Unraveling the Challenges of Reverse Engineering Flutter Applications

Axelle Apvrille, Fortinet

BlackAlps, October 2023

Who am I?

Axelle Apvrille

Principal Security Researcher at **Fortinet**, @cryptax

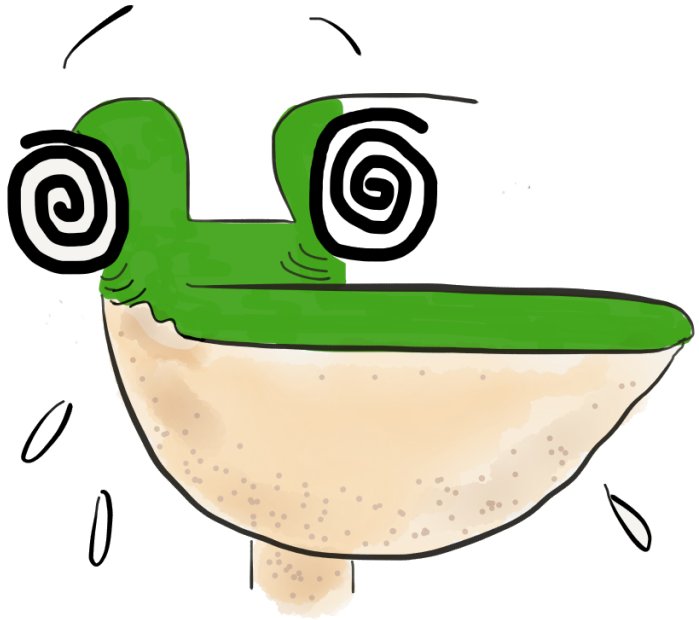
Lead organizer of Ph0wn CTF

I analyze **Android malware** and **IoT** malware

My slides are created with LaTeX, my reports with *Pandoc*

I disassemble with JEB Pro, Radare2 on Linux

Audience check



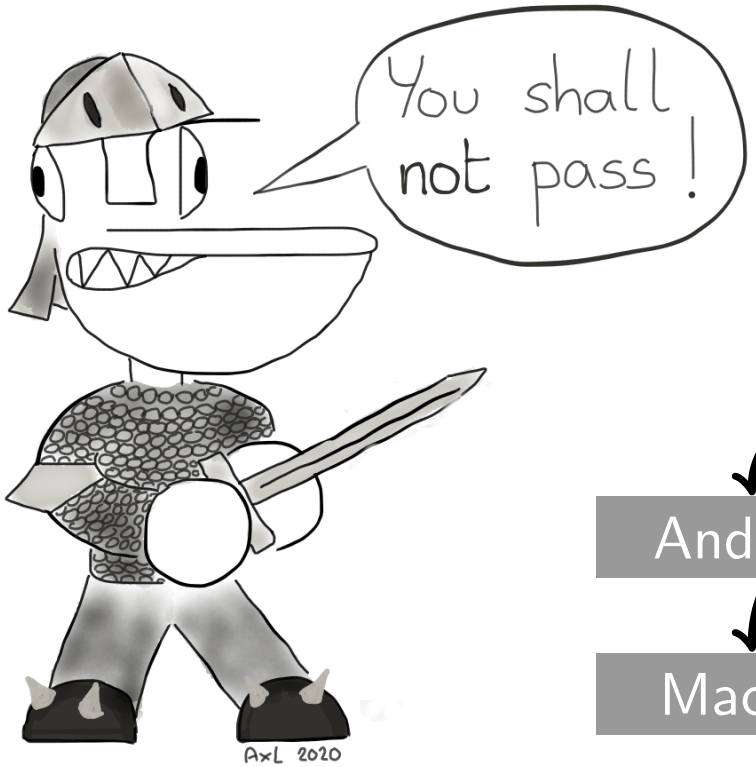
Dart is an **object**-oriented programming language with a **C-style** syntax

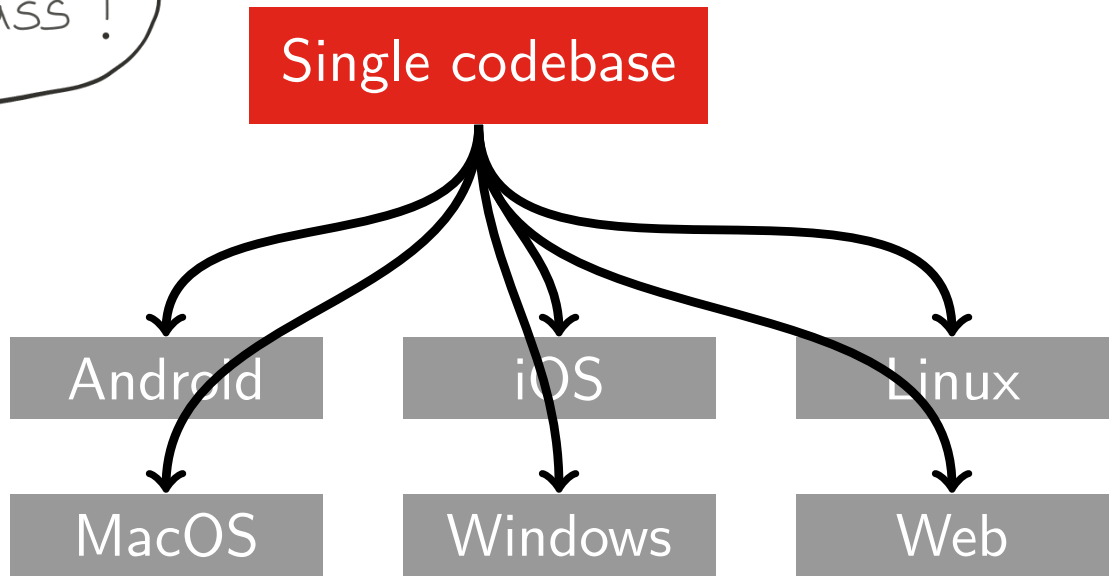
```
import 'dart:io';

void printSeparatorLine(int len) {
  for (int i=0; i<len; i++) {
      stdout.write('=');
  }
  stdout.write('\n');
}

void main(List<String> arguments) {
  printSeparatorLine(25);
  print('Welcome to BlackAlps 2023');
  printSeparatorLine(25);
}
```

One codebase to rule them all





Example: the same codebase runs on Aarch32, AAarch64, x86_64

```
blackalps.aot:         ELF 64-bit LSB shared object,
↪   x86-64, version 1 (SYSV), dynamically linked,
↪   BuildID[md5/uuid]=..., with debug_info, not
↪   stripped
blackalps.arm32.aot: ELF 32-bit LSB shared object,
↪   ARM, EABI5 version 1 (SYSV), dynamically
↪   linked, BuildID[md5/uuid]=..., with debug_info,
↪   not stripped
blackalps.arm64.aot: ELF 64-bit LSB shared object,
↪   ARM aarch64, version 1 (SYSV), dynamically
↪   linked, BuildID[md5/uuid]=..., with debug_info,
↪   not stripped
```

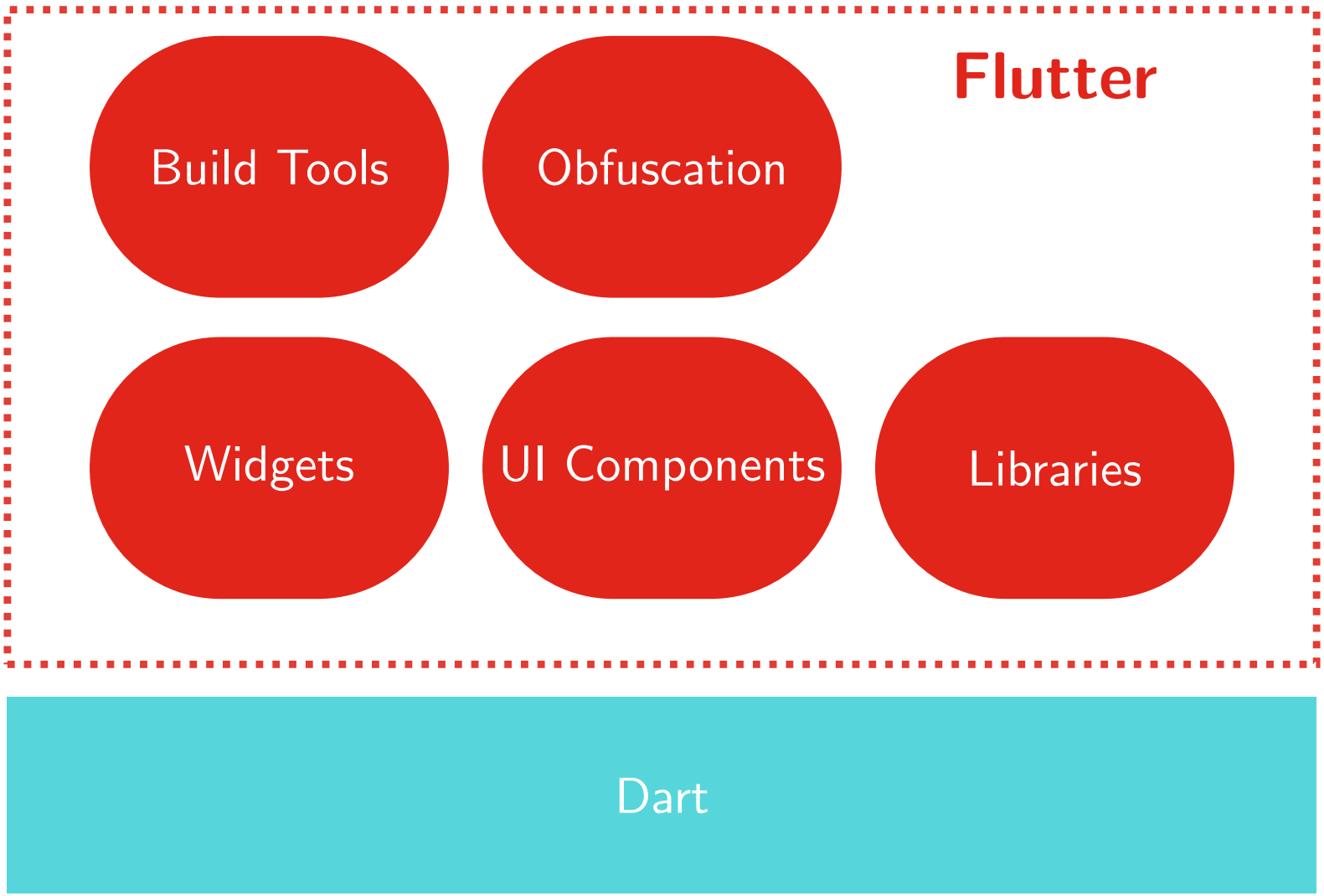
Have you noticed? AOT!

```
    blackalps.aot :       ELF 64-bit LSB shared
↪   object, x86-64, version 1 (SYSV), dynamically
↪   linked, BuildID[md5/uuid]=..., with debug_info,
↪   not stripped
```

`dart compile aot-snapshot|kernel|jit-snapshot|exe sourcefile`

Format	**Size**	**Speed**
Kernel snapshot	Small	Slow
Just In Time (JIT) snapshot	Big	Slow+
Ahead of Time (AOT) snapshot	Medium	Fast
Self contained executable	Big+	Fast

Flutter is a UI framework which uses the Dart language

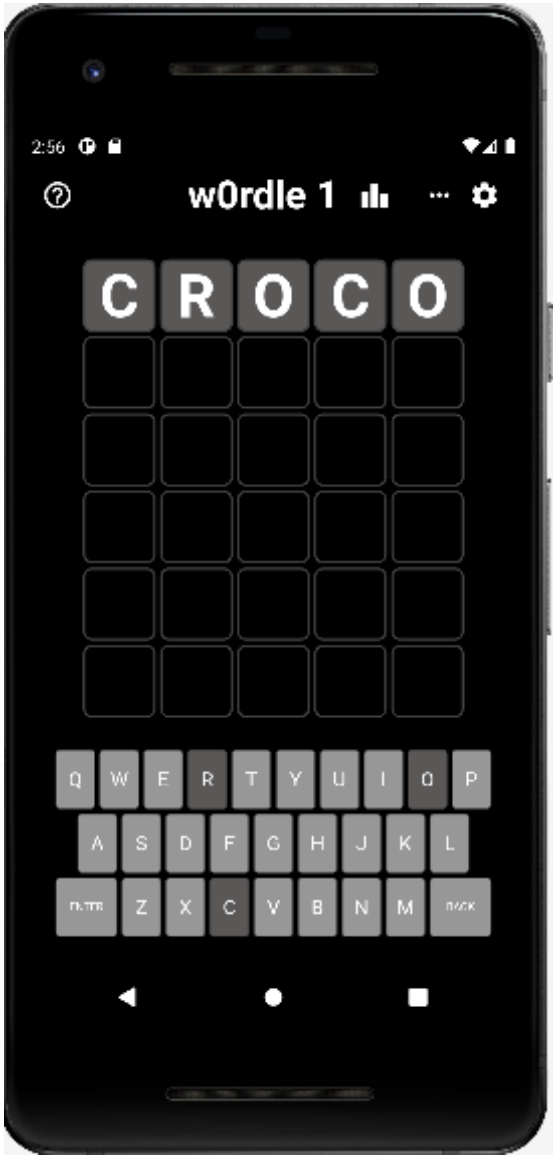


Flutter execution formats

Format	Speed	Comments
Kernel snapshot	Slow	**Portable**. Used for Flutter **Debug** builds. Easy to reverse
AOT snapshot	Fast	**Natively** compiled. Used for Flutter **Release** builds. Difficult to reverse

A **single codebase** for **multiple** platforms
Compiled **natively** for performance

Ph0wn CTF 2022 challenge



- A clone of the wordle game, in Flutter
- Game code from https://github.com/johnnysbug/flutterdle
- W0rdle 1 and 2 at Ph0wn CTF 2022
- Write-ups: https://github.com/ph0wn/writeups/tree/master/2022/reverse/w0rdle
- **Ph0wn 2023 is November 24-25** - just sayin' ! https://ph0wn.org

W0rdle 3

- Released on January 27, 2023
- **Unsolved**
- Download it from GitHub
- Difficult because it requires reversing **Flutter**

The code is very simple!

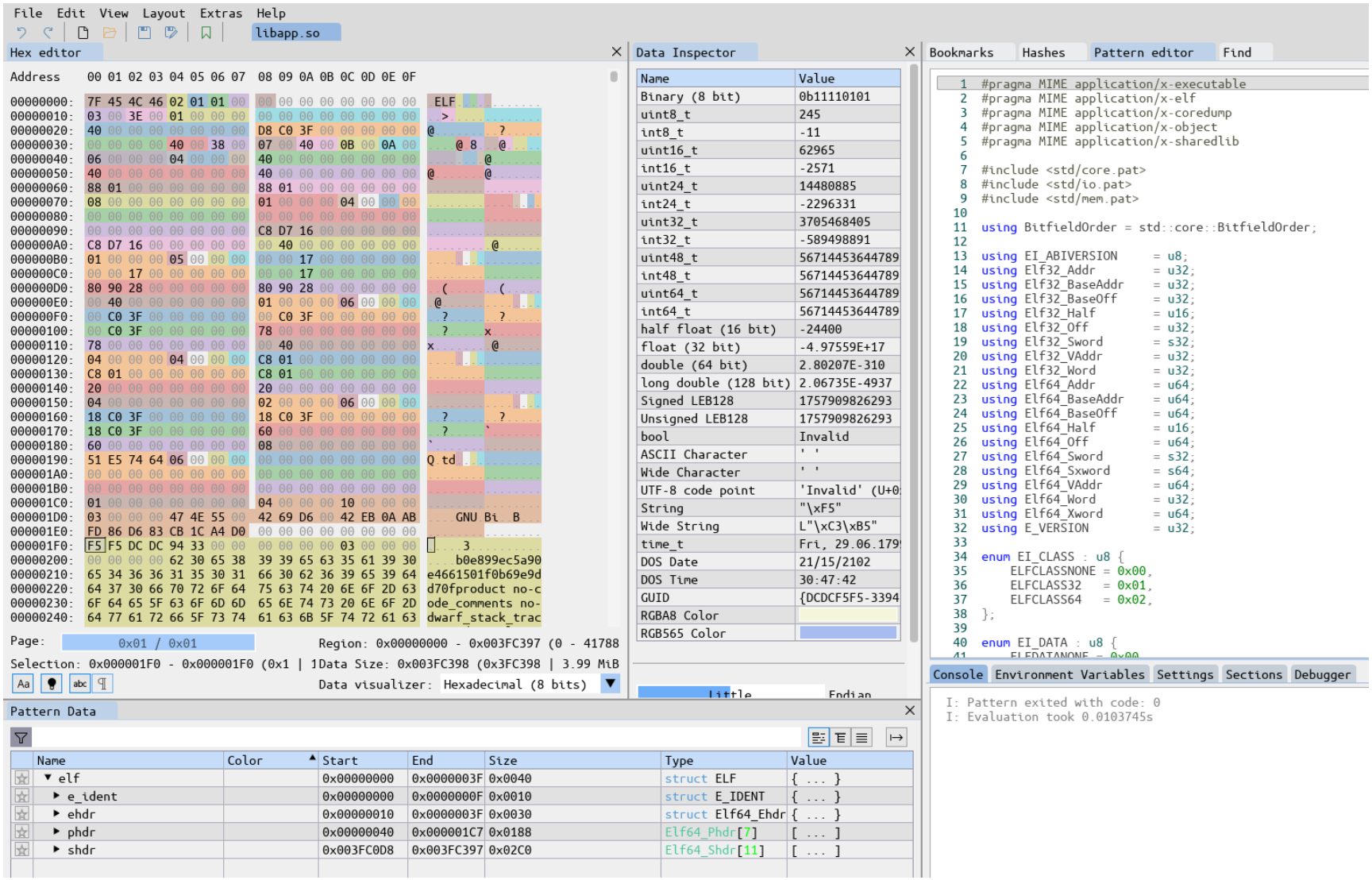
```
String hiddenName() {
  if (stage == 3) {
    hidden;
    hidden;
    List <int> encrypted_flag = [ ... ];
    int i = 0;

    for (i = 0; i < encrypted_flag.length; i++) {
      hidden; // are you up to it?
    }
    print( THEFLAG );
    return THEFLAG;
  }
}
```

Reconnaissance: where is Flutter code located?

- `classes.dex`: contains Dalvik to Flutter glue
- `./lib/arm64-v8a`
- `./lib/armeabi-v7a`
- `./lib/x86_64`: the **same** code is compiled natively for each architecture. Select the one you prefer.
- `./lib/PLATFORM/libflutter.so`: Flutter implementation
- `./lib/PLATFORM/libapp.so`: Dart payload code!

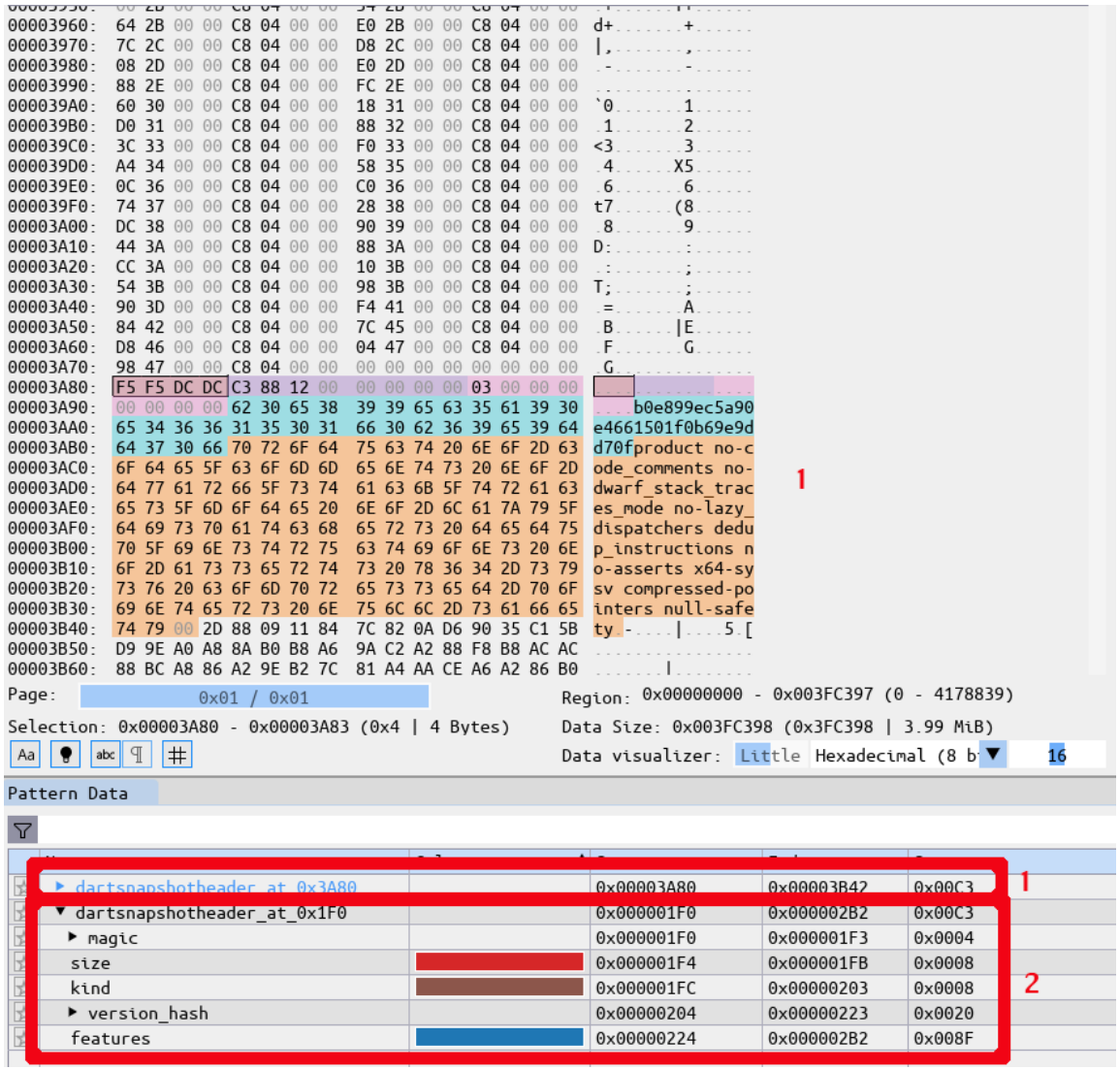
libapp.so is an ELF



ImHex: https://github.com/WerWolv/ImHex

ELF Pattern: https://github.com/WerWolv/ImHex-Patterns/blob/master/patterns/elf.hexpat

Dart AOT Format



- Documentation is ... the code
- It changes *regularly*, sometimes in-depth
- Doldrums and Darter are **obsolete**
- ImHex Dart AOT Pattern: https://github.com/cryptax/ImHex-Patterns/blob/master/patterns/dartaot.hexpat (Work in Progress)

2 snapshots: 1 VM snapshot, 1 isolate snapshot

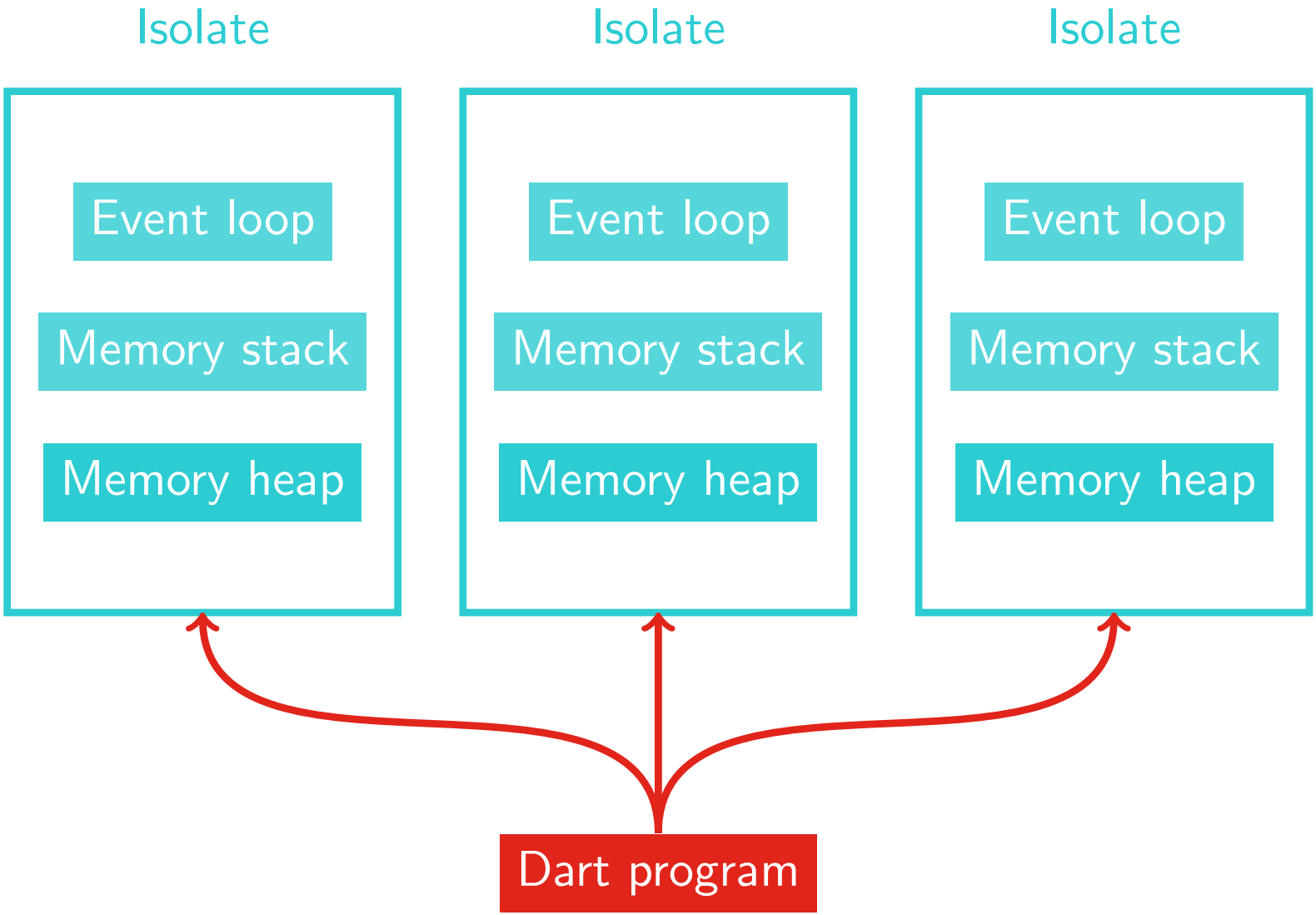
libapp.so contains **2 snapshots**

```
$ objdump -T libapp.so

libapp.so:     file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
0170000 g    DO .text   00047a0  _kDartVmSnapshotInstructions
01747a0 g    DO .text   02848e0  _kDartIsolateSnapshotInstructions
00001f0 g    DO .rodata 00003890 _kDartVmSnapshotData
0003a80 g    DO .rodata 00169bb0 _kDartIsolateSnapshotData
00001c8 g    DO .note.gnu.build-id 0000020 _kDartSnapshotBuildId
```

Dart Isolates



Locating the Dart payload

- VM snapshot is for Flutter's implementation
- **Payload** is in an **Isolate snapshot**.

```
objdump -T libapp.so
...
00000000000001f0 g    _kDartVmSnapshotData
0000000000003a80 g    _kDartIsolateSnapshotData <<<<<<

python3 flutter-header.py -i ./libapp.so
========== Flutter Snapshot Header Parser ============
...
Snapshot
        offset  = 14976 (0x3a80)
        size    = 1214659
        kind    = SnapshotKindEnum.kMessage
        version = 3.3.4
        features= product no-code_comments no-dwarf_stack_traces_mode
↪   no-lazy_dispatchers dedup_instructions no-asserts x64-sysv
↪   compressed-pointers null-safety
```

https://github.com/cryptax/misc-code/blob/master/flutter/flutter-header.py

We get stage 1 flag but not stage 3

strings libapp.so ||grep ph0wn

```
ph0wn
file:///home/axelle/prog/challenges/ph0wn2022/...
ph0wn{
ph0wn w0rdle
ph0wn CTF is awesome
ph0wn{you_w1n_darts_stage1}
I love ph0wn CTF. I guessed w0rdle
```

Who is using stage 1 flag? Nobody?!

```
[0x0004a5e7]> f~ph0wn
0x0004a5e7 28 str.ph0wnyou_w1n_darts_stage1
0x000572ab 37
 ↪  str.I_love_ph0wn_CTF._I_guessed_w0rdle__
[0x0004a5e7]> axt str.ph0wnyou_w1n_darts_stage1
[0x0004a5e7]>
```

Flutter obfuscates class, function, variable names

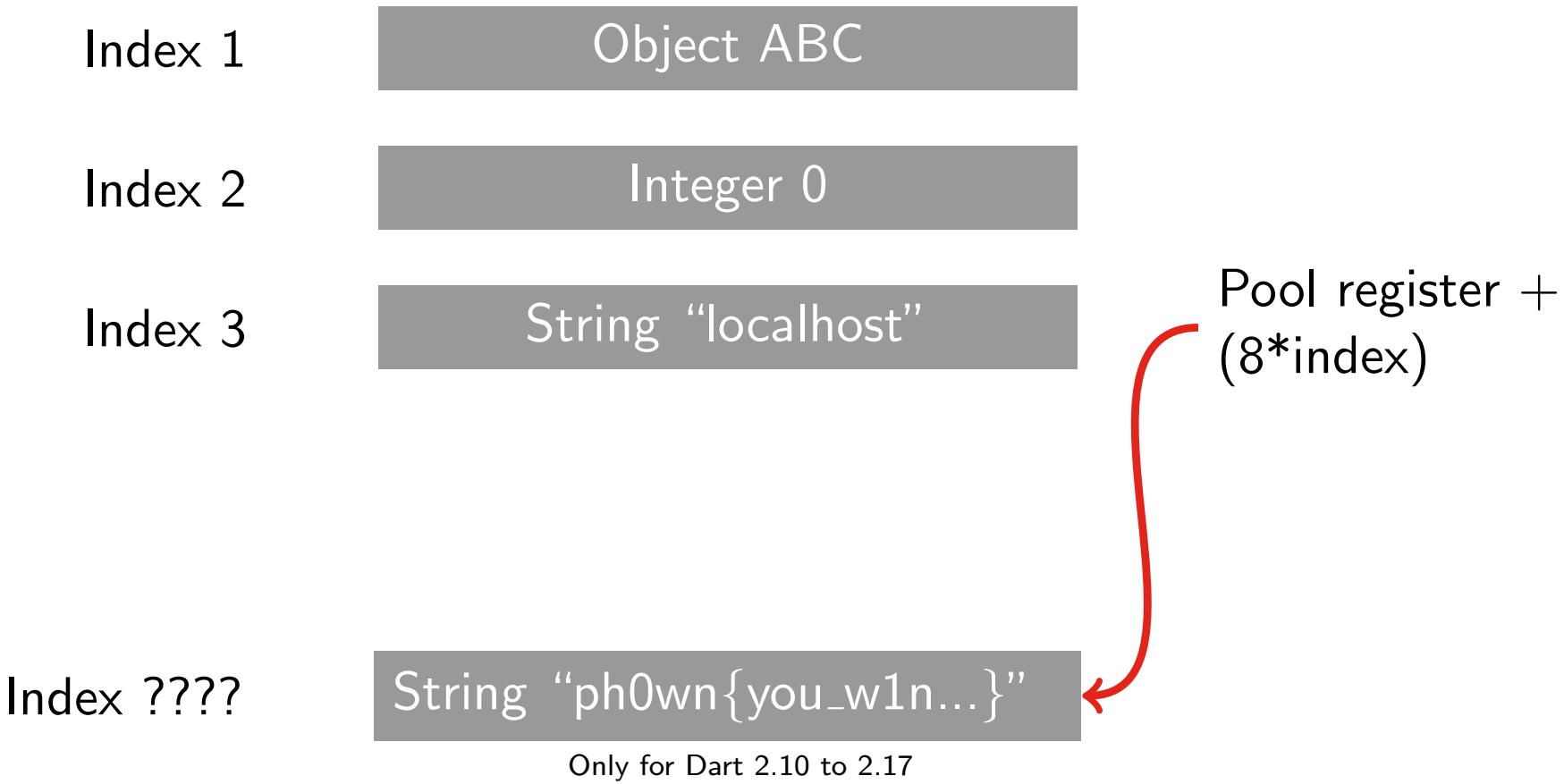
Normally, **Dart** code is *not* obfuscated:

```
[0x00050000]> afl <--- list functions
0x00077568   53   1119 sym._Uri._makeHost
0x000779c8  114   2286 sym._Uri._normalizeRegName
0x000782b8  126   2344 sym.Uri.parseIPv6Address
0x00078be0   37    888 sym.Uri._parseIPv4Address
```

Flutter adds obfuscation

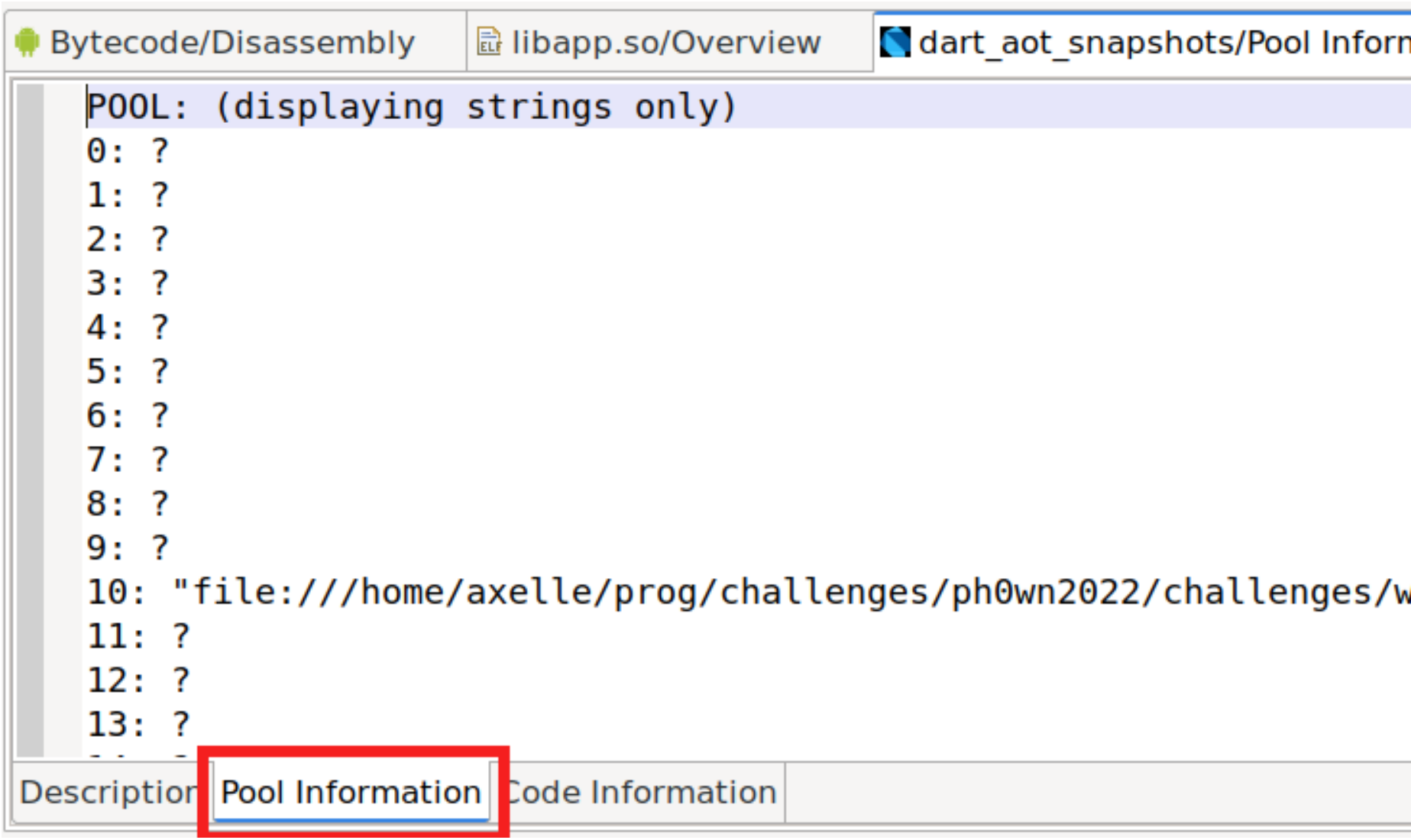
```
[0x00170000]> afl
0x003f7b80    9    212 fcn.003f7b80
0x003db6c0   28    464 fcn.003db6c0
0x0036f024   14    232 fcn.0036f024
0x00332678   14    232 fcn.00332678
```

Strings, constants etc are accessed indirectly through an Object Pool

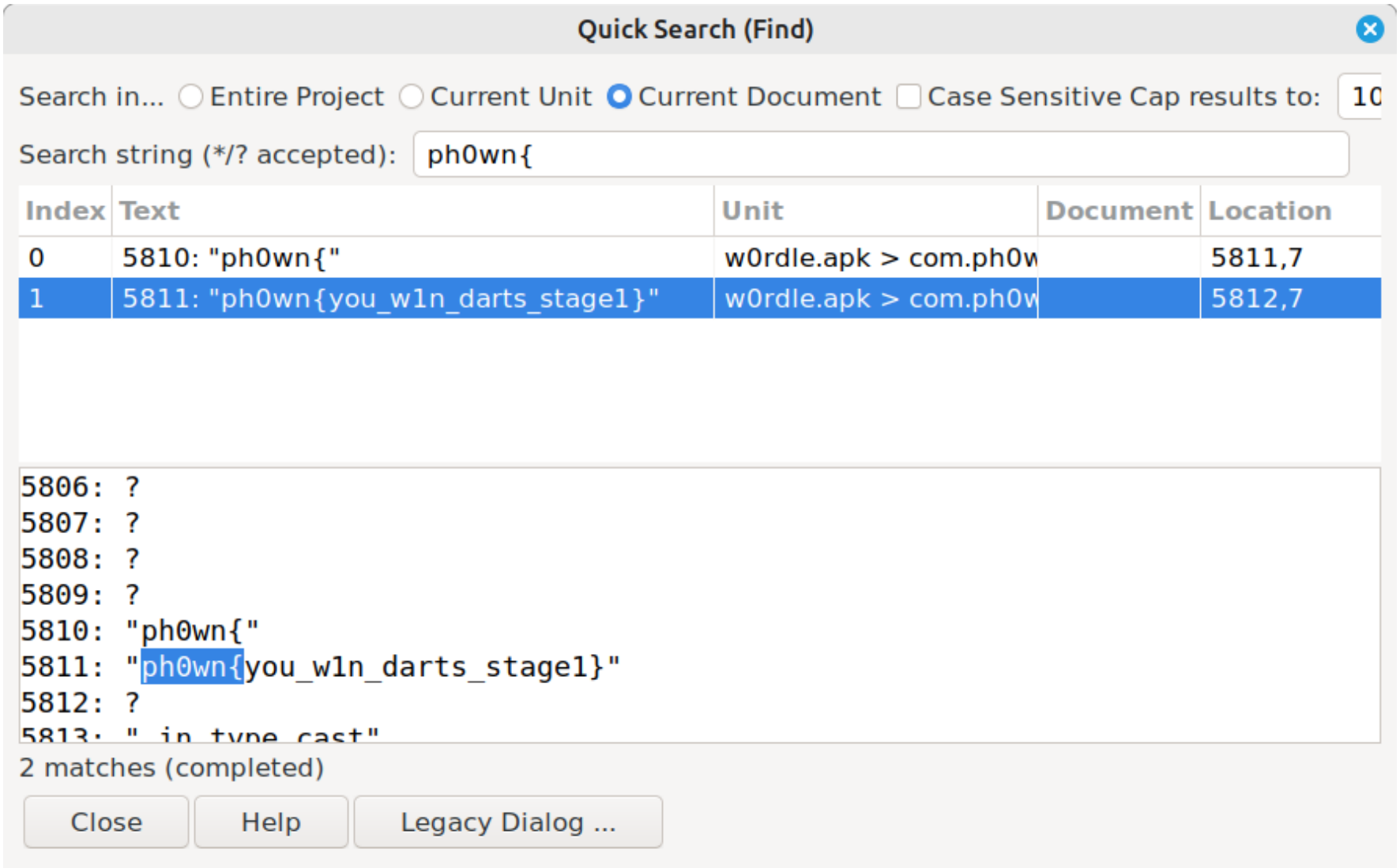


Only for Dart 2.10 to 2.17

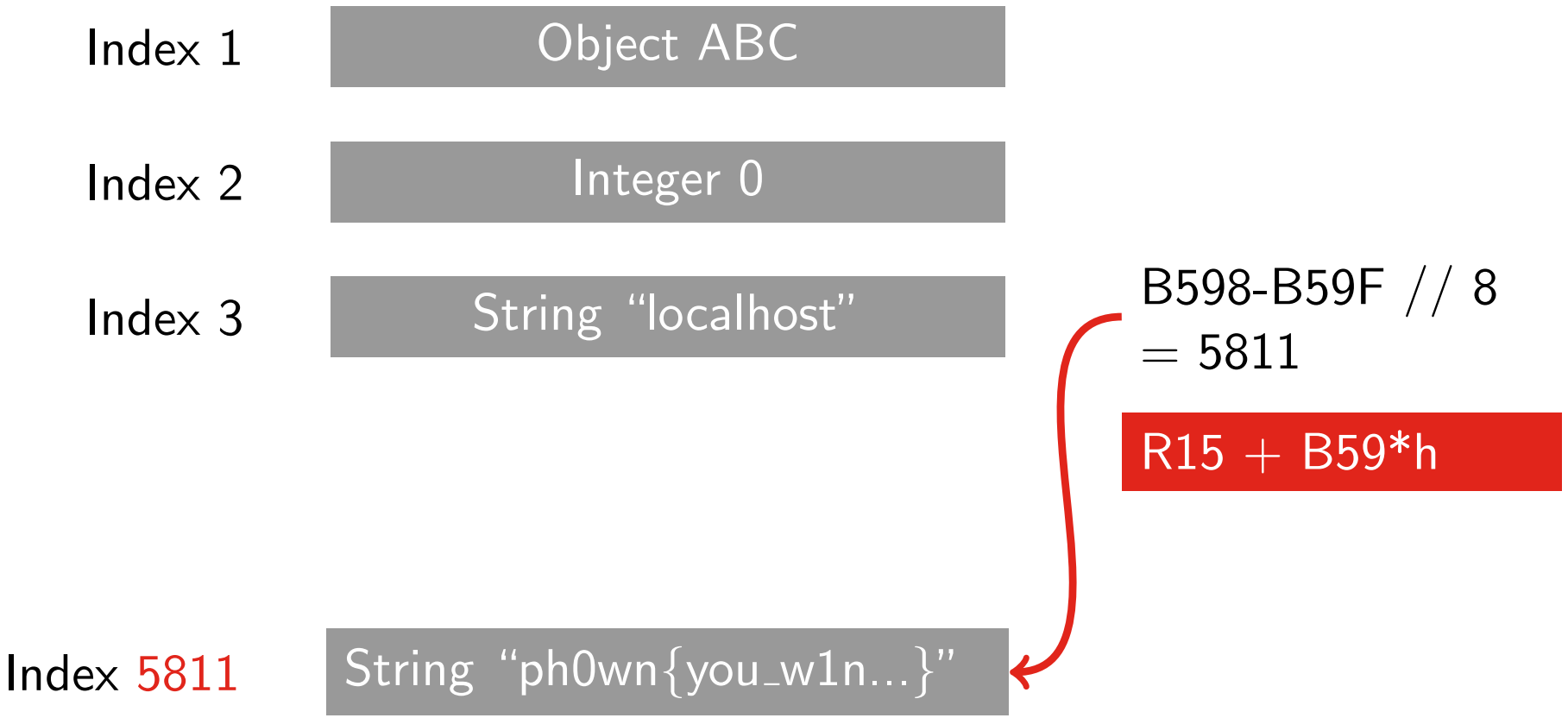
JEB is able to read strings in the Object Pool



JEB is able to read strings in the Object Pool



Accessing pool object index 5811



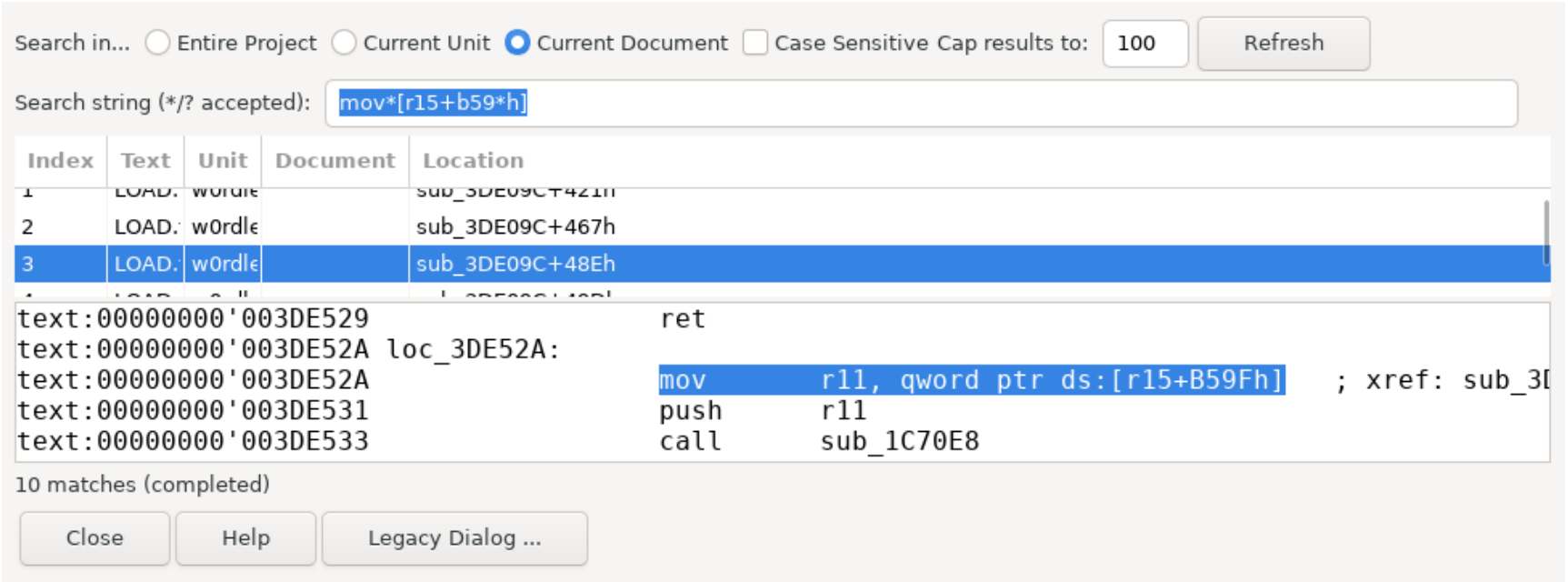
Dart assembly defines its own registers!

	x86_64	**Aarch32**	**Aarch64**
Object Pool	R15	R5	X27
Custom Stack Pointer	*(rsp)*	*(sp)*	X15
Pointer to VM thread	R14	R10	X26

Loading a Pool Object

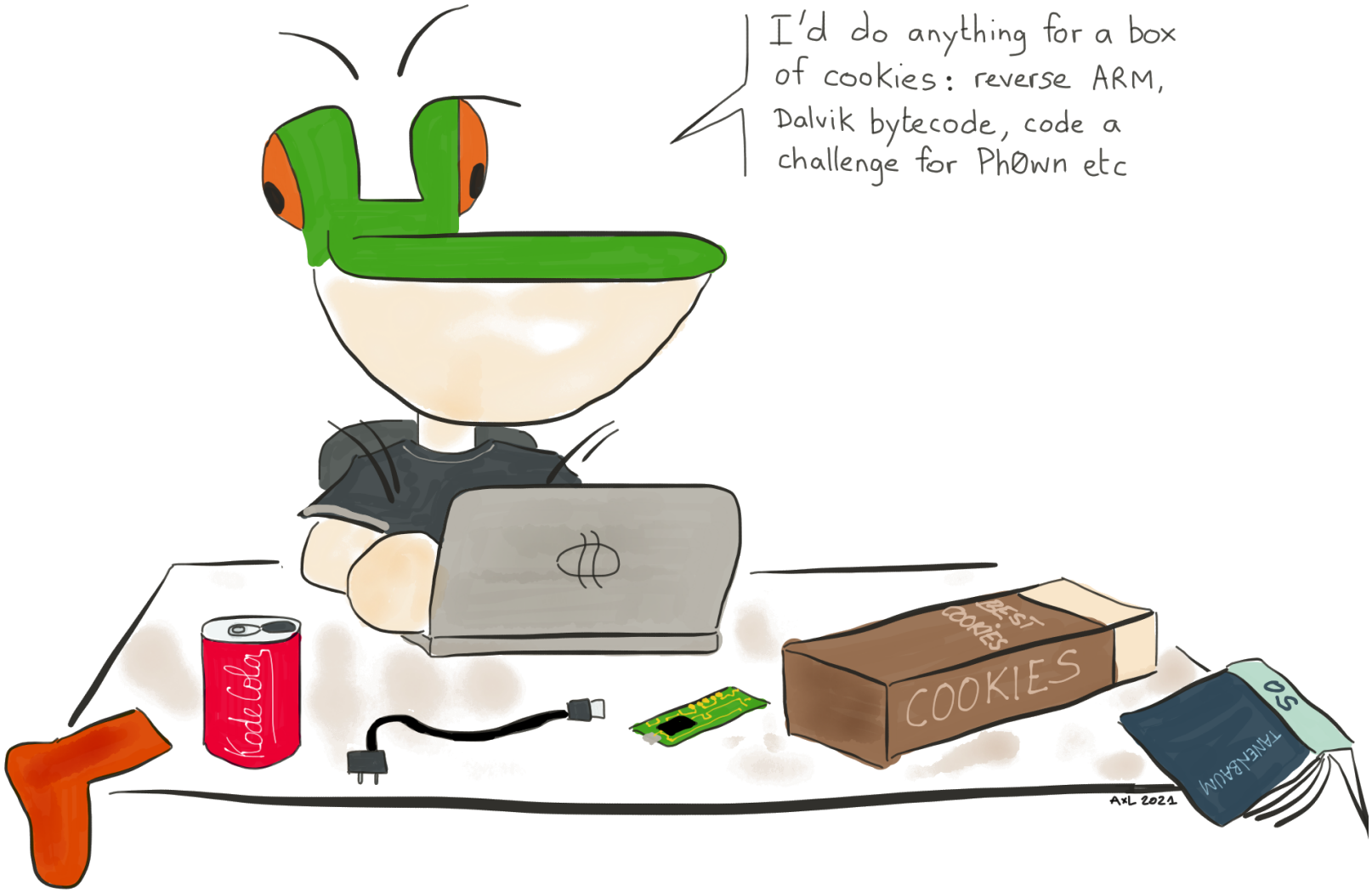
- x86_64: `mov REGISTER, qword ptr ds:[R15 + VALUEh]`
- Aarch32: `LDR REGISTER, [R5, #VALUEh]`
- Aarch64: `LDR REGISTER, [X27, #VALUEh]`

Who is using stage 1 flag?

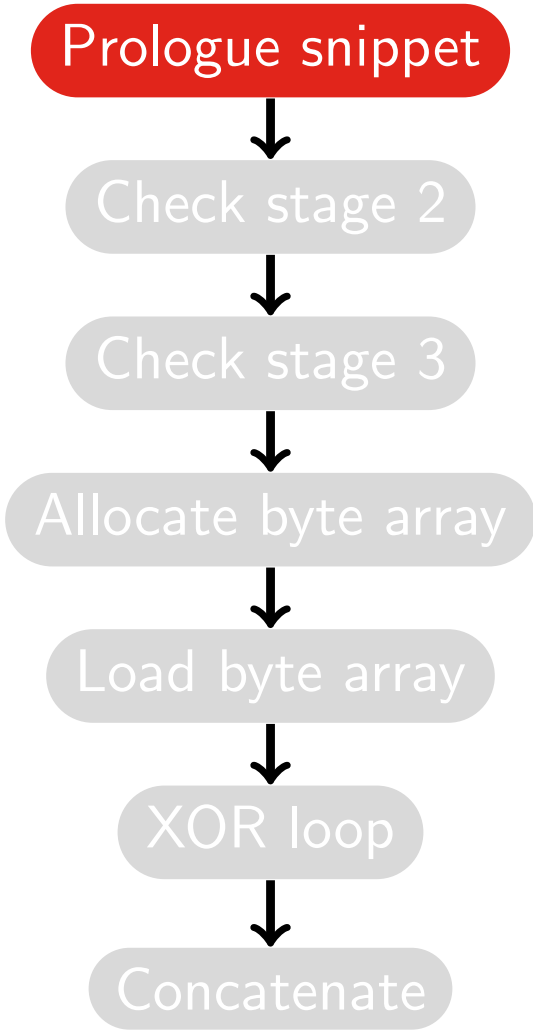


Search for the instruction

You won't eat cookies, you'll eat assembly



Disassembling sub_3DE09C



Function prologue snippet

```
sub_3DE09C        proc
  ; push base pointer on the stack
  push rbp
  mov rbp, rsp
  ; allocate 16 bytes
  sub rsp, 10h
  ; compare stack pointer with r14+38h
  cmp  rsp , qword ptr ds:[ r14  + 38h]
  ; if stack pointer is <= [r14 + 0x38 ]
  ; jump stack overflow error
  jbe        loc_3DE545
```

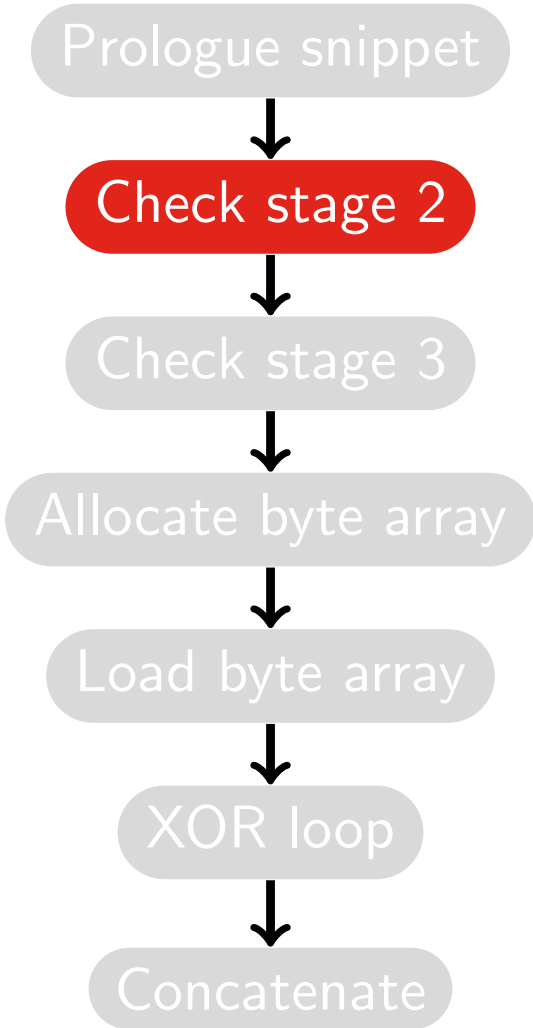
- Function name is **obfuscated** thanks to Flutter...
- **r14** holds a pointer to the current thread
- **rsp** is the (standard) stack pointer register
- stack overflow protection

Same mechanims for Aarch64 (and Aarch32)

```
; push frame pointer and link register on the stack
STP         X29, X30, [X15, #FFFFFFF0h]!
; update frame pointer
MOV         X29, X15
; allocate 16 bytes on the stack
SUB         X15, X15, #10h
; stack overflow check
LDR         X16, [X26, #38h]
CMP         X15, X16
B.LS        loc_3D75DC
```

- **Specific**: AAarch64 uses a **custom stack pointer: X15**
- **X26** holds a pointer to the **current thread**

Disassembling sub_3DE09C

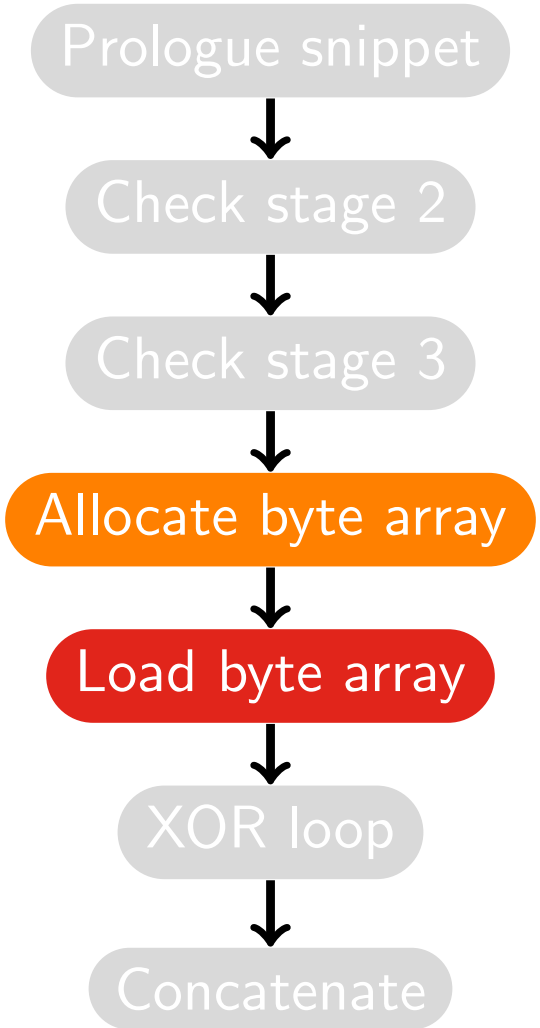


Compare stage value

```
; check argument == 2
mov       rax, qword ptr ss:[rbp+10h]
mov       rcx, qword ptr ds:[rax+2Bh]
cmp       rcx, 2
jnz       loc_3DE2FC
```

- W0rdle has 3 stages
- We guess there is a case for each stage
- We need to look for **stage 3**

Disassembling sub_3DE09C



This code loads byte values in an array

```
mov         qword ptr ss:[rbp-8], rax
mov         r11d, A2h
mov         dword ptr ds:[rax+Fh], r11d
mov         r11d, BCh
mov         dword ptr ds:[rax+13h], r11d
mov         r11d, B2h
mov         dword ptr ds:[rax+17h], r11d
mov         r11d, A6h
mov         dword ptr ds:[rax+1Bh], r11d
mov         r11d, D0h
mov         dword ptr ds:[rax+1Fh], r11d
mov         r11d, BCh
mov         dword ptr ds:[rax+23h], r11d
mov         r11d, 86h
mov         dword ptr ds:[rax+27h], r11d
```

Dart has no character type

- No `char`, `unsigned char`, `byte`
- An array of characters is a list of **integers**, or better, `Uint8List`

```
List <int> myArray1 = ... ;
Uint8List myArray2 = ... ;
```

- **Strings** are immutable, use `StringBuffer` to manipulate them

```
StringBuffer buffer = StringBuffer(''Black'');
buffer.write(''Alps'')
print(buffer.toString());
```

How to store character 'A' = 0x41 ?

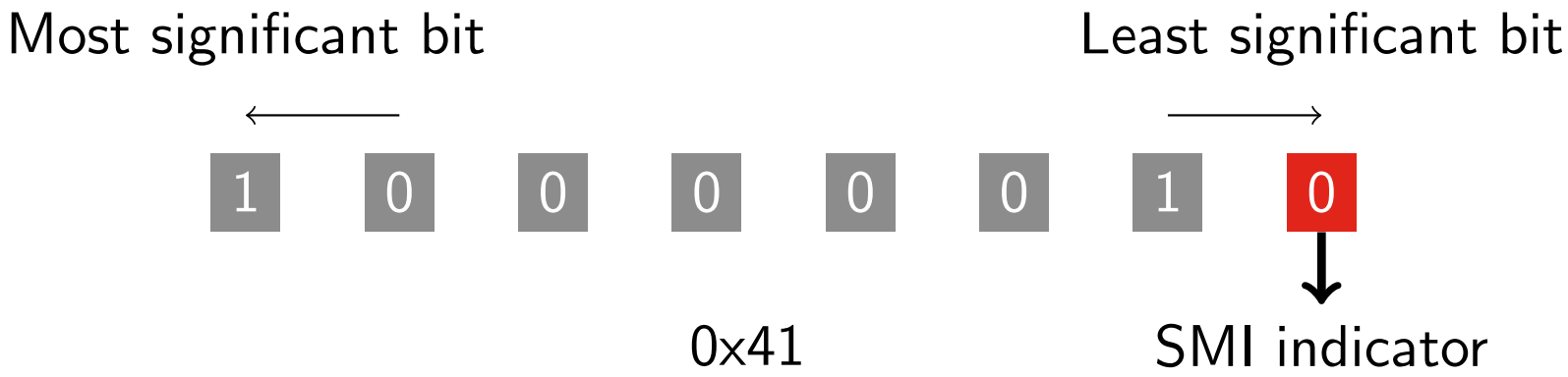
Is `mov REGISTER, 41h` correct?

How to store character 'A' = 0x41 ?

Is `mov REGISTER, 41h` correct?

No because Dart has 2 different representations for integers:

- **Small Integers** (SMI). They fit on **31 bits**. Least significant bit set to 0.
- **Medium Integers** (Mint). Bigger.



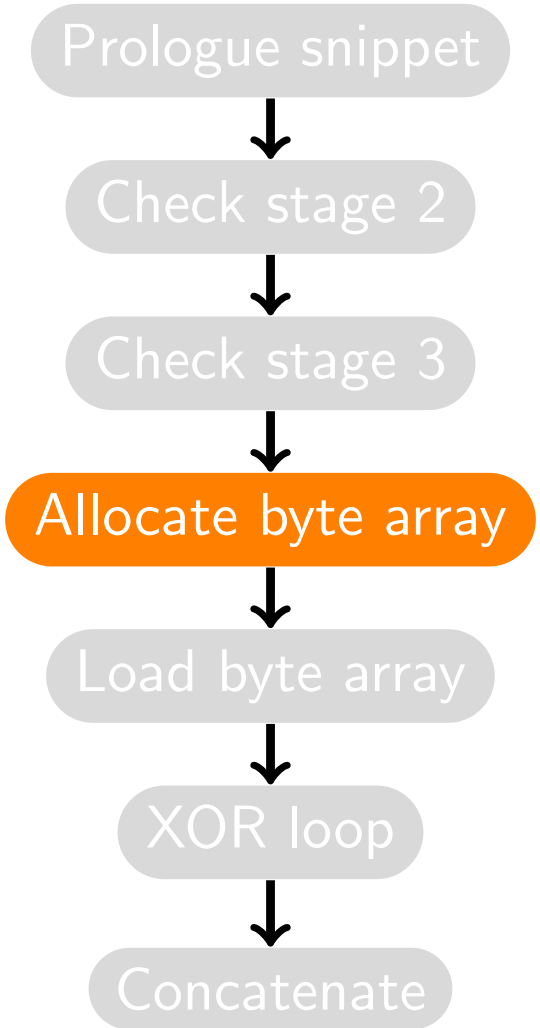
The correct instruction is `mov REGISTER, 82h`

Array of SMIs

```
mov         qword ptr ss:[rbp-8], rax
; 0xA2 is a SMI. Represents 0x51
mov         r11d, A2h
mov         dword ptr ds:[rax+Fh], r11d
; 0xBC is a SMI. Represents 0x5e
mov         r11d, BCh
mov         dword ptr ds:[rax+13h], r11d
mov         r11d, B2h
mov         dword ptr ds:[rax+17h], r11d
mov         r11d, A6h
mov         dword ptr ds:[rax+1Bh], r11d
```

byte array: 51 5E 59 53

Disassembling sub_3DE09C

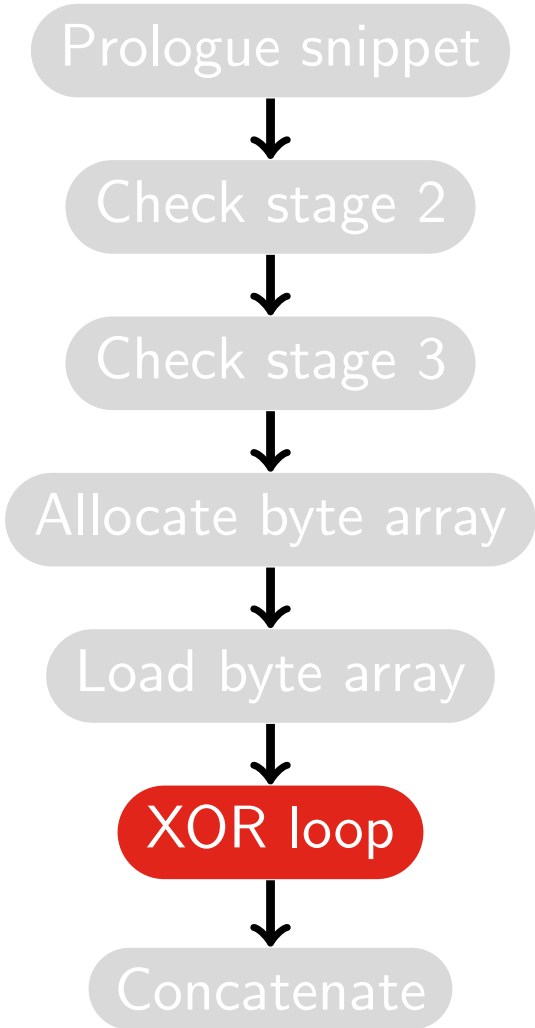


Array allocation

```
mov     rbx, qword ptr ds:[r15+A5Fh]
mov     r10d, 2Ch
call    sub_40277C
```

- The text is **22** characters long
- $0x2C = 44$. This is the **SMI encoding for 22**.
- sub_40277C: **stub for array allocation**

Disassembling sub_3DE09C



Loop end condition

```
; check loop end condition
  cmp       rdi, 22
  jnl       loc_3DE4A2
```

- **rdi** is the counter
- Loop until counter reaches value = 22
- Parses each byte of the byte array

Loop

```
; get array[i]
  mov       eax, dword ptr ds:[rcx+4*rdi+Fh]
  add       rax, qword ptr ds:[r14+48h]
  movsxd    rdx, eax
; it's an SMI: divide by 2!
  sar       rdx, 1
  jnb       loc_3DE44F
  mov       rdx, qword ptr ds:[rax+7]
  loc_3DE44F:
; XOR encryption
  xor       rdx, 37h
```

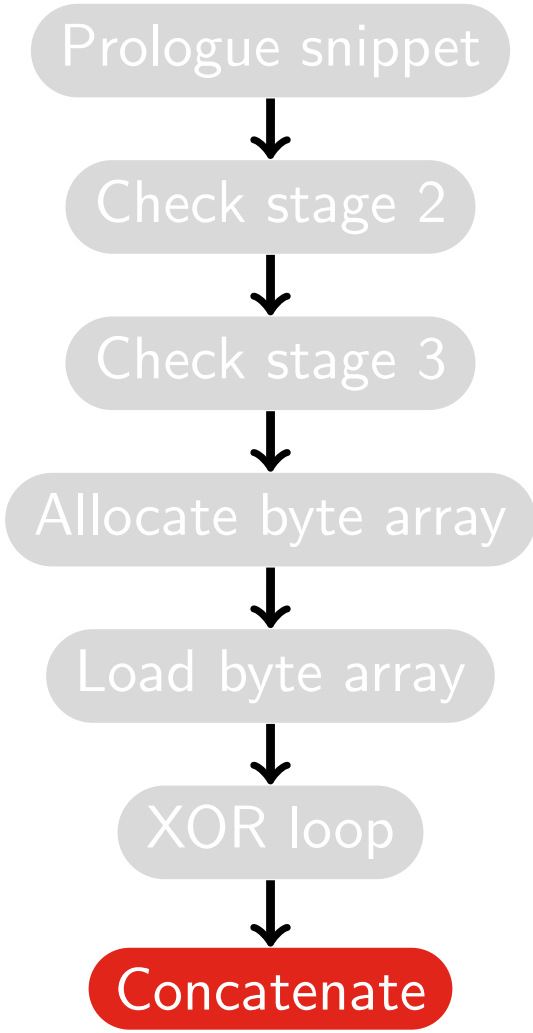
- Read each byte
- SMI representation is double, so assembly divides the value by 2 to recover the real byte value
- XOR with 0x37

Decrypting the array

```
table = [0xa2, 0xbc, 0xb2, 0xa6, 0xd0, 0xbc, 0x86,
 ↪  0xd0, 0xe6, 0xbc, 0xa2, 0xa2, 0xbc, 0xa8, 0x84,
 ↪  0xb6, 0x86,0xd0, 0xb2, 0x0e, 0x80, 0x10]
ciphertext = [i // 2 for i in table]
plaintext = [chr(i ^ 0x37) for i in ciphertext]
print("Plaintext: ", ''.join(plaintext))
```

Plaintext: find_it_Difficult_n0w?

Disassembling sub_3DE09C



Dart's calling conventions

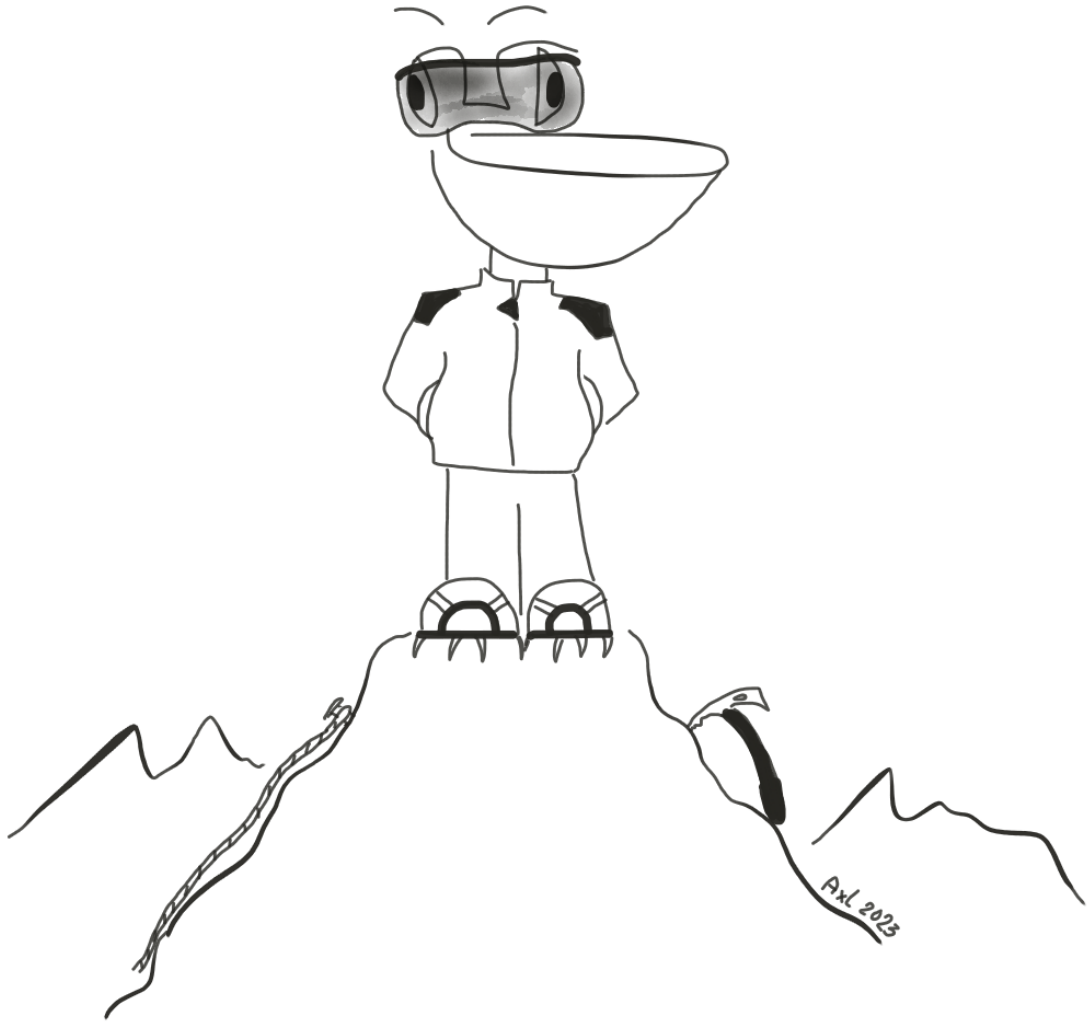
```
; load object index 5810: ''ph0wn{''
mov       r11, qword ptr ds:[r15+B597h]
push      r11
; rax holds ''find_it_Difficult_n0w?''
push      rax
call      string_concat
```

- Both arguments are pushed on the stack: this is *unusual*
- x86_64 usually gets the first arguments from registers, and only subsequent argument from the stack
- Same behaviour on AAarch32 and AAarch64: pushed on the stack
- We concatenate `ph0wn{` with `find_it_Difficult_n0w?`

Building the flag

```
; load object index 5810: ''ph0wn{''
mov       r11, qword ptr ds:[r15+B597h]
push      r11
; rax holds ''find_it_Difficult_n0w?''
push      rax
call      string_concat
pop       rcx
pop       rcx
; rax = ''ph0wn{find_it_Difficult_n0w?''
push      rax
; load object index 583 = '}'
mov       r11, qword ptr ds:[r15+123Fh]
push      r11
call      string_concat
```

ph0wn{find_it_Difficult_n0w?}



Easy, wasn't it?

Is it more difficult on Aarch32 or Aarch64?

- **Dart registers change**, e.g. Object Pool register is R5 for Aarch32 and X27 for Aarch64
- Specificities: **custom stack pointer X15 on Aarch64**

Several different choices:

XOR in a single instruction on Aarch32

```
EOR R3, R9, #37h
```

XOR: Two instructions on Aarch64:

```
MOVZ X16, #37h
EOR X4, X3, X16
```

Divide with a shift right

```
ASRS R9, R0, #1
```

Divide by copying bit fields

```
SBFX X1, X0, #1, #31
```

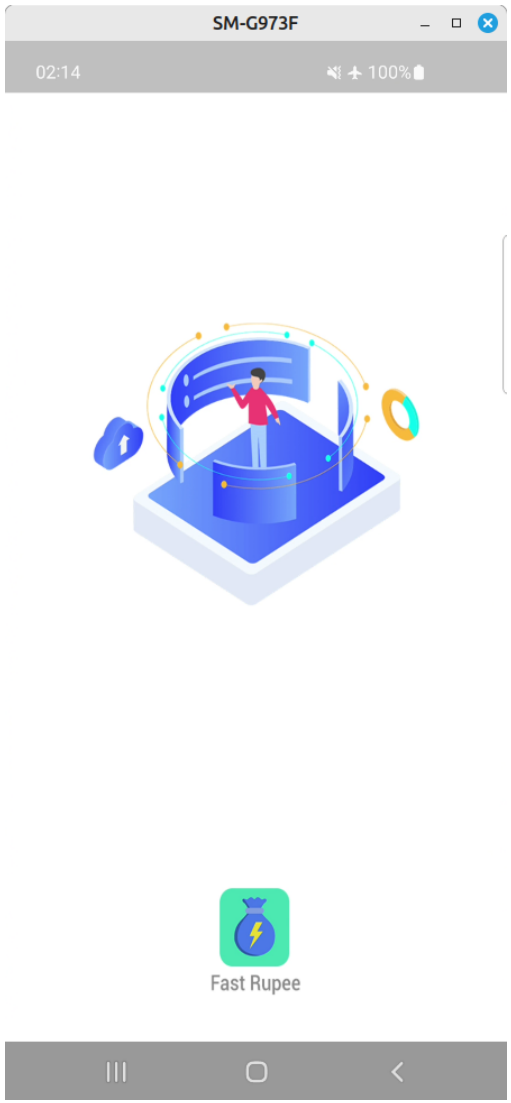
Summary of Dart specificities

1. Object Pool
2. Small integers are represented with least significant bit set to 0
3. Use of specific registers for Object Pool, VM thread
4. Use of a specific stack register on Aarch64
5. Unconventional passing of arguments: all on the stack

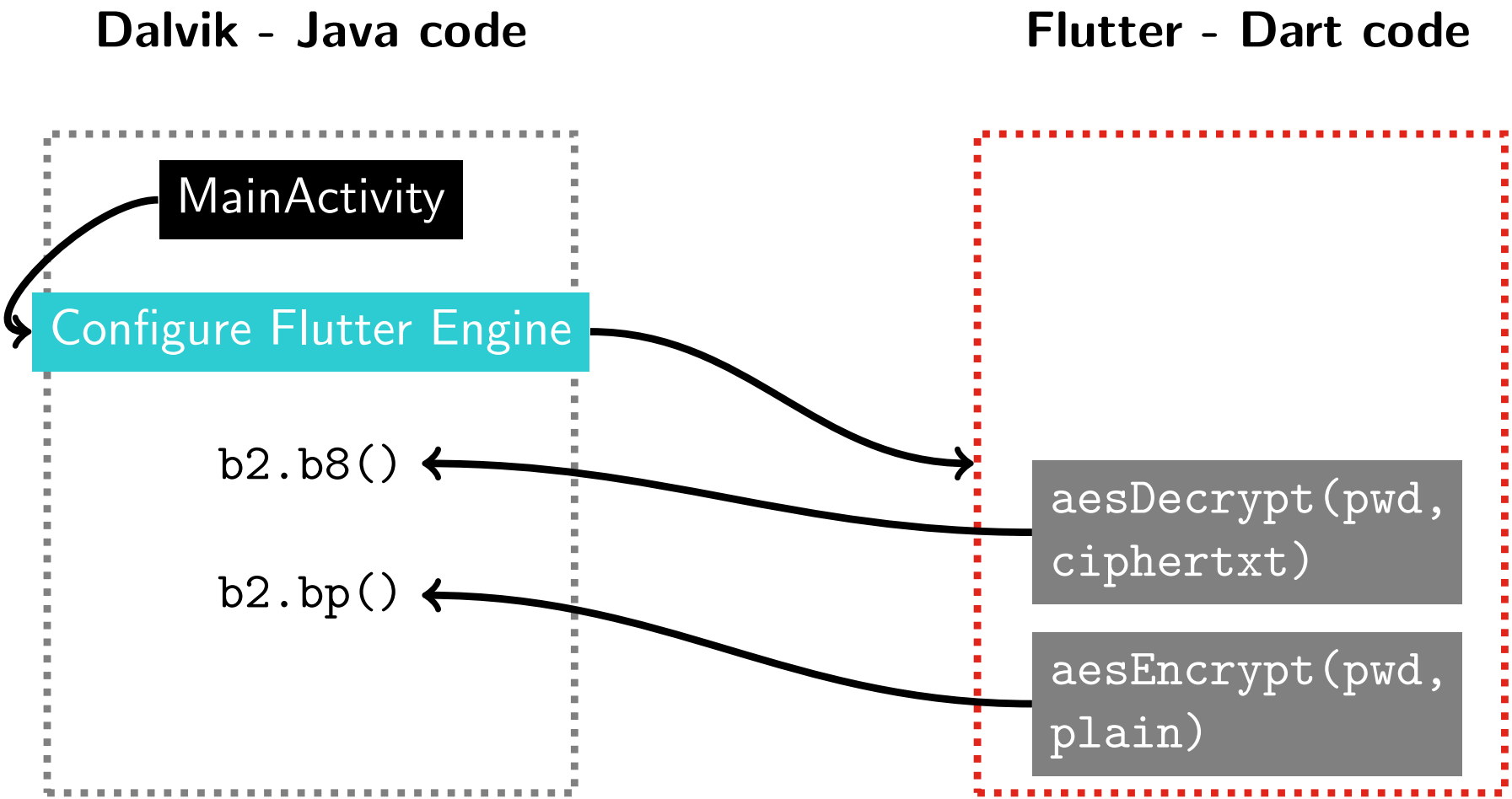
$+$ Flutter obfuscation

MoneyMonger

- **Loan scam** in India starting May 2022
- **Flutter** version noticed in **December 2022**
- **Threatens to leak pictures to contacts, harrasses victims**
- Leaks GPS, call log, SMS list, installed apps, contact list...

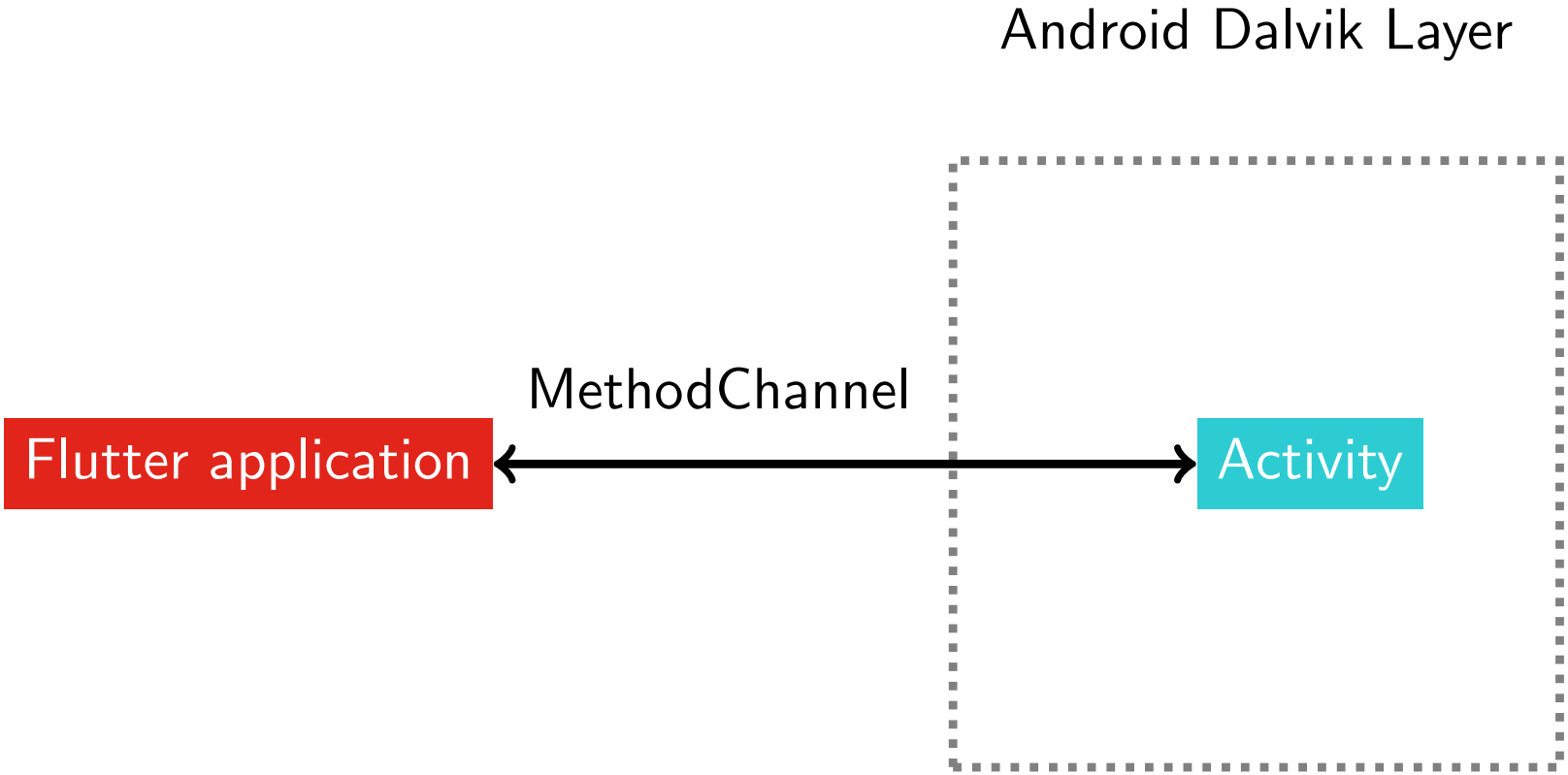


MoneyMonger Java/Flutter architecture



Platform Channel communication

- **Platform Channels** are implemented by **Flutter**, to pass messages between *client (Flutter UI)* and *host (Android)*.
- **MoneyMonger** uses it to **access code they already have in Java**, or don't know how to implement in Dart



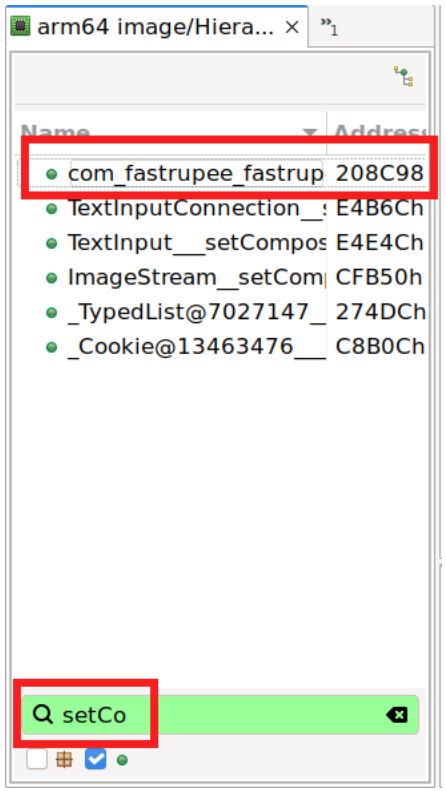
MethodChannel glue in Java

```
if(l8.a6(methodCall0.method, "setConfig")) {
   String s = (String)methodCall0.argument("smsCount");
   if(s == null) {
     return null;
   }

   sharedPreferences$Editor0 = this.getSharedPreferences("data",
↪   0).edit().putString("smsCount", s);
   sharedPreferences$Editor0.commit();
   return null;
}
```

- Dart function `setConfig` has 1 argument: `smsCount`
- Adds a `smsCount` entry with the specified value in the shared preferences

Locate setConfig in Dart



With JEB,

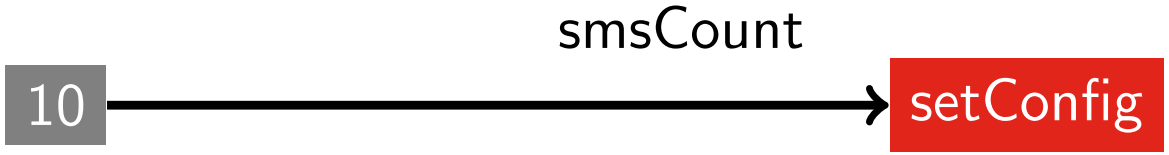
- Filter function names to `setConfig`,
- Search for `setConfig` in Code Information of Dart AOT Snapshot

```
com_fastrupee_fastrupeepeefaMyUtilsPlugin__setConfig
↪  proc
   STP        X29, X30, [X15, #FFFFFFF0h]!
   MOV        X29, X15
   SUB        X15, X15, #18h
   LDR        X16, [X26, #40h] ; VM thread
   CMP        X15, X16
   B.LS       check_overflow
```

It's a wrapper

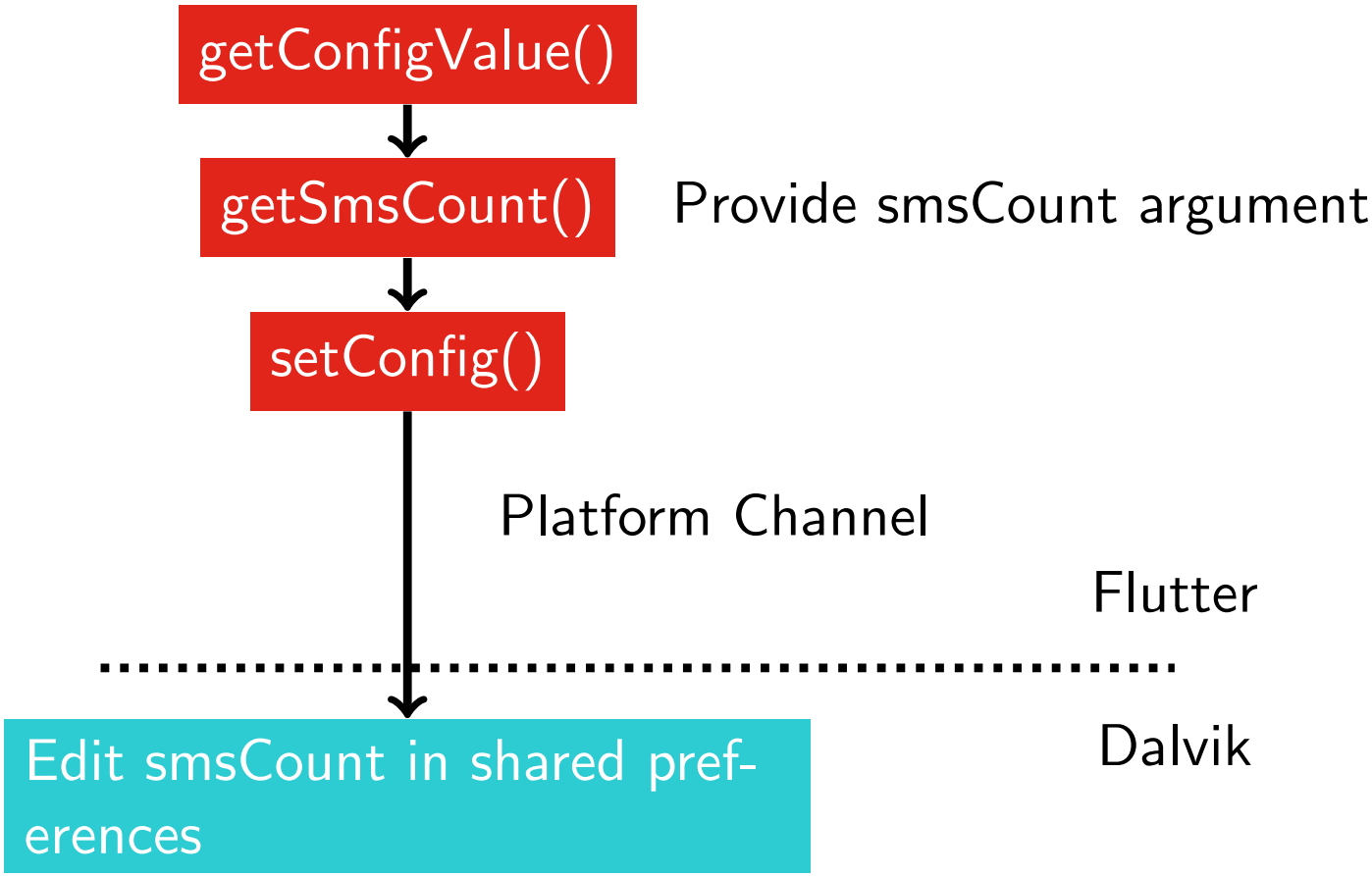
```
com_fastrupee_fastrupeepeefaMyUtilsPlugin__setConfig proc
    ...
                ADD         X17, X27, #13h, LSL #12          ; setConfig
                LDR         X17, [X17, #E48h]
                STUR        X17, [X0, #1Fh]
                LDUR        X1, [X29, #FFFFFFF0h]
                STUR        X1, [X0, #27h]
                STUR        X0, [X1, #5Fh]
                STR         X0, [X15, #FFFFFFF8h]!
                BL          ::___asyncThenWrapperHelper@4048458
```

Retrieve SMS count and pass as argument




```
BL
↪  com_fastrupee_fastrupeepeefaPermissionConfig__get:getSmsCount
STR       X0, [X15, #FFFFFFF8h]!
BL        com_fastrupee_fastrupeepeefaMyUtilsPlugin__setConfig
ADD       X15, X15, #8
```

MoneyMonger: setConfig



MoneyMonger Example: Calling AES code

```
if(l8.a6(methodCall0.method, "aesDecrypt")) {
  return
 ↪  b2.b8(((String)methodCall0.argument("password")),
 ↪  ((String)methodCall0.argument("encrypt")));
}
```

- When Dart function `aesDecrypt(password, encrypt)` is called, the job is done by Java `b2.b8(password, encrypt)`
- password and encrypt are the *name* of the 2 arguments
- Same for AES Encryption

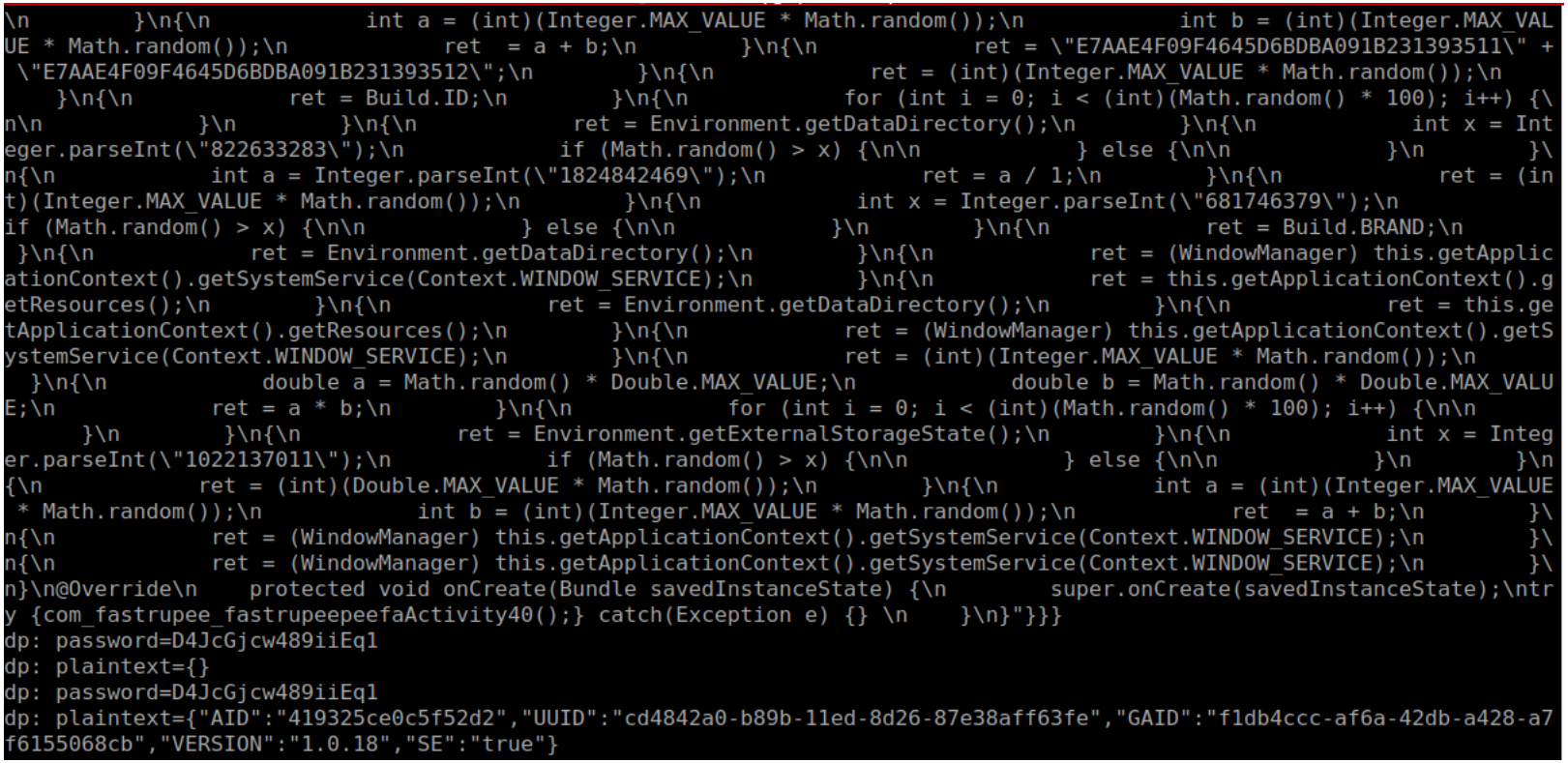
MoneyMonger: where is the key?

```
$ strings libapp.so
...
7085EC154BCD59FD103796D25254F3521C9F2C
D4JcGjcw489iiEq1 8
.8&D4
```

The key is **D4JcGjcw489iiEq1** (16 bytes)

MoneyMonger: what's encrypted/decrypted?

Frida hook `b2.b8()` and `b2.bp()`



- **Device information** is encrypted by the malware
- **Configuration** file is decrypted
- Easy way to find the **password**

Flutter-based malware: Lessons Learned

In **MoneyMonger**,

1. The malicious code is located in Dalvik
2. The decryption key is located in Flutter

With **FluHorse**, the maliciouse code is in Flutter.
It listens to incoming SMS and sends it to a remote server.

Thanks for your attention!

Ready for Raclette!

- https://github.com/cryptax/talks
- @cryptax (X, Mastodon.social)