

Quarkslab

**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
 - ...

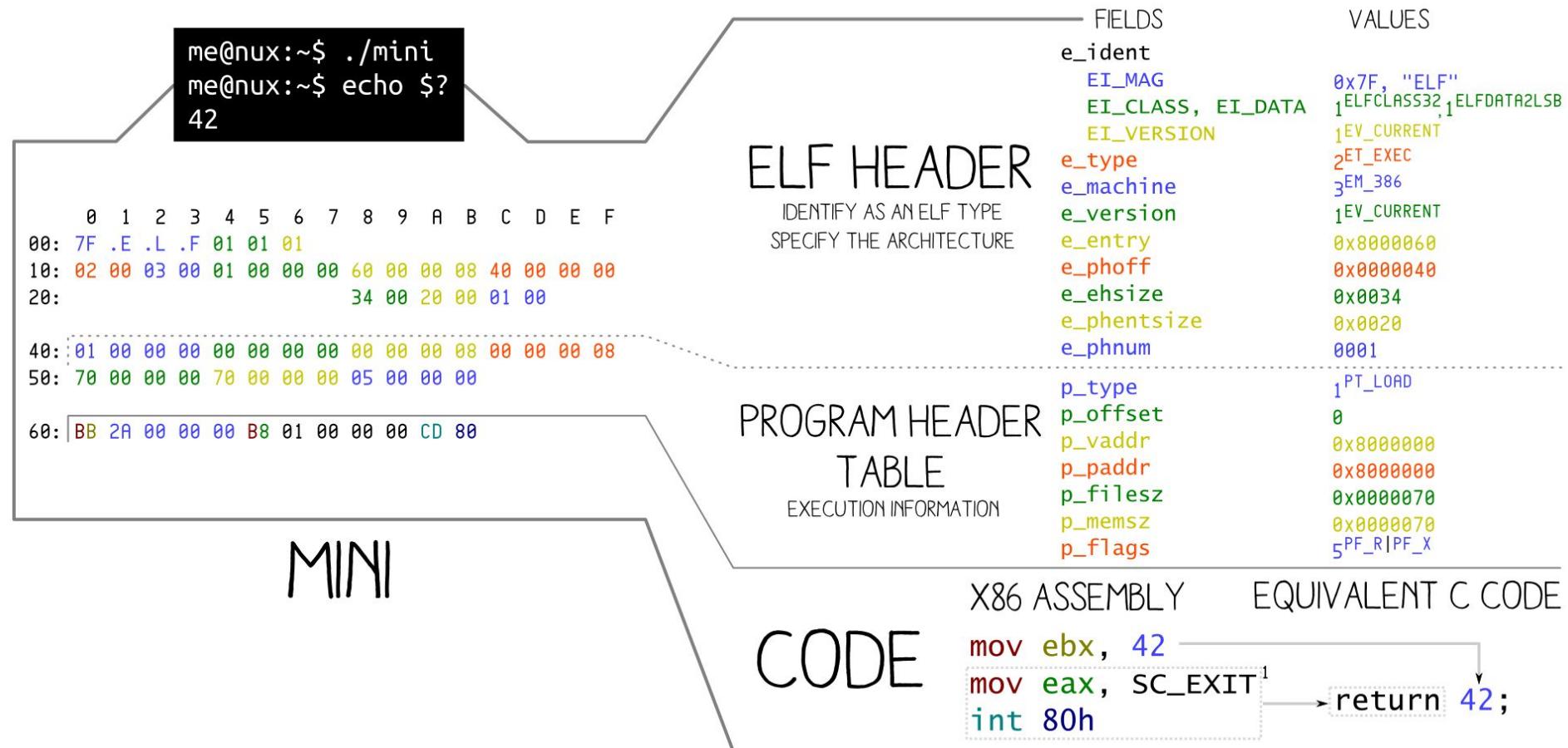
Ejemplo de pipeline de Análisis

Binario/Bytecode

```
62000300  
70200900  
08006e10  
0b000800  
0a000000
```

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
<http://www.corkami.com>



Parsing - 2

Elf Program Header:					
TYPE	FLAGS	Offset	V.Addr	M.Size	Align
		P.Addr	F.Size		
PHDR	R	0x0000000000000040	0x0000000000000040		
		0x0000000000000040	0x0000000000002d8	0x0000000000002d8	0x8
INTERP	R	0x00000000000000318	0x00000000000000318		
		0x00000000000000318	0x000000000000001c	0x000000000000001c	0x1
	[Program Interpreter: /lib64/ld-linux-x86-64.so.2]				
LOAD	R	0x0000000000000000	0x0000000000000000		
		0x0000000000000000	0x000000000036f8	0x000000000036f8	0x1000
					(DATA)
LOAD	R X	0x0000000000004000	0x0000000000004000		
		0x0000000000004000	0x000000000014db1	0x000000000014db1	0x1000
					(TEXT)
LOAD	R	0x0000000000019000	0x0000000000019000		
		0x0000000000019000	0x0000000000071b8	0x0000000000071b8	0x1000
					(DATA)
LOAD	RW	0x0000000000020f30	0x0000000000021f30		
		0x0000000000021f30	0x000000000001348	0x0000000000025e8	0x1000
					(DATA)
DYNAMIC	RW	0x0000000000021a38	0x0000000000022a38		
		0x0000000000022a38	0x00000000000200	0x00000000000200	0x8

Desensamblado - 1

Disassembly
Process



```
const/16 v0, 91
invoke-virtual {v4, v0}, int java.lang.String->indexOf(int)
move-result v0
const/16 v1, 93
invoke-virtual {v4, v1}, int java.lang.String->indexOf(int)
move-result ...
```

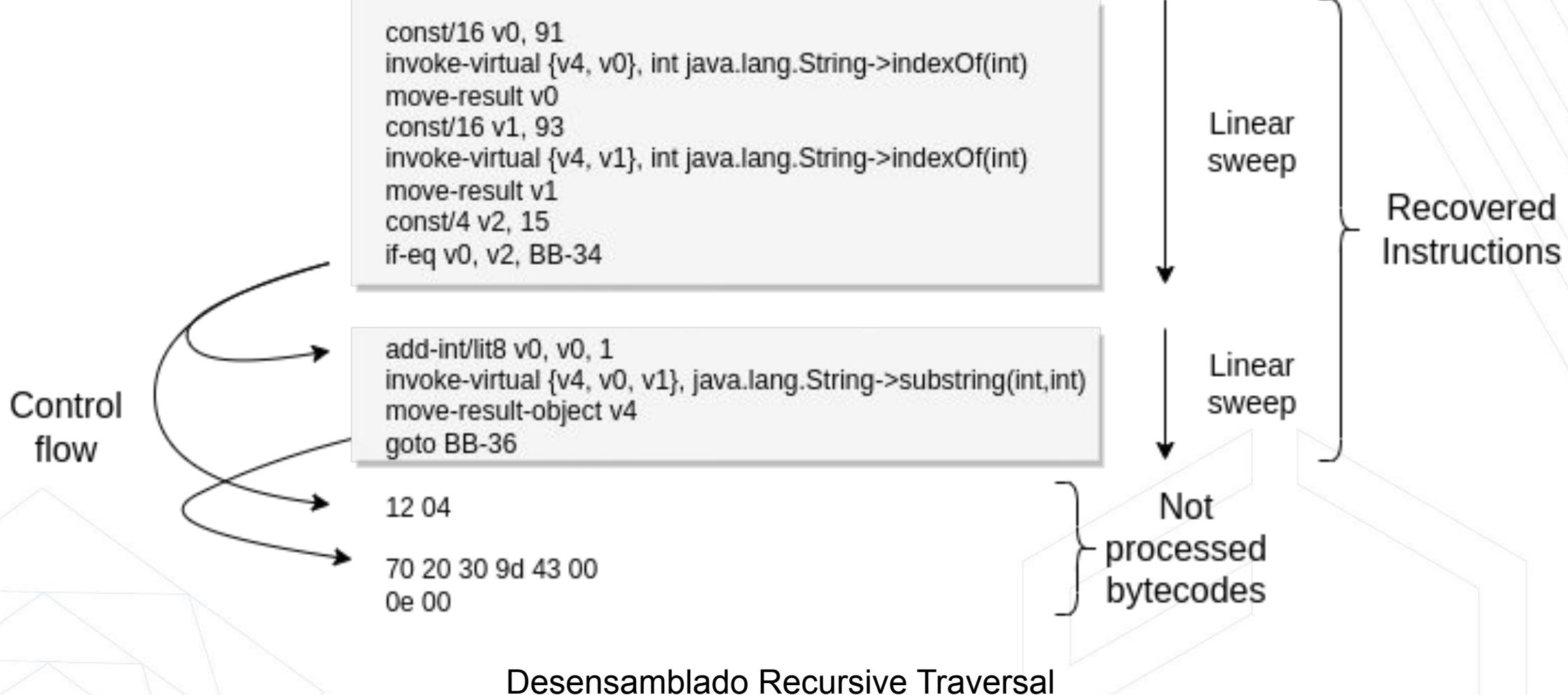
```
... 01
12 f2
32 20 0d 00
32 21 0b 00
35 10 09 00
d8 00 00 01
6e 30 28 9f 04 01
0c 04
28 02
12 04
70 20 30 9d 43 00
0e 00
```

Desensamblado Linear Sweep

Recovered
Instructions

Not
processed
bytecodes

Desensamblado - 2



Desensamblado - 3 & Análisis - 1

The screenshot shows a debugger interface with two main panes. On the left, a list of global variables is displayed:

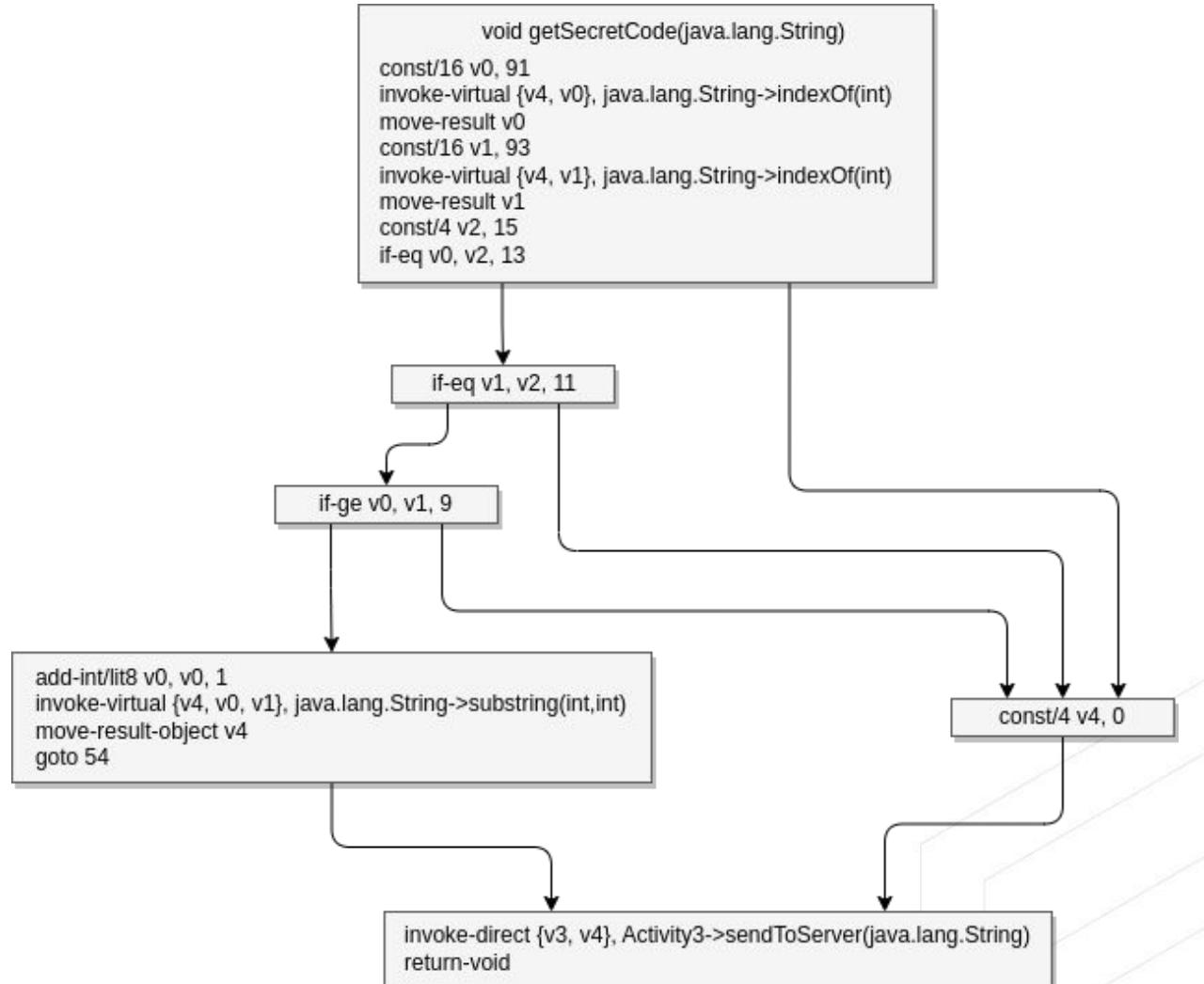
Variable	Address
firstFlagCheck	0x0004
secondFlagCheck	0x0004
thirdFlagCheck	0x0004
FourthFlagCheck	0x0004
sub_4021d0	0x0004
keyMethod	0x0004
puts_exit	0x0004
read	0x0004
firstFlagCheckNot0bfu...	0x0004
non0bfuscatedRead	0x0004
sub_402520	0x0004
cmpSecondFlag	0x0004
finalCheck	0x0004
init	0x0004
fini	0x0004
_fini	0x0004
__gmon_start__	0x0004
__libc_start_main	0x0004
puts	0x0004
exit	0x0004
strcmp	0x0004
RSA_set0_key	0x0004
strlen	0x0004
BN_hex2bn	0x0004

The right pane displays the assembly code for the function `int64_t firstFlagCheck()`:

```
00401870 int64_t firstFlagCheck()
00401870 f30f1efa    endbr64
00401874 4157        push   r15 {__saved_r15}
00401876 660fefc0    pxor   xmm0, xmm0
0040187a 488d3d8f170000 lea    rdi, [rel firstMessage] {"You have heard that a rival hist..."}
00401881 4156        push   r14 {__saved_r14}
00401883 4155        push   r13 {__saved_r13}
00401885 4154        push   r12 {__saved_r12}
00401887 53          push   rbx {__saved_rbx}
00401888 4881ec10080000 sub   rsp, 0x810
0040188f 4c8d642440    lea    r12, [rsp+0x40 {buff}]
00401894 0f29442440    movaps xmmword [rsp+0x40 {buff[0].o}], xmm0
00401899 0f29442450    movaps xmmword [rsp+0x50 {buff[0x10].o}], xmm0
0040189e 0f29442460    movaps xmmword [rsp+0x60 {buff[0x20].o}], xmm0
004018a3 0f29442470    movaps xmmword [rsp+0x70 {buff[0x30].o}], xmm0
004018a8 0f29842480000000 movaps xmmword [rsp+0x80 {buff[0x40].o}], xmm0
004018b0 0f29842490000000 movaps xmmword [rsp+0x90 {buff[0x50].o}], xmm0
004018b8 0f298424a0000000 movaps xmmword [rsp+0xa0 {buff[0x60].o}], xmm0
004018c0 0f298424b0000000 movaps xmmword [rsp+0xb0 {buff[0x70].o}], xmm0
004018c8 0f298424c0000000 movaps xmmword [rsp+0xc0], xmm0
004018d0 0f298424d0000000 movaps xmmword [rsp+0xd0 {var_768}], xmm0
004018d8 0f298424e0000000 movaps xmmword [rsp+0xe0 {var_758}], xmm0
004018e0 0f298424f0000000 movaps xmmword [rsp+0xf0 {var_748}], xmm0
004018e8 48c7842400010000... mov    dword [rsp+0x100 {var_738}], 0x0
```

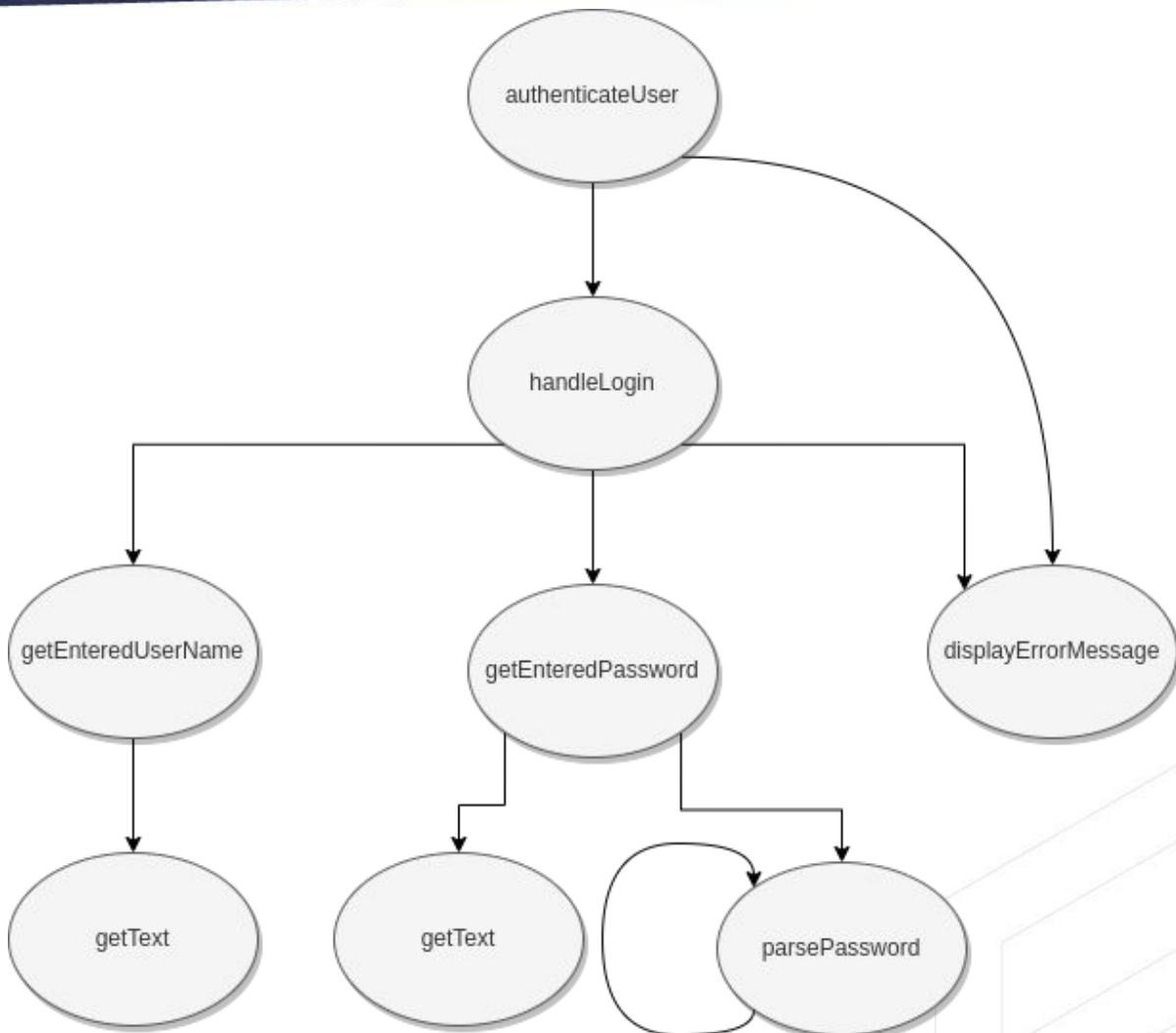
Presentación de Desensamblado y Reconocimiento de Funciones

Análisis - 2



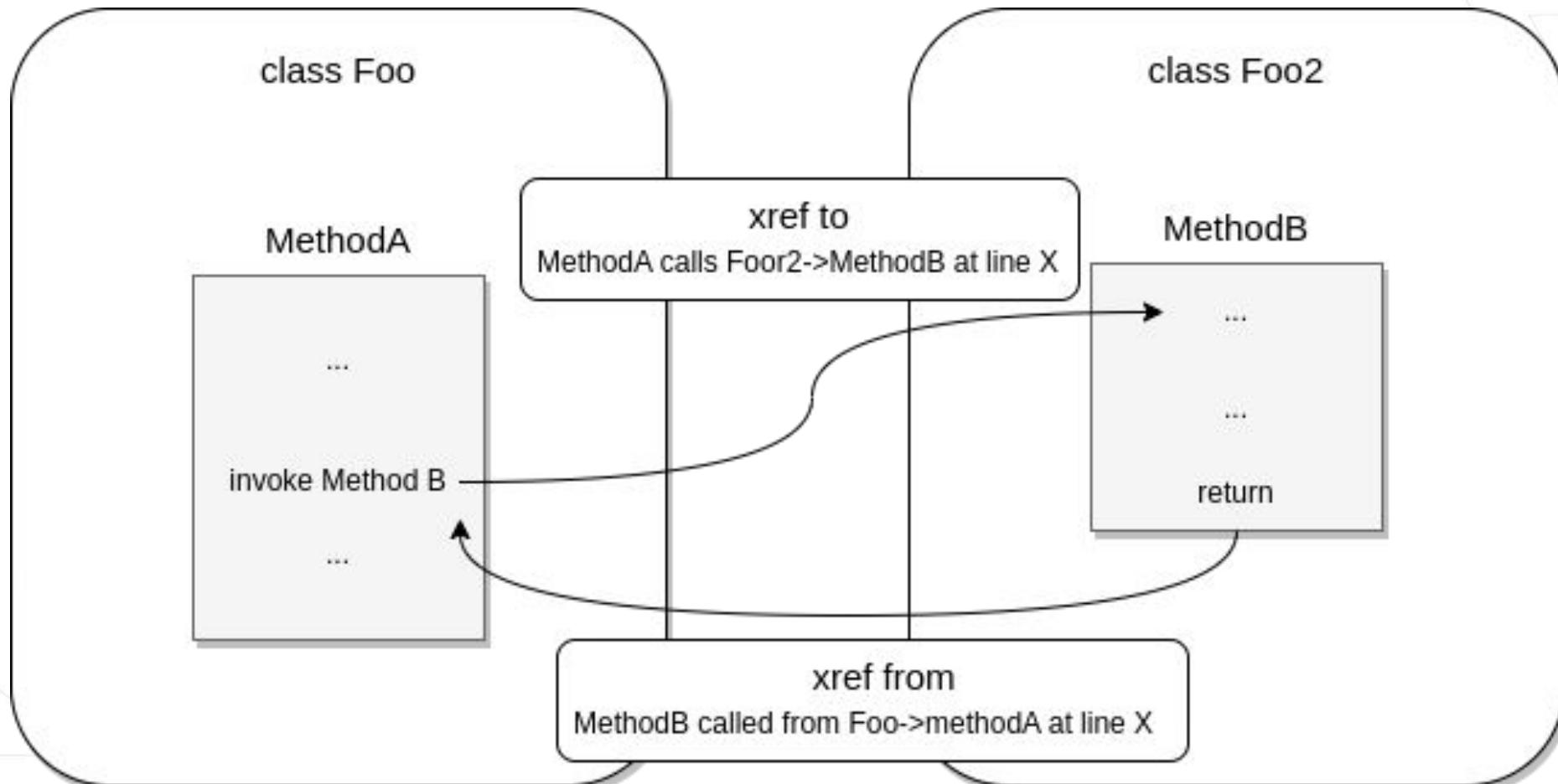
Control-Flow Graph

Análisis - 3



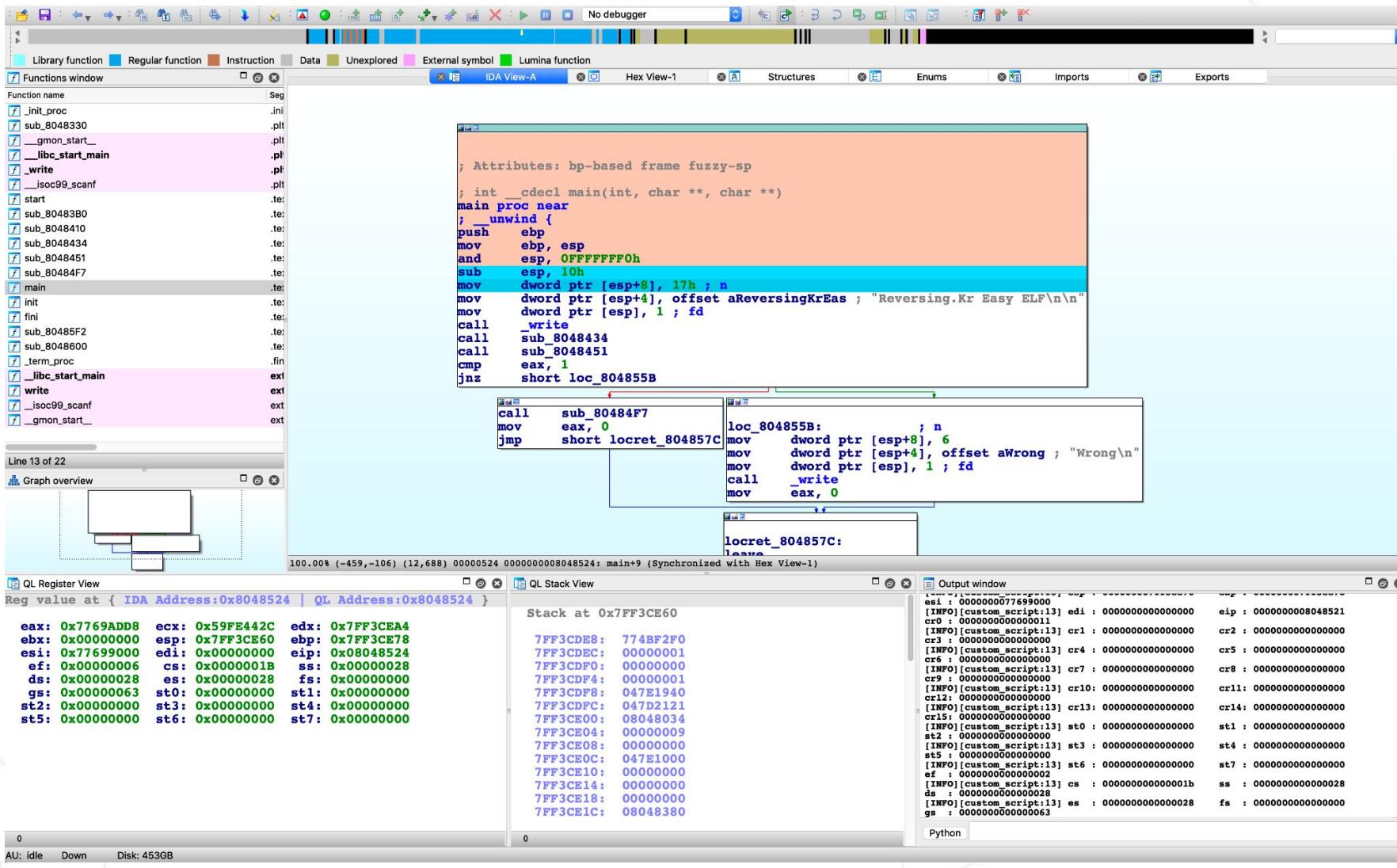
Call Graph

Análisis - 4



Cross-References

Herramientas de Análisis Estático - IDA Pro



Herramientas de Análisis Estático - Binary Ninja

madame.bnbd — Binary Ninja Personal 4.2.6204-dev

File Edit View Analysis Debugger Plugins Window Help

madame.bnbd

Symbols

Name	Address
strlen	0x0004021ef
BN_hex2bn	0x0004021fc
RSA_new	0x000402205
AES_decrypt	0x00040220a
memcmp	0x00040220e
strncpy	0x000402218
RSA_public_encrypt	0x00040221d
AES_set_decrypt_key	0x000402220
getc	0x000402228
main	0x000402229
_start	0x00040222c
sub_401240	0x00040222d
deregister_tm_clones	0x00040222e
sub_401280	0x00040222f
_FINI_0	0x000402230
_INIT_0	0x000402231
thirdCheck	0x000402232
checkFourthFlag	0x000402233
sub_4017c0	0x000402234
firstFlagCheck	0x000402235
secondFlagCheck	0x000402236
thirdFlagCheck	0x000402237
FourthFlagCheck	0x000402238
sub_4021d0	0x000402239
keyMethod	0x00040223a

keyMethod:

```
0 @ 004021ef int64_t module = 0
1 @ 004021fc int64_t exponent = 0
2 @ 00402205 rsa_key = RSA_new()
3 @ 0040220a zmm0.xmm0 = zx.o(0)
4 @ 0040220e int64_t* rdi = &module
5 @ 00402218 __builtin_memset(s: &user_buffer_encrypted:0, c: 0, n: 0x100)
6 @ 0040221d int64_t r12 = rsa_key
7 @ 00402286 BN_hex2bn(rdi, "8e449627141446d50a3bfab5d9fc0d58...")
8 @ 0040228b int64_t* rdi_1 = &exponent
9 @ 00402297 BN_hex2bn(rdi_1, "3")
10 @ 0040229c int64_t rdx = exponent
11 @ 004022a1 int64_t rsi = module
12 @ 004022a7 int64_t rdi_2 = r12
13 @ 004022aa RSA_set0_key(rdi_2, rsi, rdx, 0)
14 @ 004022af char (* rdx_1)[0x100] = &user_buffer_encrypted
15 @ 004022b7 int64_t rcx = r12
16 @ 004022c7 RSA_public_encrypt(0x100, &global_user_buffer, rdx_1, rcx, DT_PLTGOT)
17 @ 004022d8 char (* rdi_3)[0x100] = &user_buffer_encrypted
18 @ 004022db CHECK3_1 = memcmp(rdi_3, &encrypted_buffer, 256)
19 @ 004022e2 if (CHECK3_1 != 0) then 20 @ 0x40230d else 21 @ 0x4022e4
```

Cross References

- Filter (5)
- Code References (5)
 - sub_4017c0 {1}
 - 0040184d keyMethod()
 - firstFlagCheck {1}
 - 00401a2d keyMethod()
 - secondFlagCheck {1}
 - 00401c9d keyMethod()
 - thirdFlagCheck {1}
 - 00401f1d keyMethod()
 - FourthFlagCheck {1}

Downloading update... (90%) Cancel

21 @ 004022e4 int32_t rdx_2 = [&CHECK2].d
22 @ 004022ec if (rdx_2 == 0) then 23 @ 0x402310 else 25 @ 0x4022ee

23 @ 00402310 [&CHECK2].d = 1
24 @ 00402325 return CHECK3_1

25 @ 004022ee int32_t CHECK3_1 = [&CHECK3].d
26 @ 004022f6 if (CHECK3_1 == 0) then 27 @ 0x402330 else 29 @ 0x4022f8

27 @ 00402330 [&CHECK3].d = 1
28 @ 00402345 return CHECK3_1

29 @ 004022f8 [&CHECK1].d = 1
30 @ 004022f8 goto 20 @ 0x40230d

linux-x86_64 0x40220e-0x402211 (0x3 bytes)

Herramientas de Análisis Estático - Radare2

File Settings Edit [View] Tools Search Emulate Debug Analyze Help

[X] Registers (dr) [Cache] Off [X] Disassembly (pd) [Cache] Off [X] Functions (af1) [Cache] On

rax = 0x00000000
rbx = 0x00000000
rcx = 0x00000000
rdx = 0x00000000
rsi = 0x00000000
rdi = 0x00000000
r8 = 0x00000000
r9 = 0x00000000
r10 = 0x00000000
r11 = 0x00000000
r12 = 0x00000000
r13 = 0x00000000
r14 = 0x00000000
r15 = 0x00000000
rip = 0x0000067d0
rbp = 0x00000000
rflags = 0x00000
rsp = 0x00000000

> Breakpoints
Classes
Comments
Console
Database
Decompiler
Decompiler With Offsets
Disassemble Summary
Disassembly
Entropy
Entropy Fire
File Hashes
Function Calls
Functions
Graph
Headers
Hexdump
Imports
Info
Methods
Relocs
Sections
Segments
Show All Decompiler Output
Stack
Strings in data sections
Strings in the whole bin
Summary
Symbols
Tiny Graph
Var READ address
Var WRITE address
Xrefs Here

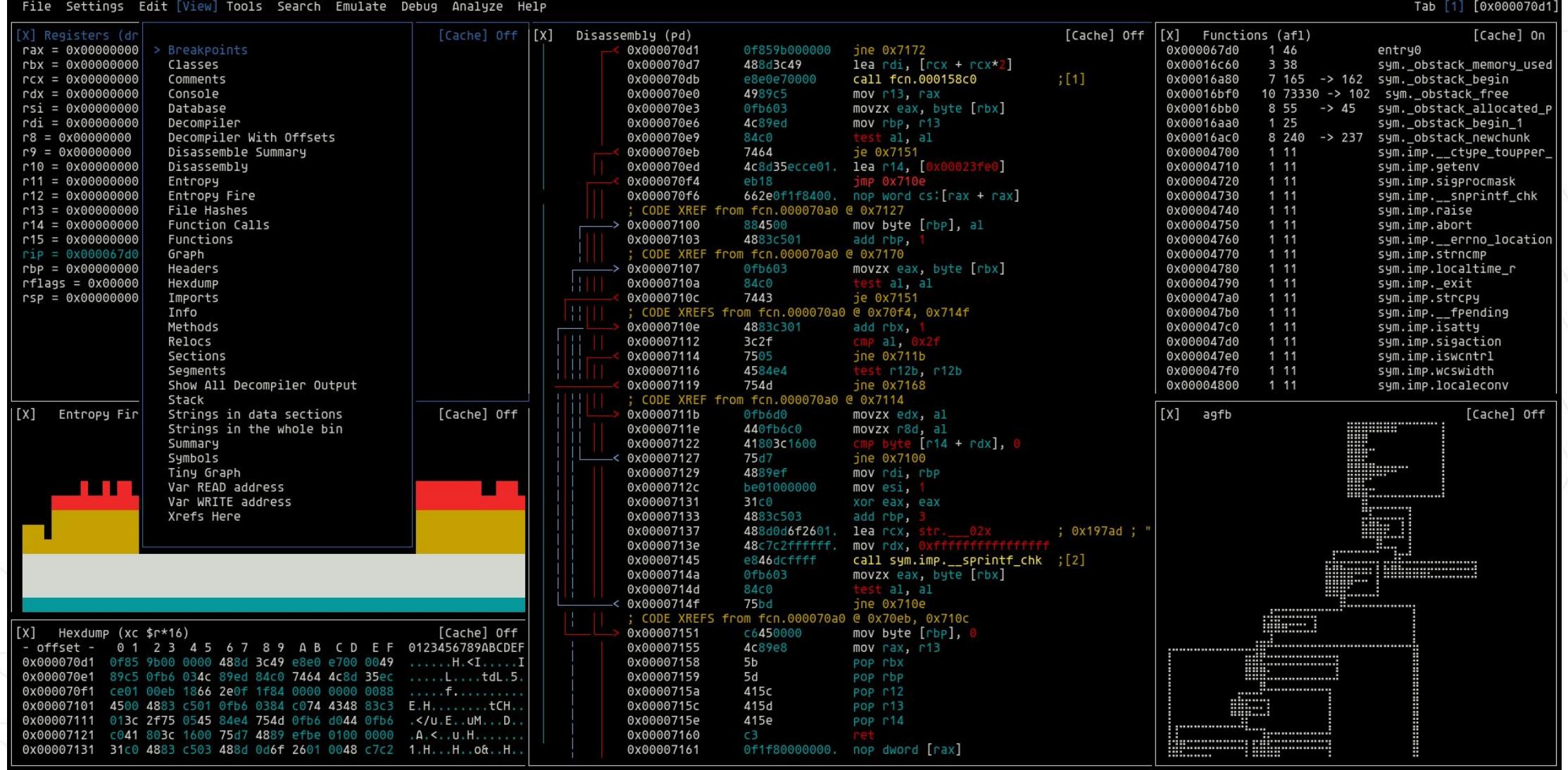
[X] Entropy Fir [Cache] Off [X] agfb [Cache] Off

[X] Hexdump (xc \$r*16) [Cache] Off

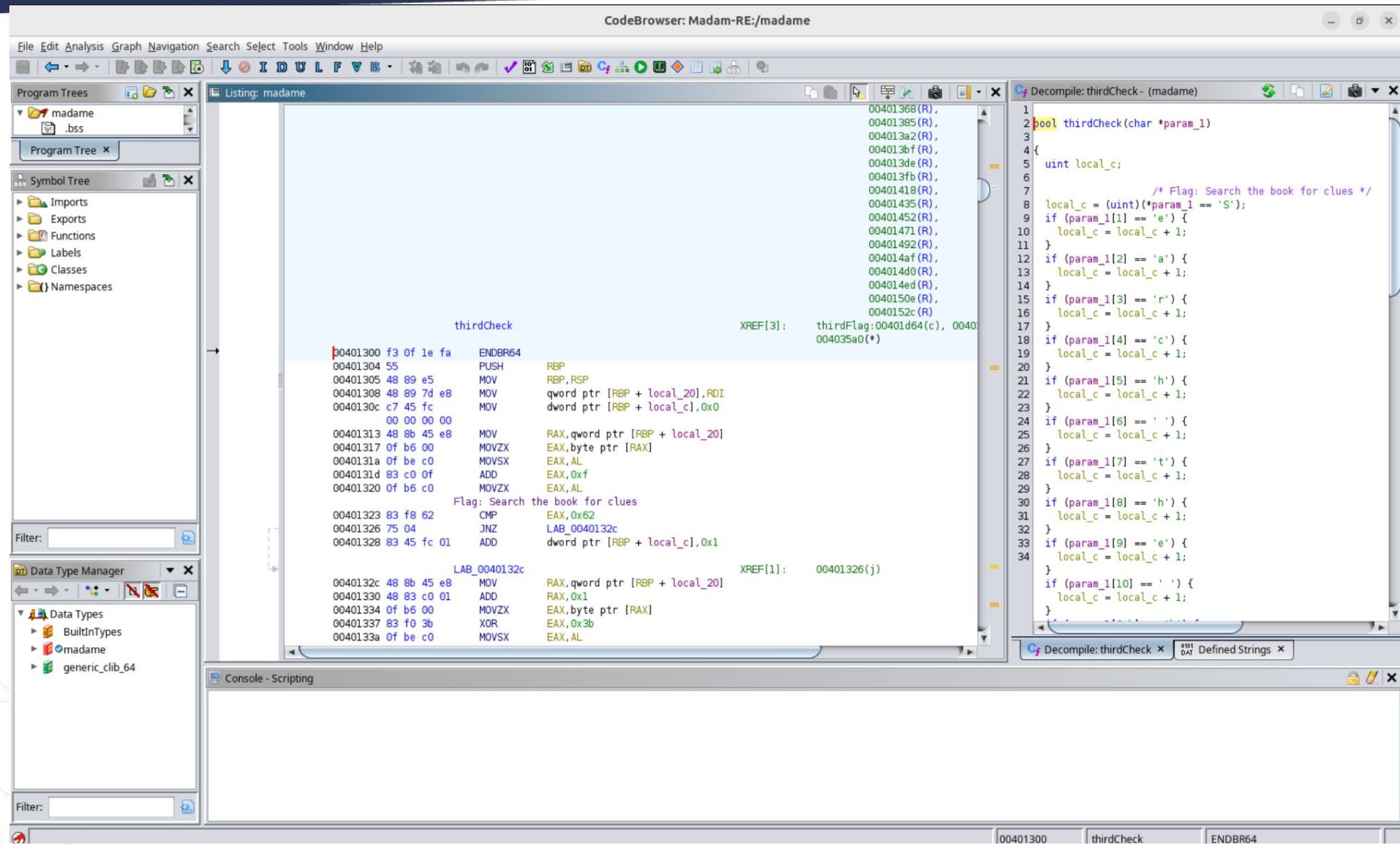
- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x0000070d1	0	785	9b00	0000	48bd	3c49	e8e0	e700	0049	.	.	H.	L.	I.H..L..I..
0x0000070e1	89c5	9fb6	034c	89ed	84c0	7464	4c8d	35ec	.	.	L.	.	tdL.	5.	.	.	
0x0000070f1	ce01	00eb	1866	2e0f	1f84	0000	0000	0088	.	.	F.	
0x000007101	4500	4883	c501	0fb6	0384	c074	4348	83c3	E.	H.	.	tCh.	
0x000007111	013c	2f75	0545	84e4	754d	0fb6	d044	0fb6	.	</U.	E..	uM..	D..	.	.	.	
0x000007121	c041	803c	1600	75d7	4889	efbe	0100	0000	.	A..	<..uH..	
0x000007131	31c0	4883	c503	488d	0d6f	2601	0048	c7c2	1.H..	H..	o&..	H..	

0x0000067d0 1 46 entry0
0x00016c60 3 38 sym._obstack_memory_used
0x00016a80 7 165 -> 162 sym._obstack_begin
0x00016bf0 10 73330 -> 102 sym._obstack_free
0x00016bb0 8 55 -> 45 sym._obstack_allocated_P
0x00016aa0 1 25 sym._obstack_begin_1
0x00016ac0 8 240 -> 237 sym._obstack_newchunk
0x00004700 1 11 sym.imp._ctype_toupper_
0x00004710 1 11 sym.imp.getenv
0x00004720 1 11 sym.imp.sigprocmask
0x00004730 1 11 sym.imp.__snprintf_chk
0x00004740 1 11 sym.imp.raise
0x00004750 1 11 sym.imp.abort
0x00004760 1 11 sym.imp.__errno_location
0x00004770 1 11 sym.imp.strncmp
0x00004780 1 11 sym.imp.localtime_r
0x00004790 1 11 sym.imp._exit
0x000047a0 1 11 sym.imp.strcpy
0x000047b0 1 11 sym.imp._fpending
0x000047c0 1 11 sym.imp.isatty
0x000047d0 1 11 sym.imp.sigaction
0x000047e0 1 11 sym.imp.iswcntrl
0x000047f0 1 11 sym.imp.wcswidth
0x00004800 1 11 sym.imp.localeconv

0x0000070d1 0f85b0000000 jne 0x7172
0x0000070d7 488d3c49 lea rdi, [rcx + rcx*2]
0x0000070db e8e0e70000 call fcn.0000158c0 ;[1]
0x0000070e0 4989c5 mov r13, rax
0x0000070e3 0fb603 movzx eax, byte [rbx]
0x0000070e6 4c89ed mov rbp, r13
0x0000070e9 84c0 test al, al
0x0000070eb 7464 je 0x7151
0x0000070ed 4c8d35ecce01 lea r14, [0x00023fe0]
0x0000070f4 eb18 jmp 0x710e
0x0000070f6 662e0f1f8400. nop word cs:[rax + rax]
; CODE XREF from fcn.0000070a0 @ 0x7127
0x000007100 884500 mov byte [rbp], al
0x000007103 4883c501 add rbp, 1
; CODE XREF from fcn.0000070a0 @ 0x7170
0x000007107 0fb603 movzx eax, byte [rbx]
0x00000710a 84c0 test al, al
0x00000710c 7443 je 0x7151
; CODE XREFS from fcn.0000070a0 @ 0x70f4, 0x714f
0x00000710e 4883c301 add rbx, 1
0x000007112 3c2f cmp al, 0x2f
0x000007114 7505 jne 0x711b
0x000007116 4584e4 test r12b, r12b
0x000007119 75d4 jne 0x7168
; CODE XREF from fcn.0000070a0 @ 0x7114
0x00000711b 0fb6d0 movzx edx, al
0x00000711e 440fb6c0 movzx r8d, al
0x000007122 41803c1600 cmp byte [r14 + rdx], 0
0x000007127 75d7 jne 0x7100
0x000007129 4889ef mov rdi, rbp
0x00000712c be01000000 mov esi, 1
0x000007131 31c0 xor eax, eax
0x000007133 4883c503 add rbp, 3
0x000007137 488d06f2601. lea rcx, str.__02x ; 0x197ad ; "
0x00000713e 48c7c2fffffff. mov rdx, 0xffffffffffffffffffff
0x000007145 e846dcffff call sym.imp.__sprintf_chk ;[2]
0x00000714a 0fb603 movzx eax, byte [rbx]
0x00000714d 84c0 test al, al
0x00000714f 75d4 jne 0x710e
; CODE XREFS from fcn.0000070a0 @ 0x70eb, 0x710c
0x000007151 c450000 mov byte [rbp], 0
0x000007155 4c89e8 mov rax, r13
0x000007158 5b pop rbp
0x000007159 5d pop rbp
0x00000715a 415c pop r12
0x00000715c 415d pop r13
0x00000715e 415e pop r14
0x000007160 c3 ret
0x000007161 0f1f80000000. nop dword [rax]



Herramientas de Análisis Estático - Ghidra



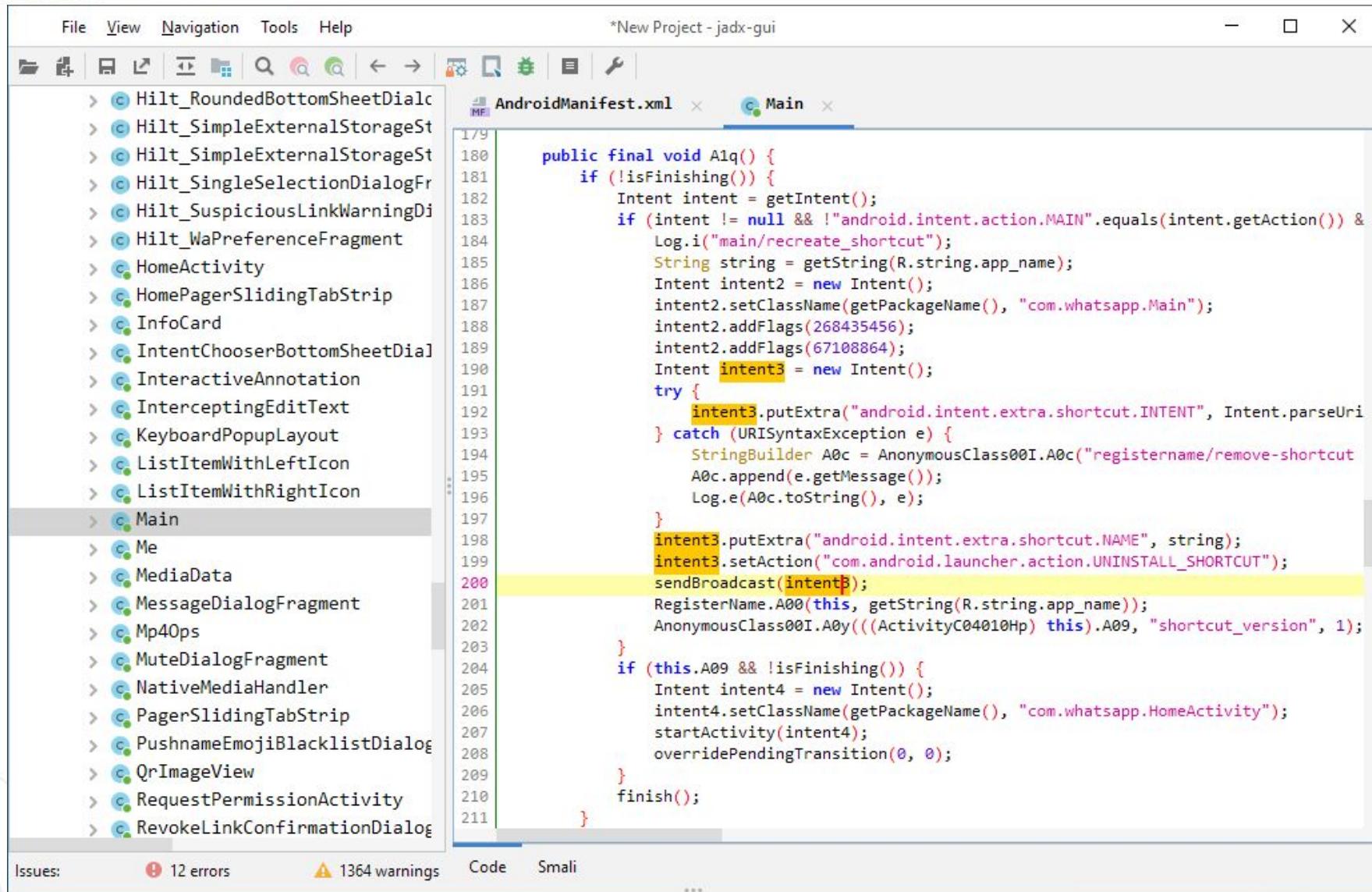
Herramientas de Análisis Estático - JEB

The screenshot displays the JEB static analysis tool interface, showing the analysis of the `raasta-classes.dex` file. The interface is divided into several panes:

- Project Explorer:** Shows the file structure of the dex file, including `raasta-classes.dex`, `raasta-classes.dex`, and `decompiler`.
- raasta-classes.dex/Disassembly:** Displays the Dalvik Disassembly of the `AppHelp` class. The assembly code shows the implementation of the `onCreate` method, which creates a `WebView` and loads HTML content.
- AppHelp/Source:** Shows the corresponding Java source code for the `AppHelp` class. It includes imports for `Activity`, `Bundle`, and `WebView`, and defines the `onCreate` method which sets up the `WebView` with `text/html` encoding.
- raasta-classes.dex/Hierarchy:** Displays the class hierarchy and field information. It shows the `com.pnsoftware.raasta` package containing `AppHelp`, `CoordinatesE6`, `GeoTrace`, `GpsManager`, `MapViewEx`, and `MockLocationGenerator`. The `MockLocationGenerator` class has fields like `PATH_TURTLE_SHELL`, `PATH_ZIGZAG`, `dir`, `dirX`, `dirY`, `dist`, `distLeft`, `distLeft`, `freqMs`, `incr`, `lat`, `lng`, and `locman`.
- Description / Source:** Provides tabs for Description and Source, allowing switching between the assembly and source code views.
- Logger / Terminal:** Shows the command-line interface used to interact with the JEB tool, including help commands and session configuration.
- raasta-classes.dex/CallGraph:** A pane for generating and viewing call graphs, currently prompting the user to "Click here to generate the callgraph".

At the bottom, the status bar shows the coordinates (3,4,34), address Lcom/pnsoftware/raasta/AppHelp;-, and memory location 0x8h.

Herramientas de Análisis Estático - Jadx



The screenshot shows the Jadx-GUI application window. On the left, a tree view displays the class hierarchy of the analyzed APK. The 'Main' class is selected. The main pane contains the decompiled Java code for the 'Main' class. A specific line of code, `sendBroadcast(intent3);`, is highlighted with a yellow background.

```
File View Navigation Tools Help
*New Project - jadx-gui
AndroidManifest.xml × Main ×
179
180     public final void A1q() {
181         if (!isFinishing()) {
182             Intent intent = getIntent();
183             if (intent != null && !"android.intent.action.MAIN".equals(intent.getAction()) &
184                 Log.i("main/recreate_shortcut");
185                 String string = getString(R.string.app_name);
186                 Intent intent2 = new Intent();
187                 intent2.setClassName(getApplicationContext(), "com.whatsapp.Main");
188                 intent2.addFlags(268435456);
189                 intent2.addFlags(67108864);
190                 Intent intent3 = new Intent();
191                 try {
192                     intent3.putExtra("android.intent.extra.shortcut.INTENT", Intent.parseUri
193                 } catch (URISyntaxException e) {
194                     StringBuilder A0c = AnonymousClass00I.A0c("registername/remove-shortcut
195                     A0c.append(e.getMessage());
196                     Log.e(A0c.toString(), e);
197                 }
198                 intent3.putExtra("android.intent.extra.shortcut.NAME", string);
199                 intent3.setAction("com.android.launcher.action.UNINSTALL_SHORTCUT");
200                 sendBroadcast(intent3);
201                 RegisterName.A00(this, getString(R.string.app_name));
202                 AnonymousClass00I.A0y((ActivityC04010H0) this).A09, "shortcut_version", 1);
203             }
204             if (this.A09 && !isFinishing()) {
205                 Intent intent4 = new Intent();
206                 intent4.setClassName(getApplicationContext(), "com.whatsapp.HomeActivity");
207                 startActivity(intent4);
208                 overridePendingTransition(0, 0);
209             }
210             finish();
211         }
    
```

Issues: 12 errors 1364 warnings

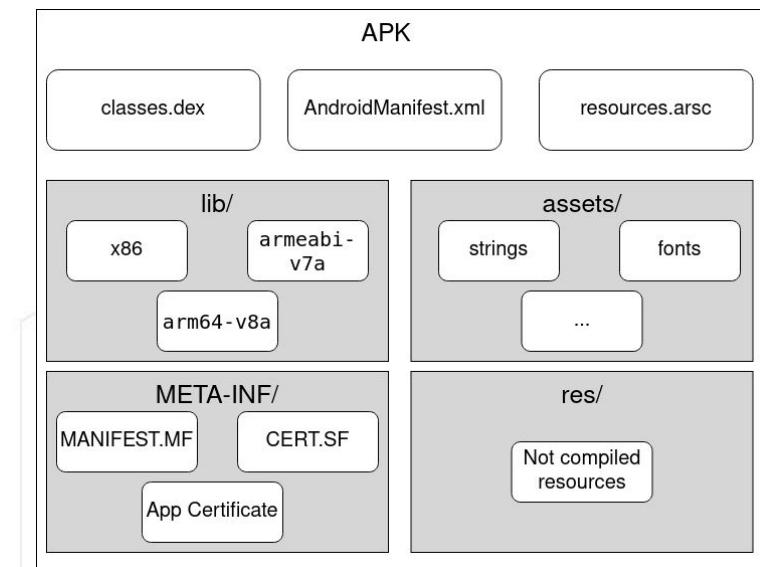
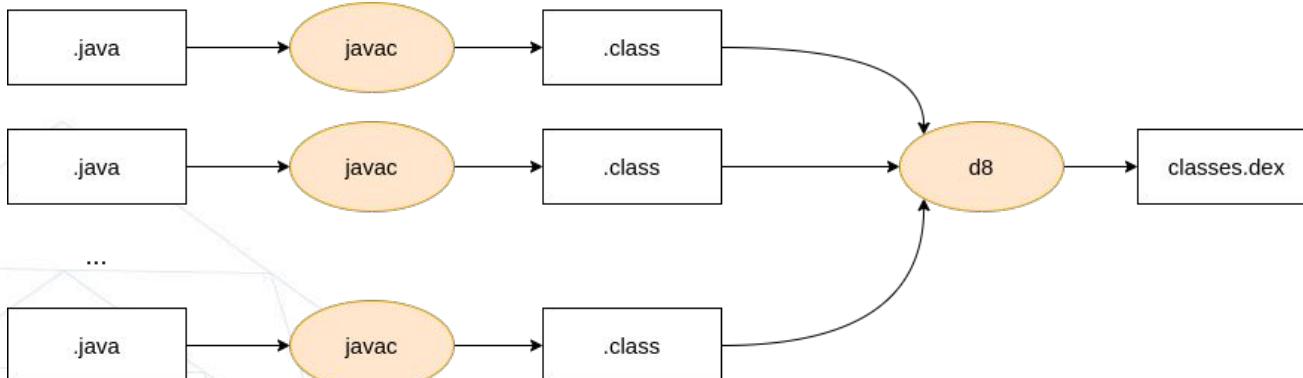
Herramientas de Análisis Estático - ...

- ❖ 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

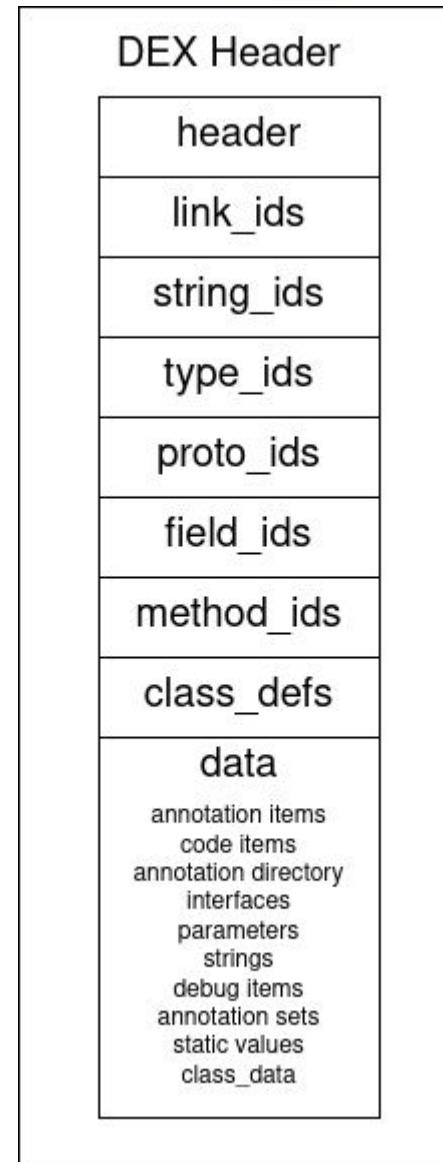
- ❖ Dentro de un APK, la forma de distribuir el código de las aplicaciones es a través de archivos Dalvik EXecutable (DEX).
- ❖ Antiguamente el código de un archivo DEX era ejecutado por la Dalvik VirtualMachine (DVM), actualmente es compilado a nativo y ejecutado por el Android RunTime (ART).
- ❖ Los 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.



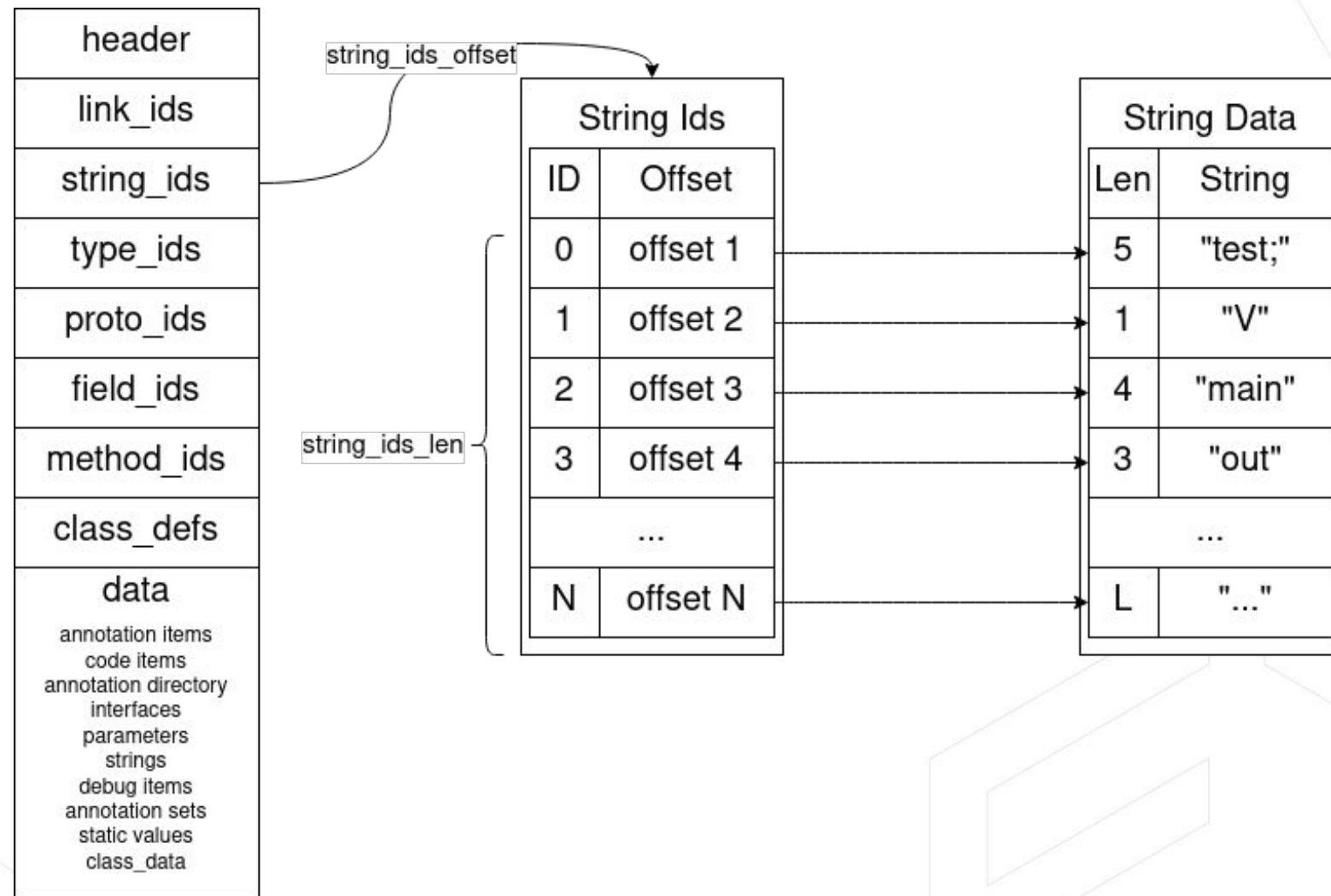
Formato DEX

- ❖ 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*.

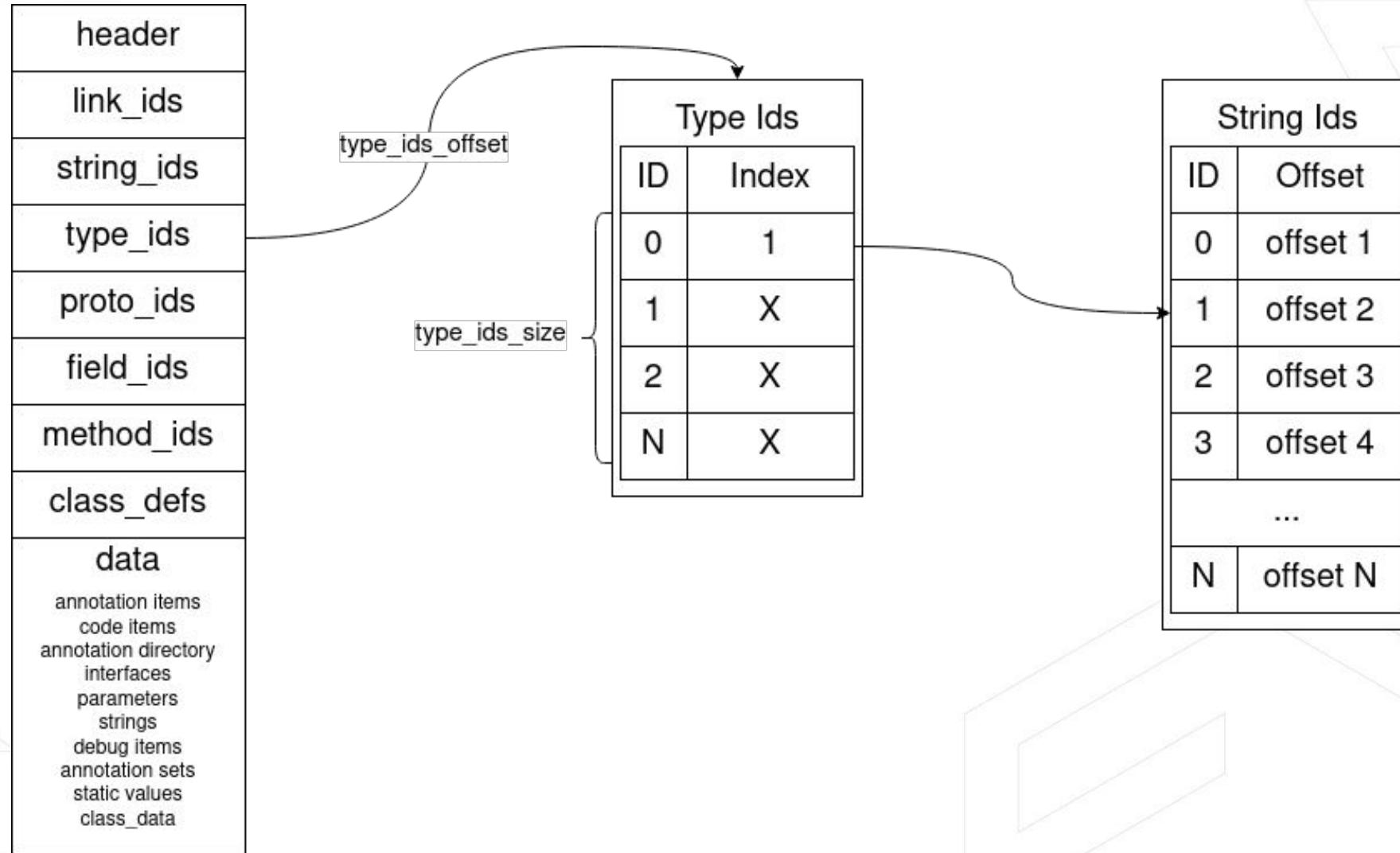
Formato DEX - Cabecera



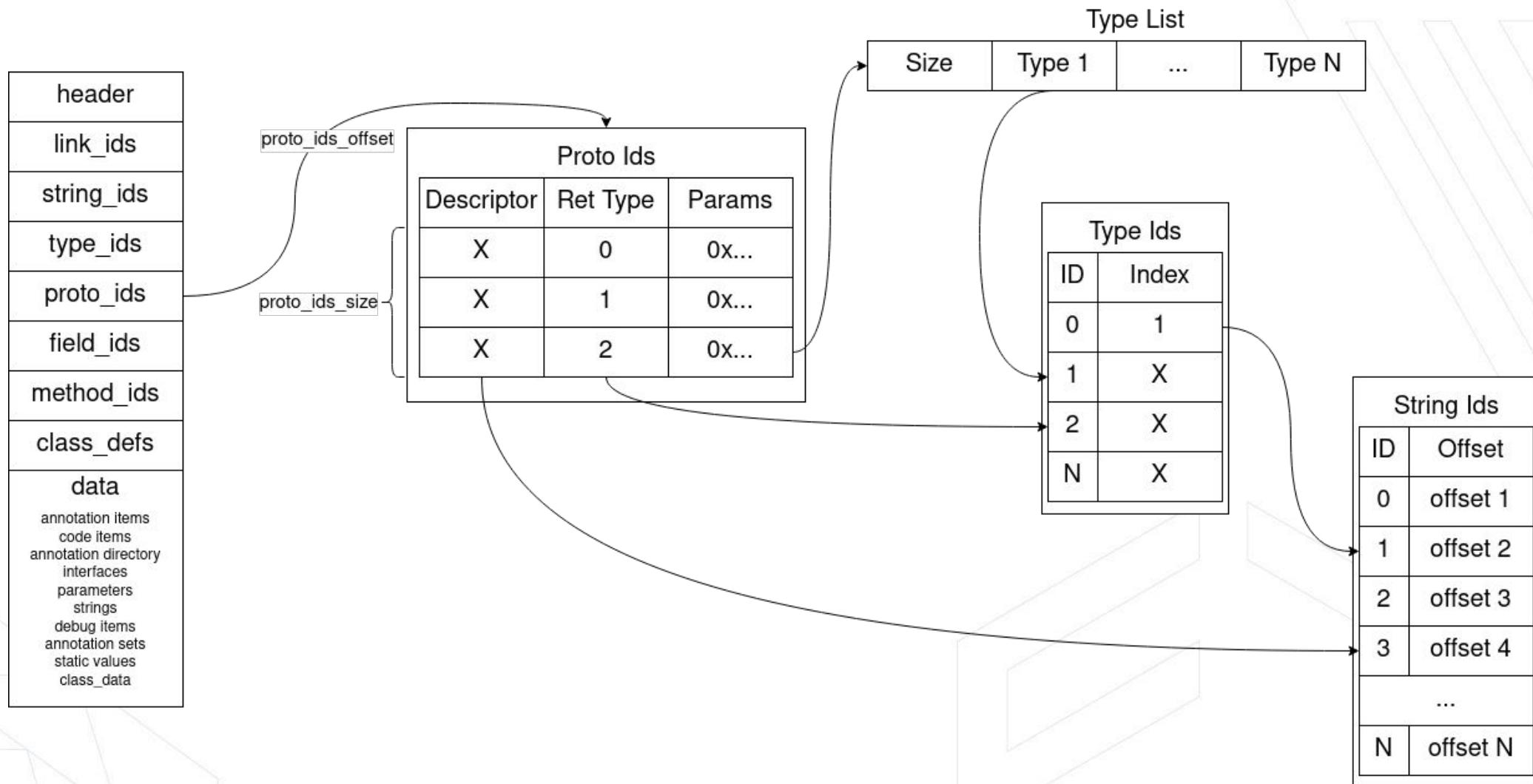
Formato DEX - String IDs



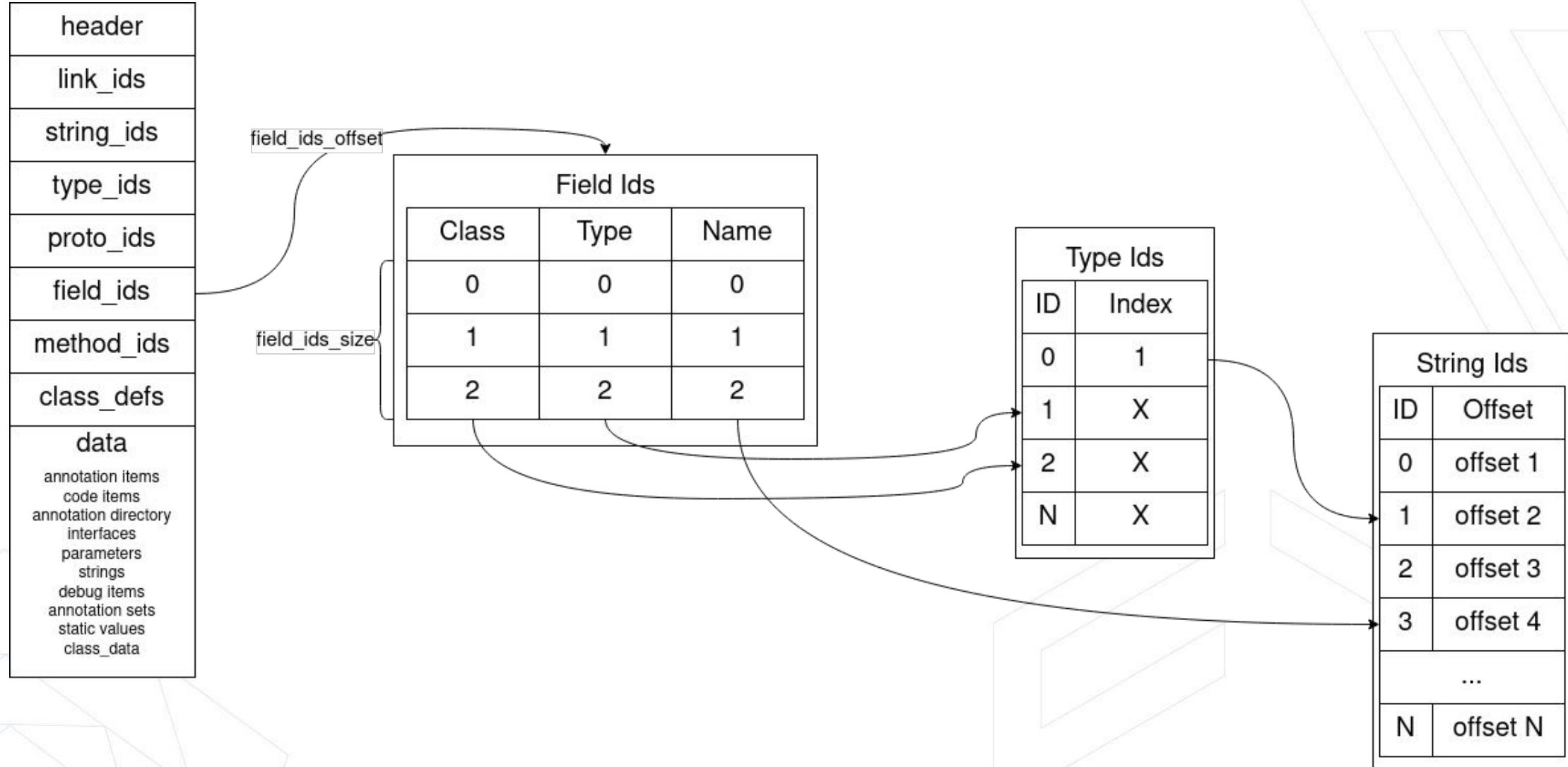
Formato DEX - Type IDs



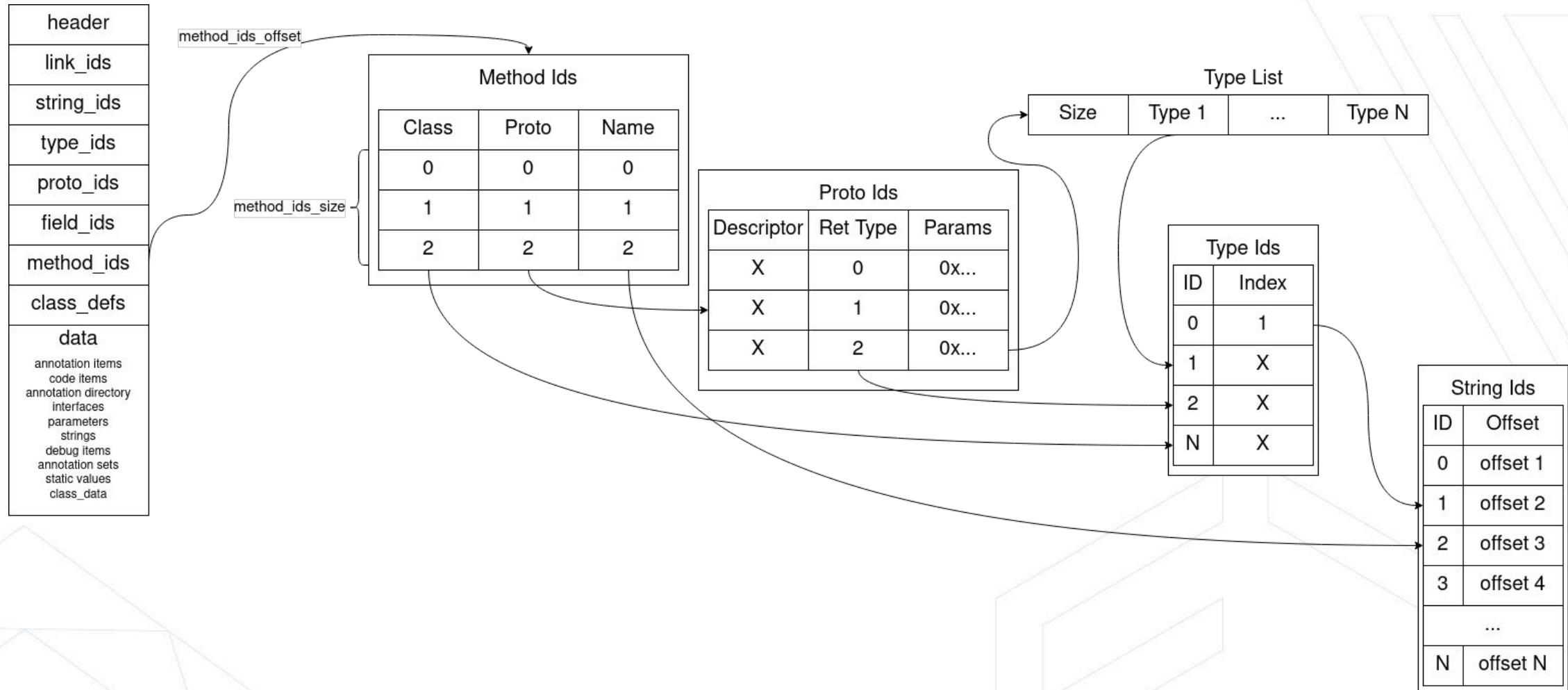
Formato DEX - Proto IDs



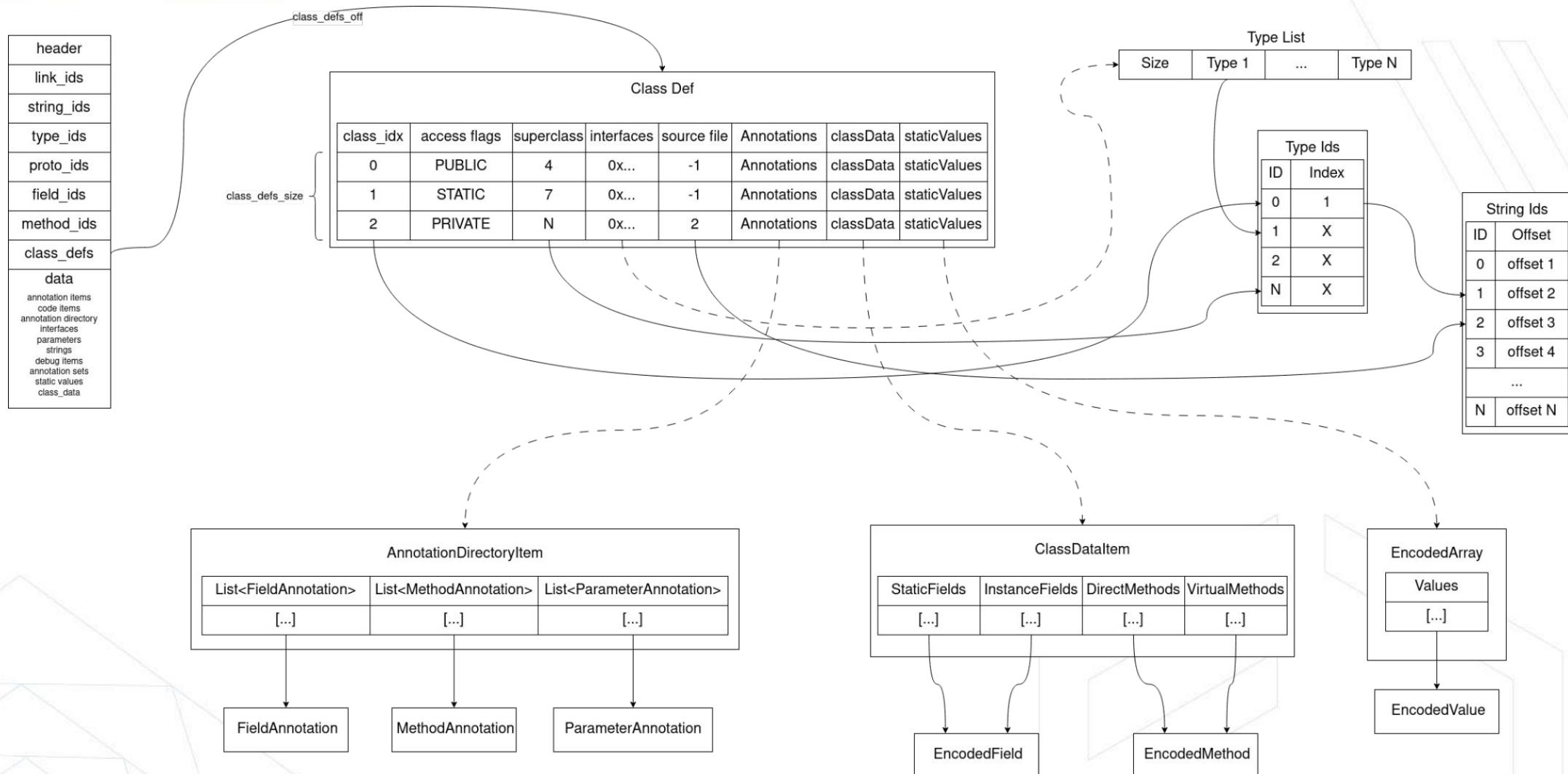
Formato DEX - Field IDs



Formato DEX - Method IDs



Formato DEX - Class Defs



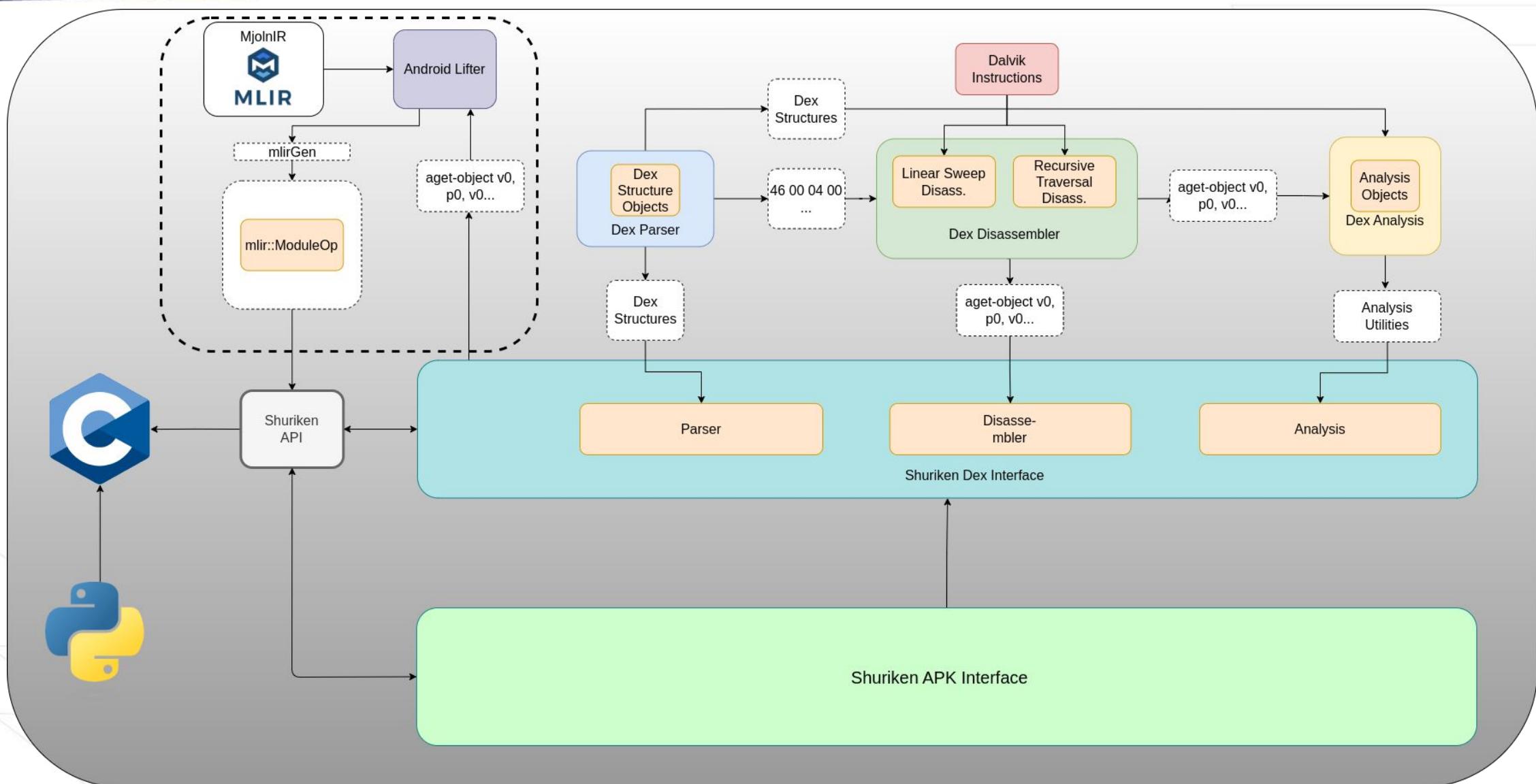
PoC||GTFO

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

```
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

```
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

```
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::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

```
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_xrefs} xrefs")

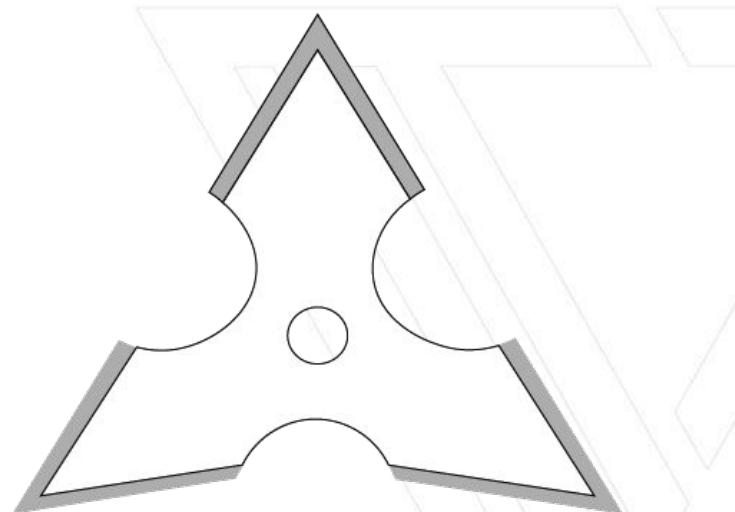
    for i in range(apk.get_number_of_classes_for_dex_file(dex_file)):
        class_: hdvmclass_t = apk.get_hdvmclass_from_dex_by_index(dex_file, i)
        class_analysis: hdvmclassanalysis_t = apk.get_analyzed_class_from_apk(class_name)
        for j in range(int(class_analysis.n_of_methods)):
            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

