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

Harekaze

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

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
- 楽曲の再生時間のチェック
- ランキング登録時のスコアの検証
- ガチャ時のレースコンディションの修正
- ユーザ情報取得時の SQL インジェクションの修正
- UUID 等のユーザ入力値の検証
- session key 生成時の古い session の削除

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
- 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)
  - ⇨ 6 個以上のスキルの不正な装備ができなくなった

```
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)

- ゲームサーバへの HTTPS によるリクエスト時に、
  証明書の拇印をあらかじめ埋め込んだものと比較することで、
  証明書チェーンによらず証明書の正当性を確認するように修正を行った
- また、HTTPS が使われていないゲームサーバに接続しようとした場合、
  通信をキャンセルするように修正を行った
  - ⇨ 対象の端末にルート証明書をインストールし、
    証明書を偽造して通信の盗聴や改ざんを行う中間者攻撃が難しくなった

USB デバッグが有効化されているかチェック

- USB デバッグが有効化されているかチェックし、
  もし有効化されている場合にはゲームを開始できないように修正を行った
  - ⇨ USB 経由でのデバッグ (adb 等の使用) が難しくなった

エミュレータが使用されていないかチェック

- API の返り値や環境変数などを参照することで、それらの差異から Android エミュレータが使用されている場合に検知できるよう修正した
  - ⇨ Android エミュレータでゲームを遊んでいる場合には、検知しゲームを終了するようになった

root 化されていないかチェック

- Android ライブラリ scottyab/rootbeer を利用し、
  端末が root 化されていた場合に検知できるよう修正した
  - ⇨ root 化された端末でゲームを遊んでいる場合には、
    検知しゲームを終了するようになった

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

- 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

```
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

- 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

- 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`)

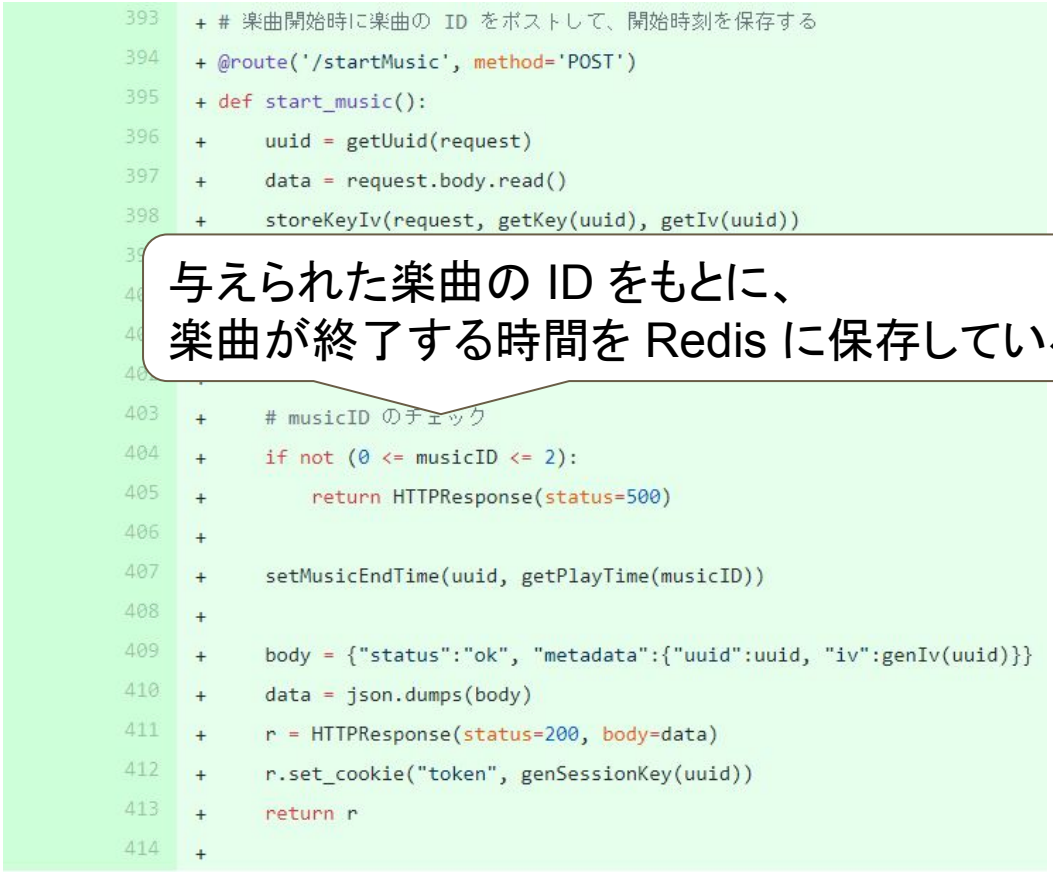
楽曲の再生時間のチェック

- 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`)

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

- 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

```
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 +
```

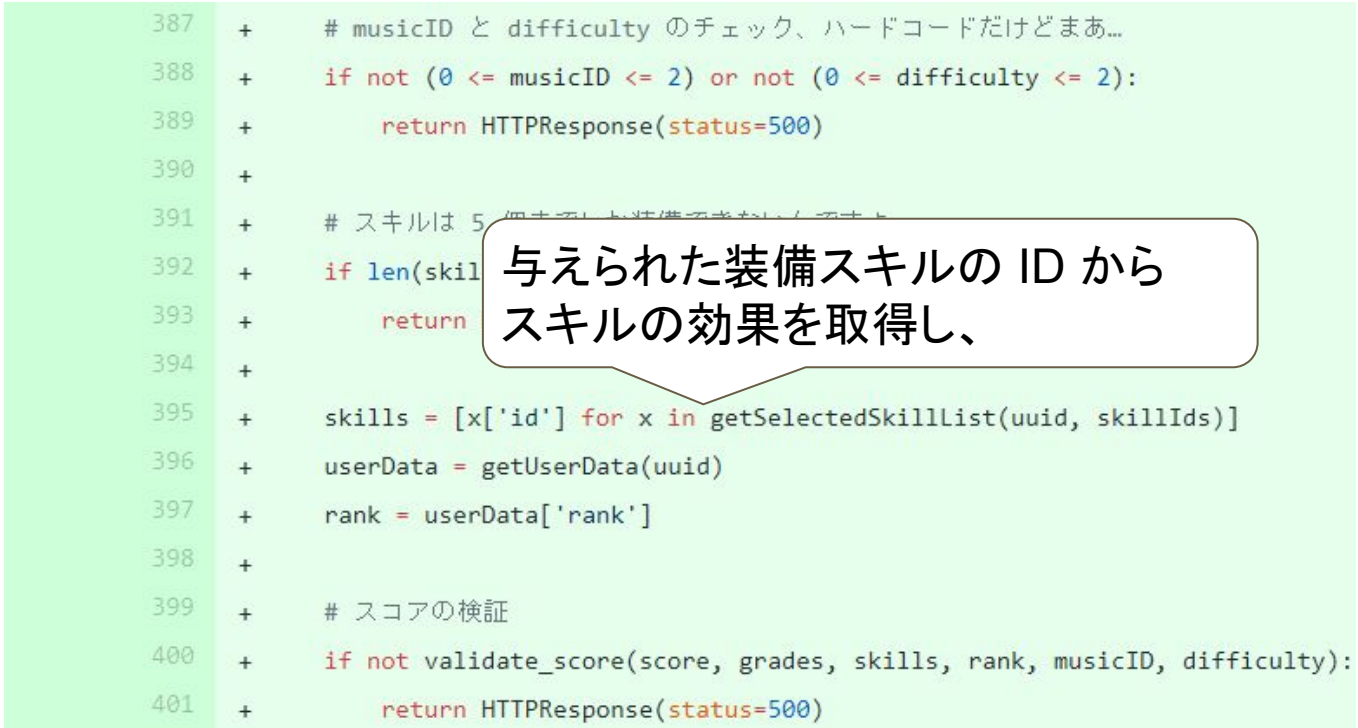
そのユーザが装備スキルを所持しているか、
重複していないかをチェックし、
スキルの情報を取得する関数

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`)



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 +     # スコアの検証
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`)

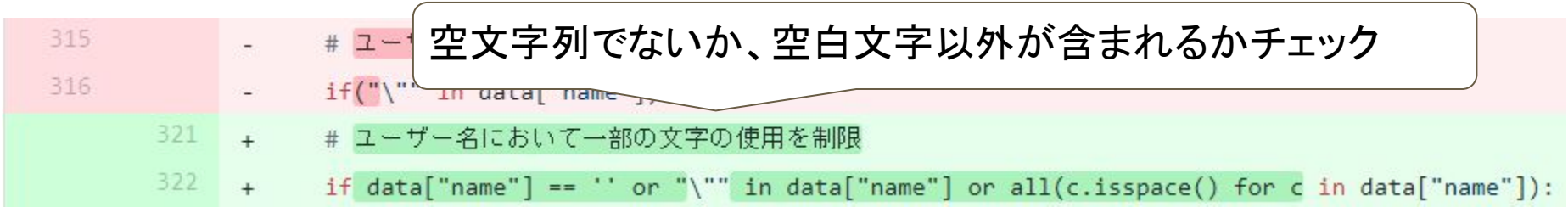
情報取得時の SQL インジェクションの修正

- 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)