

Unraveling the Challenges of Reverse Engineering Flutter Applications

Axelle Apvrille, Fortinet

BlackAlps, October 2023



① Introduction

What is Dart?

What is Flutter?

② W0rdle

Isolate snapshots

Object pool

Prologue snippet

Strings and Integers

Decryption Loop

Aarch32 and Aarch64

③ Reverse Malware

Platform Channel

setConfig

AES encryption

④ Conclusion



Who am I?



Axelle Apvrille

Principal Security Researcher at **Fortinet**, @cryptax
Lead organizer of **PhOwn 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

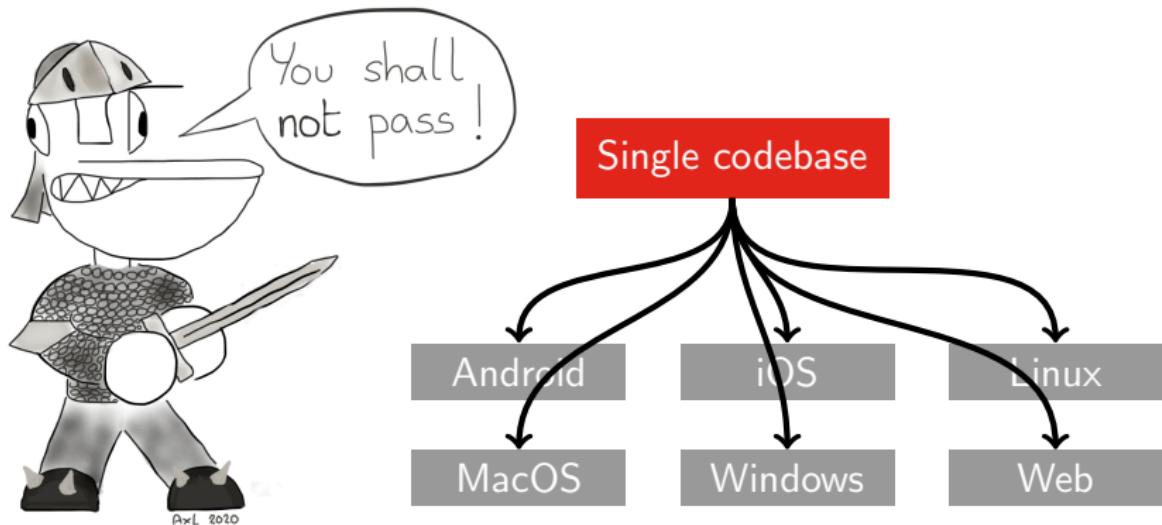
```
import 'dart:io';

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!

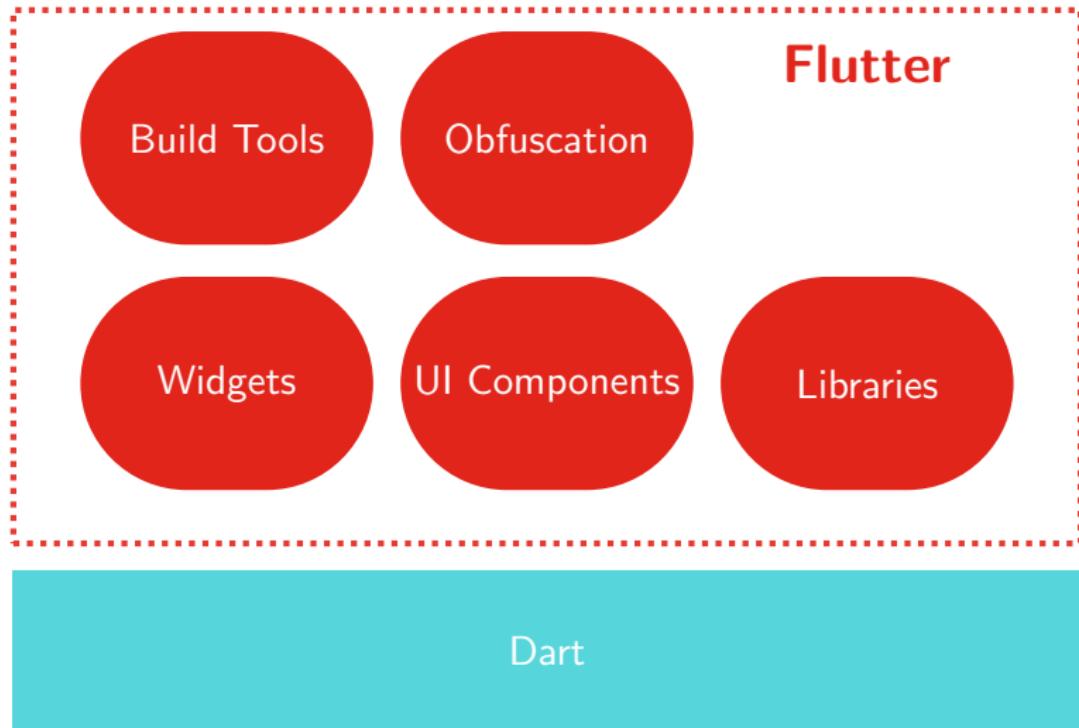
```
blackalps.aot: ELF 64-bit LSB shared  
object, x86-64, version 1 (SYSV), dynamically  
linked, BuildID[md5/uuid]=..., with debug_info,  
not stripped
```

```
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



① Introduction

What is Dart?

What is Flutter?

② W0rdle

Isolate snapshots

Object pool

Prologue snippet

Strings and Integers

Decryption Loop

Aarch32 and Aarch64

③ Reverse Malware

Platform Channel

setConfig

AES encryption

④ Conclusion



Ph0wn CTF 2022 challenge



- A clone of the wordle game, in Flutter
- Game code from <https://github.com/johnnysbug/flutterdile>
- 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**

Axelle Ap. @cryptax @mastodon.social @cryptax · Jan 27
W0rdle **Stage 3** is out!
Feel like some Android reverse engineering challenge?
- Download app: github.com/phOwn/writeups...
- SHA256:
99068666d845bdf2513bca731ad95e4b21f6d0460925beaf881a649ce16
a9c2e
#CTF #Android #phOwn #flutter

C R O C O

Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
ENTER BACK

1 4 22 2,517



The code is very simple!

```
1 String hiddenName() {
2     if (stage == 3) {
3         hidden;
4         hidden;
5         List <int> encrypted_flag = [ ... ];
6         int i = 0;
7
8         for (i = 0; i < encrypted_flag.length; i++) {
9             hidden; // are you up to it?
10        }
11        print( THEFLAG );
12        return THEFLAG;
13    }
14 }
```



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

File Edit View Layout Extras Help
libapp.so
Hex editor
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00
00000010: B3 00 3E 00 B1 00 50 00 00 00 00 00 00 00 00 00
00000020: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 40 00 38 00 00 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0: C8 D7 16 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0: 01 00 00 00 B5 00 00 00 00 00 00 00 00 00 00 00
000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0: 00 98 28 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0: 00 00 00 00 C8 3F 00 00 00 00 00 00 00 00 00 00
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110: 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000130: C8 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140: 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160: 00 00 00 00 C8 3F 00 00 00 00 00 00 00 00 00 00
00000170: 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190: 51 E5 74 64 86 00 00 00 00 00 00 00 00 00 00 00
000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0: B1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0: 03 00 00 00 47 4E 55 00 00 00 00 00 00 00 00 00
000001E0: FD 86 D6 83 BC 1C A4 D0 00 00 00 00 00 00 00 00
000001F0: [REDACTED] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200: 00 00 00 00 62 3B 65 3B 39 39 65 63 35 61 39 30
00000210: 65 34 36 36 31 35 30 31 66 62 36 39 65 39 64 e4661501fb69e9d
00000220: 64 37 30 66 70 72 6F 64 75 63 74 20 6E 60 63 d78fprud no-C
00000230: 64 65 57 63 6F 60 60 65 6E 74 73 28 6F 62 odr_comments no-
00000240: 64 77 63 72 66 5F 73 74 63 68 5F 74 72 61 63 dwarf_stack_trac
Page: 0x01 / 0x01 Region: 0x00000000 - 0x003FC397 (0 - 41788B
Selection: 0x000001F0 - 0x000001F0 (0x1 | 1 Data Size: 0x003FC398 (0x3FC398 | 3.99 MB
Data visualizer: Hexadecimal (8 bits) ▾
Pattern Data
Name Color Start End Size Type Value
elf 0x00000000 0x0000003F 0x0040 struct ELF { ... }
e_ident 0x00000000 0x0000000F 0x0010 struct E_IDENT { ... }
ehdr 0x00000010 0x0000003F 0x0030 struct Elf64_Ehdr { ... }
phdr 0x00000040 0x000001C7 0x0018 Elf64_Phdr [7] { ... }
shdr 0x003FC0D8 0x003FC397 0x02C0 Elf64_Shdr [11] { ... }
I: Pattern exited with code: 0
I: Evaluation took 0.0103745s

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

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

Dart AOT Format

The screenshot shows a memory dump of the Dart AOT format in ImHex. The dump includes fields like `magic`, `size`, `knd`, `version_hash`, and `features`. The `magic` field contains the value `b0e899ec5a90`. The `size` field contains the value `a4661501f0b69e9d`. The `knd` field contains the value `d7ffproduct no-code_comments no-dwarf_stack_traces_mode no-lazy_dispatchers deduplicated_instructions n-asserts x64-syntx compressed-payloads null-safe`. The `version_hash` field contains the value `5 [`. The `features` field contains the value `1`.

Field	Value
<code>magic</code>	<code>b0e899ec5a90</code>
<code>size</code>	<code>a4661501f0b69e9d</code>
<code>knd</code>	<code>d7ffproduct no-code_comments no-dwarf_stack_traces_mode no-lazy_dispatchers deduplicated_instructions n-asserts x64-syntx compressed-payloads null-safe</code>
<code>version_hash</code>	<code>5 [</code>
<code>features</code>	<code>1</code>

The screenshot shows the pattern data for the Dart AOT format. It includes fields like `magic`, `size`, `knd`, `version_hash`, and `features`. The `magic` field is set to `b0e899ec5a90`. The `size` field is set to `a4661501f0b69e9d`. The `knd` field is set to `d7ffproduct no-code_comments no-dwarf_stack_traces_mode no-lazy_dispatchers deduplicated_instructions n-asserts x64-syntx compressed-payloads null-safe`. The `version_hash` field is set to `5 [`. The `features` field is set to `1`.

Field	Value
<code>magic</code>	<code>b0e899ec5a90</code>
<code>size</code>	<code>a4661501f0b69e9d</code>
<code>knd</code>	<code>d7ffproduct no-code_comments no-dwarf_stack_traces_mode no-lazy_dispatchers deduplicated_instructions n-asserts x64-syntx compressed-payloads null-safe</code>
<code>version_hash</code>	<code>5 [</code>
<code>features</code>	<code>1</code>

- 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**

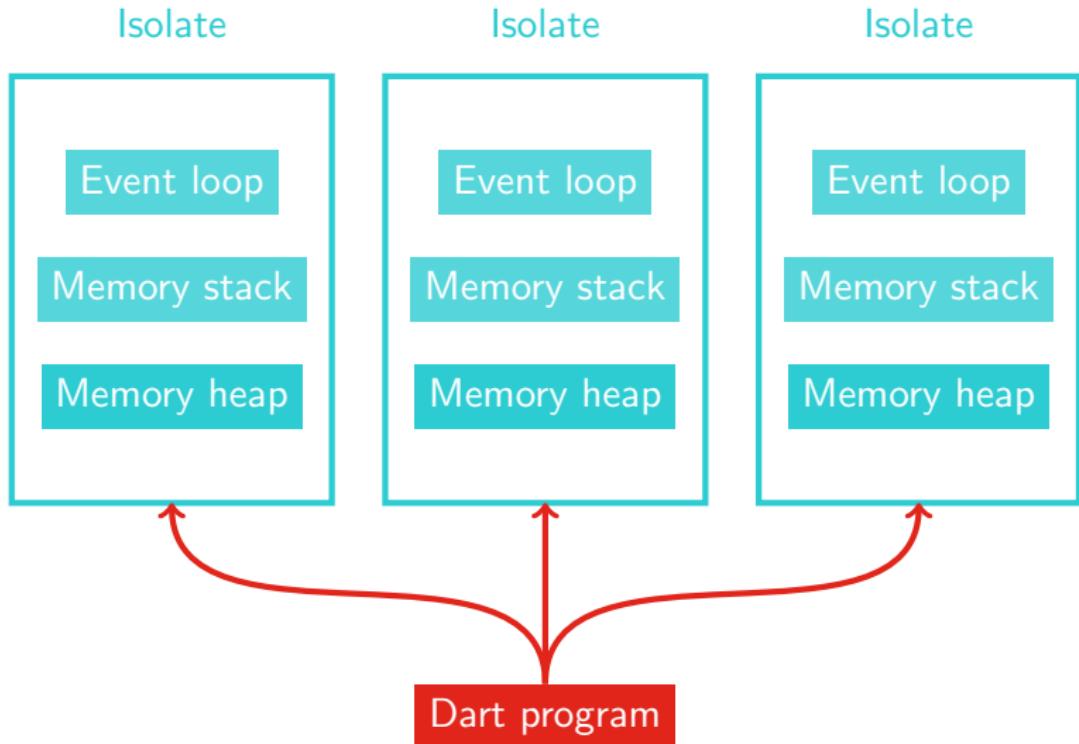
```
$ objdump -T libapp.so

libapp.so:      file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
0170000 g    D .text    00047a0  _kDartVmSnapshotInstructions
01747a0 g    D .text    02848e0  _kDartIsolateSnapshotInstructions
00001f0 g    D .rodata  00003890 _kDartVmSnapshotData
0003a80 g    D .rodata  00169bb0 _kDartIsolateSnapshotData
00001c8 g    D .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
...
0000000000000001f0 g    _kDartVmSnapshotData
00000000000003a80 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:

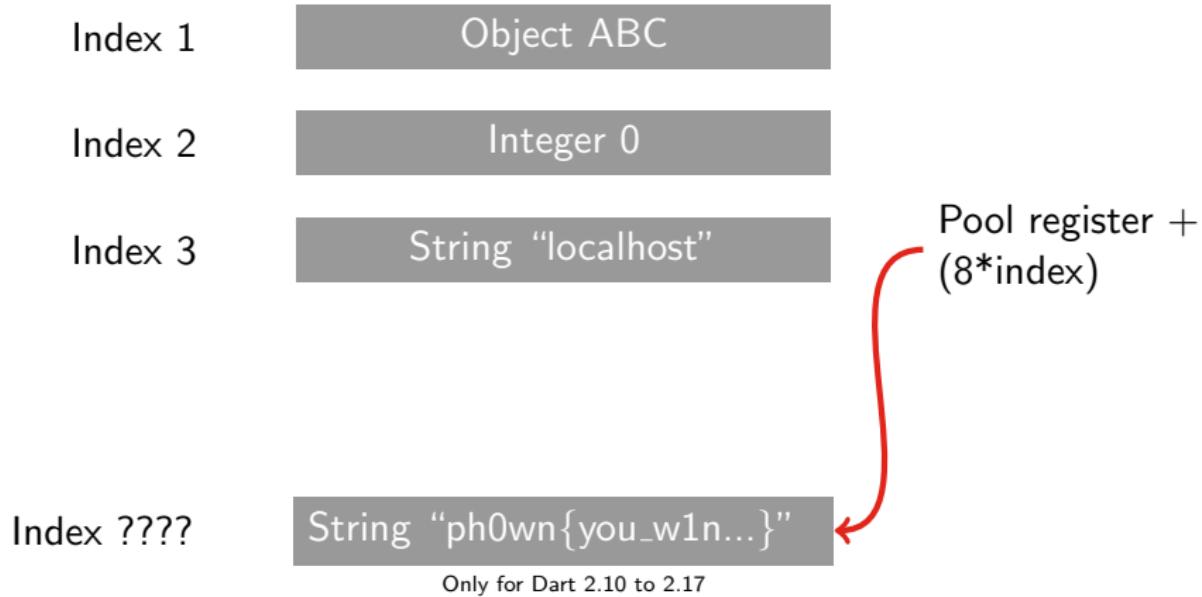
```
[0x00050000]> afl <--- list functions
0x00077568    53    1119 sym._Uri._makeHost
0x000779c8   114    2286 sym._Uri._normalizeRegName
0x000782b8   126    2344 sym.Uri.parseIPv6Address
0x00078be0    37     888 sym.Uri._parseIPv4Address
```

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



JEB is able to read strings in the Object Pool

The screenshot shows the JEB debugger interface with the "Pool Information" tab selected. The main pane displays a list of memory addresses, mostly containing question marks, except for address 10 which contains a file path. A red box highlights the "Pool Information" tab at the bottom.

Bytecode/Disassembly libapp.so/Overview dart_aot_snapshots/Pool Inform

POOL: (displaying strings only)

0: ?
1: ?
2: ?
3: ?
4: ?
5: ?
6: ?
7: ?
8: ?
9: ?
10: "file:///home/axelle/prog/challenges/ph0wn2022/challenges/w
11: ?
12: ?
13: ?
...

Description Pool Information Code Information



JEB is able to read strings in the Object Pool

Quick Search (Find) X

Search in... Entire Project Current Unit Current Document Case Sensitive Cap results to: 10

Search string (*? accepted): ph0wn{

Index	Text	Unit	Document	Location
0	5810: "ph0wn{"	w0rdle.apk > com.ph0w		5811,7
1	5811: "ph0wn{you_wln_darts_stage1}"	w0rdle.apk > com.ph0w		5812,7

5806: ?
5807: ?
5808: ?
5809: ?
5810: "ph0wn{"
5811: "ph0wn{you_wln_darts_stage1}"
5812: ?
5813: " in_type_cast"
2 matches (completed)

Close Help Legacy Dialog ...



Accessing pool object index 5811

Index 1	Object ABC
Index 2	Integer 0
Index 3	String "localhost"
Index 5811	String "ph0wn{you_w1n...}"

B598-B59F // 8
= 5811

R15 + B59*h



Dart assembly defines its own registers!

	x86_64	Aarch32	Aarch64
Object Pool	R15	R5	X27
Custom Stack Pointer	(<i>rsp</i>)	(<i>sp</i>)	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 in... Entire Project Current Unit Current Document Case Sensitive Cap results to: 100

Search string (*? accepted): `mov*[r15+b59*h]`

Index	Text	Unit	Document	Location
1	LOAD. wordt			sub_3DE09C+421H
2	LOAD. wOrdle			sub_3DE09C+467H
3	LOAD. wOrdle			sub_3DE09C+48EH

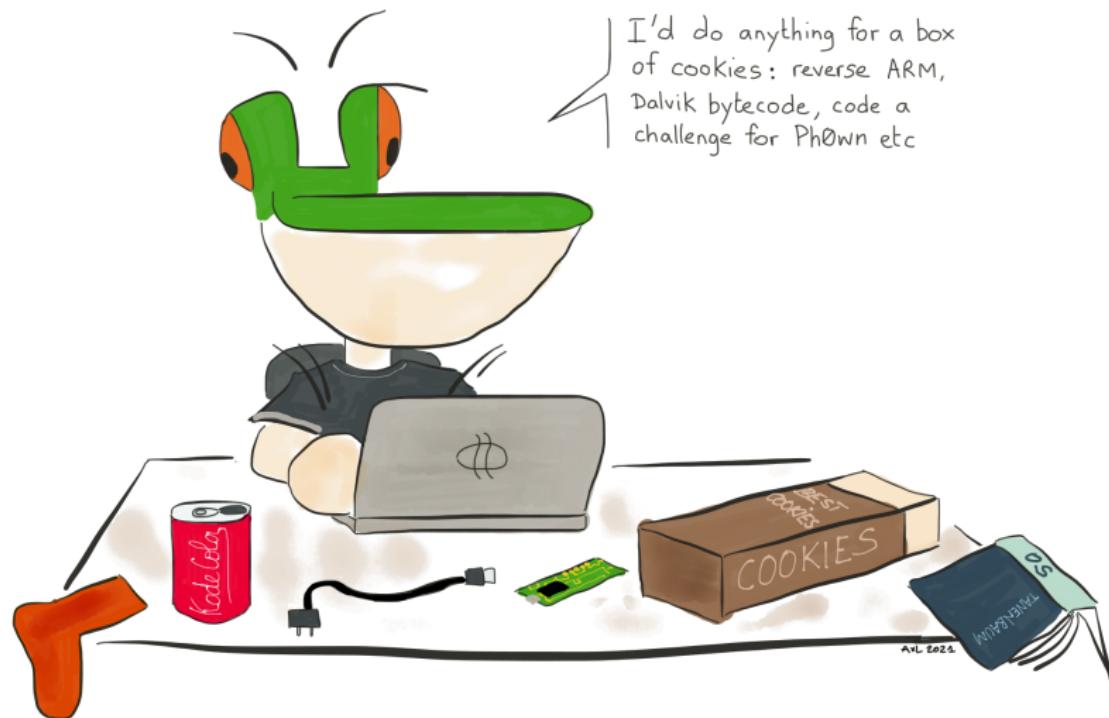
```
text:00000000'003DE529          ret
text:00000000'003DE52A loc_3DE52A:
text:00000000'003DE52A          mov     r11, qword ptr ds:[r15+B59Fh] ; xref: sub_3D
text:00000000'003DE531          push    r11
text:00000000'003DE533          call    sub_1C70E8
```

10 matches (completed)

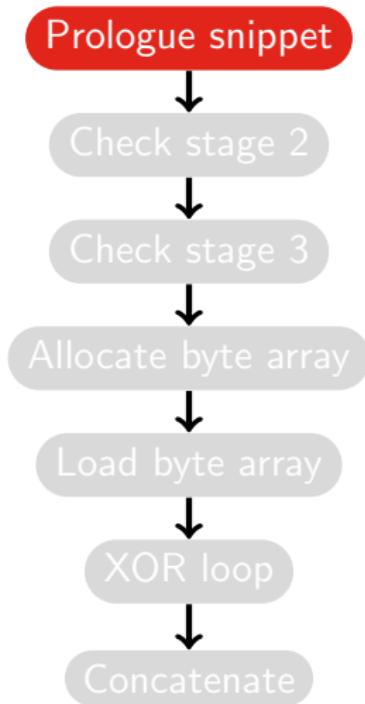
Search for the instruction



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



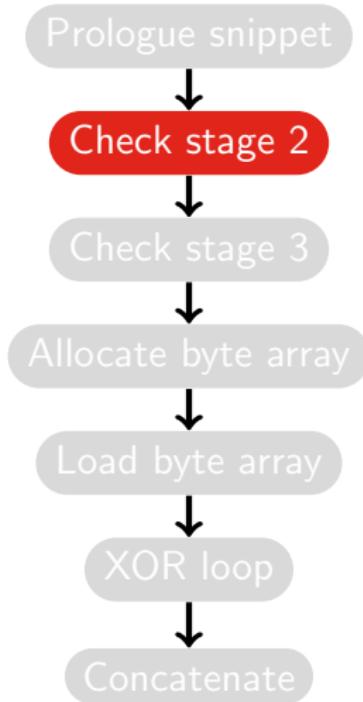
Same mechanisms for AArch64 (and AArch32)

```
; 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



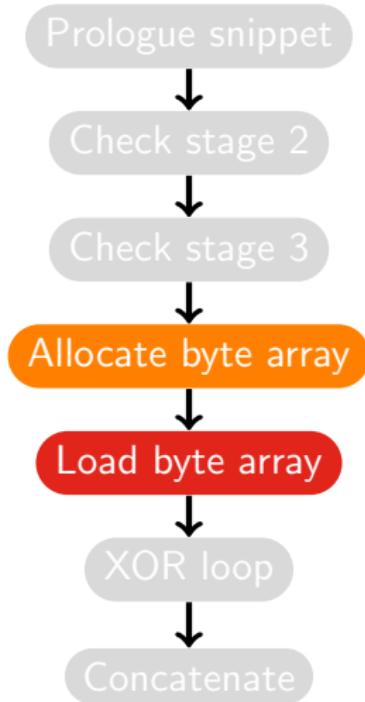
Compare stage value

```
; 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 ?

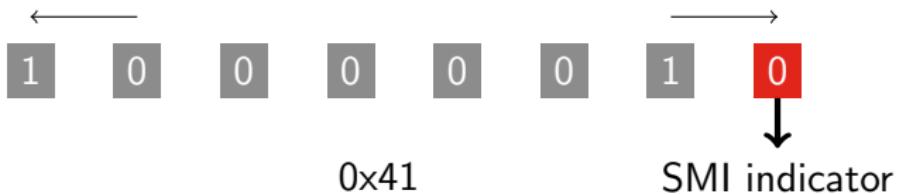
Is mov REGISTER, 41h correct?

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.

Most significant bit

Least significant bit



The correct instruction is mov REGISTER, 82h



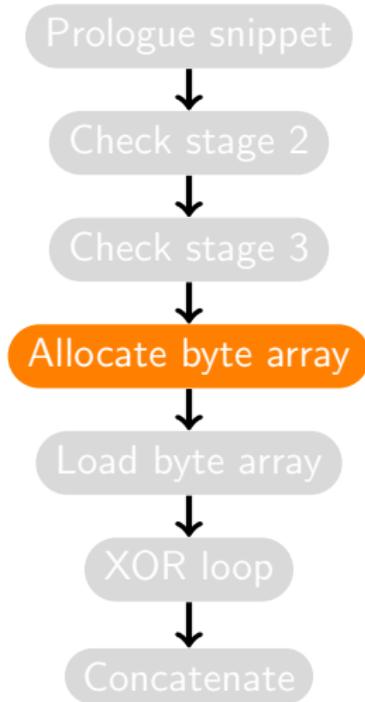
Array of SMIs

```
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



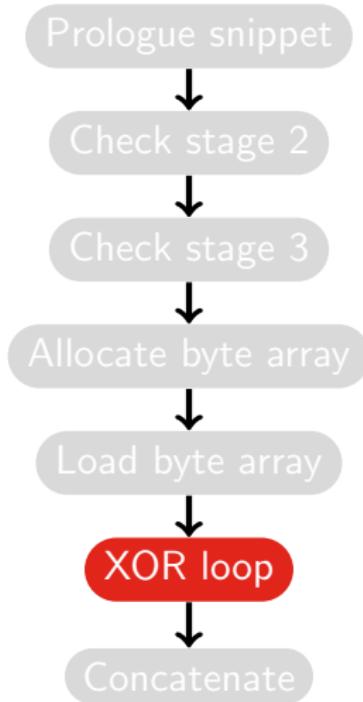
Array allocation

```
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



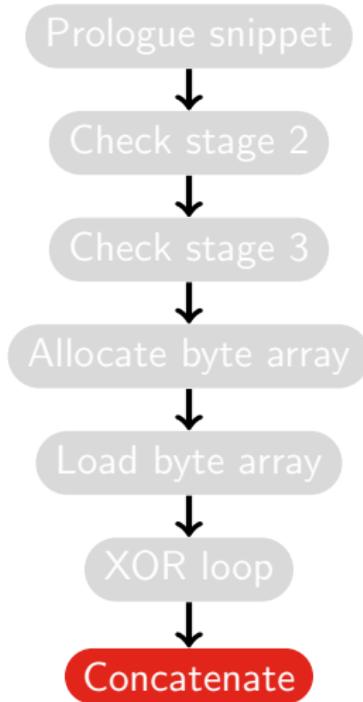
Decrypting the array

```
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?

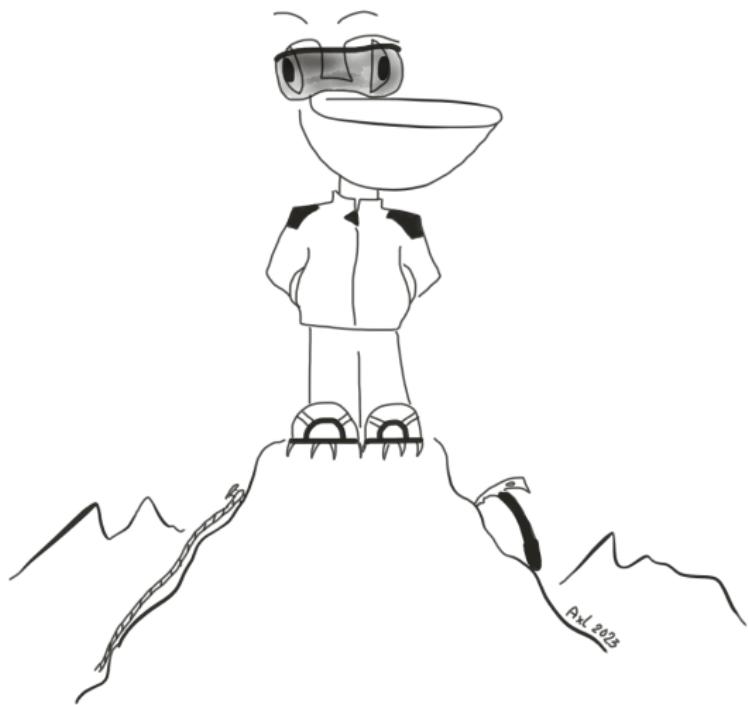


Building the flag

```
; 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
```

Divide with a shift right

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

XOR: Two instructions on Aarch64:

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

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



① Introduction

What is Dart?

What is Flutter?

② W0rdle

Isolate snapshots

Object pool

Prologue snippet

Strings and Integers

Decryption Loop

Aarch32 and Aarch64

③ Reverse Malware

Platform Channel

setConfig

AES encryption

④ Conclusion



MoneyMonger



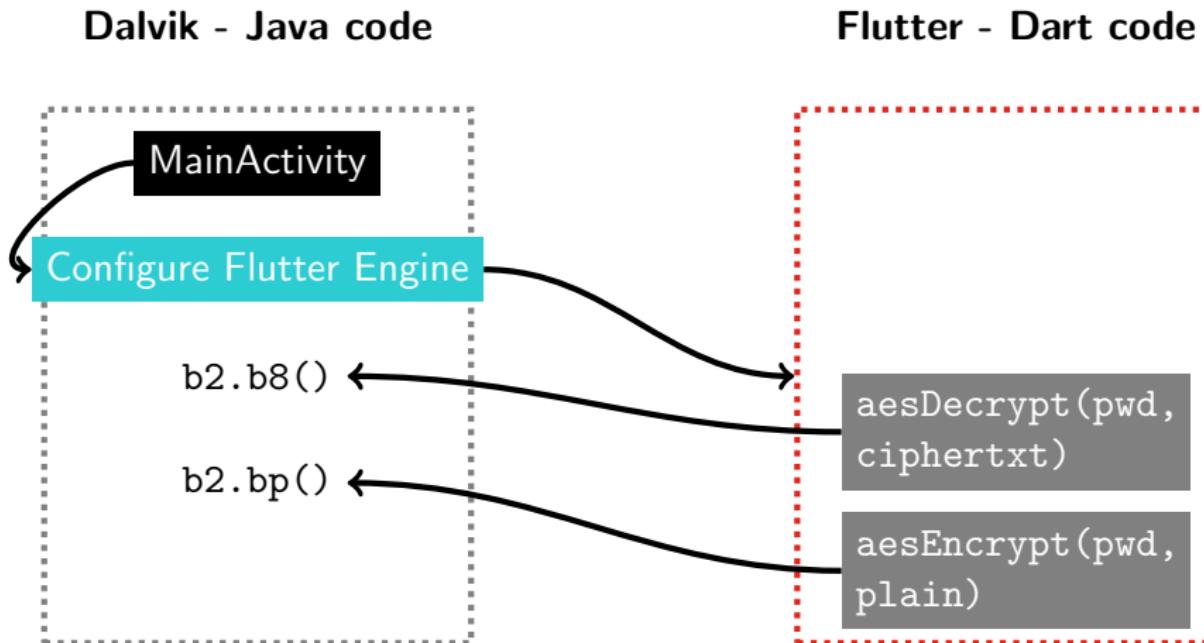
- **Loan scam** in India starting May 2022
- **Flutter** version noticed in December 2022
- **Threatens to leak pictures to contacts, harasses victims**
- Leaks GPS, call log, SMS list, installed apps, contact list...



Fast Rupee



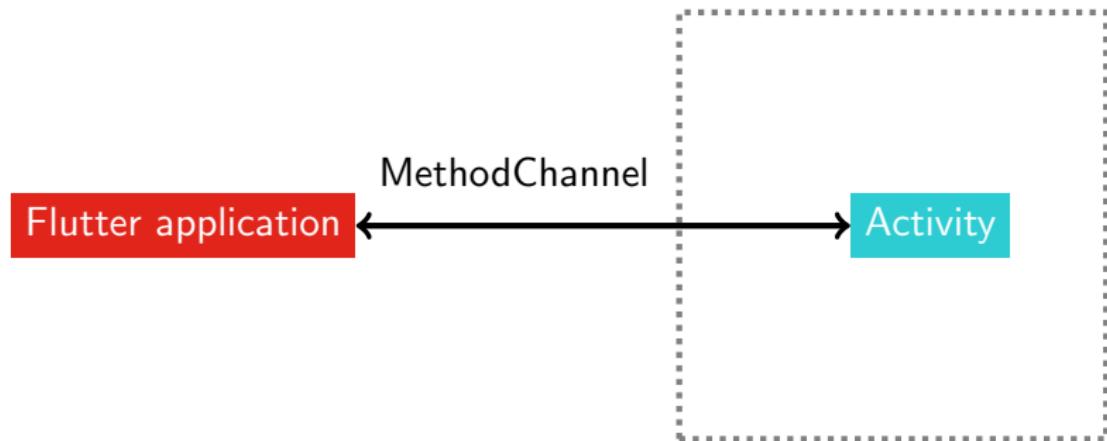
MoneyMonger Java/Flutter architecture



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

Android Dalvik Layer



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;  
}
```

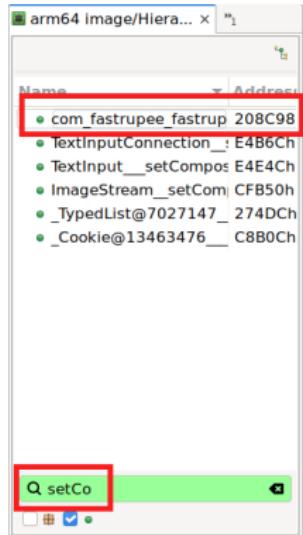
- Dart function `setConfig` has 1 argument: `smsCount`
- Adds a `smsCount` entry with the specified value in the shared preferences



Locate setConfig in Dart

With JEB,

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



```
com_fastrupee_fastrupeepeefafMyUtilsPlugin__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
```



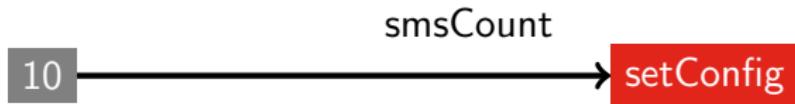
It's a wrapper



```
com_fastrupee_fastrupeepeefafMyUtilsPlugin__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
```

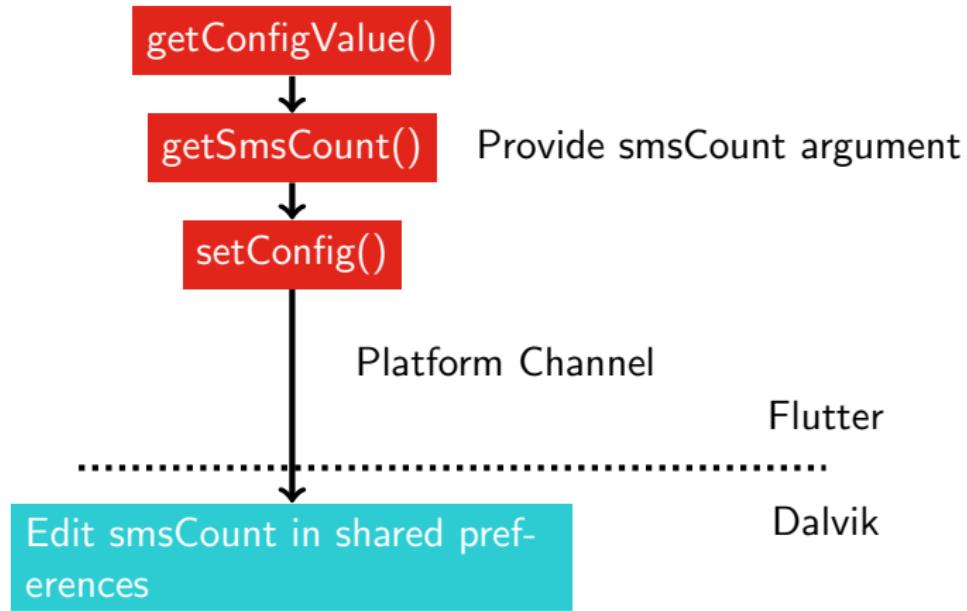


Retrieve SMS count and pass as argument



```
BL
↪ com_fastrupee_fastrupeepeefafPermissionConfig__get:getSmsCount
STR      X0, [X15, #FFFFFFF8h]!
BL
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()

```
\n    }\n}\n\n        int a = (int)(Integer.MAX_VALUE * Math.random());\n        int b = (int)(Integer.MAX_VALUE *\nUE * Math.random());\n        ret = a + b;\n    }\\n{\n        ret = \"E7AAE4F09F4645D6BDBA091B231393511\" +\n        \"E7AAE4F09F4645D6BDBA091B231393512\";\n    }\\n{\n        ret = (int)(Integer.MAX_VALUE * Math.random());\n        }\\n{\n        ret = Build.ID;\n        }\\n{\n        ret = Environment.getDataDirectory();\n        for (int i = 0; i < (int)(Math.random() * 100); i++) {\n            }\\n{\n            int x = Integer.parseInt(\"822633283\");\n            if (Math.random() > x) {\n                } else {\n                }\\n{\n                int a = Integer.parseInt(\"1824842469\");\n                ret = a / 1;\n            }\\n{\n            ret = (int)(Integer.MAX_VALUE * Math.random());\n            }\\n{\n            int x = Integer.parseInt(\"681746379\");\n            if (Math.random() > x) {\n                } else {\n                }\\n{\n                ret = Build.BRAND;\n            }\\n{\n            ret = Environment.getDataDirectory();\n            }\\n{\n            ret = (WindowManager) this.getSystemService(Context.WINDOW_SERVICE);\n            }\\n{\n            ret = this.getApplicationContext().getResources();\n            }\\n{\n            ret = Environment.getDataDirectory();\n            }\\n{\n            ret = (WindowManager) this.getApplicationContext().getResources();\n            }\\n{\n            ret = (int)(Integer.MAX_VALUE * Math.random());\n            }\\n{\n            double a = Math.random() * Double.MAX_VALUE;\n            double b = Math.random() * Double.MAX_VALUE;\n            ret = a * b;\n            }\\n{\n            for (int i = 0; i < (int)(Math.random() * 100); i++) {\n                }\\n{\n                int x = Integer.parseInt(\"1022137011\");\n                if (Math.random() > x) {\n                    } else {\n                    }\\n{\n                    int a = (int)(Integer.MAX_VALUE *\n                    * Math.random());\n                    int b = (int)(Integer.MAX_VALUE * Math.random());\n                    ret = a + b;\n                }\\n{\n                ret = (WindowManager) this.getApplicationContext().getSystemService(Context.WINDOW_SERVICE);\n                }\\n{\n                ret = (WindowManager) this.getApplicationContext().getSystemService(Context.WINDOW_SERVICE);\n            }\\n{\n            @Override\n            protected void onCreate(Bundle savedInstanceState) {\n                super.onCreate(savedInstanceState);\n                try {\n                    com_fastrupee_fastrupeeefaActivity40();\n                } catch (Exception e) {}\\n            }\n        }\n        dp: password=D4JcGjcw489iiEq1\n        dp: plaintext={}\n        dp: password=D4JcGjcw489iiEq1\n        dp: plaintext={"AID":"419325ce0c5f52d2","UUID":"cd4842a0-b89b-11ed-8d26-87e38aff63fe","GAID":"f1db4ccc-af6a-42db-a428-a7f6155068cb","VERSION":"1.0.18","SE":"true"}\n    }
```

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



① Introduction

What is Dart?

What is Flutter?

② W0rdle

Isolate snapshots

Object pool

Prologue snippet

Strings and Integers

Decryption Loop

Aarch32 and Aarch64

③ Reverse Malware

Platform Channel

setConfig

AES encryption

④ Conclusion



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 malicious 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)

