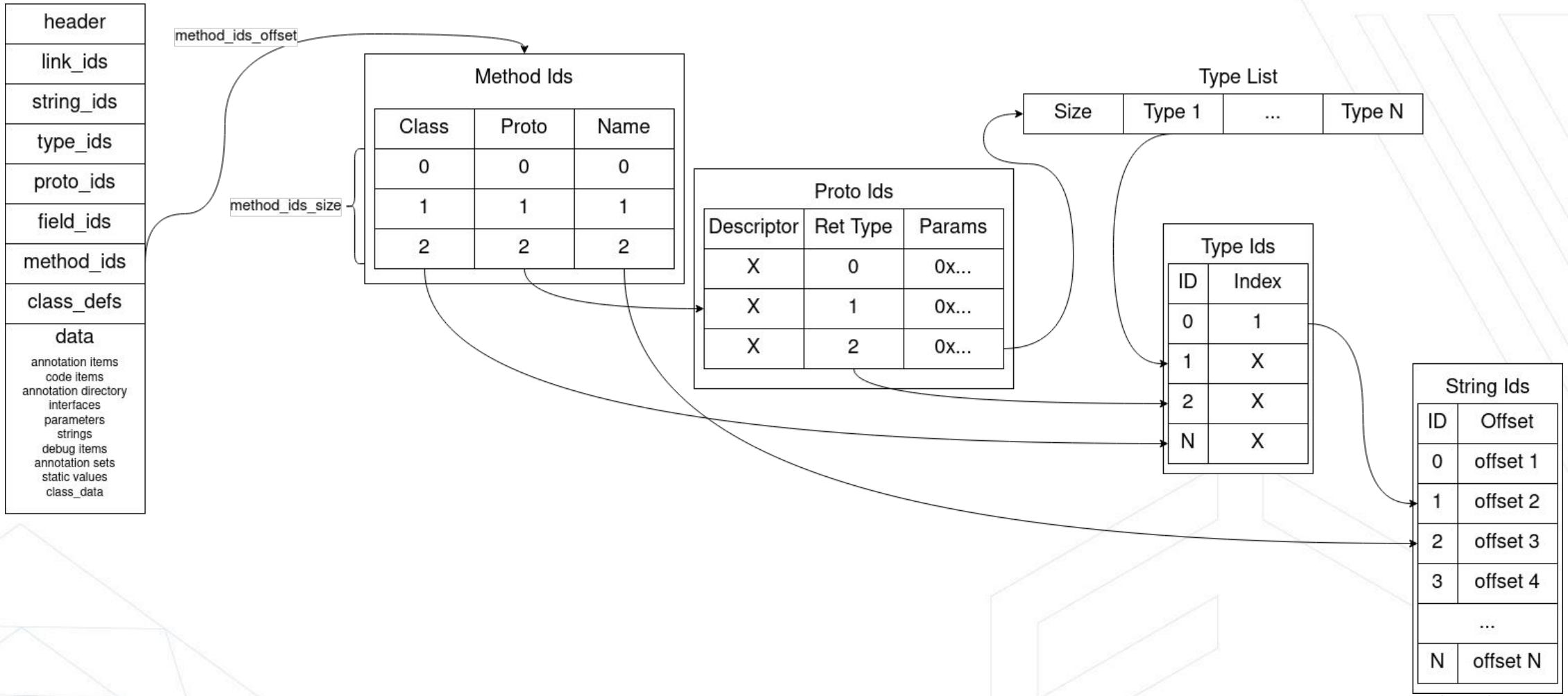
Formato DEX - Proto IDs



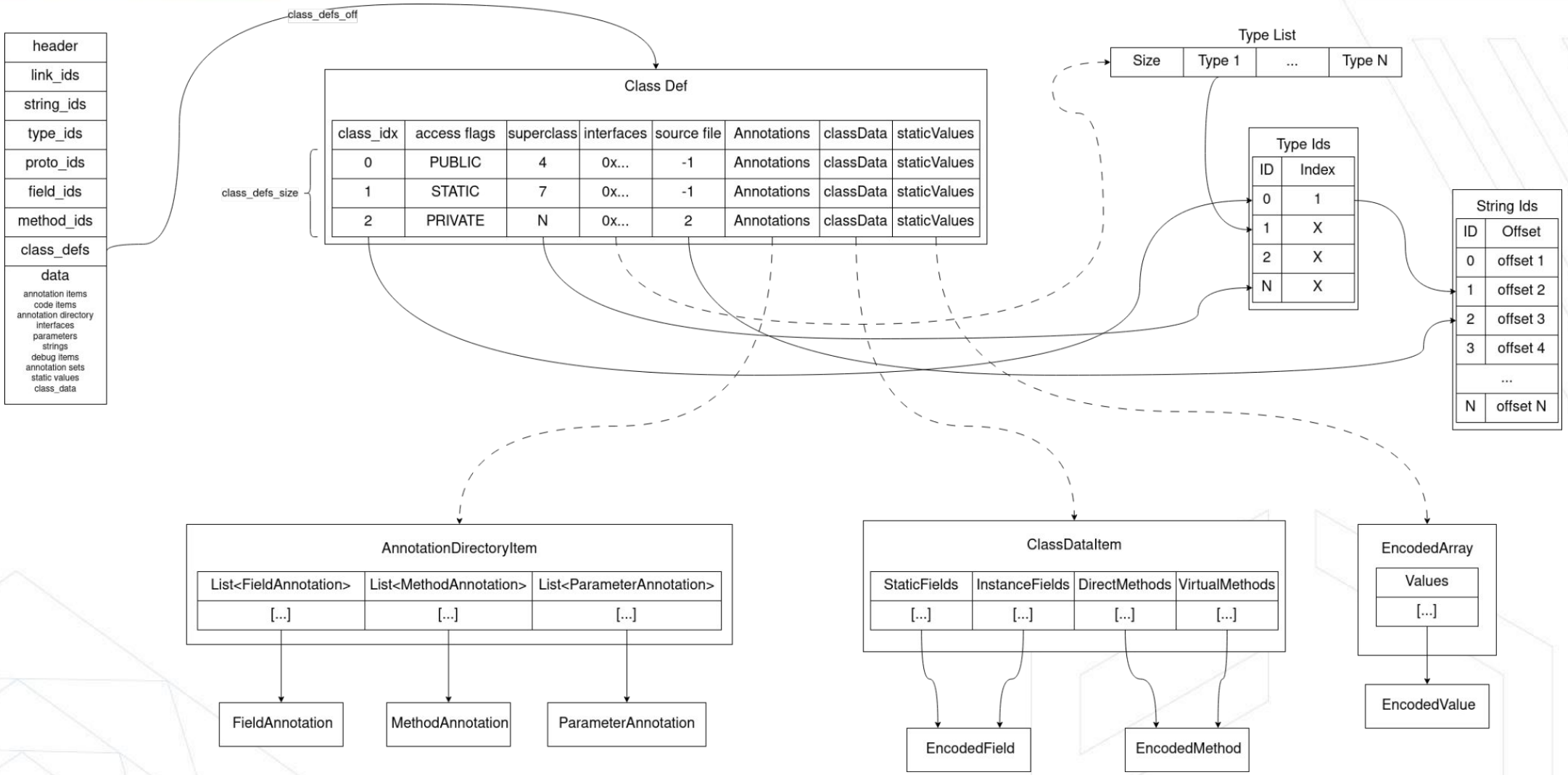
Formato DEX - Field IDs

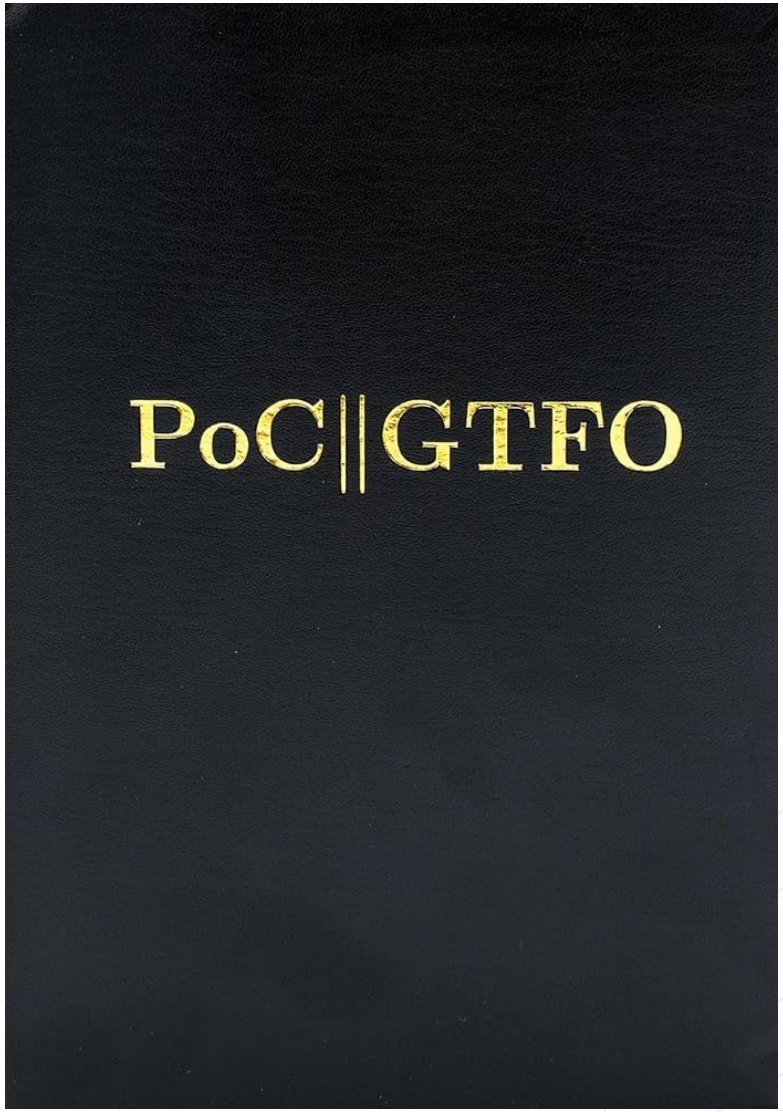


Formato DEX - Method IDs



Formato DEX - Class Defs



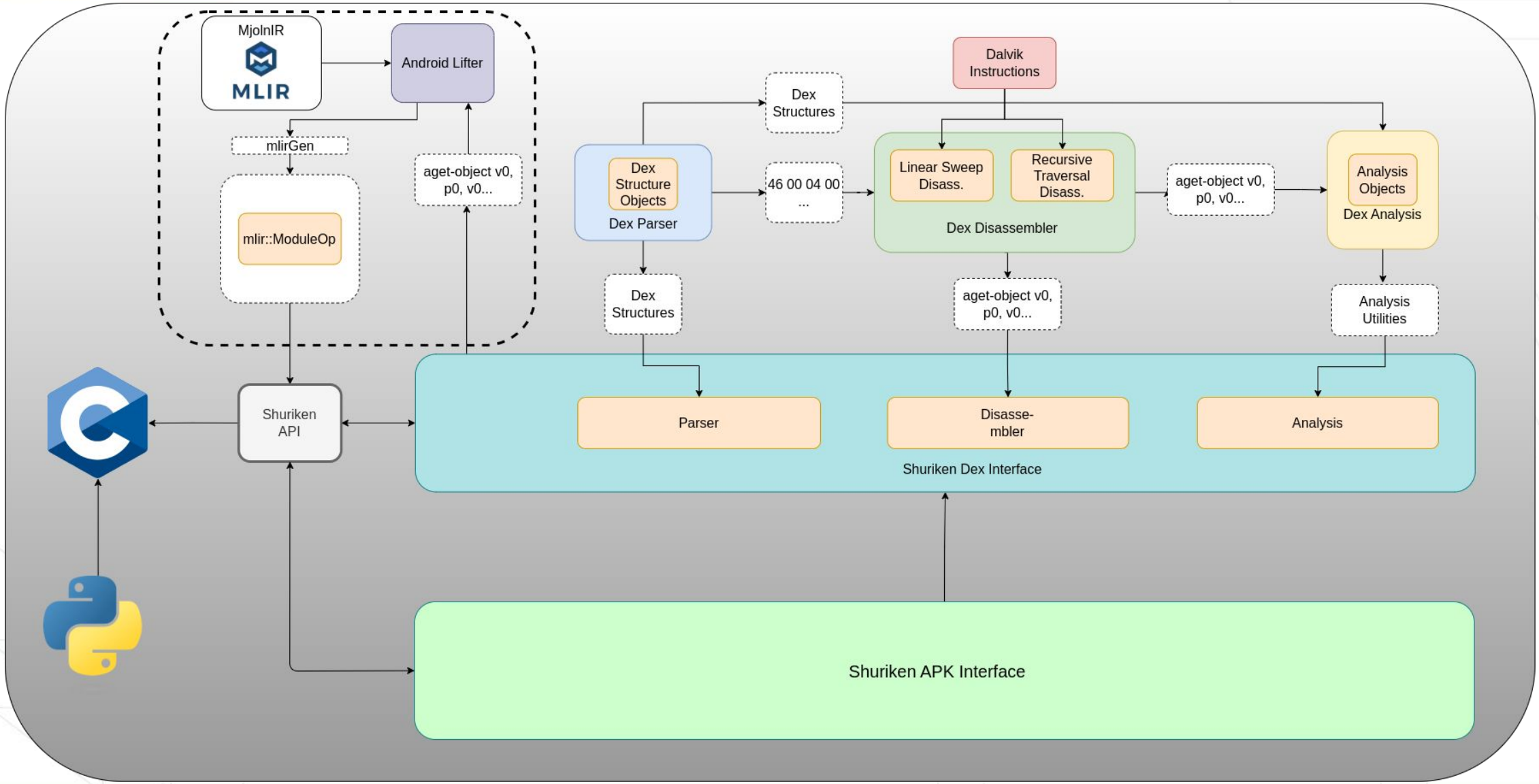

PoC||GTFO

Shuriken-Analyzer

Shuriken-Analyzer

- Librería escrita en C++ para el análisis de archivos DEX y APKs.
- 3 módulos principales en la librería:
  - Parser del formato DEX: permite obtener todas las estructuras de la cabecera, todas las tablas antes mencionadas, y más información como los bytes del código de cada método, información de depuración (si está disponible), etc.
  - Desensamblador de DEX: una vez obtenido el bytecode de cada método, un desensamblador obtiene las instrucciones y genera objetos para cada instrucción. Cada instrucción tiene un formato diferente, y permite mostrar la instrucción como un string.
  - Clases de análisis: clases que permiten generar el CFG de los métodos, y las cross-references de *strings, fields, métodos y clases.*
- Como *work in progress* se está diseñando una representación intermedia utilizando MLIR (un framework parte del proyecto LLVM).
- La librería provee de una API en C, esta API en lugar de objetos, permite acceder a los datos a través de estructuras.
- Utilizando la API en C, varios bindings están también en progreso. Oficialmente soportado un binding en Python, y en desarrollo uno en Rust.

Shuriken-Analyzer



Shuriken-Analyzer - C++ API - 1

```cpp
std::string test_file = "path/to/dex/file.dex";

std::unique_ptr<shuriken::parser::dex::Parser> dex_parser =
        shuriken::parser::parse_dex(test_file);

auto &header = dex_parser->get_header();
auto &classes = dex_parser->get_classes();
```

Shuriken-Analyzer - C++ API - 2

```cpp
std::string test_file = "path/to/dex/file.dex";

std::unique_ptr<shuriken::parser::dex::Parser> dex_parser = nullptr;
std::unique_ptr<shuriken::disassembler::dex::DexDisassembler> dex_disassembler = nullptr;

dex_parser = shuriken::parser::parse_dex(test_file);
dex_disassembler = std::make_unique<shuriken::disassembler::dex::DexDisassembler>(dex_parser.get());
dex_disassembler->disassembly_dex();

for (auto disassembled_method: disassembled_methods) {
    auto method = dex_disassembler->get_disassembled_method(disassembled_method.first);
}
```

Shuriken-Analyzer - C++ API - 3

```cpp
std::string test_file = "path/to/dex/file.dex";

std::unique_ptr<shuriken::parser::dex::Parser> dex_parser = nullptr;
std::unique_ptr<shuriken::disassembler::dex::DexDisassembler> dex_disassembler = nullptr;
std::unique_ptr<shuriken::analysis::dex::Analysis> dex_analysis = nullptr;

dex_parser = shuriken::parser::parse_dex(test_file);
dex_disassembler = std::make_unique<shuriken::disassembler::dex::DexDisassembler>(dex_parser.get());

dex_disassembler->disassembly_dex();

dex_analysis = std::make_unique<shuriken::analysis::dex::Analysis>(dex_parser.get(),
                                                                   dex_disassembler.get(),
                                                                   true);


dex_analysis->create_xrefs();

dex_analysis->get_fields();

for (auto &clazz: dex_analysis->get_classes()) {
    auto &clazz_value = clazz.second.get();
    std::cout << clazz_value.name() << "\n";
    for (auto &method: clazz_value.get_methods()) {
        auto method_value = method.second;
        if (!method_value->external())
            std::cout << method_value->toString();
    }
}
```

Shuriken-Analyzer - C API

```
hDexContext dexContext = parse_dex(file);
disassemble_dex(dexContext);
create_dex_analysis(dexContext, true);
analyze_classes(dexContext);

size_t nr_of_classes = get_number_of_classes(dexContext);
uint16_t i;
for (i = 0; i < nr_of_classes; ++i) {
    hdvmclass_t * class_ = get_class_by_id(dexContext, i);
    const char * name = class_->class_name;
    hdvmclassanalysis_t * class_analysis = get_analyzed_class(dexContext, name);

    for (uint32_t j = 0; j < class_analysis->n_of_methods; j++) {
        hdvmmethodanalysis_t * method_analysis = class_analysis->methods[j];
        printf("%s\n", method_analysis->full_name);
        basic_blocks_t * basic_blocks = method_analysis->basic_blocks;
        for (uint32_t z = 0; z < basic_blocks->n_of_blocks; z++) {
            hdvmbasicblock_t * basic_block = basic_blocks->blocks[z];
            printf("%s\n", basic_block.block_string);
        }
    }
}

destroy_dex(dexContext);
```

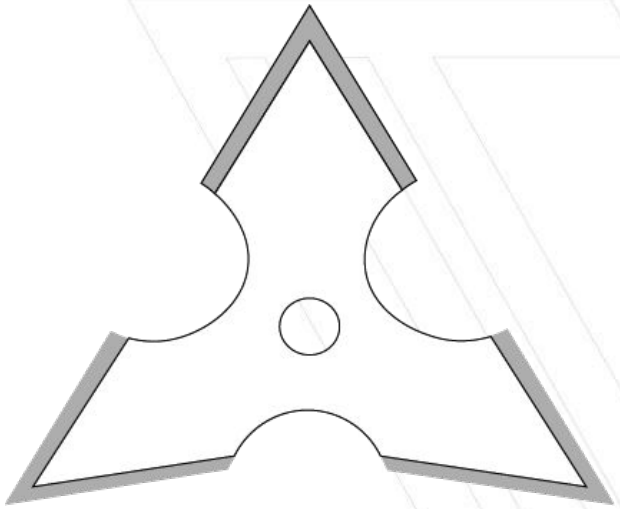
Shuriken-Analyzer - Python API

```python
file = "path/to/apk/file.apk"

apk = Apk(file, True)

for j in range(apk.get_number_of_dex_files()):
    dex_file = apk.get_dex_file_by_index(j)

    for i in range(apk.get_number_of_strings_from_dex(dex_file)):
        str_raw = apk.get_string_by_id_from_dex(dex_file, i)
    print(f"Retrieved string: {str_raw}")
        str_analysis = apk.get_analyzed_string_from_apk(str_raw)
        if str_analysis:
            print(f"{str_analysis.value} contains analysis {str_analysis.n_of_xreffrom} xrefs")

    for i in range(apk.get_number_of_classes_for_dex_file(dex_file)):
        class_: hdvmclass_t = apk.get_hdvmclass_from_dex_by_index(dex_file, i)
        class_analysis: hdvmclassanalysis_t = apk.get_analyzed_class_from_apk(class_name)
        for j in range(int(class_analysis.n_of_methods)):
            method_analysis: hdvmmethodanalysis_t = class_analysis.methods[j].contents
        for j in range(int(class_analysis.n_of_fields)):
            field_analysis: hdvmfieldanalysis_t = class_analysis.fields[j].contents
```

Manos a la obra :D

Eso es todo amigos

- Shuriken es un proyecto open source, y mantenido por Shuriken Development Group, podéis descargar y contribuir en: https://github.com/Shuriken-Group/Shuriken-Analyzer
- Gracias a:
  - Robert Yates (@yates82)
  - Jasmine Tang (@thisisjjasmine)
  - Antonio Nappa (@jeppojeps)
- Si queréis saber más sobre mi trabajo, podéis consultar: https://www.quarkslab.com/software-protection-qshield/



Muchas Gracias

¿Dudas, Preguntas?

Quarkslab

Contact and follow us