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

チームurandom

Sandbag - 通信内容の漏洩 概要

- API通信が平文(HTTP)
- 傍受されれば識別情報(UUID)等が漏れる
  - 信頼できないWi-Fiアクセスポイントに繋ぐと？
  - ユーザのなりすましに利用できる

```
POST /score/ranking/ HTTP/1.1
X-Unity-Version: 5.0.2f1
Content-Type: application/json; charset=UTF-8
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.1.1; Google Nexus 4 - 4.1.1 - A
Host: api.sandbag2015.net
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 76

{"uuid":"6c781665-21ac-4a50-bf29-e35709e3624c","name":"meumeu","point":4950}
```

Sandbagの平文通信を傍受したもの

Sandbag - 通信内容の漏洩 評価と対策

- リスク評価:**参考情報**
  - 現時点の機能はゲームとランキングのみ
  - 機能が増えるとなりすましの利益が生まれる

- 可能なら、HTTPSで通信する
- 無理なら、ユーザーに平文通信を告知
  - 信頼できないWi-Fiアクセスポイントでは通信しない

Sandbag - アイテム不正解放 概要

- 解放状況がセーブデータに記録されている
  - 平文のXMLの為、解析と改ざんが容易
  - 書き換える事でアイテムを不正に解放可能

Sandbag - アイテム不正解放 評価と対策

- リスク評価: 中リスク
  - チートは容易, ランキングに間接的影響

- セーブデータを暗号化
- `android:allowBackup=false`
  - 書き換えにパッチかrootが必要になる
- サーバーでセーブデータを管理

Sandbag - スコア書き換え 概要

- サーバーに報告するスコアを書き換える
  - 平文JSONでスコア送っているので書き換えが容易
  - 任意のスコアでランクインできる

Sandbag - スコア書き換え 評価と対策

- リスク評価: **中リスク**
  - チートは比較的容易, ランキングに直接影響
  - 別途PCやソフト等が必要

- API通信の内容を暗号化する
- 得点推移などをサーバーに送信させる

SUNIDRA - 無敵状態チート 概要

- 時間・体力が減らないようにできる
  - 理論値最大スコアでランクイン可能

SUNIDRA - 無敵状態チート 手法 1/4

- 通信はHTTPS + 独自の暗号化
  - 暗号の処理を解析する必要がある
- ゲーム本体には難読化 + 改ざん検知
  - 解析が比較的難しく、パッチを当てられない
- どうするか？

SUNIDRA - 無敵状態チート 手法 2/4

- 改ざん検知処理にパッチを当てて無力化
  - 自由にパッチを当てられるようになる
- 体力・時間の減算処理にパッチを当てる

SUNIDRA - 無敵状態チート 手法 3/4

- ILSpyで逆アセンブル + デコンパイル
  - IL(≒機械語)とC#のソースコードを得る
- SHA1, AESなどの処理を探す
  - 改ざん検知などでよく使われる
  - 自身のSHA1ハッシュを計算していた
  - 常時正しい値になるようにパッチ

```
object obj3 = new System.Security.Cryptography.SHA1CryptoServiceProvider();
object obj4 = calli(System.Byte[](System.Byte[]), obj3, obj2, i.a[num + 12]);
TitleSceneCtrl.systemcode = calli(System.String(System.Byte[]), obj4, i.a[num + 4]);
```

```
TitleSceneCtrl.systemcode = "g6J6KPwHXeLG+7aqoaUS+fvJDTo=";
```

改ざん検知部のパッチ適用前と適用後

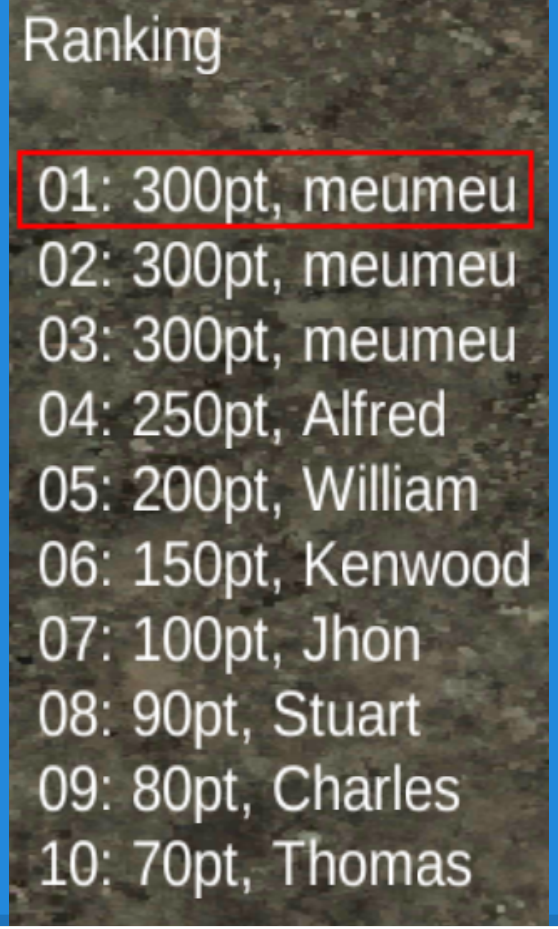
SUNIDRA - 無敵状態チート 手法 4/4

- 変数名からチートに使えそうな物を探す
  - “HP”, “timeRemaining” という変数がある
- それぞれ減算している場所がある
  - 減算処理を無効化

```
ldfld float32 GameRuleCtrl::timeRemaining
call float32 [UnityEngine]UnityEngine.Time::get_deltaTime()
sub
stfld float32 GameRuleCtrl::timeRemaining
```

```
ldfld float32 GameRuleCtrl::timeRemaining
ldc.r4 0.0
sub
stfld float32 GameRuleCtrl::timeRemaining
```

時間減算処理のパッチ前とパッチ後



理論値最大スコアでランクインした様子

SUNIDRA - 無敵状態チート 評価

- リスク評価: 中リスク
- 解析は比較的難しい
- パッチを当ててしまえばroot等不要
  - ただプレーするだけでチートできる
- 全体ランキングに直接影響

SUNIDRA - 無敵状態チート 対策 1/4

- 改ざん検知の強化
  - 複数箇所に改ざん検知処理を置く
  - 関係無い処理の中に隠す

SUNIDRA - 無敵状態チート 対策 2/4

- デコンパイルできないようにする
  - 特定の命令パターンを入れると
    デコンパイル出来なくなる
  - IL(≒機械語)しか得られなくなるので
    解析の手間が増す
  - あまり知られていない手法

```
public static void Main(string[] args)
{
  ((IGreeter) new Test()).Greet();
}
```

```
public static void Main(string[] args)
{
  // ISSUE: unable to decompile the method.
}
```

デコンパイル対策前と対策後

SUNIDRA - 無敵状態チート 対策 3/4

- 難読化を強化する
  - “HP”, “timeRemaining”などのヒントを残さない
  - 変数名をなるべく無意味なものに変換
- 難読化はカジュアルハック対策
  - 解析の手間がやや増す程度
  - 難読化の種類によっては自動で解除可能
  - 無意味な演算はLLVMによる最適化等で除去可能

```
if ((4294962356u & 4294958788u + ((uint)num >> 26)) != 0u)
{
    if (1 / 32 == 1796913459)
    {
        arg_3B9_0 = (int)-1888521640.0;
    }
    else if ((num % 22 / 647 ^ 1073742750) != 0)
    {
        arg_3B9_0 = System.Type.EmptyTypes.Length + 17068;
    }
    else
    {
        num2 = CharacterMove.a;
        if ((num2 | -2151 | 2798) != -1)
        {
            if (((1 - (1 | 1 / 2428114)) % 8 | -793) != -793)
            {
                int num3 = checked((int)2132832988L);
                arg_3B9_0 = num3;
            }
            else
            {
                arg_3B9_0 = System.Type.EmptyTypes.Length + -1962116309;
            }
        }
        else
        {
            arg_3B9_0 = checked(-1080575834 + -406099384);
        }
    }
}
else
{
    arg_3B9_0 = -462640917 * 2;
}
```

無意味な演算をLLVMで除去する前と後

```
if ((4294962356u & 4294958788u + ((uint)num >> 26)) != 0u)
{
    if (1 / 32 == 1796913459)
    {
        arg_3B9_0 = (int)-1888521640.0;
    }
    else if ((num % 22 / 647 ^ 1073742750) != 0)
    {
        arg_3B9_0 = System.Type.EmptyTypes.Length + 17068;
    }
    else
    {
        num2 = CharacterMove.a;
        if ((num2 | -2151 | 2798) != -1)
        {
            if (((1 - (1 | 1 / 2428114)) % 8 | -793) != -793)
            {
                int num3 = checked((int)2132832988L);
                arg_3B9_0 = num3;
            }
            else
            {
                arg_3B9_0 = System.Type.EmptyTypes.Length + -1962116309;
            }
        }
        else
        {
            arg_3B9_0 = checked(-1080575834 + -406099384);
        }
    }
}
else
{
    arg_3B9_0 = -462640917 * 2;
}
```

```
mov     edi, [rax+18h]
add     edi, 42ACh
```

無意味な演算をLLVMで除去する前と後

SUNIDRA - 無敵状態チート 対策 4/4

- 詳細な動作ログをサーバーに送信させる
  - 敵との交戦推移など
  - 人間によるBAN判断の助けにする
  - チーターも対策側も最後は人間

終わりに - チート対策の考え方

- 根本的な対策は無い
  - クライアントがビュアーに徹するなら別だが
- やれる事はある
  - 暗号化などでカジュアルハックを防ぐ
  - root検知や改ざん検知を多重的に施す
  - 追跡可能にして人間がBAN判断
  - チートのコストがチートの利益を上回るように