Unity アプリの脆弱性と難読化

AG-TECH CORP.

目次

I. はじめに

Dotfuscator では、Unity アプリの難読化が可能です。Unity アプリは中間言語を実行ファイル内に持ち、実行時に機械語へ変換するような仕組みになっていますので、容易に復元が可能です。このドキュメントでは、Unity アプリがどのように保管され、どのようにソースレベルまで戻せてしまうかを説明します。また、その対策についてもご説明します。

(1) 公式マニュアルについて

Dotfuscator の公式マニュアルでは、Dotfuscator を使用する上での詳細が記述されています。Dotfuscator を使用する上で、より詳しい内容は、WEB ページをご覧ください。

https://www.agtech.co.jp/download/manual/preemptive/

II. 保護されていない Unity アプリの実態

Mono および .NET スクリプト バックエンドは、プロジェクト用に記述されたすべての C# スクリプトを、Assembly-CSharp.dll という DLL ライブラリにまとめます。 この Assembly-CSharp.dll には ゲーム開発者が書いたコードが保持されており、Dotfuscator の保護対象となります。保護されていないアプリをリバース エンジニアリングすると、どのように見えるかを確認する手順を紹介します。今回はターゲットとして「PC, Mac & Linux Standalone」にて作業を行います。iOS プラットフォームと Android プラットフォームには追加の構成を行う必要がありますが、全体の流れとしては変わりません。

※IL2CPP スクリプト バックエンドは、Dotfuscator への入力として使用できる出力を生成しません。

（1）ビルドと出力

Unity で Build Settings ウィンドウの［Build］ボタンを押して、出力ファイルを生成します。

（2）出力ファイルの確認

プロジェクトフォルダの Managed フォルダ以下を見ると、Assembly-CSharp.dll ファイルが作成されています。このファイルの中にゲームのコードが入っています。

Assembly-CSharp.dll

（3）Assembly-CSharp.dll を逆コンパイル

Assembly-CSharp.dll は、.NET の DLL ファイルですので、逆コンパイラでソースレベルまで戻せます。ソースレベルまで復元した結果の一部を載せておきます。ほとんどオリジナルのソースコードレベルに戻っていることが分かります。また、Unity では、仕様が公開されている API を活用するケースが多いので、開発が簡単な分、解析側も容易に目的のコード部にたどり着けます。（Update や Start など）

```
// Token: 0x06000052 RID: 82 RVA: 0x0000328C File Offset: 0x0000148C
private void Update()
{
    Vector2 inputVector = Util.GetInputVector();
    float d = this.MoveSpeed * Time.deltaTime;
    base.ClampScreenAndMove(inputVector * d);
    float axis = CrossPlatformInputManager.GetAxis("Horizontal");
    float axis2 = CrossPlatformInputManager.GetAxis("Vertical");
    base.transform.position += (Vector3.right * axis + Vector3.up * axis2) * Time.deltaTime *
      this.MoveSpeed;
    if (CrossPlatformInputManager.GetButton("Jump"))
    {
        float x = base.X + UnityEngine.Random.Range(0f, base.SpriteWidth / 2f);
        float direction = UnityEngine.Random.Range(-3f, 3f);
        Shot.Add(x, base.Y, direction, 10f);
    }
    if (Input.GetKey(KeyCode.Space))
    {
        float x2 = base.X + UnityEngine.Random.Range(0f, base.SpriteWidth / 2f);
        float direction2 = UnityEngine.Random.Range(-3f, 3f);
        Shot.Add(x2, base.Y, direction2, 10f);
    }
}
```

```
// Token: 0x06000053 RID: 83 RVA: 0x000033BC File Offset: 0x000015BC
private void Start()
{
    float width = base.SpriteWidth / 2f;
    float height = base.SpriteWidth / 2f;
    base.SetSize(width, height);
}
```

III. Unity アプリの難読化の方法

Assembly-CSharp.dll の難読化方法を紹介します。

全体の流れとしては、

・Dotfuscator への入力

・Dotfuscator の構成

・Dotfuscator プロジェクトのビルド

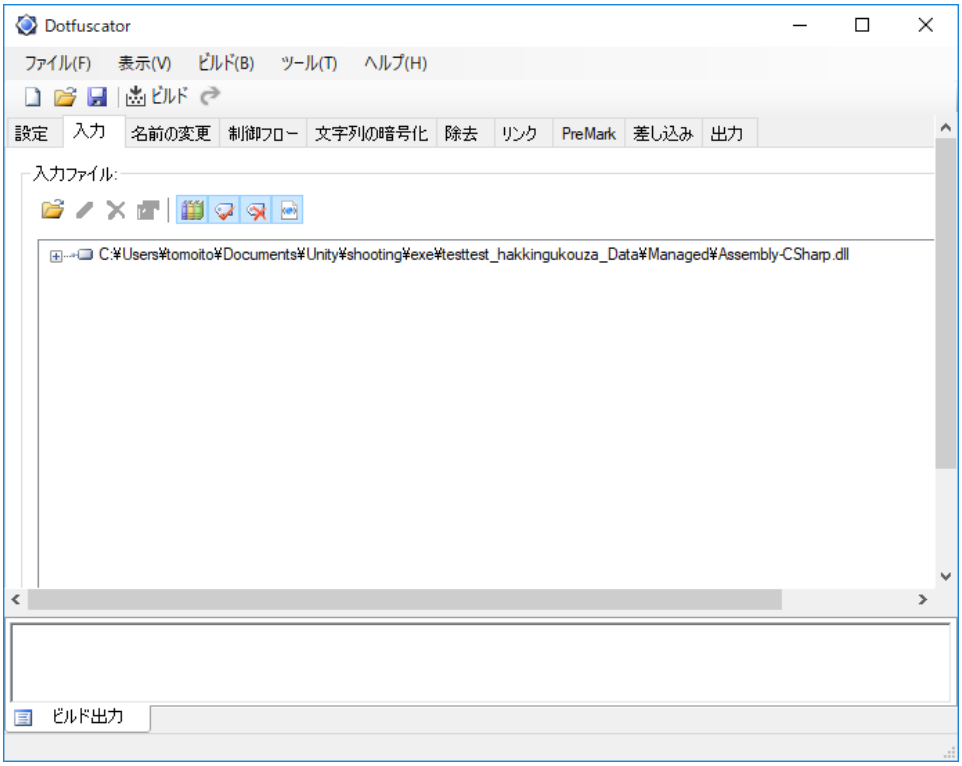
・アセンブリの置き換え

となります。

※ソースプログラムは書き換えませんので、ご安心ください。

（1）Dotfuscator への入力

Dotfuscator ウィザードの「入力」タブを選択し、対象の Assembly-CSharp.dll をドラッグアンドドロップします。



（2）Dotfuscator の構成

難読化の設定を行います。一般的な難読化の設定方法については、マニュアルをご参照ください。Unity においては、以下の対象は名前の変更の除外をする必要があります。

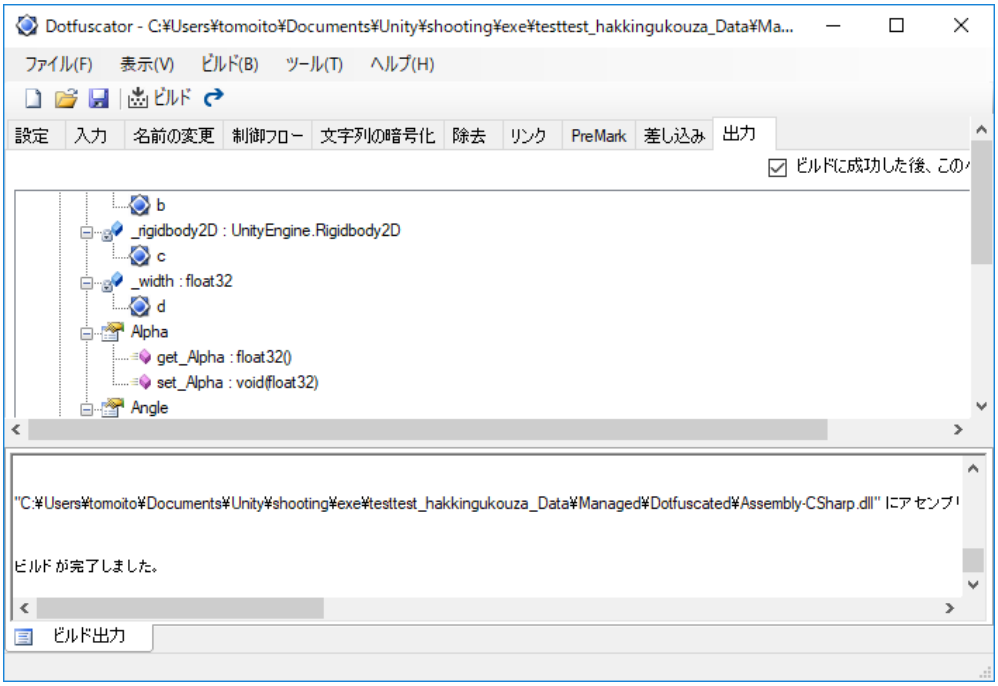
・すべての Start() メソッド

・Update() や FixedUpdate() など、更新に関係するすべてのメソッド

・OnEnable()、OnTriggerEnter()、OnTriggerExit() など、イベント時に実行されるすべてのメソッド

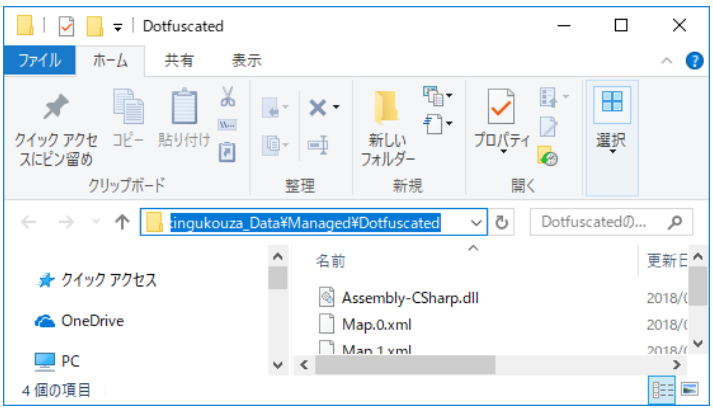
（3）Dotfuscator プロジェクトのビルド

Dotfuscator GUI にて［ビルド］を行います。 GUI の下部にある出力を調べて、ビルドが成功したことを確認します

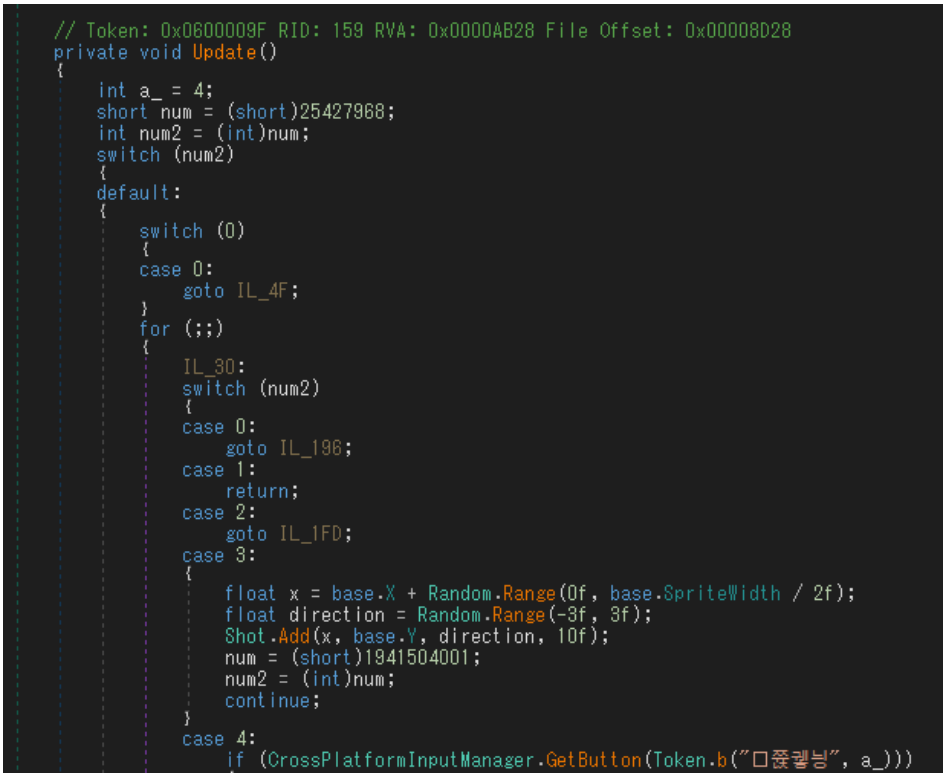


（4）アセンブリの置き換え

保護されていない Assembly-CSharp.dll を、Dotfuscator の出力からコピーした保護されているバージョンに置き換えます。これで作業は完了です。Android/iOS の場合、使用するアプリ全体をビルドする必要があります。手順については、マニュアルをご参照ください。



ちなみに、逆コンパイラで難読化された DLL を戻してみると、解析がほぼ困難な状態になっていることが分かります。



```
// Token: 0x0600009F RID: 159 RVA: 0x0000AB28 File Offset: 0x00008D28
private void Update()
{
    int a_ = 4;
    short num = (short)25427968;
    int num2 = (int)num;
    switch (num2)
    {
    default:
    {
        switch (0)
        {
        case 0:
            goto IL_4F;
        }
        for (;;)
        {
            IL_30:
            switch (num2)
            {
            case 0:
                goto IL_196;
            case 1:
                return;
            case 2:
                goto IL_1FD;
            case 3:
            {
                float x = base.X + Random.Range(0f, base.SpriteWidth / 2f);
                float direction = Random.Range(-3f, 3f);
                Shot.Add(x, base.Y, direction, 10f);
                num = (short)1941504001;
                num2 = (int)num;
                continue;
            }
            case 4:
                if (CrossPlatformInputManager.GetButton(Token.b("口쯎궿닝", a_)))
```

IV. まとめ

便利な開発ツールが発展することにより、開発に掛かるコストを大幅に削減することができますが、ソフトウェアをセキュリティ面で脅威にさらす危険が高まる可能性があります。より強固にアプリケーションを保護したい方は、弊社が販売している Dotfuscator または DashO をご検討ください。

※Dotfuscator ⇒ NET 開発環境向け難読化ツール

DashO ⇒ Java/Android 開発向け難読化ツール