
SECCON 2015 x CEDEC CHALLENGE

ゲームクラッキング & チートチャレンジ

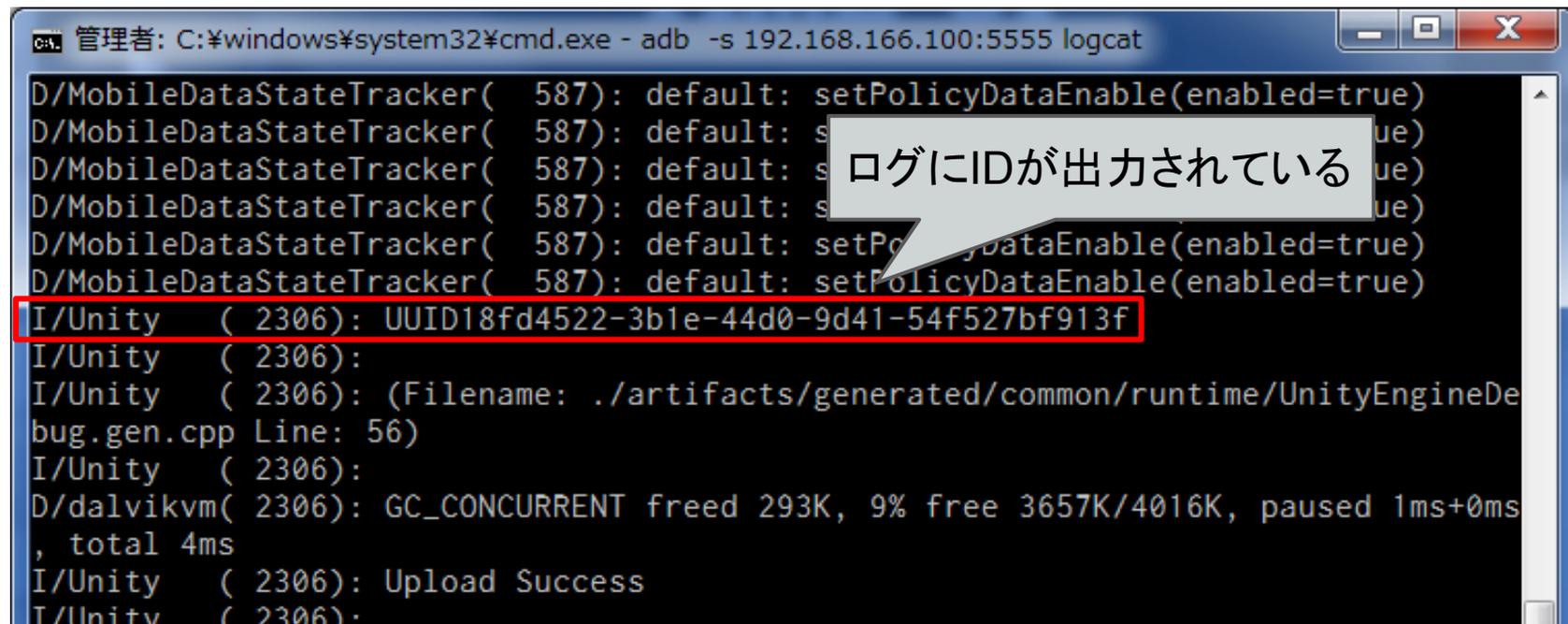
調査結果

vulscryptos

[1] sandbagの問題点

1. 不要なログ出力
 2. アイテム不正入手
 3. 任意のスコアデータの登録
-

[1]-1. 不要なログ出力 概要



```
管理者: C:\windows\system32\cmd.exe - adb -s 192.168.166.100:5555 logcat
D/MobileDataStateTracker( 587): default: setPolicyDataEnable(enabled=true)
D/MobileDataStateTracker( 587): default: s
D/MobileDataStateTracker( 587): default: s
D/MobileDataStateTracker( 587): default: s
D/MobileDataStateTracker( 587): default: setPolicyDataEnable(enabled=true)
D/MobileDataStateTracker( 587): default: setPolicyDataEnable(enabled=true)
I/Unity ( 2306): UUID18fd4522-3b1e-44d0-9d41-54f527bf913f
I/Unity ( 2306):
I/Unity ( 2306): (Filename: ./artifacts/generated/common/runtime/UnityEngineDe
bug.gen.cpp Line: 56)
I/Unity ( 2306):
D/dalvikvm( 2306): GC_CONCURRENT freed 293K, 9% free 3657K/4016K, paused 1ms+0ms
, total 4ms
I/Unity ( 2306): Upload Success
I/Unity ( 2306):
```

ログにIDが出力されている

[1]-1. 不要なログ出力 再現手法

難易度: 簡単

Android SDKの入ったPCと、
USBデバッグモードが有効になったAndroidを
USBケーブルで繋ぎ、
PCで `adb logcat` を実行

[1]-1. 不要なログ出力 影響度

今回の"UUID"は個人情報とは紐付かない、
スコア送信用のユーザ識別IDなので、特に影響はない

しかし、
ログ出力から個人情報・パスワードがダダ漏れになっていた
.....というケースも過去にあったため、細心の注意が必要

[1]-1. 不要なログ出力 対策

「ログ出力しても問題ない情報」と、
「ログ出力してはならない情報」とを吟味する

リリース時によく確認を！

[1] sandbagの問題点

1. 不要なログ出力
 2. アイテム不正入手
本来はプレイ回数に応じて入手できるボールが、
プレイ回数に関係なく入手できてしまう
 3. 任意のスコアデータの登録
-

[1]-2. アイテムの不正入手 問題点

```
# cat /data/data/net.sandbag2015/shared_prefs/net.sandbag2015.xml
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes'
```

```
<map>
```

```
  <int name="Screenmanager Is Fullscre
```

```
  <int name="Screenmanager Resolution
```

```
  <int name="scorePoint" value="305" />
```

```
  <string name="playerName">Charo</st
```

```
  <int name="Screenmanager Resolution Height" value="1920" />
```

```
  <int name="item" value="0" />
```

```
  <int name="playcount" value="1" />
```

```
  <string name="5.0.2">5bdc25aea3242a84c567ddb45a63b91e</string>
```

```
  <string name="uuid">18fd4522-3b1e-44d0-9d41-54f527bf913f</string>
```

```
</map>
```

アプリの設定値等を保存するファイル。
本来は自由にアクセスできないファイルだが、
改造(root化)した端末だと自由にアクセス可能

[1]-2. アイテムの不正入手 問題点

```
# cat /data/data/net.sandbag2015/shared_prefs/net.sandbag2015.xml
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
```

```
<map>
```

```
  <int name="Screenmanager Is Fullscreen mode" value="0" />
```

```
  <int name="Screenmanager Resolution Width" value="1024" />
```

```
  <int name="scorePoint" value="305" />
```

```
  <string name="playerName">Charo</string>
```

```
  <int name="Screenmanager Resolution Height" value="768" />
```

```
  <int name="item" value="0" />
```

```
  <int name="playcount" value="1" />
```

```
  <string name="5.0.2">5bdc25aea3242a84c567ddb45a63b91e</string>
```

```
  <string name="uuid">18fd4522-3b1e-44d0-9d41-54f527bf913f</string>
```

```
</map>
```

プレイ回数・入手アイテムが
平文でファイルに書いてある

[1]-2. アイテムの不正入手 再現手法

難易度: root化さえできれば簡単

1. 端末をroot化する("端末の型番 root"でGoogle検索)
 2. 以下のファイルを編集する
`/data/data/net.sandbag2015/shared_prefs/net.sandbag2015.xml`
-

[1]-2. アイテムの不正入手 影響度

- アイテムの入手難易度が変わるため、ゲームバランスが崩壊
 - ゲームの寿命を著しく縮める

 - 仮に、個人情報平文で保存されていた場合、悪意のあるアプリに読み取られて外部に送信される恐れがある
-

[1]-2. アイテムの不正入手 対策

- プレイ回数・入手アイテムはサーバ側で管理
 - 見せたくない・改ざんされたくない情報を端末に置かない
 - 効果は大きい一方で、
ゲームプレイにインターネット環境が必須になる、
といったデメリットも
 - 端末に置く場合、暗号化して保存する
-

[1] sandbagの問題点

1. 不要なログ出力
2. アイテム不正入手
3. 任意のスコアデータの登録
実際にはプレイしていないのに、
任意のスコアデータを登録できる

Hall of Fame	
1 Charo	2147483647
2 TestPlayer	2000000000
3 na_ga	20000
4 na_ga	20000
5 oki	20000
6 ???	5200
7 aaaa	3300
8 na_ga	200
9 Player 01	10
10 Player 02	9

BACK

[1]-3. 任意のスコアデータの登録 問題点

No.	Time	Source	Destination	Protocol	Length	Info
98	13.2454780	192.168.0.5	210.140.172.74	HTTP	443	POST /score/ranking/ HTTP/1.1 (application/json)
101	13.3612680	210.140.172.74	192.168.0.5	HTTP	453	HTTP/1.1 200 OK (text/html)

No.	Time	Source	Destination	Protocol	Length	Info
98	13.2454780	192.168.0.5	210.140.172.74	HTTP	443	POST /score/ranking/ HTTP/1.1 (application/json)
101	13.3612680	210.140.172.74	192.168.0.5	HTTP	453	HTTP/1.1 200 OK (text/html)

Frame 98: 443 bytes on wire (3544 bits), 443 bytes captured (3544 bits) on interface 0
Ethernet II, Src: Micro-ST_33:b6:d9 (6c162:6d33:b6:d9), Dst: a2:12:42:ba:38:8f (a2:12:42:ba:38:8f)
Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 210.140.172.74 (210.140.172.74)
Transmission Control Protocol, Src Port: 56680 (56680), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 389
Hypertext Transfer Protocol

JavaScript Object Notation: application/json
Object
Member Key: "uid"
Member Key: "name"
Member Key: "point"

送信したスコアデータ

```
0150 07 3a 20 07 7a 09 70 0d 0a 45 0f 0e 74 05 0e 74
0160 2d 4c 65 6e 67 74 68 3a 20 37 36 0d 0a 0d 0a 7b
0170 22 75 75 69 64 22 3a 22 31 38 66 64 34 35 32 32
0180 2d 33 62 31 65 2d 3a 34 64 30 2d 39 64 34 31 2d
0190 35 34 66 35 32 37 62 66 39 31 33 66 22 2c 22 6e
01a0 61 6d 65 22 3a 22 43 68 61 72 6f 22 2c 22 70 6f
01b0 69 6e 74 22 3a 32 31 30 30 30 7d
```

スコアの送信先

Intel Core i7-2670QM
CPU core #1: 2704 MHz
CPU core #2: 2704 MHz
CPU core #3: 2704 MHz
CPU core #4: 2704 MHz
CPU core #5: 2704 MHz
CPU core #6: 2704 MHz
CPU core #7: 2704 MHz
CPU core #8: 2704 MHz
CPU core #9: 2704 MHz
CPU core #10: 2704 MHz
CPU core #11: 2704 MHz
CPU core #12: 2704 MHz
CPU core #13: 2704 MHz
CPU core #14: 2704 MHz
CPU core #15: 2704 MHz
CPU core #16: 2704 MHz
CPU core #17: 2704 MHz
CPU core #18: 2704 MHz
CPU core #19: 2704 MHz
CPU core #20: 2704 MHz
CPU core #21: 2704 MHz
CPU core #22: 2704 MHz
CPU core #23: 2704 MHz
CPU core #24: 2704 MHz
CPU core #25: 2704 MHz
CPU core #26: 2704 MHz
CPU core #27: 2704 MHz
CPU core #28: 2704 MHz
CPU core #29: 2704 MHz
CPU core #30: 2704 MHz
CPU core #31: 2704 MHz
CPU core #32: 2704 MHz
CPU core #33: 2704 MHz
CPU core #34: 2704 MHz
CPU core #35: 2704 MHz
CPU core #36: 2704 MHz
CPU core #37: 2704 MHz
CPU core #38: 2704 MHz
CPU core #39: 2704 MHz
CPU core #40: 2704 MHz
CPU core #41: 2704 MHz
CPU core #42: 2704 MHz
CPU core #43: 2704 MHz
CPU core #44: 2704 MHz
CPU core #45: 2704 MHz
CPU core #46: 2704 MHz
CPU core #47: 2704 MHz
CPU core #48: 2704 MHz
CPU core #49: 2704 MHz
CPU core #50: 2704 MHz

NVIDIA GeForce GTX
GPU core: 720 MHz
GPU memory: 1247 MB
GPU power: 1500 mW
GPU core: 80.0 °C

15 ぽーせんと!

20:45

DISK
C:\ 37.3 GB/232.8 GB free
D:\ 76.3 GB/232.8 GB free

[1]-3. 任意のスコアデータの登録 再現手法

<http://api.sandbag2015.net/score/ranking/> に

`{"uuid":"任意の文字列", "name":"任意の名前", "point":整数}`

をPOSTするだけ

[1]-3. 任意のスコアデータの登録 影響度

ランキングに不正なスコアが登録される



Hall of Fame	
1 Charo	2147483647
2 TestPlayer	2000000000
3 na_ga	20000
4 na_ga	20000
5 oki	20000
6 ???	5200
7 aaaa	3300
8 na_ga	200
9 Player 01	10
10 Player 02	9

BACK

[1]-3. 任意のスコアデータの登録 対策

- httpsにする
 - 「通信内容の取得」を難しくする
 - が、のぞき見用の環境を整えればhttpsも取得可能
ということで効果はそこそこ

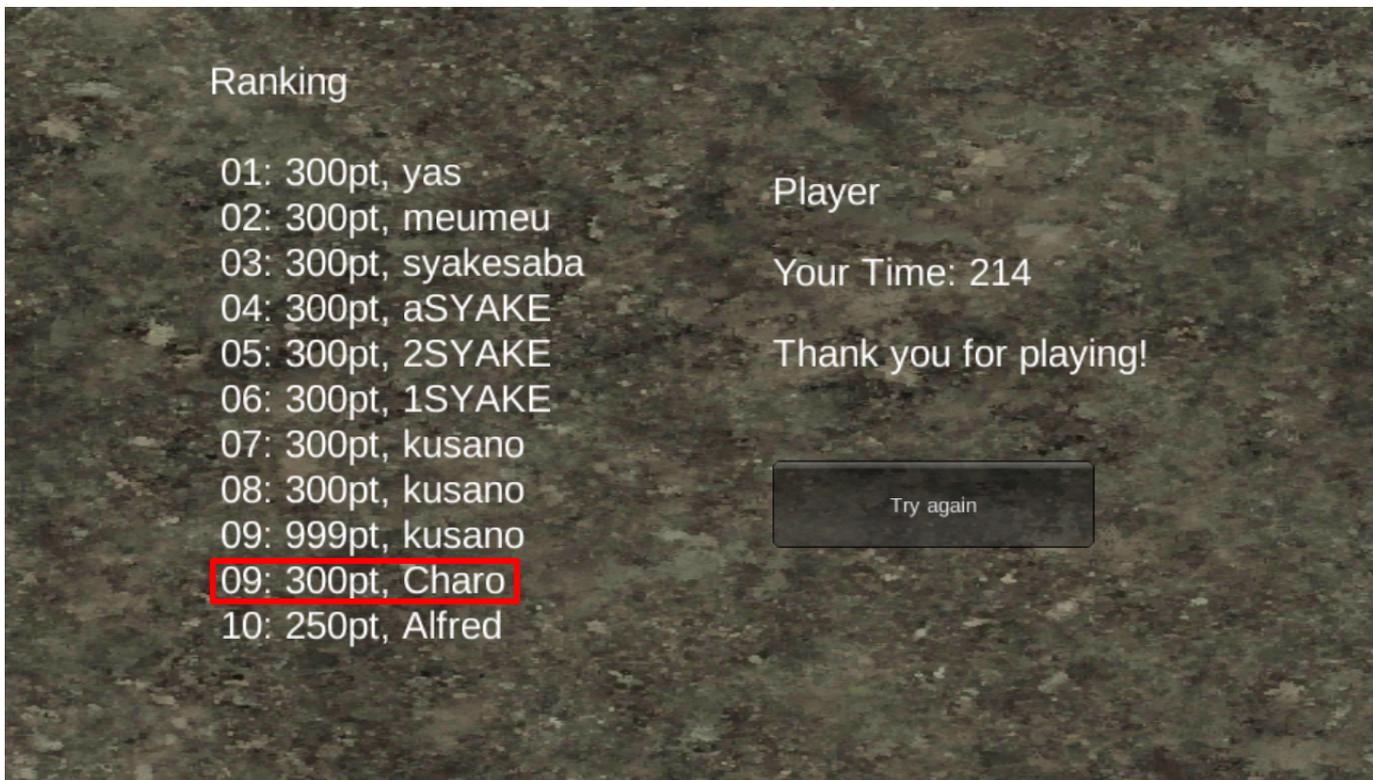
[1]-3. 任意のスコアデータの登録 対策

- スコアデータを暗号化して送信する
 - のぞき見されても、「どんなデータを送ればいいのか」がわからないようにする
 - 突破にはアプリの解析が必要になるため、コードの難読化と併せると効果大
-

[2] SUNIDRAの問題点

1. 任意のスコアデータの登録
 2. メモリの書き換えによるチート
 3. サーバ側のデータ検証不備
-

[2]-1. 任意のスコアデータの登録 概要



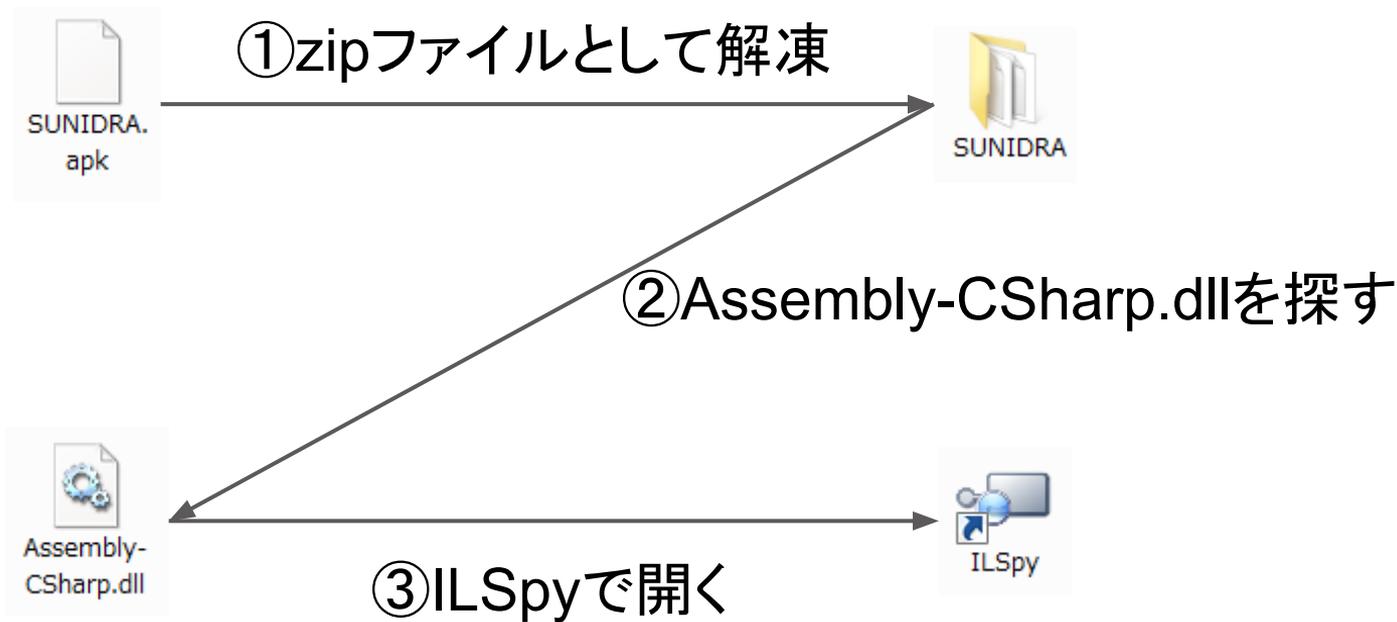
[2]-1. 任意のスコアデータの登録 影響度

ランキングに不正なスコアが登録される

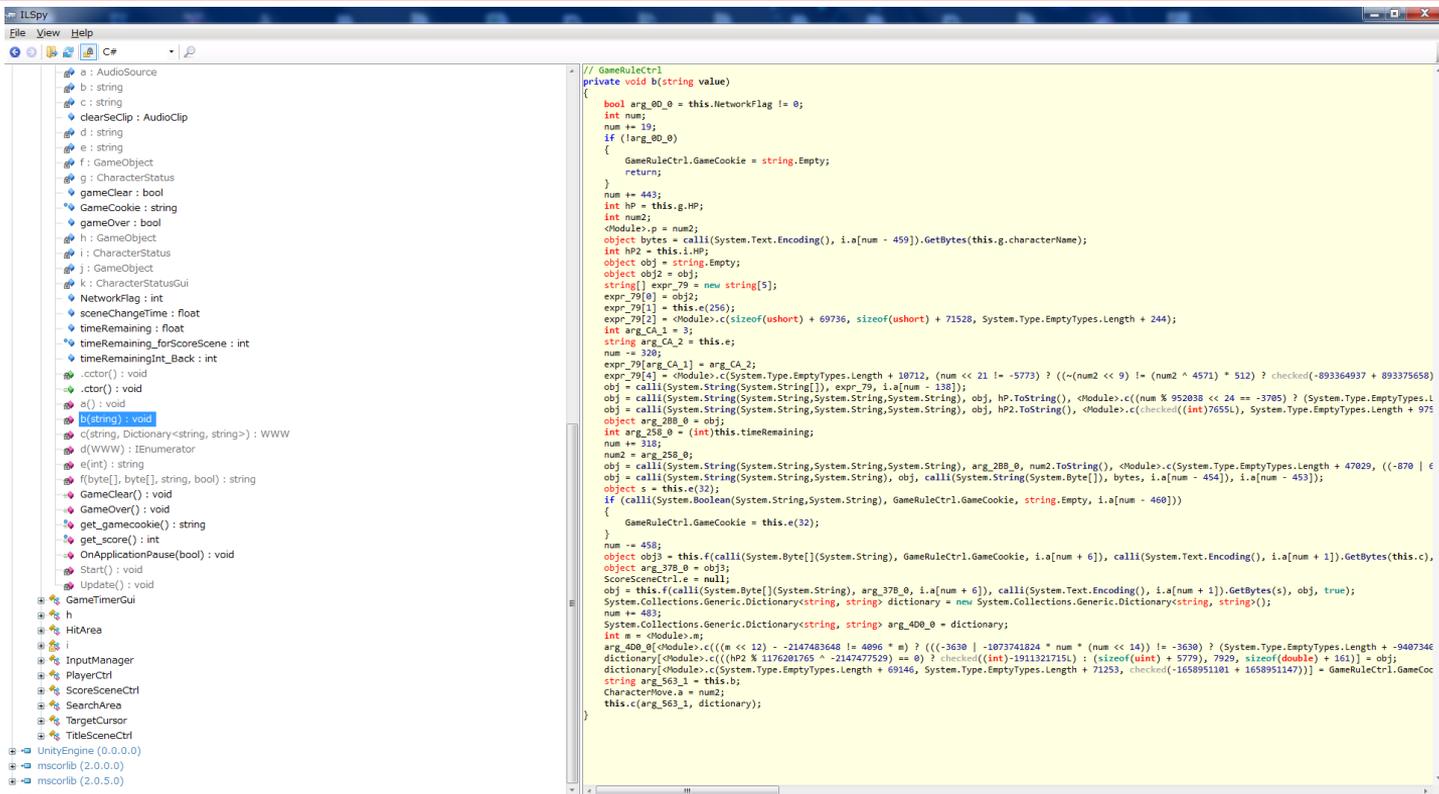
[2]-1. 任意のスコアデータの登録 再現手法

アプリがどうやって送信データを作っているかを
リバースエンジニアリングで解析し、
アプリと同じ手順で送信データを作って送る

[2]-1. 任意のスコアデータの登録 再現手法



[2]-1. 任意のスコアデータの登録 再現手法



[2]-1. 任意のスコアデータの登録 再現手法

Screenshot of Microsoft Visual Studio (VS) showing the source code for a program named SUNIDRA. The code is in C# and defines a class with static variables and a method to build a payload for an HTTP request.

```
static int sizeof_guid = System.Runtime.InteropServices.Marshal.SizeOf(System.Guid.Empty);
static int hp = 100;
static int hp2 = 100;
static int line = 300;
static string playerName = "Ocharo";
static string GameCookie = "";

static readonly string URL = "https://cscac2015.seccon.jp/cscac2015/GameCtrl/";

static void Func() {
    GameCookie = SendRequest("Start");
    Console.WriteLine("GameCookie: {0}", GameCookie);

    hp2 = 0;
    line = 300;
    Console.WriteLine("GameClear");
}

static string SendRequest(string status) {
    string data = string.Format("{0}{1}{2}", Uri.EscapeDataString(status), Uri.EscapeDataString(BuildPayload()), Uri.EscapeDataString(GameCookie));
    HttpRequest request = (HttpRequest)(HttpWebRequest.Create(URL));
    request.Method = "POST";
    request.ContentType = "application/x-www-form-urlencoded";
    request.ContentLength = data.Length;
    using(Stream stream = request.GetRequestStream()) {
        using(StreamWriter writer = new StreamWriter(stream)) {
            writer.Write(data);
            writer.Flush();
            writer.Close();
        }
    }

    HttpResponseMessage response = (HttpResponse)request.GetResponse();
    using(Stream stream = response.GetResponseStream()) {
        using(StreamReader reader = new StreamReader(stream)) {
            return reader.ReadToEnd();
        }
    }
}

static string BuildPayload() {
```

The error console at the bottom shows a warning: CS0219 変数 'num' は割り当てられていますが、その値は使用されていません。

エラー一覧	プロジェクト	ファイル	行
CS0219 変数 'num' は割り当てられていますが、その値は使用されていません。	SUNIDRA	Program.cs	91

System Information (bottom right):

- Intel Core i7-2670QM
- CPU Core #1: 2728 MHz
- CPU Core #2: 2728 MHz
- CPU Core #3: 2728 MHz
- CPU Core #4: 2728 MHz
- CPU Core #5: 28.0 MHz
- CPU Core #6: 48.0 MHz
- CPU Core #7: 48.0 MHz
- CPU Core #8: 48.0 MHz
- NVIDIA GeForce GTX
- GPU Core: 51 MHz
- GPU Memory: 1.52 GB
- GPU Shader: 101 MHz
- GPU Core: 49.0 MHz

DISK: C:\ 37.3 GB/237.8 GB free, D:\ 26.3 GB/237.8 GB free

[2]-1. 任意のスコアデータの登録 問題点

httpsが使用され、送信データは暗号化され、
コードは難読化されていた

が.....

解析者が根負けするレベルの難読化ではなかった

[2]-1. 任意のスコアデータの登録 対策

- 解析の手間を増やす
 - イタチごっこなのでキリが無いが、
「チートの手間 > チートの恩恵」になれば開発側の勝ち
 - 個人的には.....
送信データの生成処理が
Javaプラグイン・Android NDKプラグインで
実装されていたら、解析を諦めていたかも？
-

[2] SUNIDRAの問題点

1. 任意のスコアデータの登録
 2. メモリ書き換えによるチート
フラグや変数、さらにはコードの内容も書き換え、
アプリを自分仕様にする
 3. サーバ側のデータ検証不備
-

[2]-2. メモリ書き換えによるチート 概要

The screenshot shows a game window titled "BlueStacks App Player for Windows (beta-1)" running a game. A character is fighting a red dragon in a grassy field. The dragon's name "Dragon" is visible above it. A green horizontal line is drawn across the dragon's body. In the foreground, a Cheat Engine 6.4 window is open, displaying a memory scan for "Dragon". The scan results show a list of addresses and values, with the address C0B15E2C highlighted. The value is 100. The Cheat Engine window also shows various options and settings.

Address	Value	Prev
833D0438	100	100
833D069C	100	100
833D09AC	100	100
C0B15E2C	100	100
13D85A148	100	100
165246C9C	100	100
165246F00	100	100
175A5106C	100	100
175A51AD0	100	100
175A51DE0	100	100
17817780C	100	100
178177D70	100	100
181759BD4	100	100

The Cheat Engine window also shows the following options:

- Scan Type: Exact Value
- Value Type: 4 Bytes
- Memory Scan Options: Unrandomizer, Enable Speedhack
- Start: 0000000000000000
- Stop: 7FFFFFFFFFFFFFFF
- Executable:
- Writable:
- CopyOrWrite:
- Fast Scan: 4
- Alignment: Last Digits
- Pause the game while scanning:

The system tray shows the following information:

- TwitterDeck widget
- System information: Intel Core i7-3670QM, CPU Core #1: 2794 MHz, GPU Core #1: 2794 MHz, GPU Core #2: 2794 MHz, GPU Core #3: 2794 MHz, GPU Core #4: 2794 MHz, GPU Core #5: 2794 MHz, GPU Core #6: 2794 MHz, GPU Core #7: 2794 MHz, GPU Core #8: 2794 MHz, GPU Core #9: 2794 MHz, GPU Core #10: 2794 MHz, GPU Core #11: 2794 MHz, GPU Core #12: 2794 MHz, GPU Core #13: 2794 MHz, GPU Core #14: 2794 MHz, GPU Core #15: 2794 MHz, GPU Core #16: 2794 MHz, GPU Core #17: 2794 MHz, GPU Core #18: 2794 MHz, GPU Core #19: 2794 MHz, GPU Core #20: 2794 MHz, GPU Core #21: 2794 MHz, GPU Core #22: 2794 MHz, GPU Core #23: 2794 MHz, GPU Core #24: 2794 MHz, GPU Core #25: 2794 MHz, GPU Core #26: 2794 MHz, GPU Core #27: 2794 MHz, GPU Core #28: 2794 MHz, GPU Core #29: 2794 MHz, GPU Core #30: 2794 MHz, GPU Core #31: 2794 MHz, GPU Core #32: 2794 MHz, GPU Core #33: 2794 MHz, GPU Core #34: 2794 MHz, GPU Core #35: 2794 MHz, GPU Core #36: 2794 MHz, GPU Core #37: 2794 MHz, GPU Core #38: 2794 MHz, GPU Core #39: 2794 MHz, GPU Core #40: 2794 MHz, GPU Core #41: 2794 MHz, GPU Core #42: 2794 MHz, GPU Core #43: 2794 MHz, GPU Core #44: 2794 MHz, GPU Core #45: 2794 MHz, GPU Core #46: 2794 MHz, GPU Core #47: 2794 MHz, GPU Core #48: 2794 MHz, GPU Core #49: 2794 MHz, GPU Core #50: 2794 MHz, GPU Core #51: 2794 MHz, GPU Core #52: 2794 MHz, GPU Core #53: 2794 MHz, GPU Core #54: 2794 MHz, GPU Core #55: 2794 MHz, GPU Core #56: 2794 MHz, GPU Core #57: 2794 MHz, GPU Core #58: 2794 MHz, GPU Core #59: 2794 MHz, GPU Core #60: 2794 MHz, GPU Core #61: 2794 MHz, GPU Core #62: 2794 MHz, GPU Core #63: 2794 MHz, GPU Core #64: 2794 MHz, GPU Core #65: 2794 MHz, GPU Core #66: 2794 MHz, GPU Core #67: 2794 MHz, GPU Core #68: 2794 MHz, GPU Core #69: 2794 MHz, GPU Core #70: 2794 MHz, GPU Core #71: 2794 MHz, GPU Core #72: 2794 MHz, GPU Core #73: 2794 MHz, GPU Core #74: 2794 MHz, GPU Core #75: 2794 MHz, GPU Core #76: 2794 MHz, GPU Core #77: 2794 MHz, GPU Core #78: 2794 MHz, GPU Core #79: 2794 MHz, GPU Core #80: 2794 MHz, GPU Core #81: 2794 MHz, GPU Core #82: 2794 MHz, GPU Core #83: 2794 MHz, GPU Core #84: 2794 MHz, GPU Core #85: 2794 MHz, GPU Core #86: 2794 MHz, GPU Core #87: 2794 MHz, GPU Core #88: 2794 MHz, GPU Core #89: 2794 MHz, GPU Core #90: 2794 MHz, GPU Core #91: 2794 MHz, GPU Core #92: 2794 MHz, GPU Core #93: 2794 MHz, GPU Core #94: 2794 MHz, GPU Core #95: 2794 MHz, GPU Core #96: 2794 MHz, GPU Core #97: 2794 MHz, GPU Core #98: 2794 MHz, GPU Core #99: 2794 MHz, GPU Core #100: 2794 MHz
- System information: NVIDIA GeForce GTX, GPU Core: 202 MHz, GPU Memory: 328 MB, GPU Shader: 402 MHz, GPU Core: 49.0 MHz
- System information: 35.4 GB/232.8 GB free
- System information: 22.9 GB/232.8 GB free

[2]-2. メモリ書き換えによるチート 再現手法

難易度: 中

1. メモリ書き換えできる環境を作る
root化したAndroidにプロセスメモリエディタを導入
または、PCにAndroidエミュレータとプロセスメモリエディタを導入
(今回はBlueStacksとCheat Engine 6.4を使用)
 2. ゲーム中の出来事(被弾・回復等)に応じて変数の値が変動(被弾なら減少、回復なら増加)することを利用し、
目当ての変数がメモリ上のどこにあるのかを絞り込む
 3. メモリを任意の値で書き換える
-

[2]-2. メモリ書き換えによるチート 影響度

- 根気さえあれば何でもできる
 - 攻撃されてもHPが減らない
 - 一撃で敵を倒せる
 - キャラクターの移動速度を速くする etc.
 - ゲームバランスの崩壊
-

[2]-2. メモリ書き換えによるチート 対策

- root化された端末・エミュレータ環境ではプレイできないようにする
 - メモリ書き換えできる環境を排除する
 - エミュレータ環境かどうかは、特定のパッケージが入っているかどうかで検出可能
(例) com.genymotion, com.bluestacks.appfinder
-

[2]-2. メモリ書き換えによるチート 対策

- 値をそのまま保持するのではなく、ビット反転する等加工して保持する
 - 目当ての変数がメモリ上のどこにあるのかを探しにくくする(かなり効果的)
 - getter/setterをうまく使えば実装も簡単
-

[2]-2. メモリ書き換えによるチート 対策

//Before:

```
public int HP;
```

画面上のHP

100

メモリ上の値

100

画面上の値=メモリ上の値
なので、探しやすい

```
void Attacked(int damage){
```

```
    this.HP -= damage;
```

```
}
```

[2]-2. メモリ書き換えによるチート 対策

```
//After:  
private int _HP;  
public int HP{  
    get{return ~_HP;}  
    set{_HP = ~value;}  
};  
  
void Attacked(int damage){  
    this.HP -= damage;  
}
```

画面上のHP

100

メモリ上の値

-101

画面上の値≠メモリ上の値
なので、かなり探しにくい

ロジック部分はそのままでOKなので、
ソースコードのメンテナンス性を損なわない

[2] SUNIDRAの問題点

1. 任意のスコアデータの登録
 2. メモリの書き換えによるチート
 3. サーバ側のデータ検証不備
-

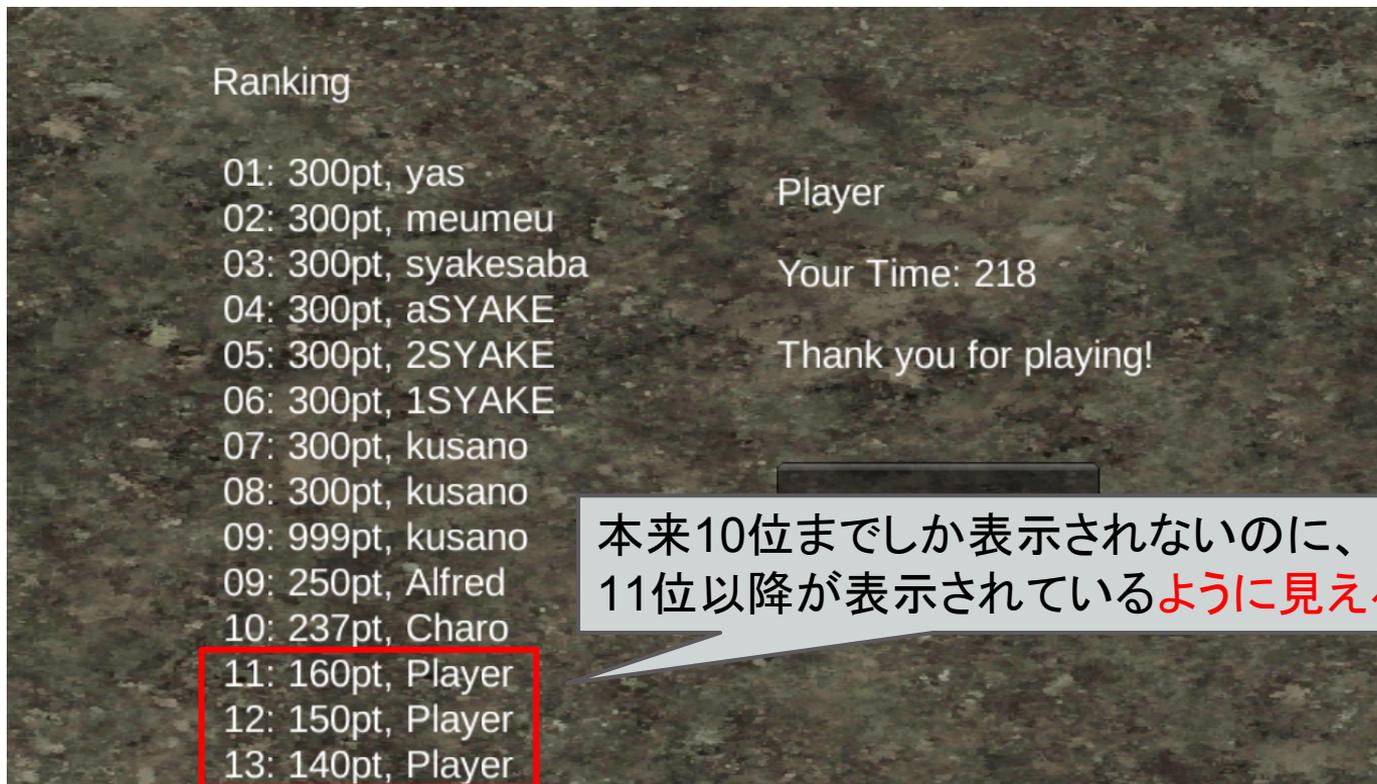
[2]-3. サーバ側のデータ検証不備 概要

Ranking

01: 300pt, yas	Player
02: 300pt, meumeu	
03: 300pt, syakesaba	Your Time: 218
04: 300pt, aSYAKE	
05: 300pt, 2SYAKE	Thank you for playing!
06: 300pt, 1SYAKE	
07: 300pt, kusano	
08: 300pt, kusano	
09: 999pt, kusano	
09: 250pt, Alfred	
10: 237pt, Charo	
11: 160pt, Player	
12: 150pt, Player	
13: 140pt, Player	

本来10位までしか表示されないのに、11位以降が表示されている

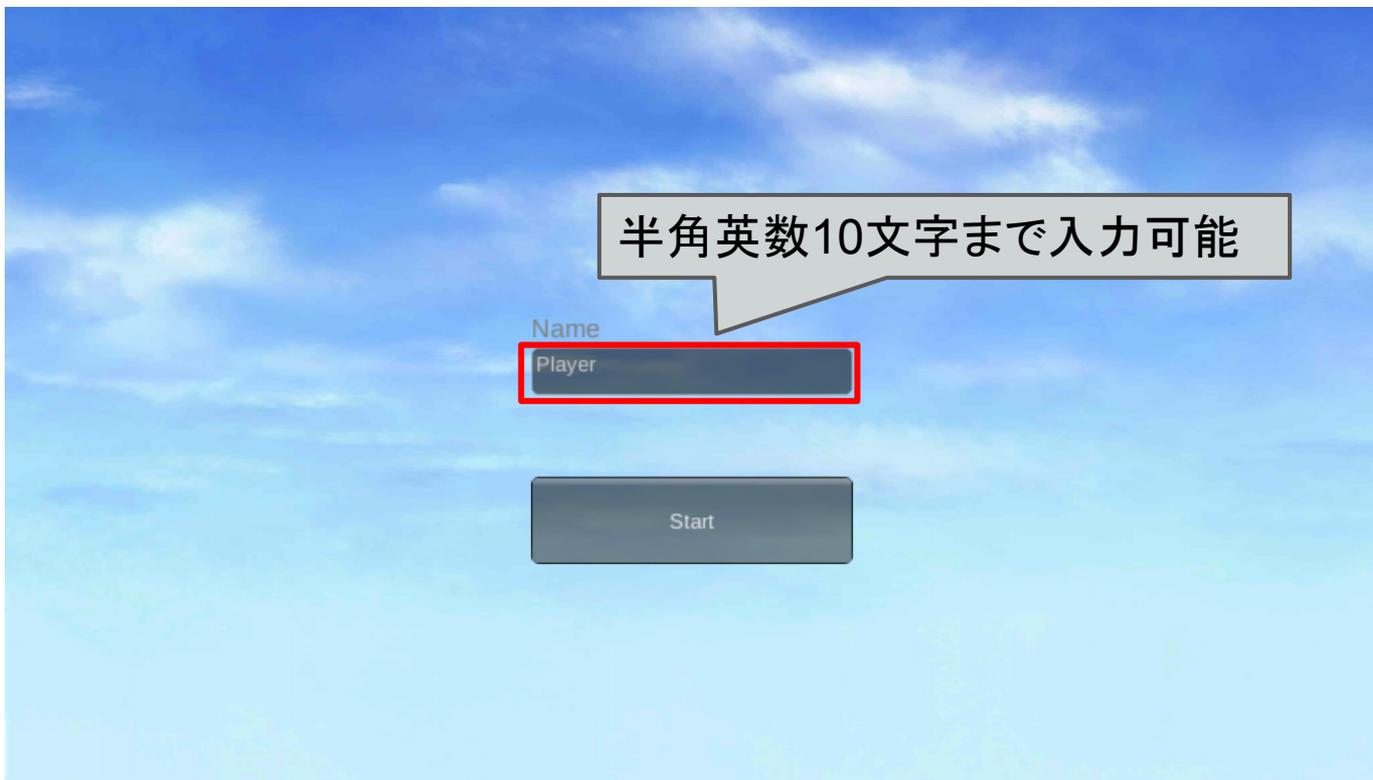
[2]-3. サーバ側のデータ検証不備 概要



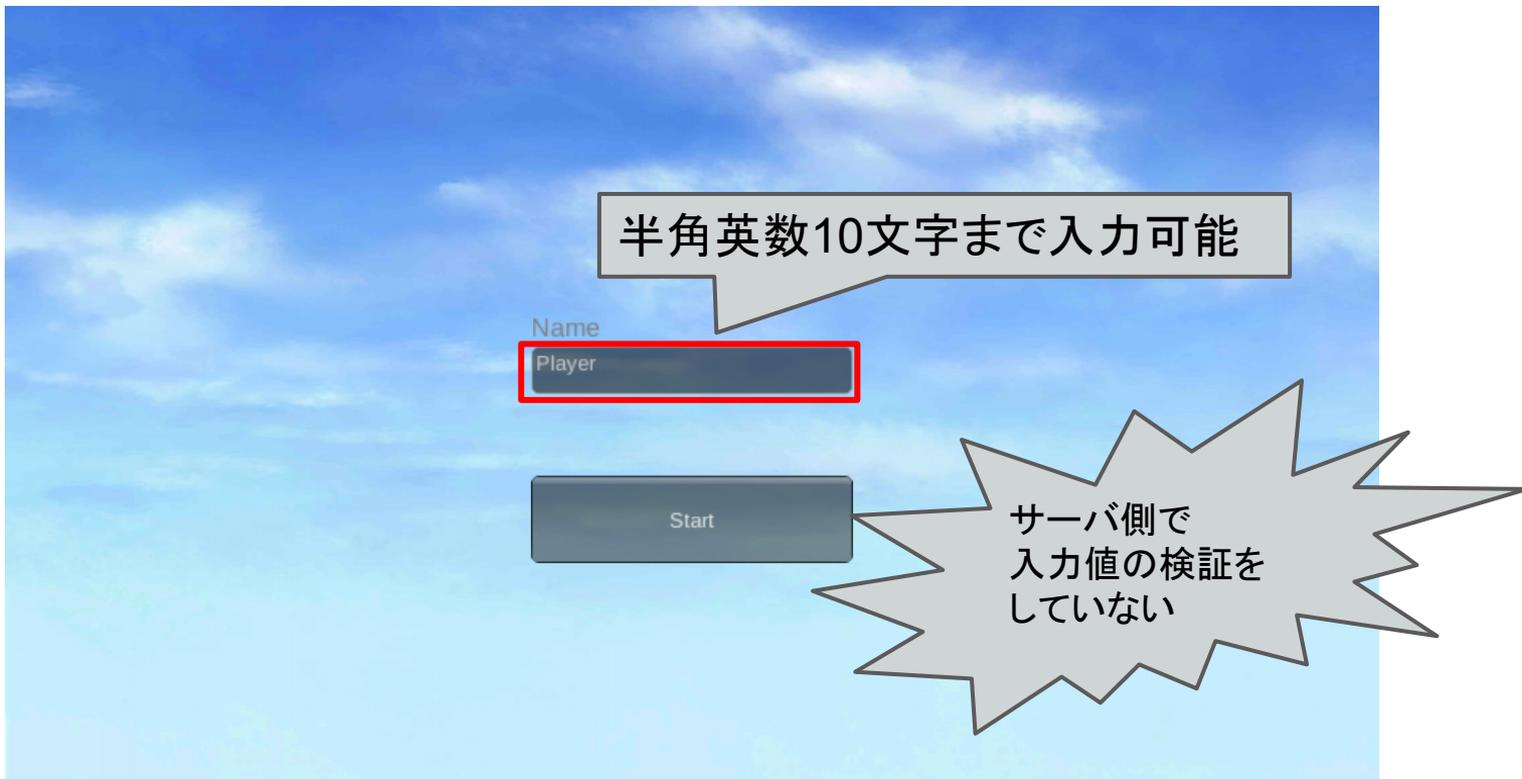
[2]-3. サーバ側のデータ検証不備 再現手法

[2]-1. 任意のスコアデータの登録のテクニックで、
"Charo\n 11: 160pt, Player\n 12: 150pt,..."
というプレイヤー名でランクインさせる

[2]-3. サーバ側のデータ検証不備 問題点



[2]-3. サーバ側のデータ検証不備 問題点



[2]-3. サーバ側のデータ検証不備 影響度

- 偽のランキングが作れる
 - サーバ側のプログラムに脆弱性があった場合、サーバを乗っ取られたり、サーバからデータを盗まれたりする恐れがある
-

[2]-3. サーバ側のデータ検証不備 対策

- 送られてきたデータを無条件に信用しない
 - クライアント側で検証するだけでは不十分
 - データ長・文字種等、サーバ側でも検証をした上で登録
-