

Prototyping and reverse engineering with frida

+44 (0)161-820-3056

www.digitalinterruption.com

jahmel@digitalinterruption.com

THIS WORKSHOP

- INTRODUCTION TO RAPID REVERSE ENGINEERING WITH FRIDA
- PRACTICAL EXERCISES (LIMIT THE THEORY)
- VIEW THE CODE! NO NEED TO READ ASSEMBLY
- LINUX/ANDROID
- !EXPLOITATION
- 2 HOURS (REALISTICALLY LESS)

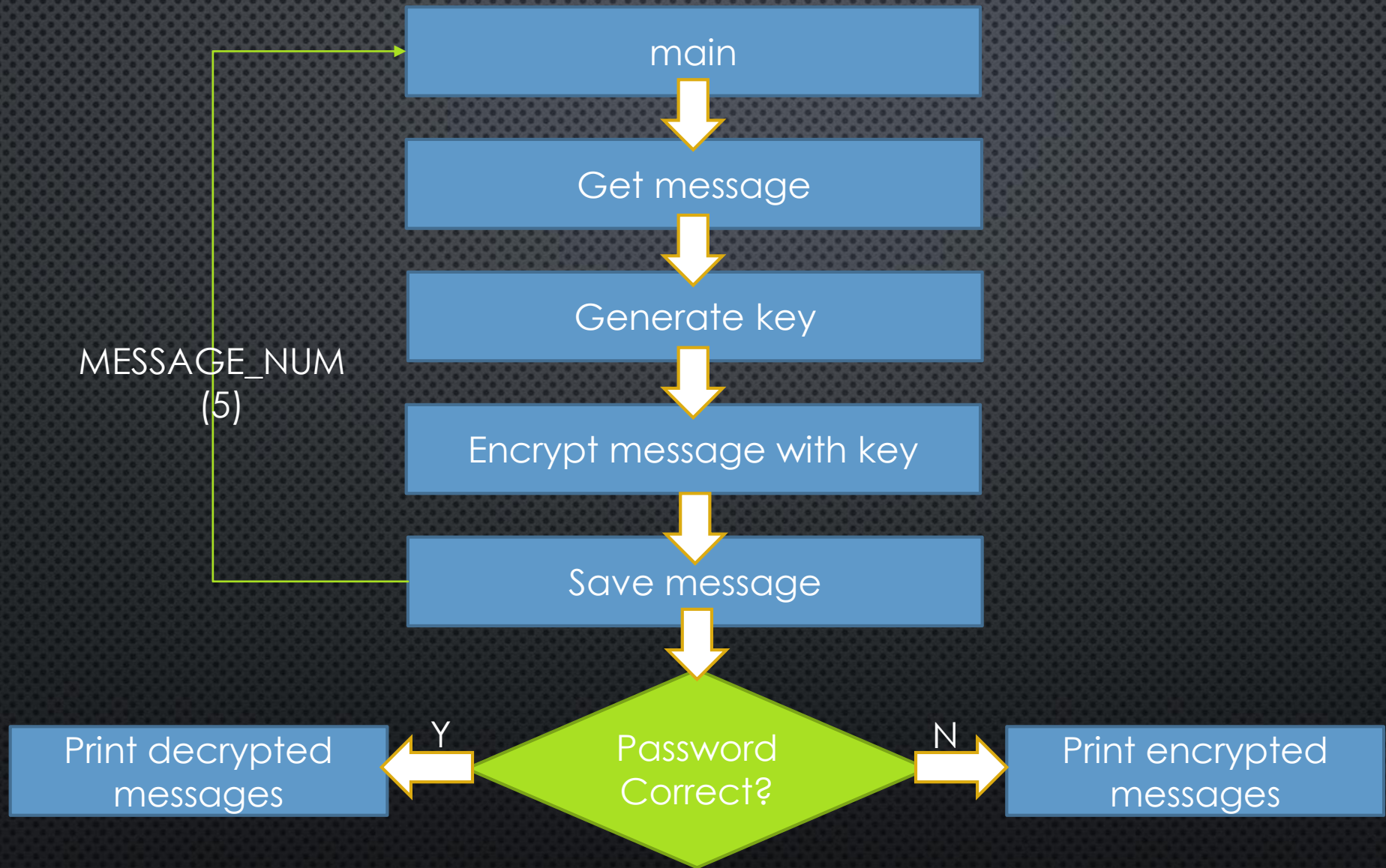
PREREQUISITES

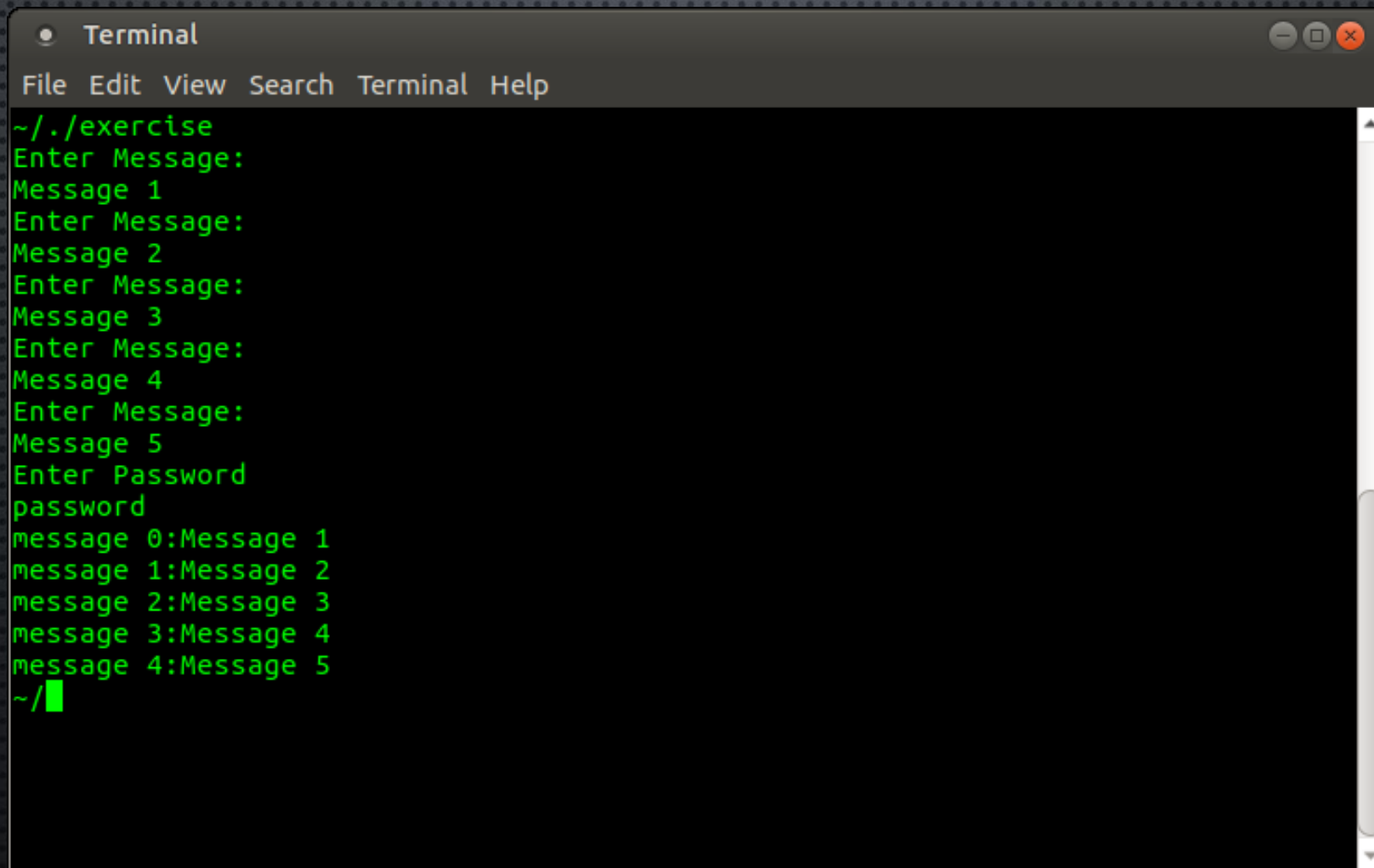
- LAPTOP RUNNING LINUX
- C COMPILER (OR TRUST MY BINARY)
- FRIDA
- READ SIMPLE C CODE
- CODE SIMPLE JAVASCRIPT

WHOAMI

- JAHMEL HARRIS
- PEN TESTER/SECURITY RESEARCHER AT DIGITAL INTERRUPTION MOBILE |
RADIO | REVERSE ENGINEERING
- @JAYHARRIS_SEC
- @MCRGREYHATS
- @DI_SECURITY

TARGET APPLICATION





```
Terminal
File Edit View Search Terminal Help
~/./exercise
Enter Message:
Message 1
Enter Message:
Message 2
Enter Message:
Message 3
Enter Message:
Message 4
Enter Message:
Message 5
Enter Password
password
message 0:Message 1
message 1:Message 2
message 2:Message 3
message 3:Message 4
message 4:Message 5
~/
```

```
Terminal
File Edit View Search Terminal Help

~/./exercise
Enter Message:
message 1
Enter Message:
message 2
Enter Message:
message 3
Enter Message:
message 4
Enter Message:
message 5
Enter Password
IDontKnowThePassword
message 0:"n_qe13g
message 1:6dg>xx
message 2:HvM1
message 3:*****13
message 4:Hx1Poooo
~/
```


CAN WE BYPASS THE NEED FOR A PASSWORD?

EXERCISE 0

- CONFIGURE ENVIRONMENT FOR WORKSHOP
 - pip install frida
 - DOWNLOAD EXERCISE.C (<http://bit.ly/2kOlzrb>)
 - \$ make exercise
 - sudo sysctl kernel.yama.ptrace.scope=0

WHAT IS REVERSE ENGINEERING?

- REPRODUCING SOMETHING BASED ON EXTRACTED KNOWLEDGE
- UNDERSTANDING THE BEHAVIOUR OF A BINARY
- LENGTHY PROCESS THAT REQUIRES SKILL

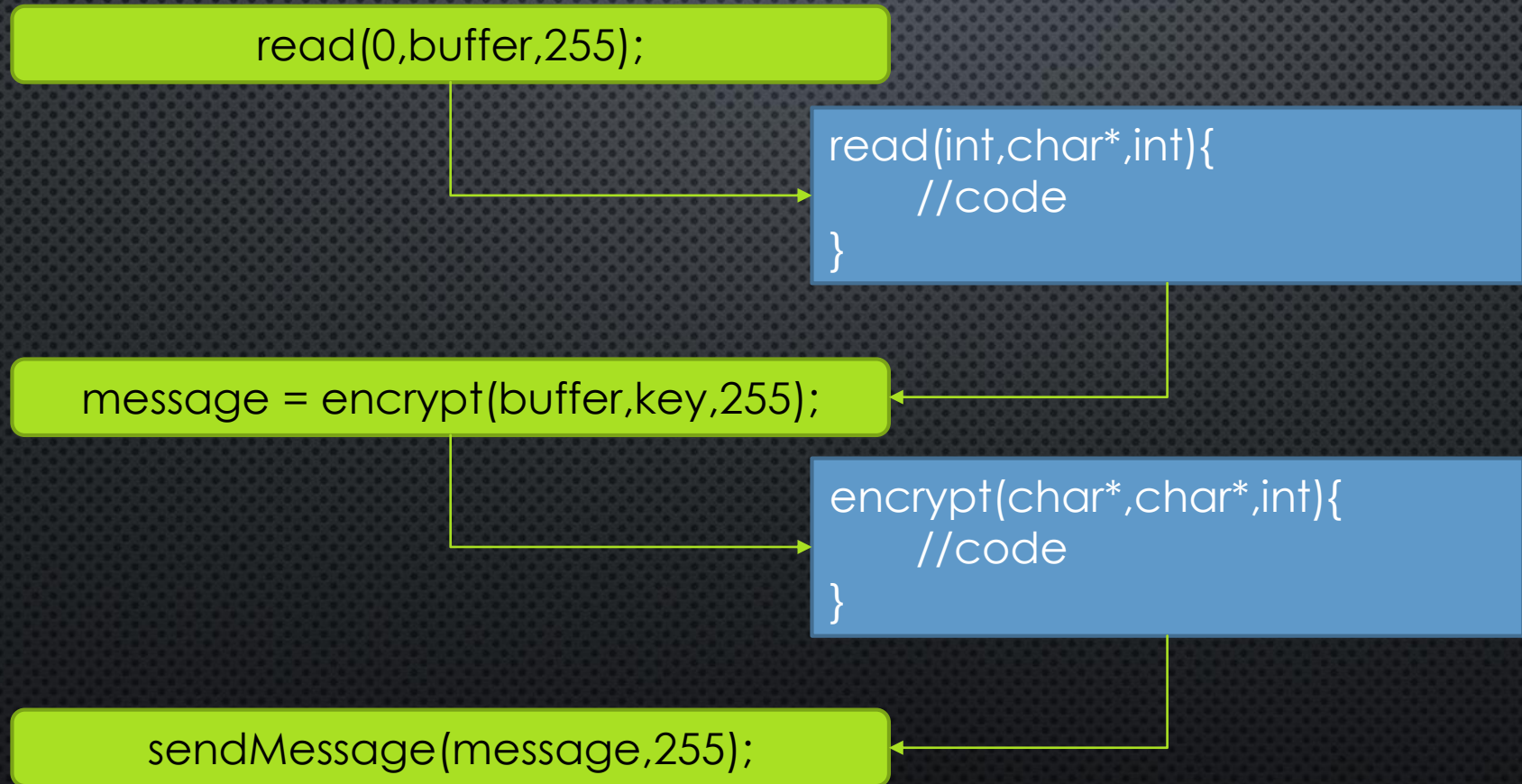
IF IT'S SO HARD, WHY DO IT?

- SOURCE CODE RECOVERY
- INTEROPERABILITY
- FUN!
- VULNERABILITY RESEARCH

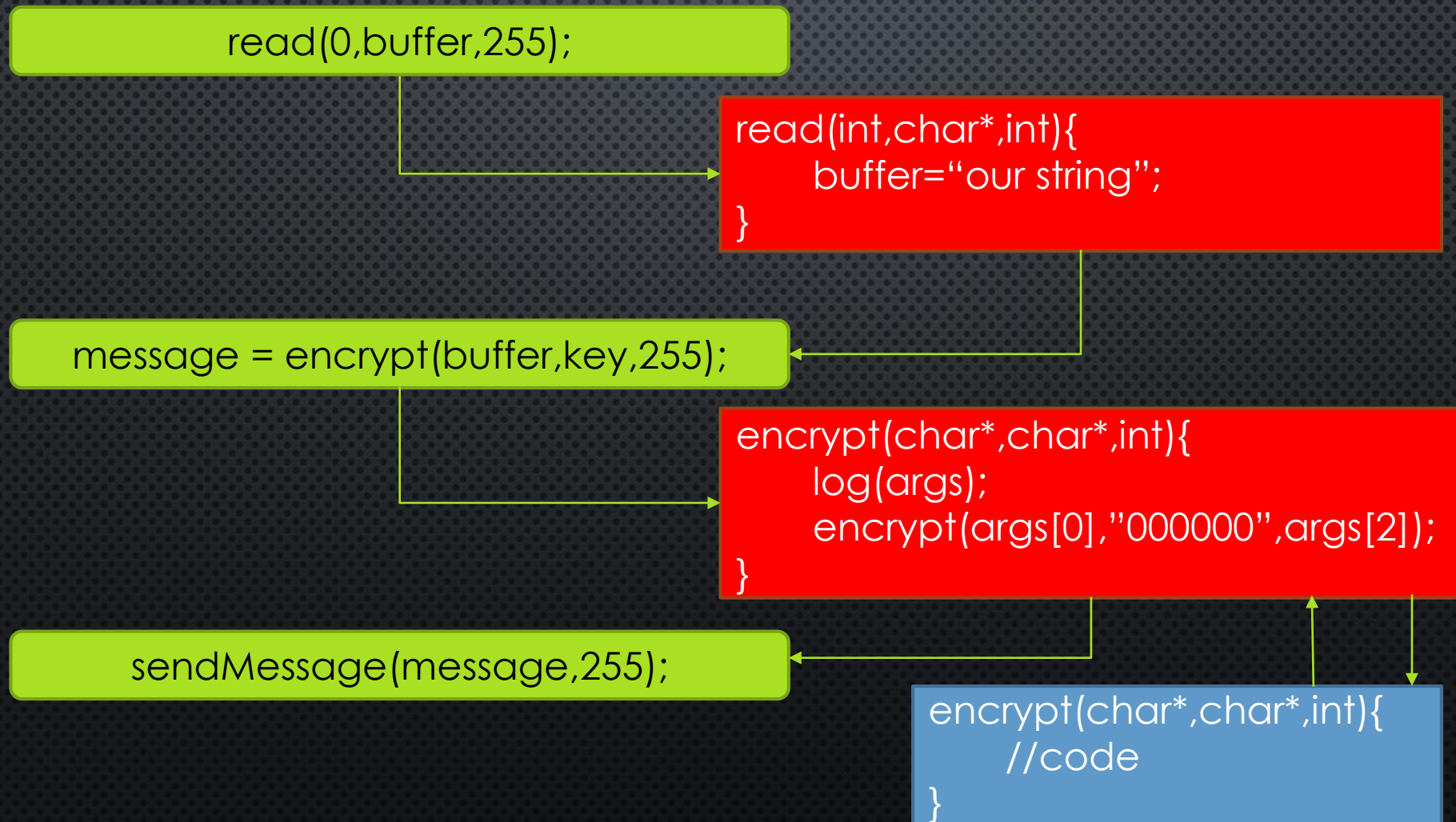
APPLICATION HOOKING

- INVALUABLE TOOL IN DYNAMIC ANALYSIS
- VIEW INTERNAL STATE
- ADD LOGGING
- CHANGE APPLICATION LOGIC

APPLICATION HOOKING



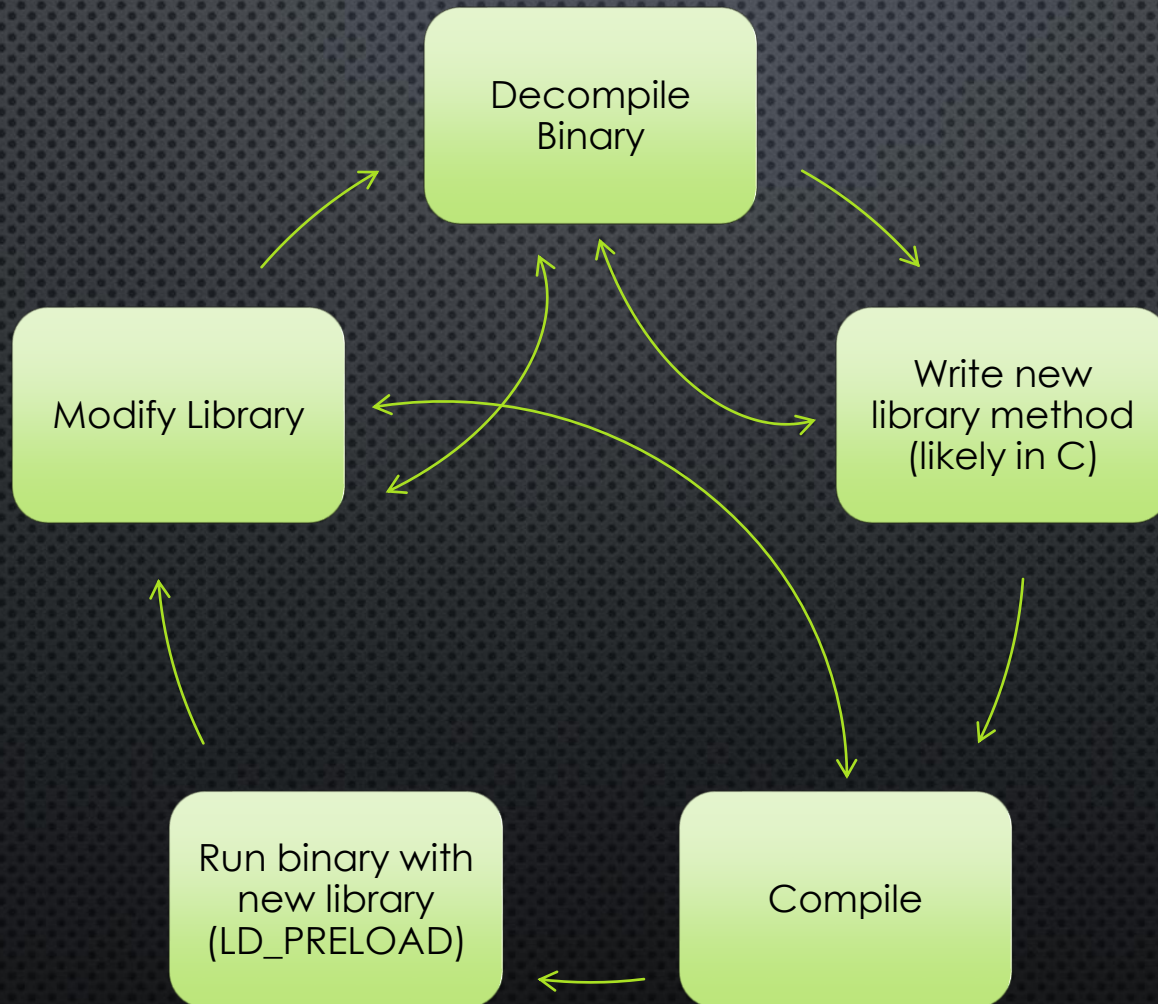
APPLICATION HOOKING



BEFORE FRIDA



BEFORE FRIDA



DEMO

AFTER FRIDA

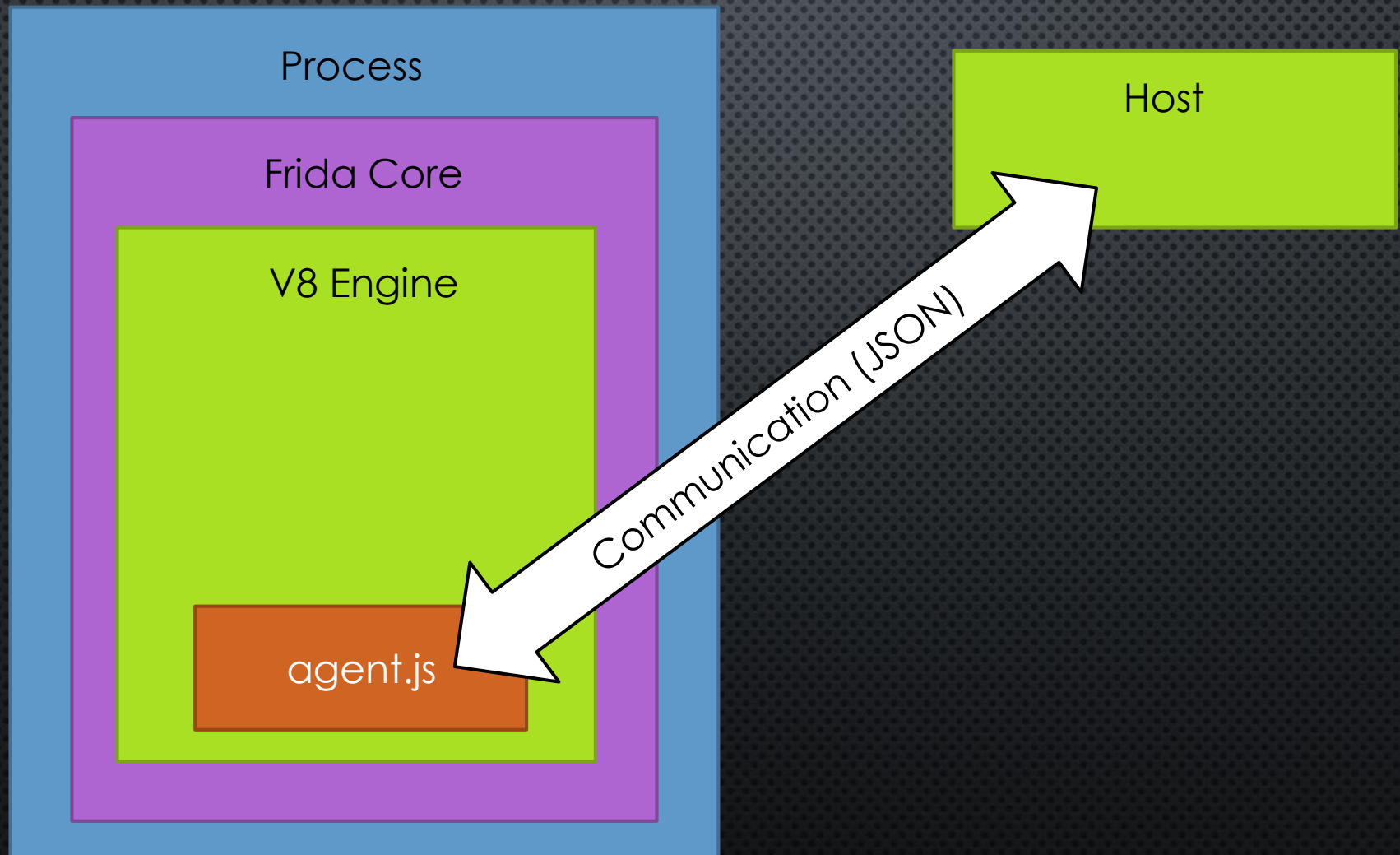


DEMO

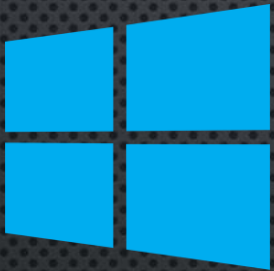
FRIDA

- FRAMEWORK FOR INSTRUMENTATION
- INJECTING JAVASCRIPT INTO APPLICATION (WHAAAA!)
- MOST IMPORTANTLY – A FRAMEWORK FOR BUILDING TOOLS

FRIDA



FRIDA



EXERCISE 1

- INJECT JAVASCRIPT INTO PROCESS
- EXPLORE FRIDA API
 - WHAT IS THE CURRENT THREADID?
 - WHAT MODULES ARE LOADED INTO THE PROCESS?
 - WHAT ARE THE ADDRESSES OF THE LIBC FUNCTIONS?

FRIDA

- FRIDA
- FRIDA-PS
- FRIDA-TRACE

FRIDA-TRACE

- WRITTEN USING FRIDA (AND INSTALLED WITH FRIDA)
- CREATES JAVASCRIPT FILE FOR HOOKED FUNCTIONS (BY NAME)
- CAN USE WILDCARDS (FRIDA-TRACE -I "*" PROCESS)

FRIDA-TRACE

```
char* encryptedMessage = encryptMessage(message,key,255);
```

```
onEnter: function(log,args,state){  
    log("encryptMessage");  
    log(Memory.readUtf8String(args[1]));  
}
```

```
onLeave: function(log,retval,state)  
{  
    retval.replace(0x00);  
}
```

FRIDA-TRACE

```
char* encryptedMessage = encryptMessage(message, key, 255);
```

```
onEnter: function(log, args, state){  
    log("encryptMessage");  
    log(Memory.readUtf8String(args[1]));  
}
```

```
onLeave: function(log, retval, state)  
{  
    retval.replace(0x00);  
}
```


FRIDA-TRACE

```
char* encryptedMessage = encryptMessage(message, key, 255);
```

```
onEnter: function(log, args, state){  
    log("encryptMessage");  
    log(Memory.readUtf8String(args[1]));  
}
```

```
onLeave: function(log, retval, state)  
{  
    retval.replace(0x00);  
}
```

FRIDA-TRACE

```
char* encryptedMessage = encryptMessage(message, key, 255);
```

```
onEnter: function(log, args, state){  
    log("encryptMessage");  
    log(Memory.readUtf8String(args[1]));  
}
```

```
onLeave: function(log, retval, state)  
{  
    retval.replace(0x00);  
}
```


FRIDA-TRACE

```
char buffer[255];  
encryptMessage(message, key, buffer, 255);
```

```
onEnter: function(log, args, state){  
    log(Memory.readUtf8String(args[2])); //garbage  
    this.buf = args[2];  
}
```

```
onLeave: function(log, retval, state)  
{  
    log(Memory.readUtf8String(this.buf));  
}
```

DEMO

EXERCISE 2

- HOOK THE PROCESS TO LOG “READ”
 - WHAT ABOUT THE ARGUMENTS?
 - POINTERS?
- MODIFY “RAND()” TO AFFECT THE ENCRYPTED DATA

FRIDA-TRACE

- REQUIRE MEMORY ADDRESS
 - EASY WITH IMPORTED FUNCTIONS + FRIDA-TRACE
- WHAT ABOUT INTERNAL FUNCTIONS?
 - HINT: OBJDUMP

FRIDA-TRACE

```
$ objdump -d exercise | grep -i "functionName"  
af3: e8 fd 01 00 00      callq cf5 <functionName>  
00000000000000cf5 <functionName>:
```

FRIDA-TRACE

```
$ objdump -d exercise | grep -i "functionName"  
af3:  e8 fd 01 00 00      callq  cf5 <functionName>  
00000000000000cf5 <functionName>:
```

WE NOW KNOW THE OFFSET

FRIDA-TRACE

```
$ objdump -d exercise | grep -i "functionName"  
af3:  e8 fd 01 00 00      callq  cf5 <functionName>  
00000000000000cf5 <functionName>:
```

```
$ frida-trace -a exercise!0x0f5
```

```
[Local::ProcName::printRandNumber]-> Process.enumerateModulesSync()
```

EXERCISE 3

- CHANGE “ENCRYPTSTRING()” TO PRINT THE KEY
- HOW CAN THE “CHECKPASSWORD()” FUNCTION BE BYPASSED?

SCRIPTING FRIDA

- BINDINGS MAKE FRIDA SCRIPTABLE!
- BINDINGS FOR NODE.JS, PYTHON, .NET, QML ETC

PYTHON TEMPLATE

```
import frida
import sys
def on_message(message, data):
    print message['payload']

jscode = """
send("hello world");
"""

session = frida.attach("process")
script = session.create_script(jscode)
script.on('message', on_message)
script.load()
sys.stdin.read()
```


DEMO

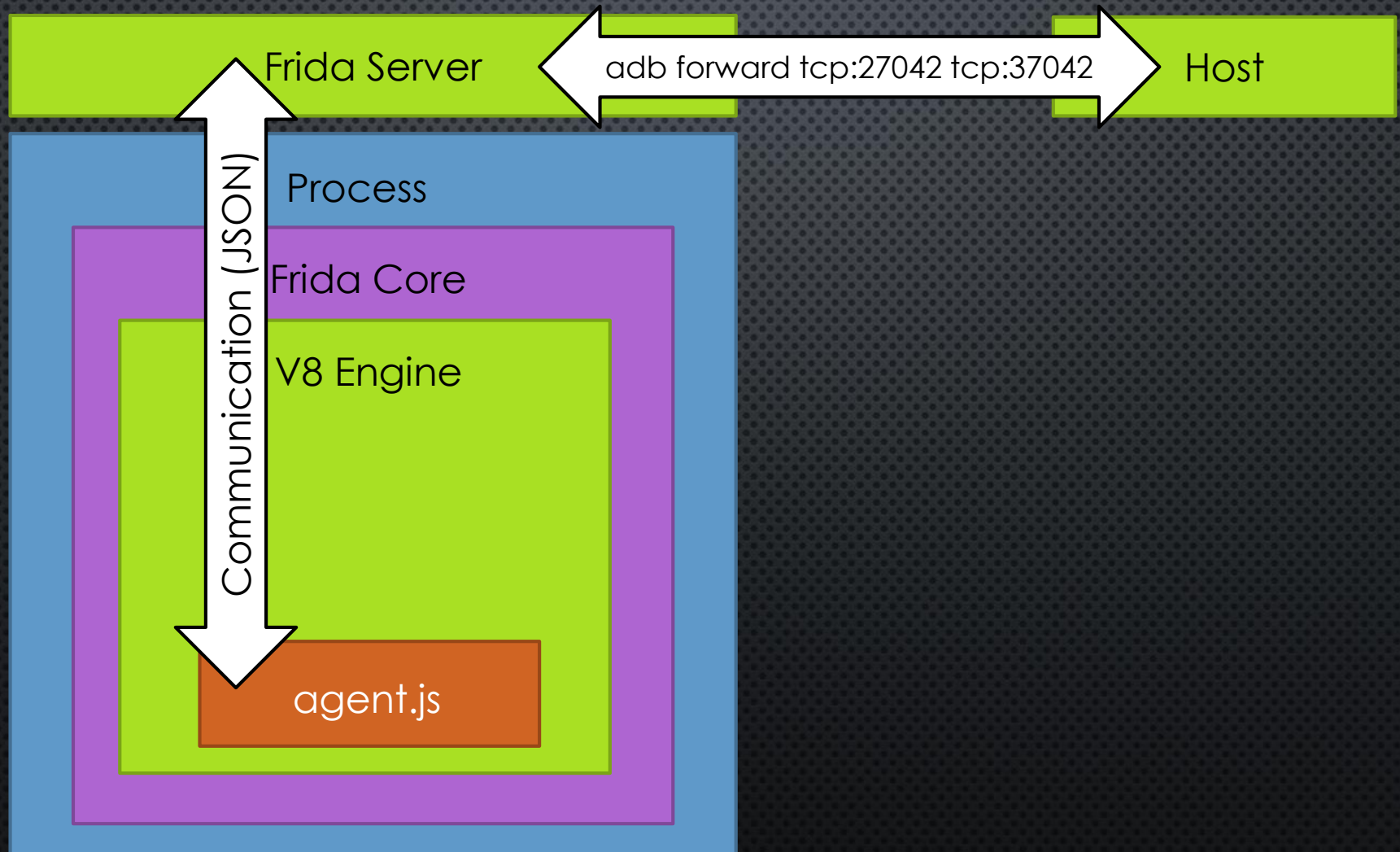
EXERCISE 4

- CREATE PYTHON SCRIPT TO RUN `DECRYPTMESSAGE()`
- CREATE PYTHON SCRIPT TO REPLACE `PRINTALLENCRYPTEDMESSAGES` WITH `DECRYPTALLMESSAGES()`
 - HINT: `NATIVEFUNCTION()`

FRIDA AND ANDROID

- ROOTED AND NON ROOTED

FRIDA AND ANDROID



DEMO – BYPASSING APP SECURITY

HOW DO WE PROTECT AGAINST THIS?

SHOULD WE PROTECT AGAINST THIS?