HackInBo®
Winter 2017 Edition
Bologna - Sabato 14 Ottobre

ADVANCED MOBILE PENETRATION TESTING WITH BRIDA

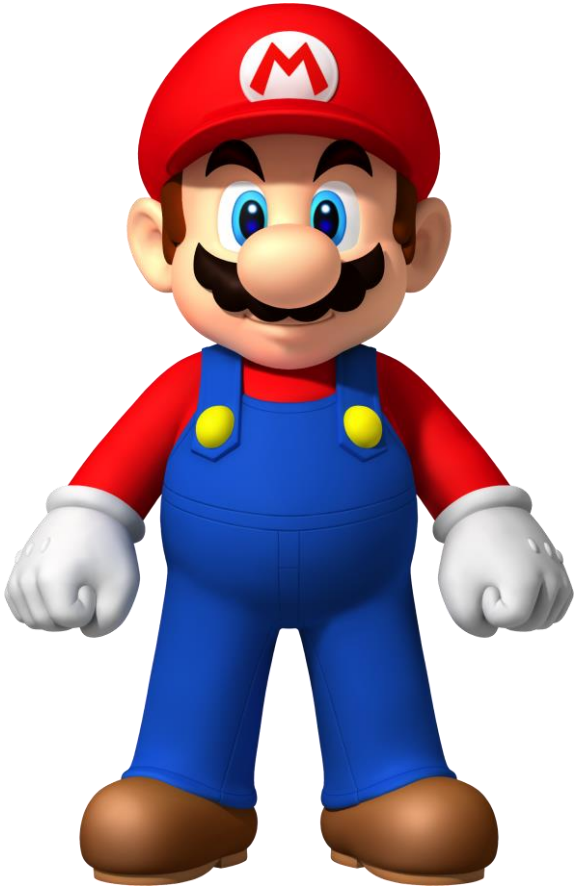
THE SPEAKER

mediaservice.net
CORPORATE SECURITY & IMAGE

Federico Dotta

Security Advisor
@ Mediaservice.net S.r.l.
(federico.dotta@mediaservice.net)

- OSCP, CREST PEN, CSSLP
- 7+ years in Penetration Testing
- Focused on application security
- Developer of sec tools: https://github.com/federicodotta
- Trainer



TOPICS

Web Application

- Fixed client (web browser)
- Logic usually is mainly on the backend components
- Client-side application code usually is coded with interpreted languages
- Provisioned directly from the application server

Mobile Application

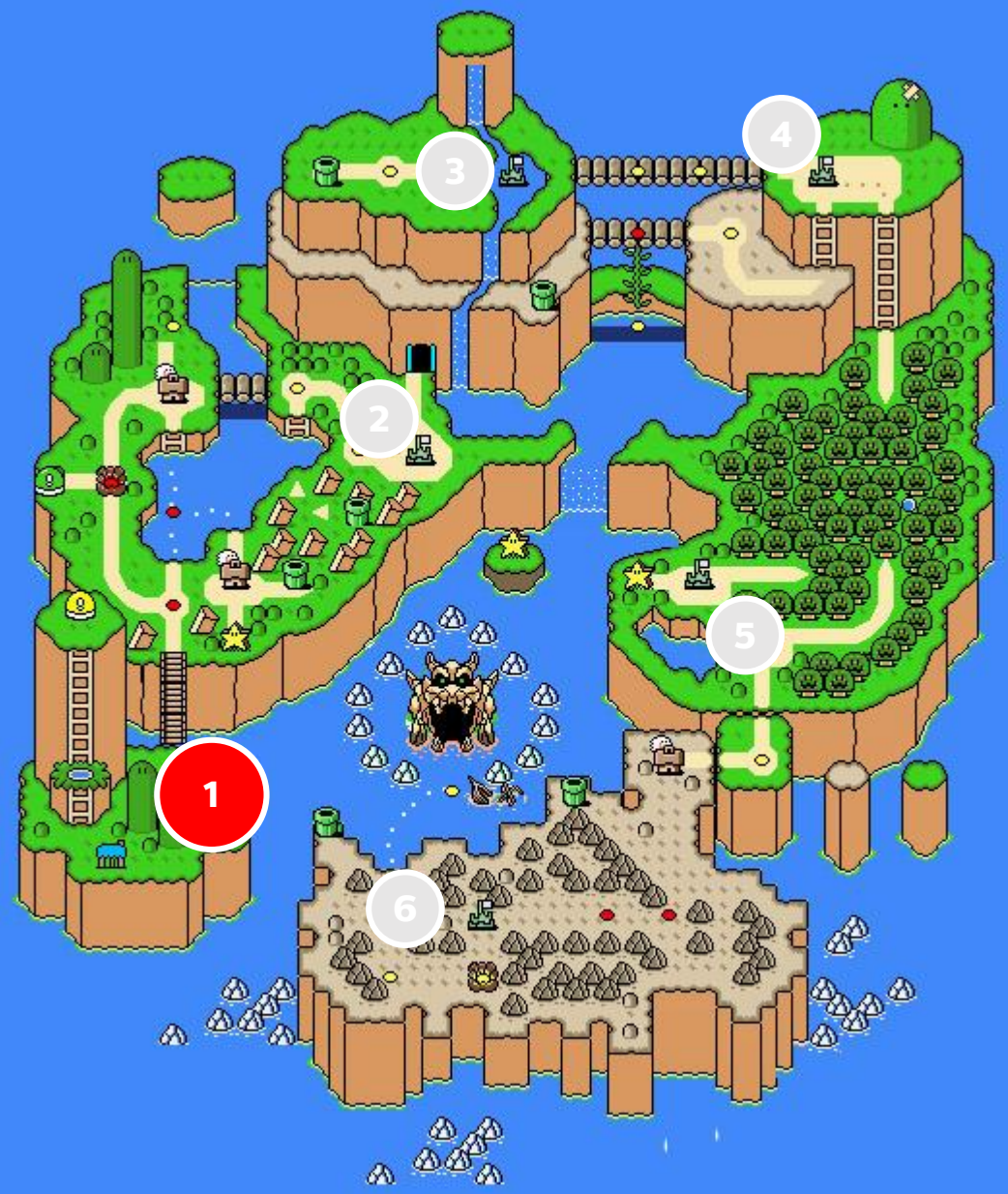
- Custom compiled client
- Logic usually divided between client and backend
- Client-side application code can be interpreted or compiled
- Provisioned from a trusted third party

Mobile Applications

It's **almost impossible** to test a complex mobile application adequately without skills in:

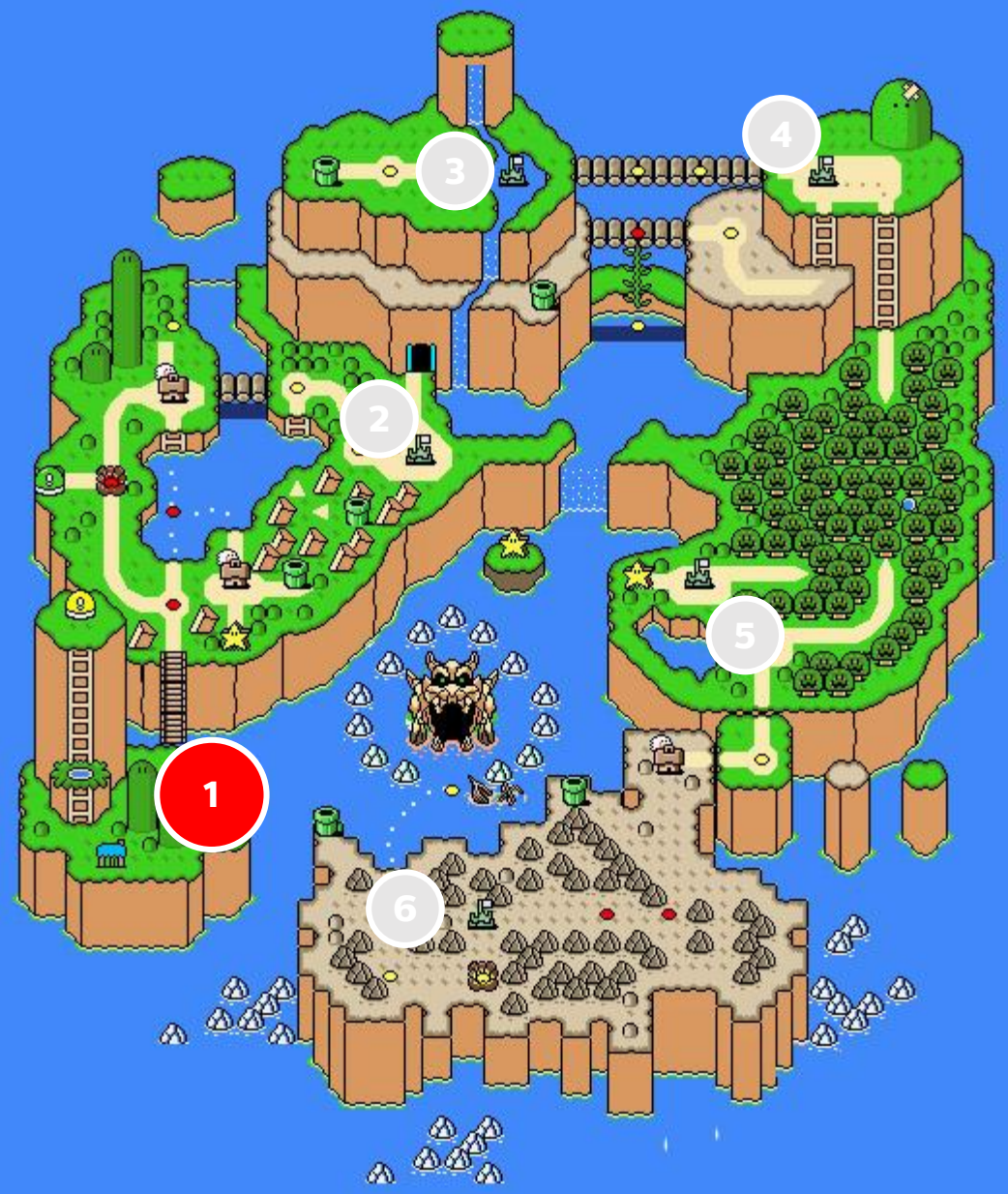
- Reversing (Java for Android but also ARM64 for iOS applications)
- Instrumentation and debugging
- Development of custom plugins for your favorite HTTP Proxy (Burp Suite, OWASP ZAP)

LET'S PLAY!



LEVEL 1

- NO SECURITY FEATURES



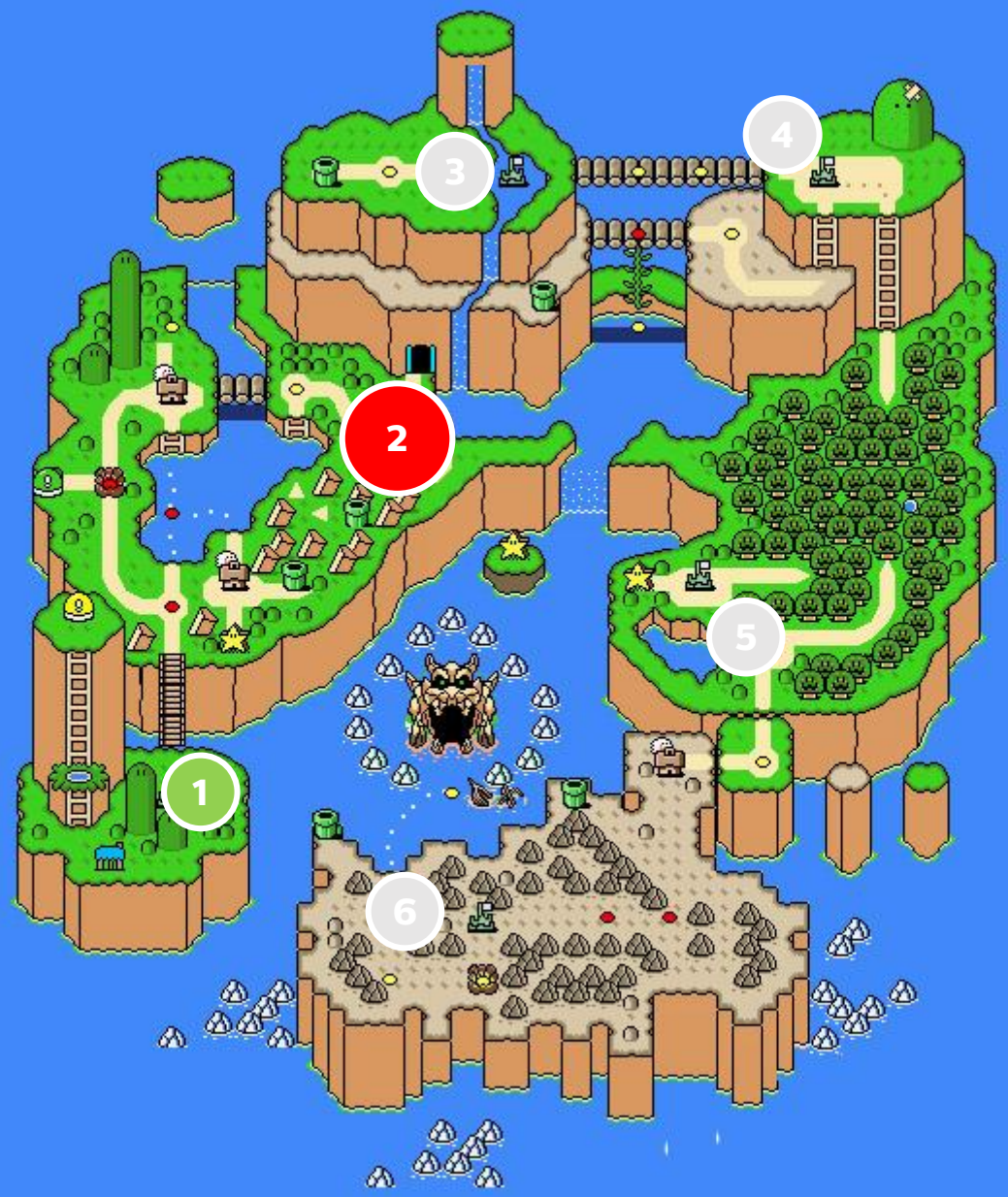
LEVEL 1

- NO SECURITY FEATURES

1. Set an HTTP proxy in the device.
2. Intercept data traffic
3. Test the backend!

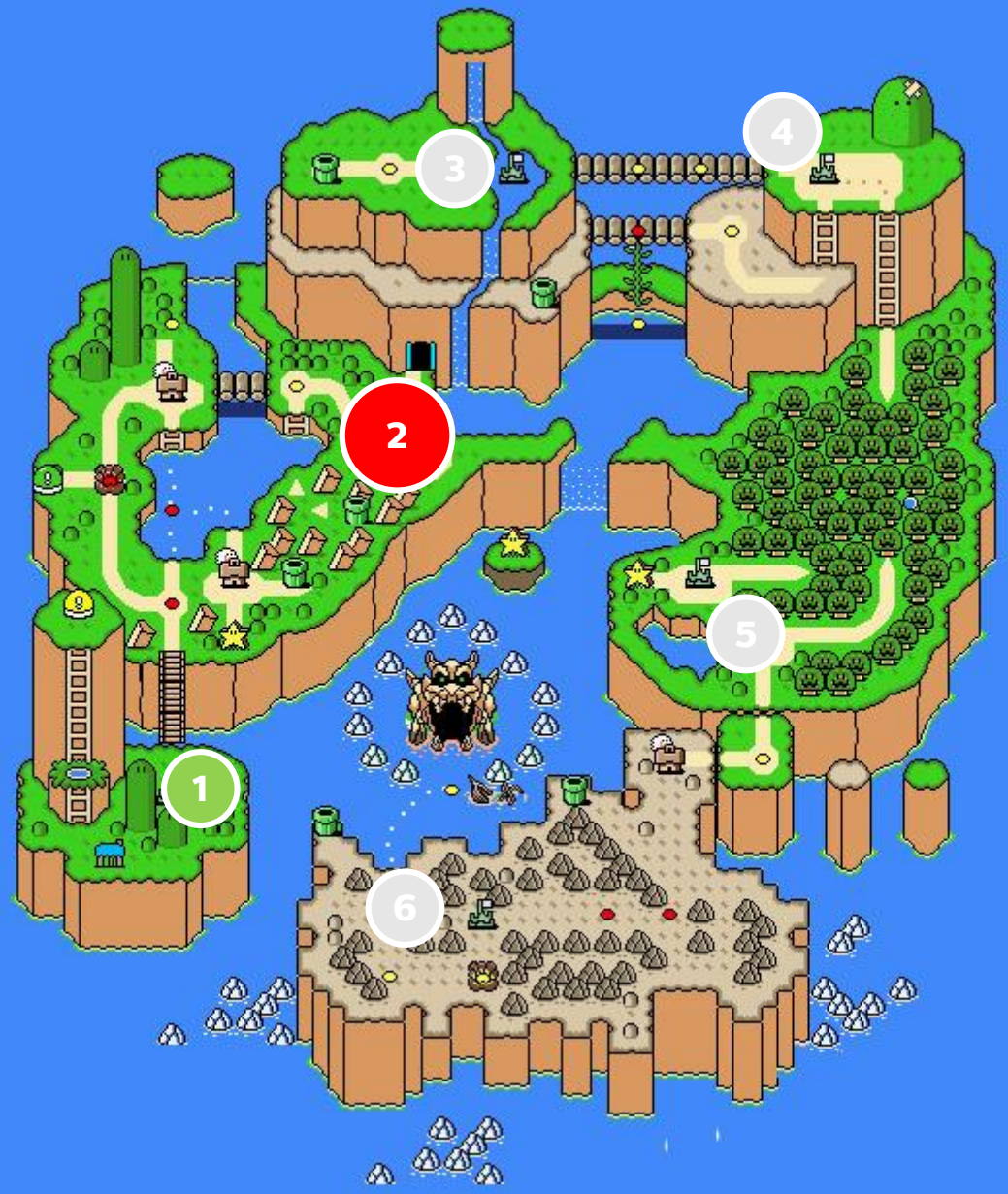
PORTSWIGGER BURP SUITE

- Suite of tools that helps penetration testers during the assessment
- Contains a lot of useful tools: HTTP Proxy, Intruder (fuzzer), a great automatic Scanner and a Repeater Tool
- Furthermore, it offers an external server very useful to test external service interactions (Collaborator) and a very good session manager
- It exports API to extend its functionalities, and consequently a huge number of plugins have been released by various developers that aid pentesters in almost every situation.
- **It is de-facto standard for web application security testing.**



LEVEL 2

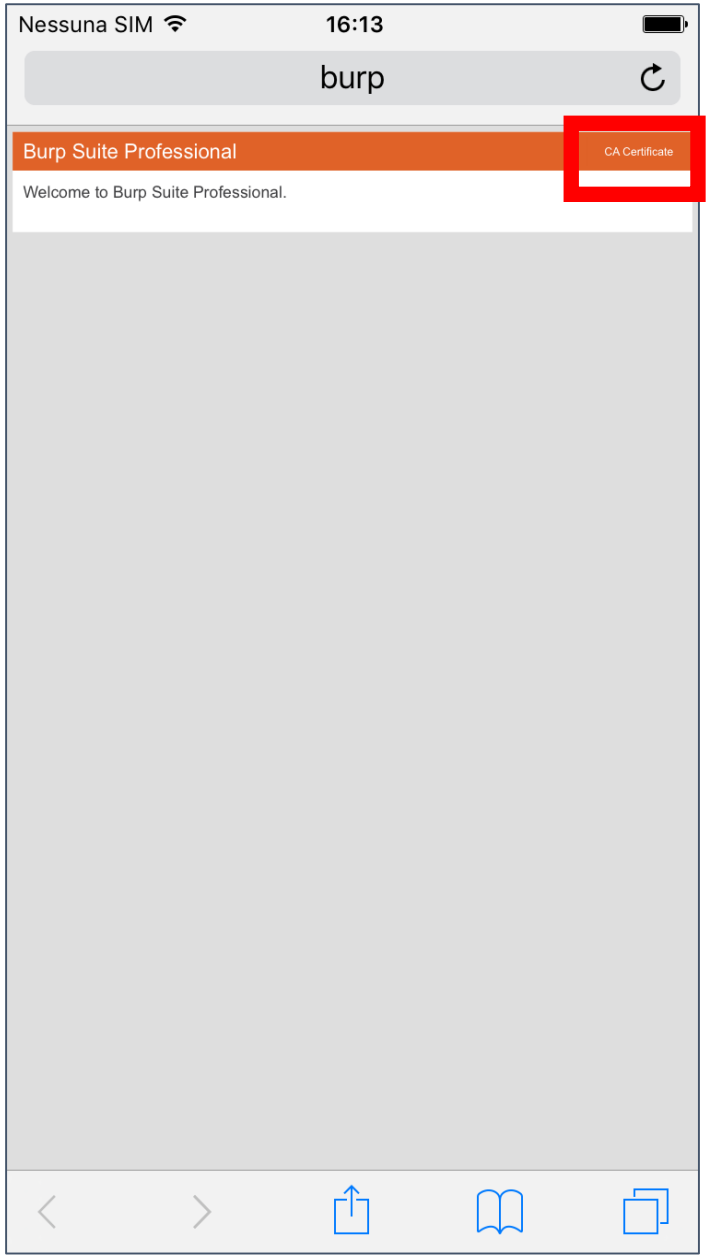
- SSL (AND THE CLIENT CHECKS FOR VALID SERVER CERTIFICATES)

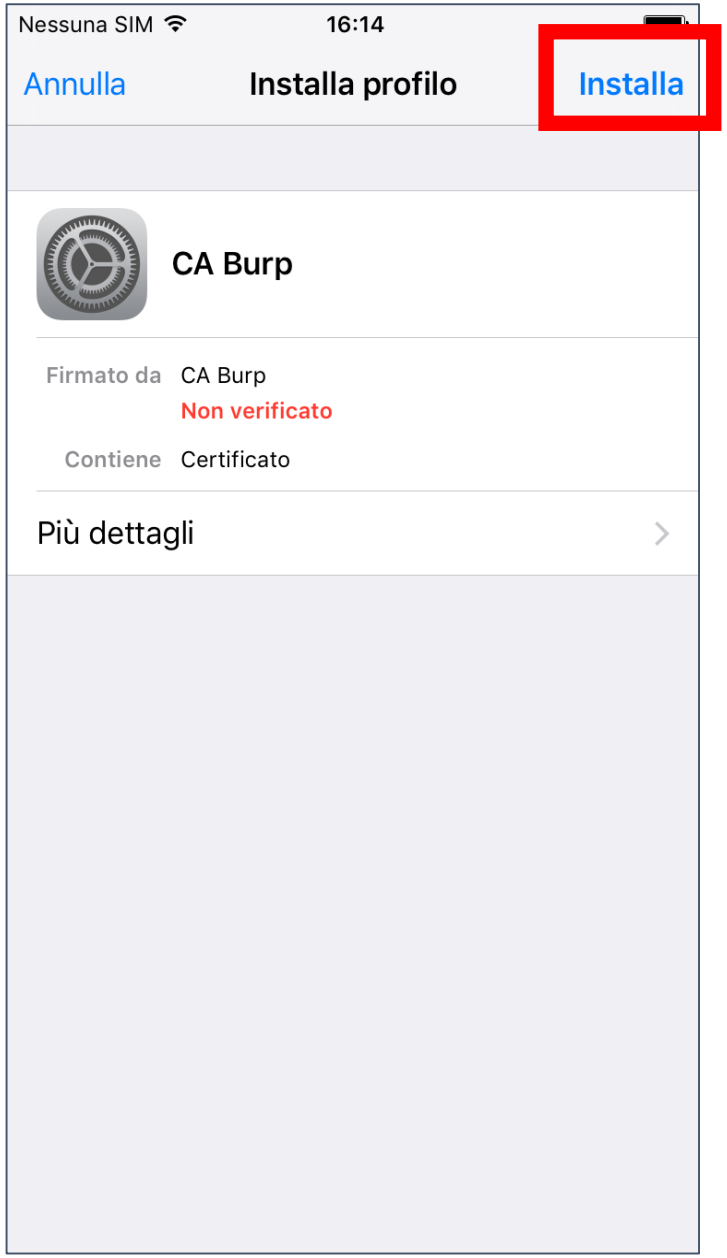


LEVEL 2

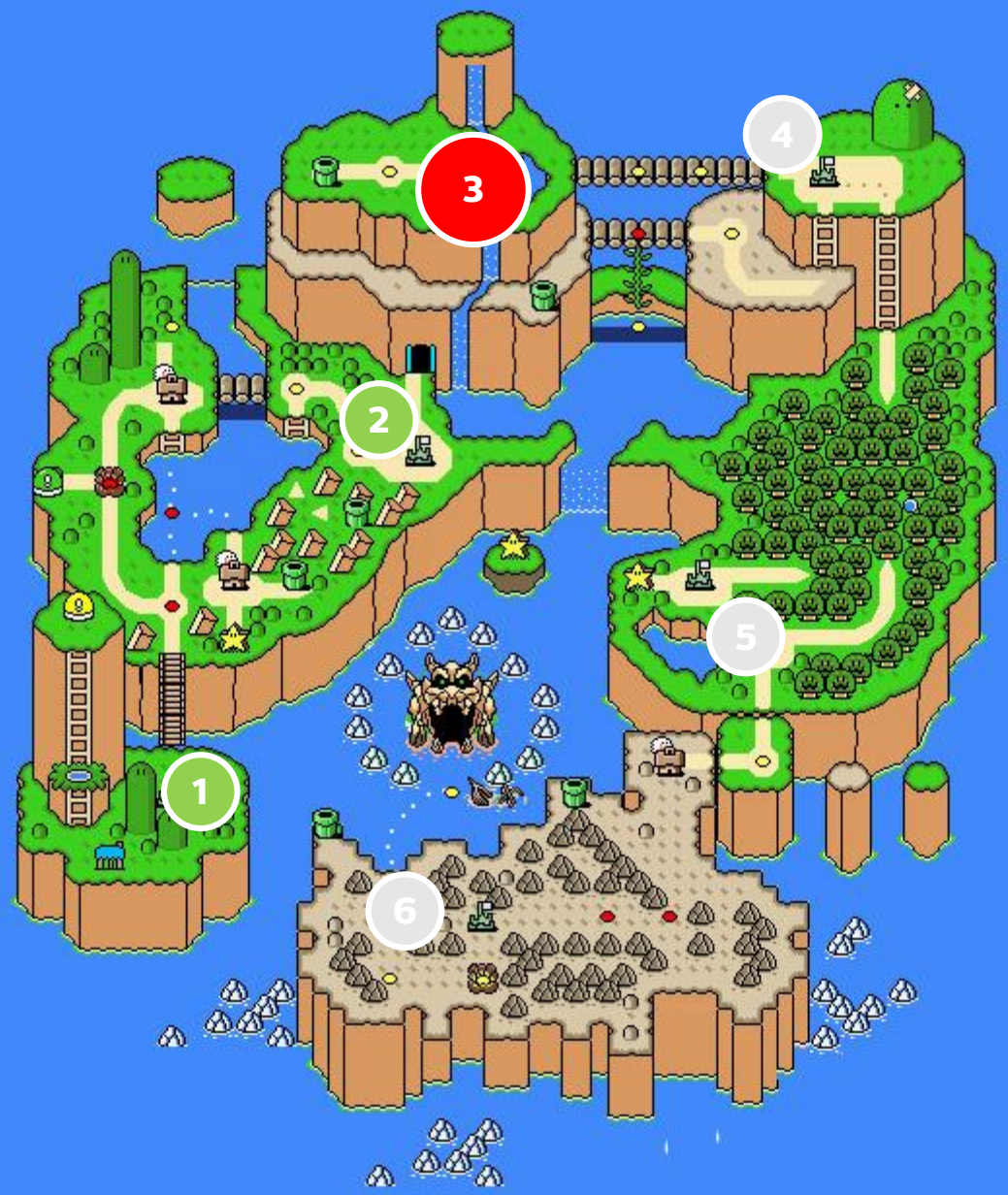
- SSL (AND THE CLIENT CHECKS FOR VALID SERVER CERTIFICATES)

1. Install Burp Suite CA certificate in the device
2. Set Burp Suite as proxy in the device
3. Intercept data traffic
4. Test the backend!


Nessuna SIM
16:13
burp
Burp Suite Professional
CA Certificate
Welcome to Burp Suite Professional.

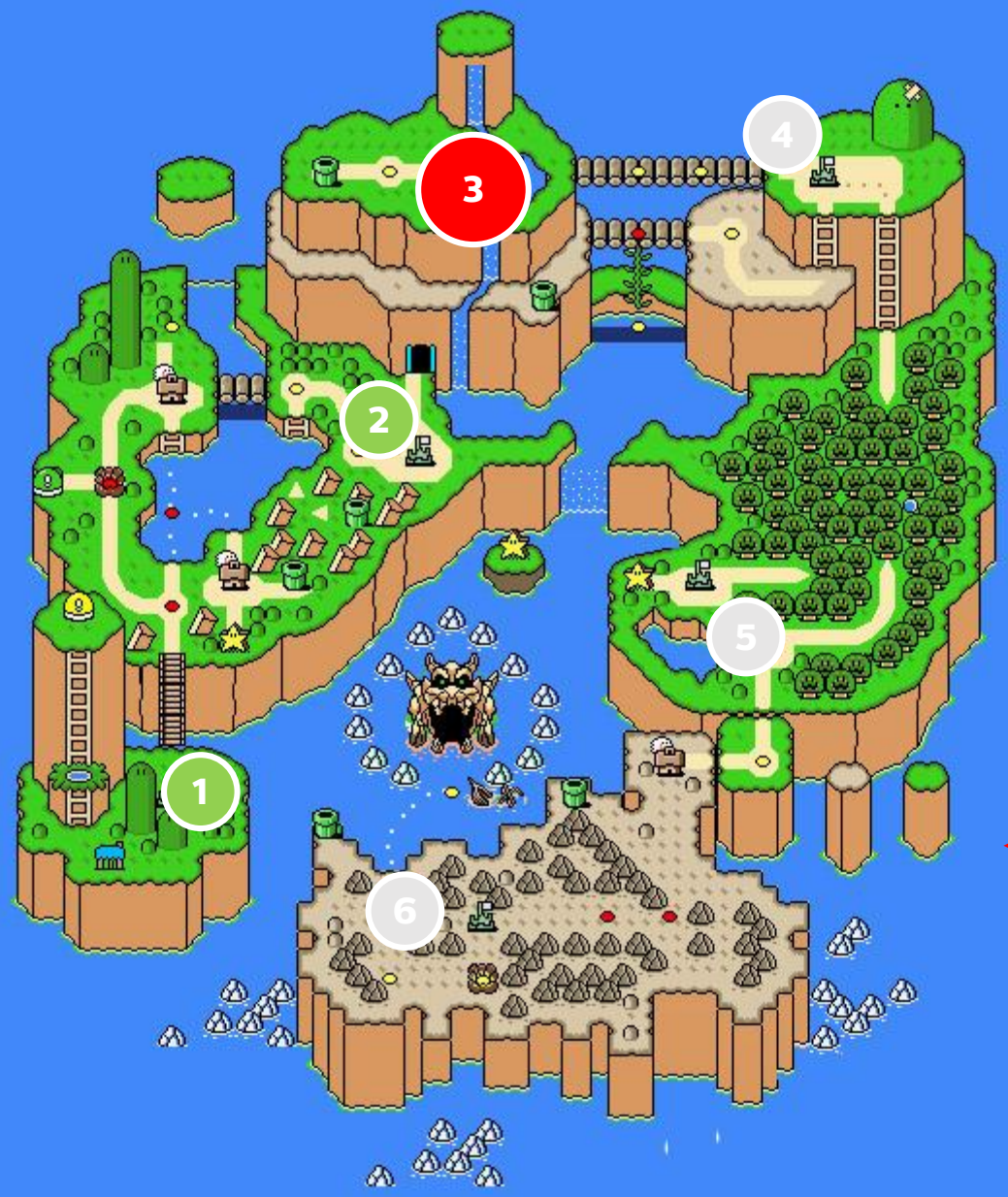

Nessuna SIM
16:14
Annulla
Installa profilo
Installa
CA Burp
Firmato da
CA Burp
Non verificato
Contiene
Certificato
Più dettagli


Nessuna SIM
16:14
Annulla
Attenzione
Installa
CERTIFICATO ROOT
L'installazione del certificato "CA Burp" fa sì che venga aggiunto all'elenco dei certificati attendibili su iPhone.
PROFILO NON VERIFICATO
Impossibile verificare l'autenticità di "CA Burp".



LEVEL 3

- SSL
- CERTIFICATE PINNING (THE CLIENT CHECKS FOR SPECIFIC SERVER CERTIFICATES)



LEVEL 3

- SSL
- CERTIFICATE PINNING (THE CLIENT CHECKS FOR SPECIFIC SERVER CERTIFICATES)

Now complications start! We can try generic tools/scripts for pinning bypass, but often we need to reverse the application and bypass the check.
For this task our favorite tool is Frida!

SSL PINNING BYPASS - 1

- **If you are lucky**, several generic tools and scripts try to bypass SSL pinning implemented in common ways.
- Android Example: Universal Android SSL Pinning Bypass with Frida (https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/)
- iOS Examples: Burp Suite Mobile Assistant (https://portswigger.net/burp/help/mobile_testing_using_mobile_assistant.html) and SSL Kill Switch 2 (https://github.com/nabla-c0d3/ssl-kill-switch2)

SSL PINNING BYPASS - 2

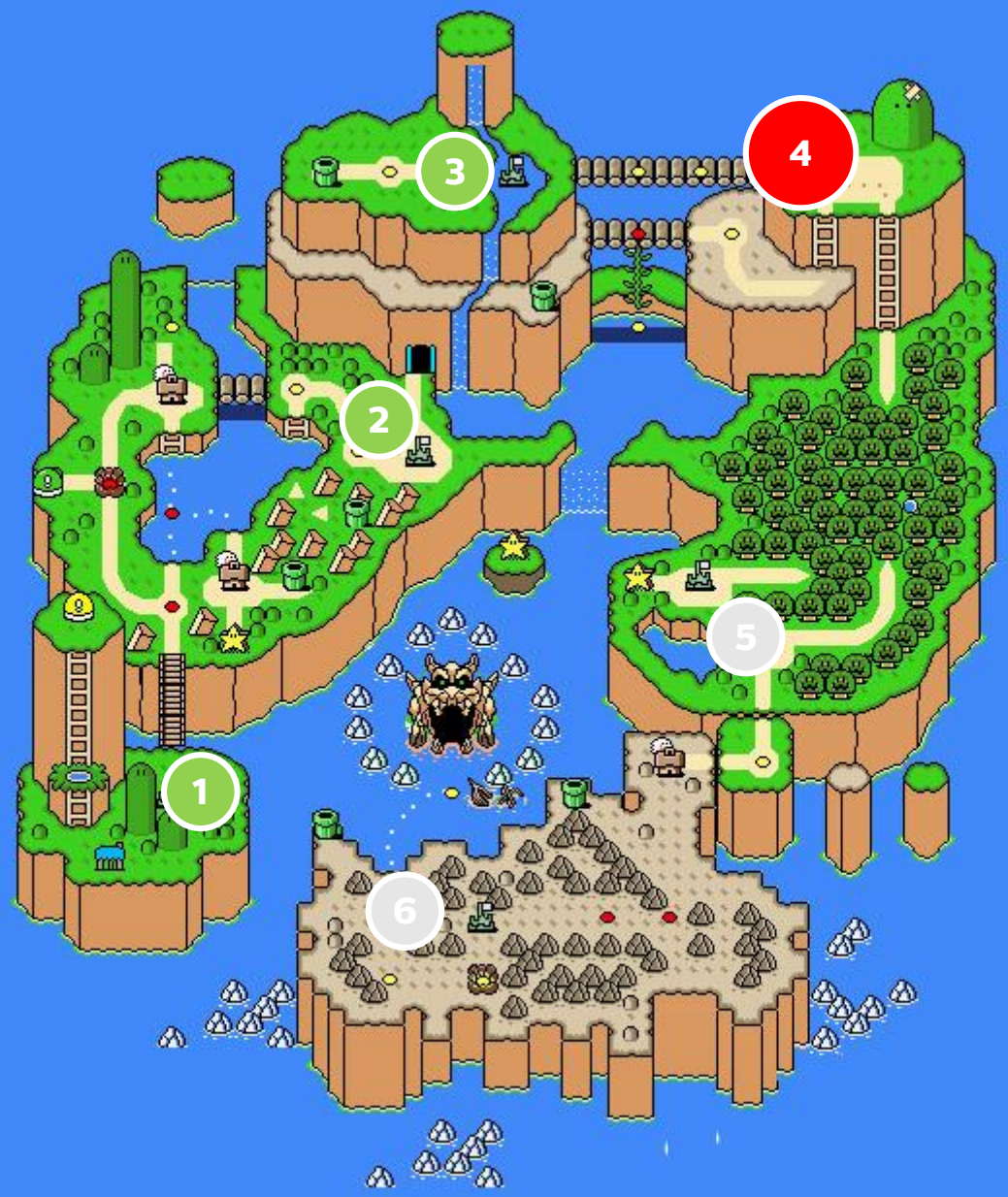
- But if you are not so lucky… it's time to reverse the application!
  - For Android applications: decompile dex and get Java code
  - For iOS applications and Android native libraries: disassemble code with IDA Pro (https://www.hex-rays.com/products/ida/), Radare2 (https://github.com/radare/radare2) or Hopper (https://www.hopperapp.com/)
- Once you locate the SSL Pinning code, you can patch the binary or you can dynamically modify code at runtime

FRIDA

- *Frida is a dynamic code instrumentation toolkit. It lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX. (cit. www.frida.re)*
- It is an amazing tool and it works both on iOS and on Android, allowing to inspect and modify running mobile code
- The **hooks** are specified with JavaScript language and can be used for **instrumentation** and **replacement** of Java and Objective-C functions

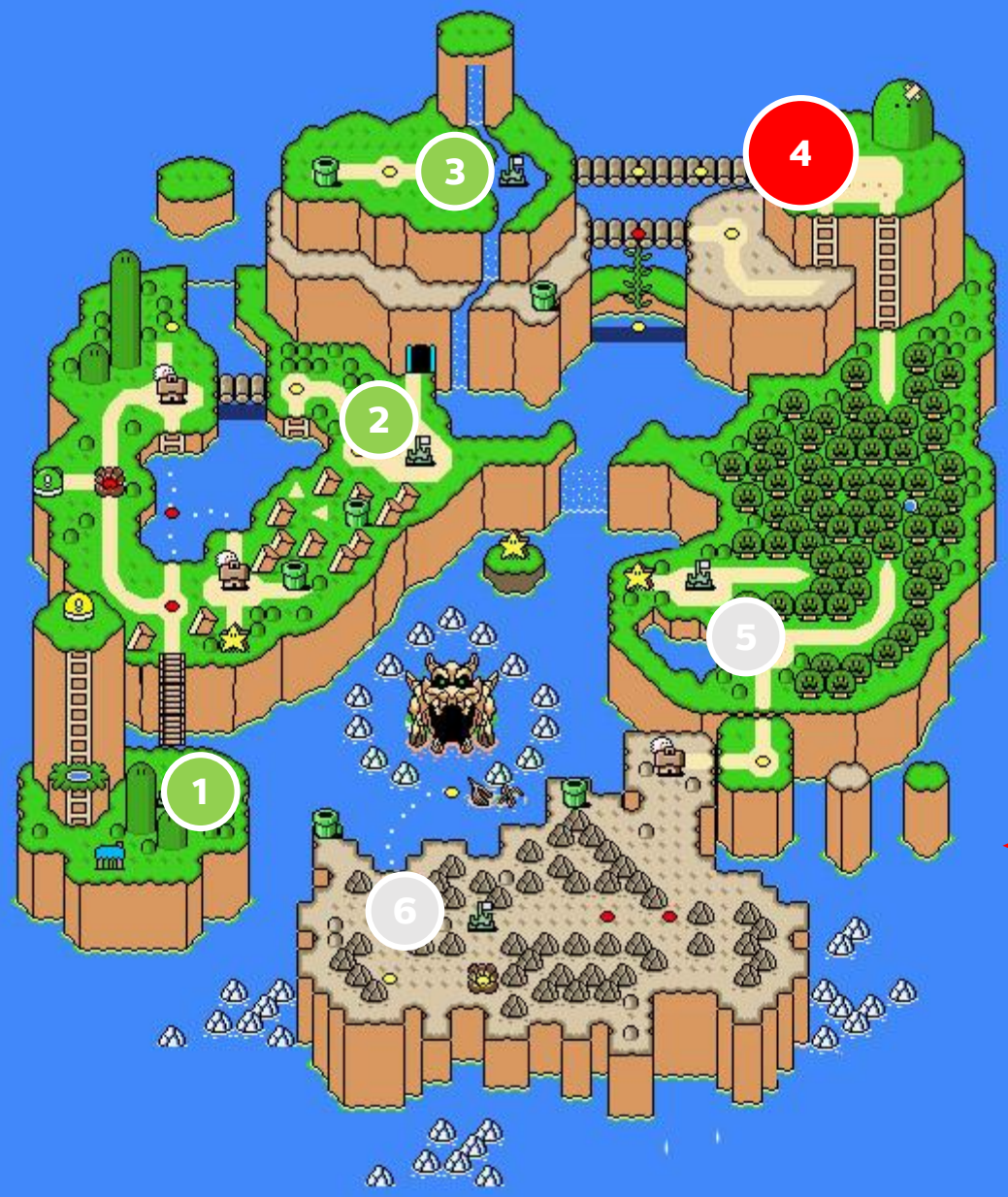
SSL PINNING BYPASS – FRIDA EXAMPLE

```
// SampleClass - (bool)checkServerCertificate:(id)
var hookSampleClass = ObjC.classes.SampleClass["- checkServerCertificate:"];

Interceptor.attach(hookSampleClass.implementation, {

    onEnter: function(args) {
    },

    onLeave: function(retval) {
        send("Bypassing Pinning");
        retval.replace(ptr(1));
    }

});
```



LEVEL 4

- SSL
- CERTIFICATE PINNING
- POST BODIES ENCRYPTED WITH SYMMETRIC ENCRYPTION



LEVEL 4

- SSL
- CERTIFICATE PINNING
- POST BODIES ENCRYPTED WITH SYMMETRIC ENCRYPTION

1. Install Burp Suite certificate in the device
2. Set Burp Suite as proxy in the device
3. Bypass SSL Pinning
4. Ouch! All POST bodies are encrypted! :'(

A BIG BASE64 BODY OF RAW BINARY DATA...

```
POST /login HTTP/1.1

Host: www.test.com

…

parameters=djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsf
jdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcndjshfj
dsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknj
skdnfjnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjb
jfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkans
djksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnv
dfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn%
3d%3d
```

LET'S REVERSE

```
SampleClass + (id)generatePostBody :(id)
SampleClass + (id)getClearTextMessage :(id)
```

LET'S REVERSE

FRIDA

```
SampleClass + (id)generatePostBody :(id)
SampleClass + (id)getClearTextMessage :(id)
```

```
// SampleClass + (id) generatePostBody :(id)
var hookGeneratePostBody =
    ObjC.classes.SampleClass["+ generatePostBody:"];
Interceptor.attach(generatePostBody.implementation, {

    onEnter: function(args) {
        var obj_input = ObjC.Object(args[2]);
        send("* generatePostBody input:");
        send(obj_input.toString());
    },

    onLeave: function(retval) {
        var obj_output= ObjC.Object(retval);
        send("* generatePostBody output:");
        send(obj_output.toString());
    }

});
```

LET'S REVERSE

FRIDA

```
SampleClass + (id)generatePostBody :(id)
SampleClass + (id)getClearTextMessage :(id)
```

```
…
* generatePostBody input:
{"username":"test","password":"testPassword"}
* generatePostBody output:
djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfj
danjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjsk
jcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfj
sfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjnd
jskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnn
dfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncx
jndjskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjx
nnvdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjks
ncxjndjskjcn==
…
```

```
// SampleClass + (id) generatePostBody :(id)
var hookGeneratePostBody =
    ObjC.classes.SampleClass["+ generatePostBody:"];
Interceptor.attach(generatePostBody.implementation, {

    onEnter: function(args) {
        var obj_input = ObjC.Object(args[2]);
        send("* generatePostBody input:");
        send(obj_input.toString());
    },

    onLeave: function(retval) {
        var obj_output= ObjC.Object(retval);
        send("* generatePostBody output:");
        send(obj_output.toString());
    }

});
```

MORE REVERSING...

```
POST /login HTTP/1.1
Host: www.test.com
…
parameters=djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjn
cjxknjskdnfjnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfnda
kfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn
djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnf
jnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnn
vdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn%3d%3d
```

Reverse → base64EncodedText = Base64(AES(clear-text))

MORE REVERSING...

```
POST /login HTTP/1.1
Host: www.test.com
…
parameters=djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjn
cjxknjskdnfjnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfnda
kfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn
djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnf
jnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnn
vdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn%3d%3d
```

Reverse 

ase64EncodedText = Base64(AES(clear-text))

FRIDA TO THE RESCUE!

```
Interceptor.attach(
  Module.findExportByName("libSystem.B.dylib","CCCrypt"),{

  onEnter: function(args) {
    send("CCOperation: " + parseInt(args[0]));
    send("CCAlgorithm: " + parseInt(args[1]));
    send("CCOptions: " + parseInt(args[2]));
    send("Key:");
    send(hexdump(ptr(args[3]), {
      offset: 0,
      length: parseInt(args[4]),
      header: true,
      ansi: true
    }));
    send("Key length: " + parseInt(args[4]));

    …

 });
```

```
…
CCOperation: 0 (encrypt)
CCAlgorithm:  0 (kCCAlgorithmAES128)
CCOptions: 1 (kCCOptionPKCS7Padding        )
Key: testPassword (in ASCII to make it more
readable)
Key length: 16
…
```

And were is the key stored?

Often it's hard-coded in the binary!

AND NOW THE CODING PART

- Great! Now we have only to code a Burp Suite plugin to decrypt requests and responses and to re-encrypt them if modified
- It seems simple, but it is not always so… We have to find a library that offers the same algorithm with the same parameters (padding, key size, etc.). Java **Bouncy Castle** is the way!
- Many hours of coding work!

How the plugin should be coded?

- We want to write a Burp Suite plugin **user-friendly enough** to test this particular application.
- We want to add a custom **editable** subtab containing the decrypted request/response
- We want be able to **modify** the decrypted requests
- **It's not an option: it's the only way to test the backend!**

AND HERE IS THE RESULT...

Request

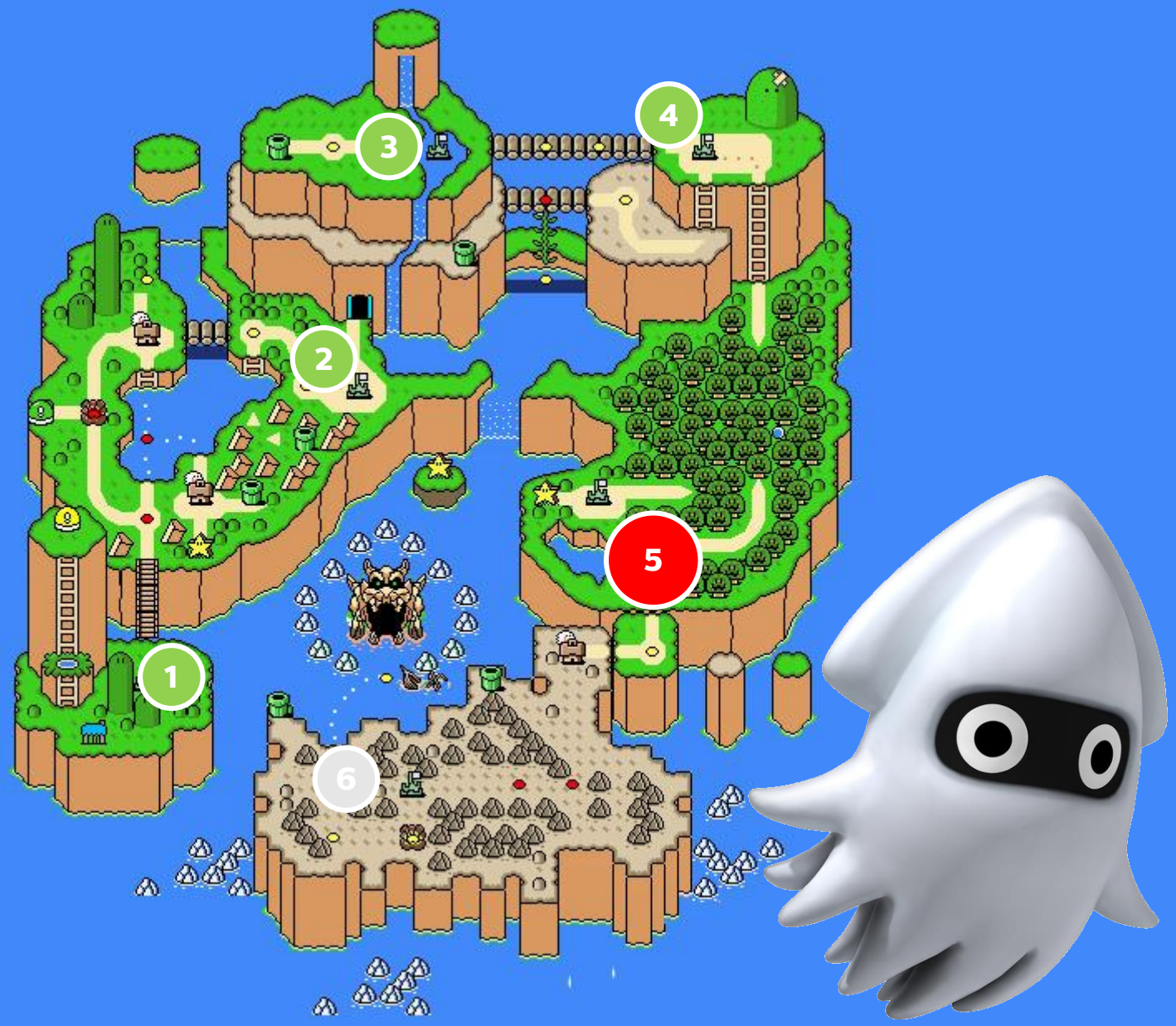
Raw | Params | Headers | Hex | Decrypted data

```
POST /login HTTP/1.1
Host: test.com
Content-Length: 234

parameters=asdhbdfhjbvkdsjkdnsfjnsjkfdhjsbfhjdsxcnckdnsjksncjndskjcndnskdnsdjncdskj
cnkdjscnkdjscnkjdscnjkdsnckjdnscjkdnscksdncksdnkjxhvusoroiewjdsfkfnxncvkjndsjkdnaks
jdnjsknfdfjnvjhcnxhjcbsdhjbvfnasdkjasfneuhfdsuknjkdncjkdnjknkj%3d%3d
```

Request

Raw | Params | Headers | Hex | Decrypted data

```
{"username":"test","password":"testPassword"}
```

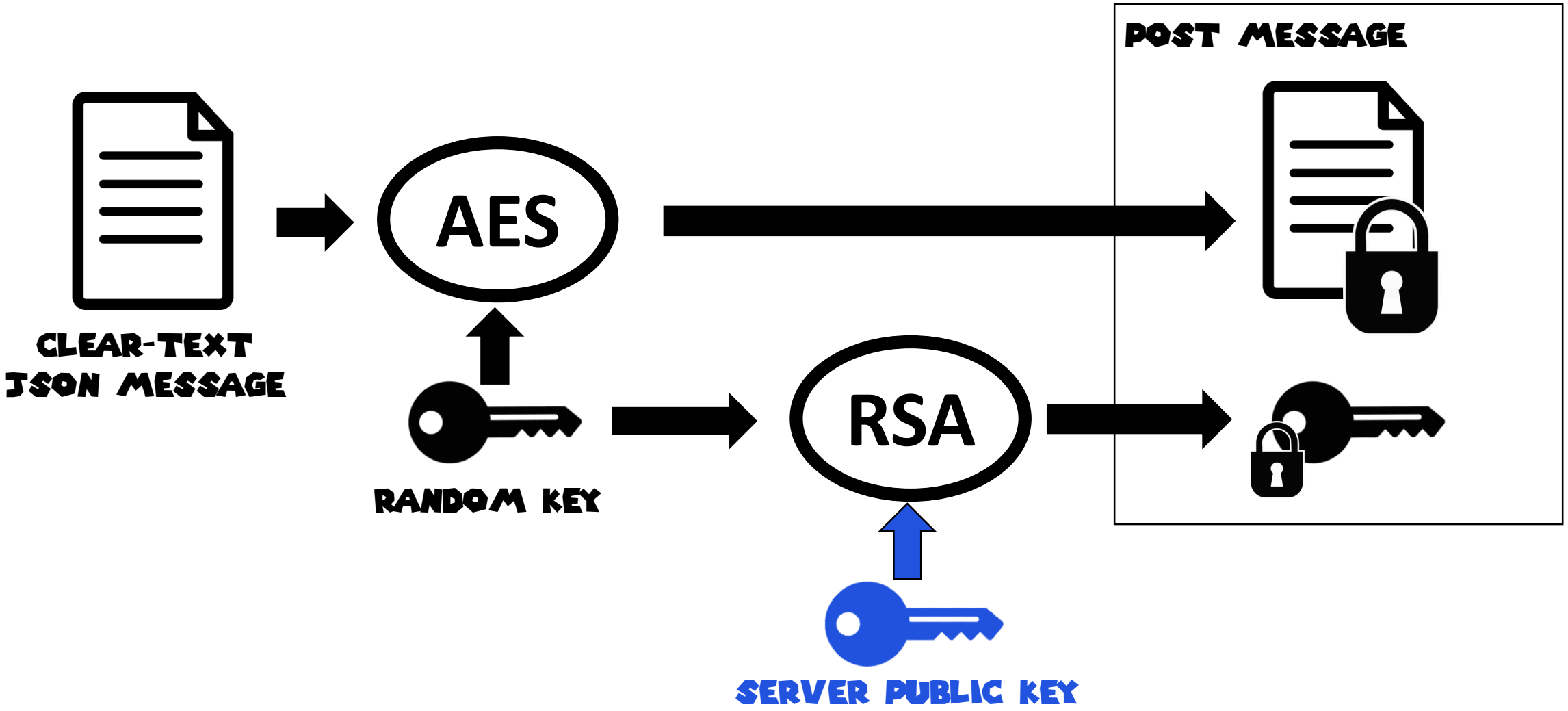


LEVEL 5

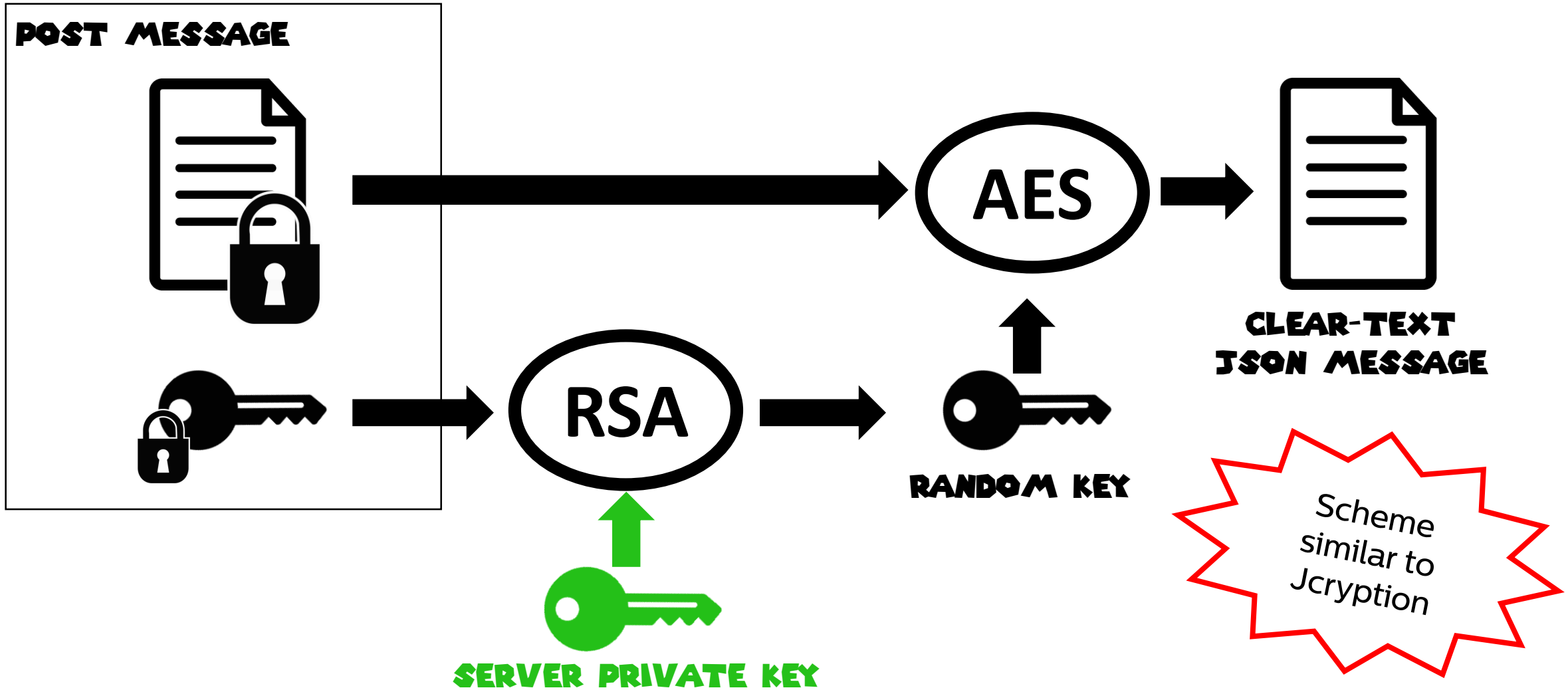
- SSL
- CERTIFICATE PINNING
- POST BODIES ENCRYPTED WITH SYMMETRIC ENCRYPTION AS THE PREVIOUS LEVEL, BUT...

... NOW EVERY REQUEST IS ENCRYPTED WITH A DIFFERENT RANDOMLY GENERATED KEY. THIS KEY IS THEN ENCRYPTED WITH ASYMMETRIC ENCRYPTION AND SENT WITH THE MESSAGE IN THE BODY!

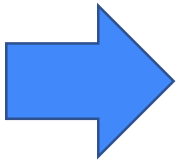
ENCRYPTION SCHEME - CLIENT SIDE



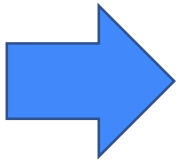
DECRYPTION SCHEME – SERVER SIDE



We don’t have the private key necessary to decrypt the random key



We can’t decrypt the random key



We can’t decrypt the body from our custom-written Burp Suite plugin. Stop.

We don't have the private key necessary to decrypt the random key

We can't decrypt the

We can't decrypt the body from our custom-written Burp Suite plugin. S

We have to find another way.

How can we code the plugin now? Option 1

We can trap CCCrypt function with Frida (as seen before) and print the asymmetric keys before they are encrypted.

Not convenient. We need to pass to the plugin a new key **for every request** (if we try 20 SQL injection vectors we have to manually insert 20 keys in the plugin)

How can we code the plugin now? Option 2

We can replace the public key used for the encryption of the key (physically if it is stored on the device or with Frida) with a public key generated by us (as a classic **MitM with SSL**). This way, Burp can decrypt the random key, and re-encrypt it with the public key of the server.

More convenient, but it requires more coding work, because the Burp Suite plugin has to deal also with public key encryption and not only with symmetric encryption.

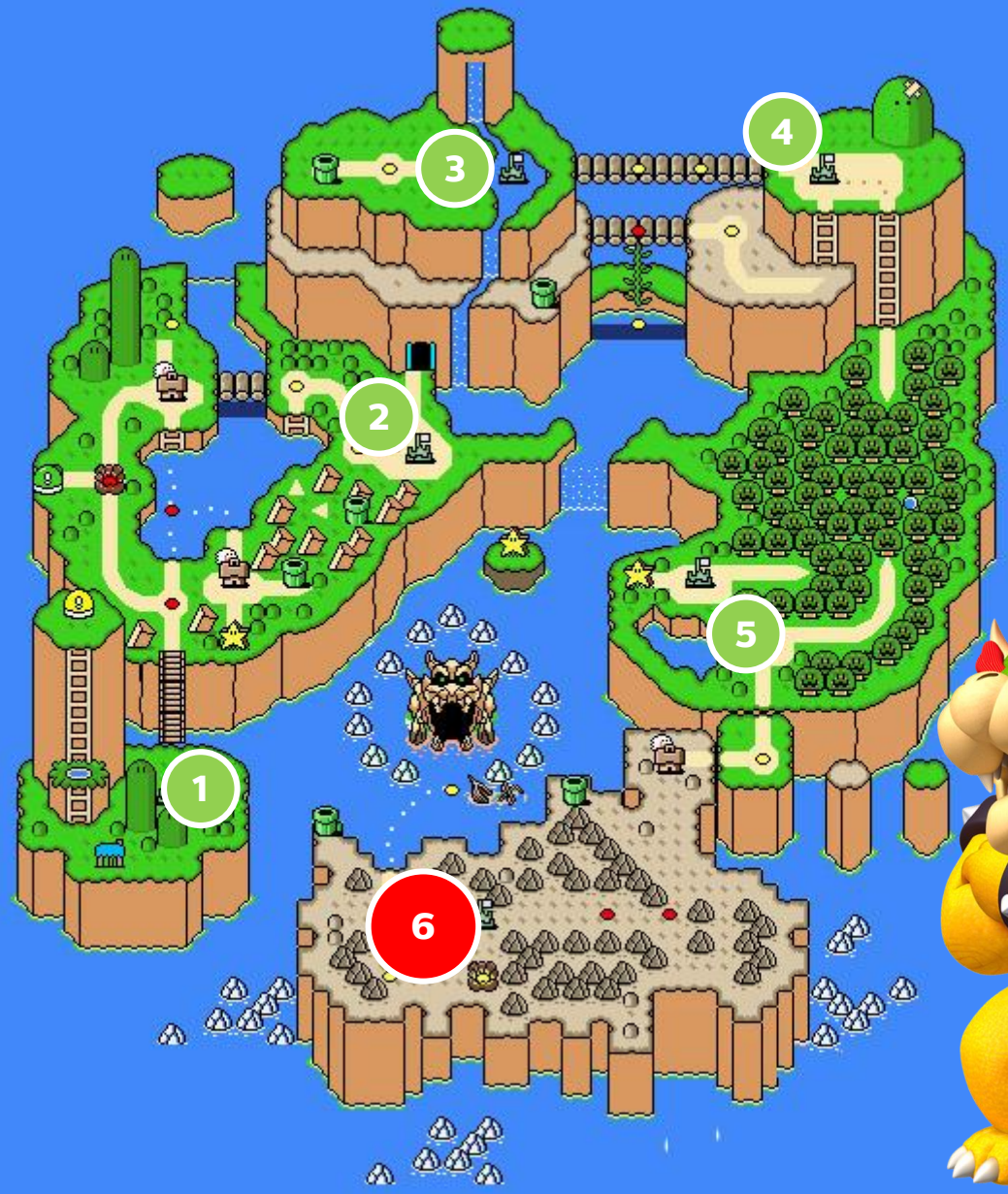
THINK OUTSIDE THE BOX

- Ok, and if we trap the function that generates the random values with Frida and replace the return value with a fixed string? For example 0x1111111111111111 ?
- In this way we can write a plugin that encrypts/decrypts the JSON of every request with the chosen fixed key without considering the part of the asymmetric encryption at all!
- And the **problem is solved!**

THE PROBLEM HAS BEEN SOLVED BUT...

- **We spent a lot of time in reversing!**
- **We spent a lot of time in coding!**
- What if the application employs a custom encryption method? We need to reverse and re-implement in Java, Python or Ruby the custom encryption method. **Very time consuming!**
- What if we can't find a library that offers the same encryption/signature algorithm with the same parameters of the mobile application?

WE NEED A MORE CONVENIENT WAY!



LEVEL 6 FINAL BOSS!

- SSL
- CERTIFICATE PINNING
- POST BODIES ENCRYPTED WITH SYMMETRIC ENCRYPTION
- KEYS ENCRYPTED WITH ASYMMETRIC ENCRYPTION AS THE PREVIOUS LEVEL, BUT...

... NOW THE BACKEND CHECKS THE INSERTED KEY AND DISCARDS IT IF IT HAS BEEN USED IN THE PREVIOUS REQUESTS!

THE FINAL BOSS!

- We can't use Frida to replace the generated key with a fixed string, because it will work only for the first request!
- We can return to the inconvenient way (print the key with Frida and manually insert every key in Burp Suite) or to the heavy-code way (change the public key with a generated one and a complex Burp Suite plugin that handles both symmetric and asymmetric encryption)
- Or… **we have to find a way to let Burp talk with Frida!**

AUTHORS

- PIERGIOVANNI CIPOLLONI
- FEDERICO DOTTA

CONTRIBUTORS

- MAURIZIO AGAZZINI

WITHOUT BRIDA

```
POST /login HTTP/1.1
Host: www.test.com
…
parameters=djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjn
cjxknjskdnfjnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfnda
kfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn
djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdnf
jnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnn
vdfjsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxjndjskjcn%3d%3d
```

Reverse [illegible] ncodedText = Base64(AES(clear-text,random_key) + RSA(random_key,public_key))

WITHOUT BRIDA

```
POST /login HTTP/1.1
Host: www.test.com
…
parameters=djshfj                    fvjbjfndak          jxnnvdfjsfjdanjfndsjn
cjxknjskdnfjnjvx          sdj          ljskjcndj          svcxuchvjsdbfvjbjfnda
kfdshfcjxnnvdfjs          fndsj          kdnfjn          kansdjksncxjndjskjcn
djshfjdsvcxuchvj          bjfndak          nnvd          danjfndsjncjxknjskdnf
jnjvxcnjkansdjksn          jskjcndjs          ru          ofvjbjfndakfdshfcjxnn
vdfjsfjdanjfndsjn          skdnfjnjvx          cxjndjskjcn%3d%3d
```

Reverse  AES(clear-text,random_key) +

KEY?

RSA(random_key,public_key))

WITH BRIDA

```
POST /login HTTP/1.1
Host: www.test.com
…
parameters=djshfjdsvcx
cjxknjskdnfjnjvxcnjkansd
kfdshfcjxnnvdfjsfjdanjf
djshfjdsvcxuchvjsdbfv
jnjvxcnjkansdjksncxjndjskjcndjs                                    fcjxnn
vdfjsfjdanjfndsjncjxknjskdnfjnj                                    d
```

We don't have to deeply reverse and implement complex plugins! We can simply ask the target application to encrypt/decrypt messages for us!


```
SampleClass + (id) generatePostBody :(id)
SampleClass + (id) getClearTextMessage :(id)
```

WITH BRIDA

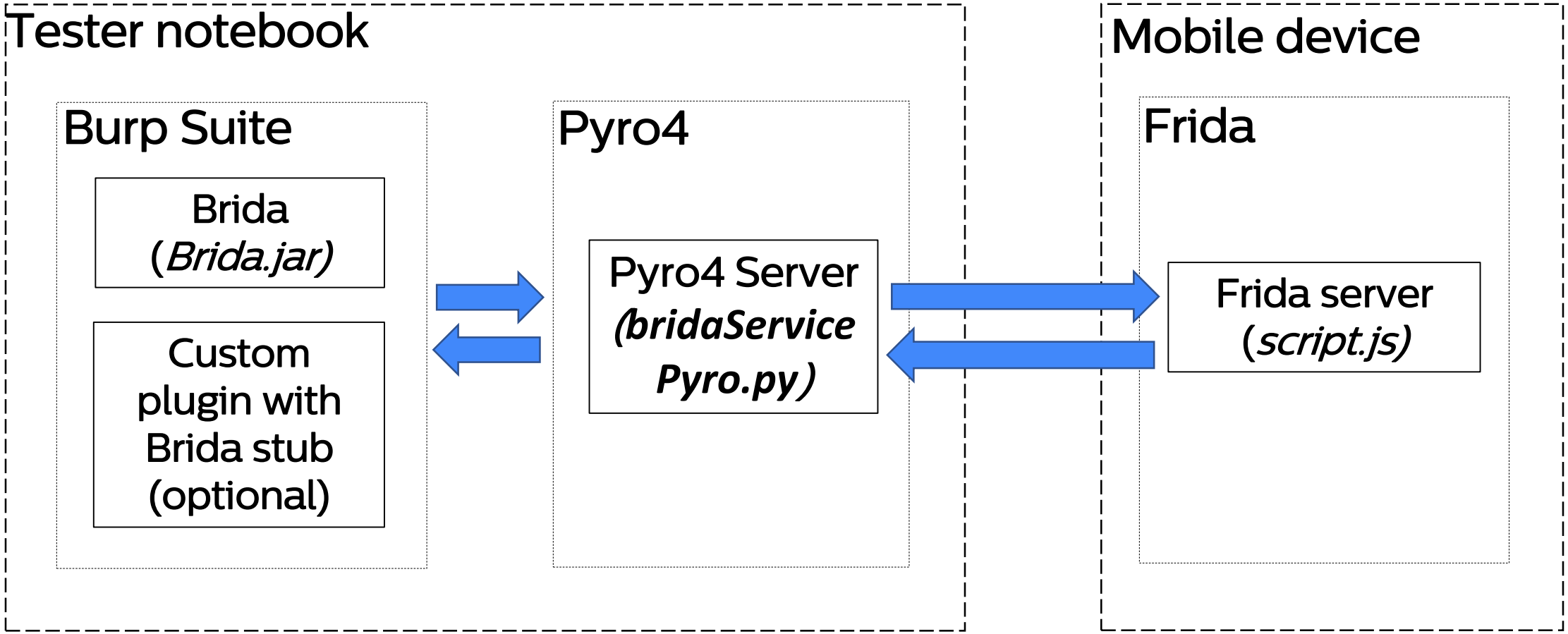
- When we have to decrypt a message, we use Brida to ask the application to decrypt the message for us
- When we have to encrypt a message, we use Brida to ask the application to encrypt the message for us
- We don't need to know how the message is encrypted/decrypted!!

WITH BRIDA

- Much less reversing! (days!)
- Much less coding! (We don’t need to reimplement encryption/decryption/signature functions, we simply use directly the iOS application functions)
- We can write a simple Burp Suite plugin with few lines of code to do the job!

LEVEL 1	
LEVEL 2	• SSL
LEVEL 3	• SSL • Certificate pinning
LEVEL 4	• SSL • Certificate pinning • POST bodies encrypted with symmetric encryption
LEVEL 5	• SSL • Certificate pinning • POST bodies encrypted with symmetric encryption • Keys encrypted with asymmetric encryption
LEVEL 6	• SSL • Certificate pinning • POST bodies encrypted with symmetric encryption • Keys encrypted with asymmetric encryption as the previous level • Check previous keys

HOW DOES IT WORK?



HOW DOES IT WORK?

- Thanks to the «rpc» object of Frida it is possible to expose RPC-style functions
- From Burp Suite we call a Pyro function that acts as a bridge
- Pyro calls the selected Frida exported function and returns the result back to Burp Suite

USING BRIDA – DEDICATED TAB

Stub generator | Execute method

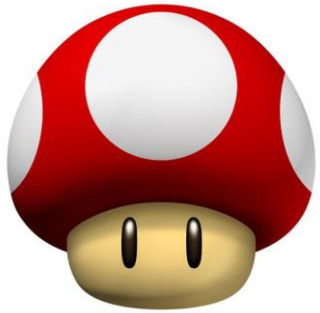
Method name: encryptbody

Argument: {"username":"test","password":"testPassword"} Add

Argument list: Remove Modify

Kill application
Reload JS
Java Stub
Python Stub
Save settings to file
Load settings from file
Execute Method

```
rpc.exports = {

    encryptbody: function(body) {
        var res = ObjC.classes.SampleClass.generatePostBody(body);
        return res.toString();
    }

}
```



USING BRIDA - CONTEXT MENU

Request

Raw | Hex

```
POST /login HTTP/1.1
Host: www.test.com

parameters=djshfjdsvcxuchvjsdbfvjbjfndakfdshf
kdnfjnjvxcnjkansdjksncxjndjskjcndjshfjdsvcxuc
jsfjdanjfndsjncjxknjskdnfjnjvxcnjkansdjksncxj
jbjfndakfdshfcjxnnvdfjsfjdanjfndsjncjxknjskdn
djshfjdsvcxuchvjsdbfvjbjfndakfdshfcjxnnvdfjsf
jkansdjksncxjndjskjcn%3d%3d
```

- Send to Spider
- Do an active scan
- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Request in browser
- Insert collaborator p
- Insert collaborator i
- Brida Custom 1
- Brida Custom 2

```
contextcustom2: function(message) {
    var res = ObjC.classes.SampleClass.generatePostBody(message);
    return res.toString();
},
```

```
contextcustom1: function(message) {
    var res = ObjC.classes.SampleClass.getClearTextMessage(message);
    return res.toString();
},
```

Request

Raw | Hex

```
POST /login HTTP/1.1
Host: www.test.com

parameters={"username":"test","password":"a'
```

- Send to Spider
- Do an active scan
- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Request in browser
- Insert collaborator p
- Insert collaborator i
- Brida Custom 1
- Brida Custom 2

USING BRIDA – CUSTOM PLUGIN

Stub generator | Execute method

```
import net.razorvine.pyro.*;

String pyroUrl = "PYRO:BridaServicePyro@localhost:9999";
try {
		PyroProxy pp = new PyroProxy(new PyroURI(pyroUrl));
		String ret = (String)pp.call("callexportfunction","METHOD_NAME",new String[]{"METHOD
_ARG_1","METHOD_ARG_2",...});
		pp.close();
} catch(IOException e) {
		// EXCEPTION HANDLING
}
```

Kill application
Reload JS
Java Stub
Python Stub
Save settings to file
Load settings from file
Execute Method

Stub generator | Execute method

```
import Pyro4

uri = 'PYRO:BridaServicePyro@localhost:9999'
pp = Pyro4.Proxy(uri)
args = []
args.append("METHOD_ARG_1")
args.append("METHOD_ARG_2")
args.append("...")
ret = pp.callexportfunction('METHOD_NAME',args)
pp._pyroRelease()
```

Kill application
Reload JS
Java Stub
Python Stub
Save settings to file
Load settings from file
Execute Method

A PRACTICAL EXAMPLE: SIGNAL

- Signal is an **encrypted communications application** for Android and iOS.
- Signal is perfect as an example because it **encrypts messages** and because it is **open source**
- We redirect iOS traffic through Burp Suite (bypassing pinning)
- We use Brida and a custom plugin to dynamically modify the content of every message in «pwned»

HANDS ON WITH SIGNAL!

Burp Intruder Repeater Window Help

Target | Proxy | Spider | Scanner | Intruder | Repeater | Sequencer | Decoder | Comparer | Extender | Project options | User options | Alerts | Logger++ | Brida

Server status: running

Application status: spawned

Python binary path: C:\Python27\python.exe

Pyro host: localhost

Pyro port: 9999

Frida JS file path: scriptSignal.js

Application ID: org.whispersystems.signal

(•) Frida Remote ( ) Frida Local

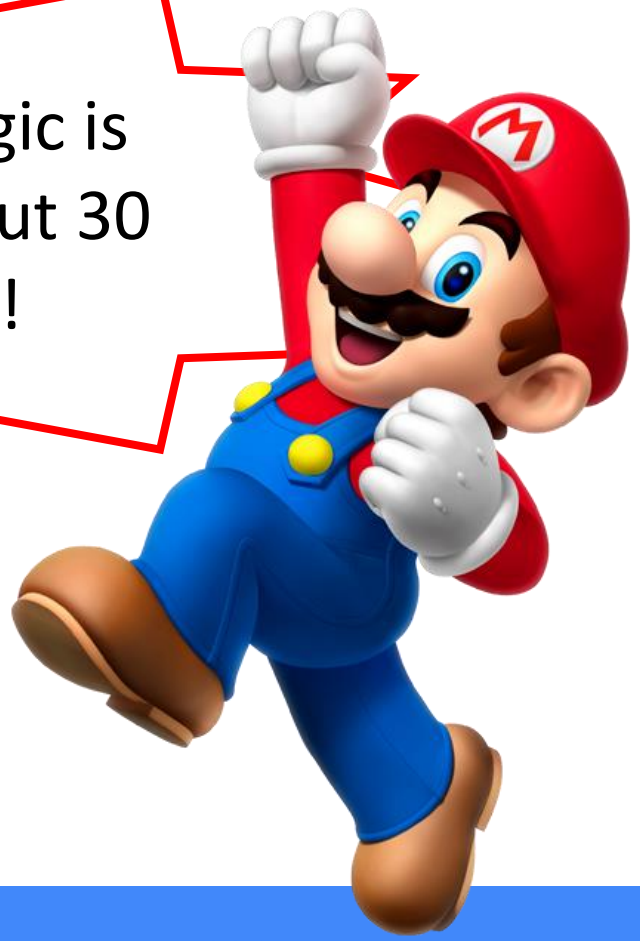
SCRIPTSIGNAL.JS

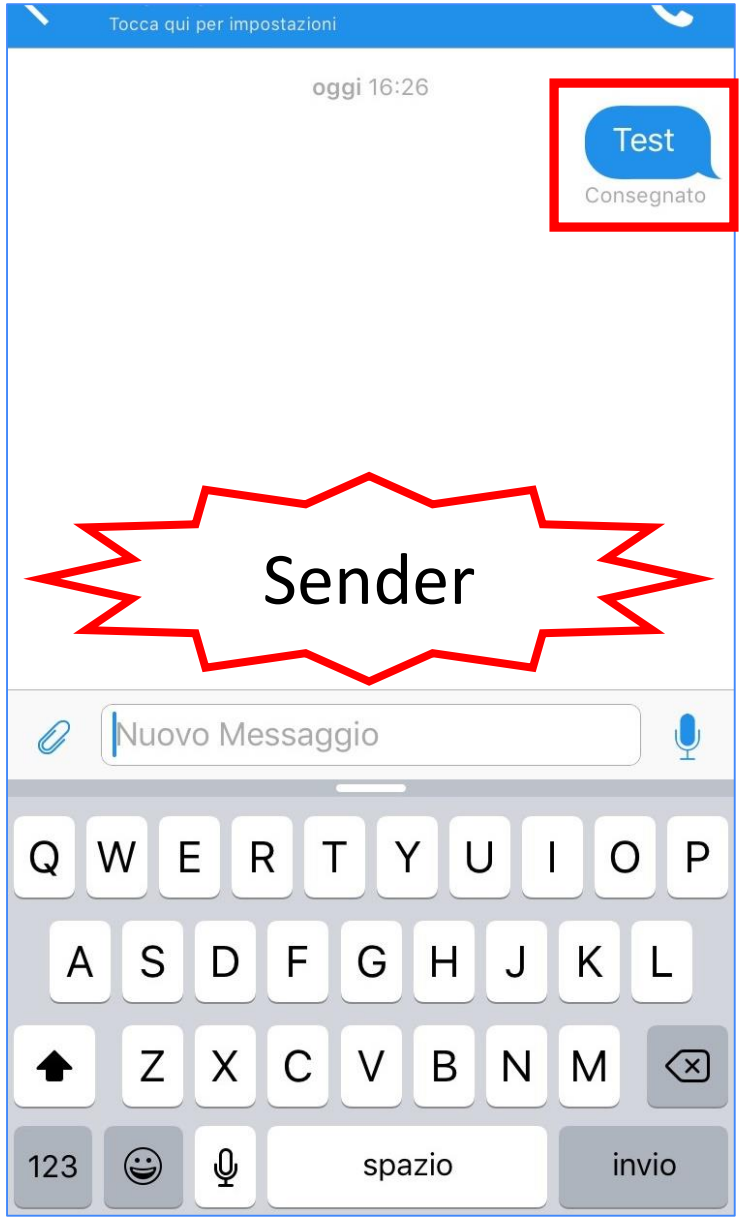
```
rpc.exports = {

    changemessage: function(message) {

        var env = ObjC.classes.Environment.getCurrent();
        var messageSender = env.messageSender();
        var signalRecipient = ObjC.classes.SignalRecipient.alloc().initWithTextSecureIdentifier_relay_(
            destNum,null);
        var contactThread = ObjC.classes.TSContactThread.alloc().initWithContactId_(destNum);
        var mex = ObjC.classes.TSOutgoingMessage.alloc().initWithTimestamp_inThread_messageBody_(
            Math.round(+new Date()/1000),null,message);
        var retVal = messageSender.deviceMessages_forRecipient_inThread_(mex,signalRecipient,contactThread);
        var retValMessage = retVal.objectAtIndex_(0);
        return retValMessage.toString();

    },
    …
```

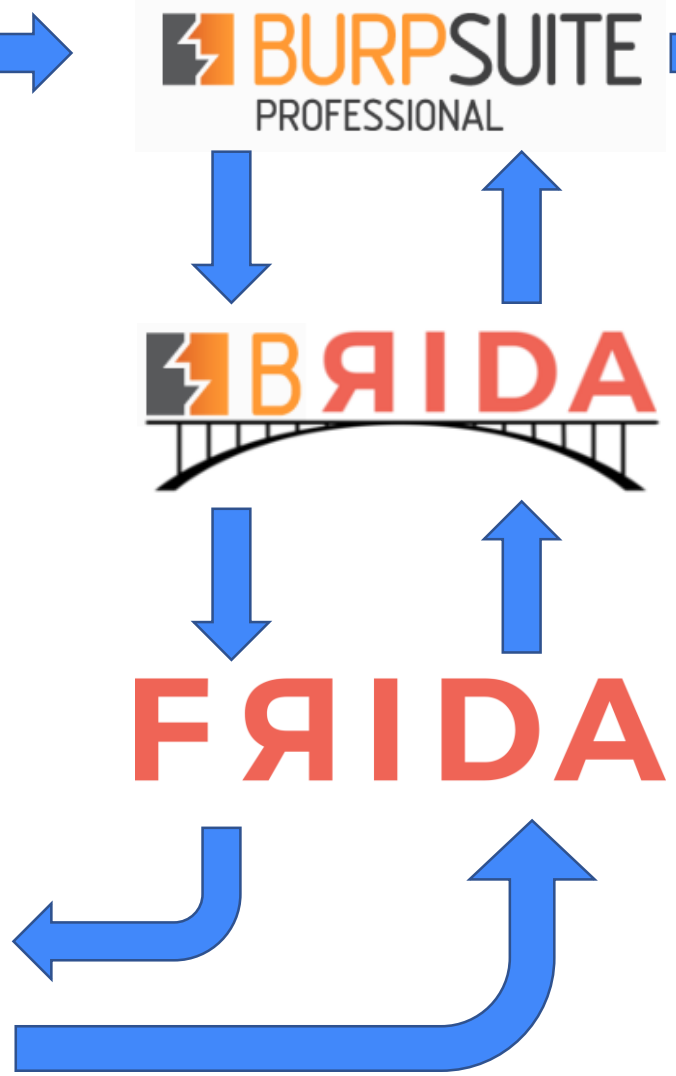
BURPEXTENDER.JAVA

```java
@Override
public void processHttpMessage(int toolFlag, boolean messageIsRequest, IHttpRequestResponse messageInfo) {
  if(messageIsRequest) {
    byte[] request = messageInfo.getRequest();
    IRequestInfo analyzedRequest = helpers.analyzeRequest(request);
    List<String> headers = analyzedRequest.getHeaders();
    int bodyOffset = analyzedRequest.getBodyOffset();
    byte[] body = Arrays.copyOfRange(request, bodyOffset, request.
    String bodyString = new String(body);
    if(bodyString.contains("destinationRegistrationId")) {
      JSONObject objRoot = new JSONObject(bodyString);
      JSONObject objMessage = objRoot.getJSONArray("messag
      String pyroUrl = "PYRO:BridaServicePyro@localhost:9999";
      try {
        PyroProxy pp = new PyroProxy(new PyroURI(pyroUrl));
        String newMessage = (String)pp.call("callexportfunction",
        pp.close();
        Pattern pattern = Pattern.compile(".*content = \"(.*?)\".*");
        Matcher matcher = pattern.matcher(newMessage);
        if (matcher.find())  {
          newMessage = matcher.group(1);
          objMessage.put("content", newMessage);
          String newBodyString = objRoot.toString();
          String newBodyString2 = newBodyString.replace("/", "\\/");
          byte[] newRequest = helpers.buildHttpMessage(headers, newBodyString2.getBytes());
          messageInfo.setRequest(newRequest);
        }
      } catch(IOException e) {
        stderr.println(e.toString());
      }
    }
  }
}
```

All the plugin logic is contained in about 30 lines of code!




Tocca qui per impostazioni
oggi 16:26
Test
Consegnato
Sender
Nuovo Messaggio
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
123
spazio
invio


BURPSUITE
PROFESSIONAL
BRIDA
FRIDA


Tap here for settings
Today 16:26
pwned
Receiver
New Message

LINKS!

- Brida repo: https://github.com/federicodotta/Brida
- Brida releases: https://github.com/federicodotta/Brida/releases
- Signal example: https://github.com/federicodotta/Brida/tree/master/examples
- Article that describes Brida: https://techblog.mediaservice.net/2017/07/brida-advanced-mobile-application-penetration-testing-with-frida/

GAME OVER!

THANKS!

ANY QUESTIONS?

FEEL FREE TO CONTACT ME AT
FEDERICO.DOTTA@MEDIASERVICE.NET

CONGRATULATIONS MARIO!

AUTHOR

FEDERICO DOTTA

REVIEW

MAURIZIO AGAZZINI

MARCO IVALDI

LICENSE