.NET アプリケーションの
リパッケージ対策

AG-TECH CORP.

目次

I. はじめに

Dotfuscator では、リバース エンジニアリングを困難にさせる手法だけではなく、コードの差し込みによってアプリケーションにチェック機能を追加できます。 アプリケーション開発者がコードを記述する必要はありません。Dotfuscator は適切な検証ロジックを自動的に差し込み、チェックが不正な改ざんを検出したタイミングに事前に作成したレスポンス（対応処理）を差し込むこともできます。

(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. 難読化だけではリパッケージ対策にならない

難読化による静的解析の抑止力は非常に効果がありますが、ソースコードの改ざんがあった場合には検知ができません。Dotfuscator が提供している改ざんチェックは、ソースコードの改ざんを検出し、アプリケーションに通知したり、攻撃者を妨害したりして対応することができます。 アプリケーションの改ざんが検知された場合、下記のような対応が可能です。

（1）Exit

アプリケーションは終了コード 0 で直ちに終了します。

（2）Exception

例外がスローされます。

（3）Hang

現在のスレッドを無期限にハングします。

（4）独自のレスポンス処理

チェックが不正な使用を検出したタイミングに事前に作成したレスポンスを差し込むこともできます。

III. 改ざんチェックを使用する方法

本資料では、改ざんチェックを導入する方法を紹介します。

全体の流れとしては、

・チェックを構成

・チェックの動作を決めるさまざまなプロパティを指定

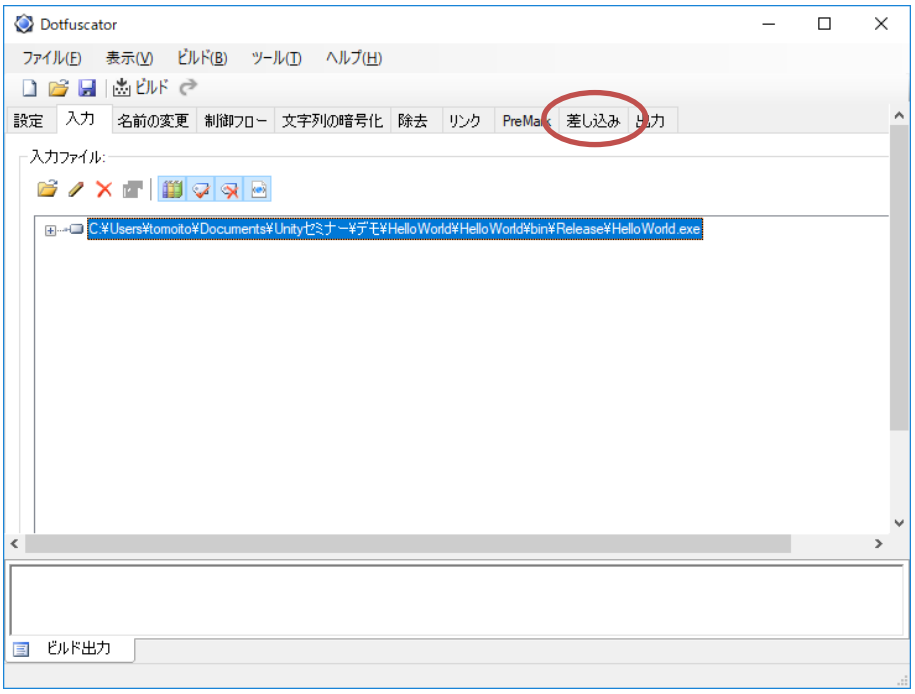
・チェックの検証

となります。

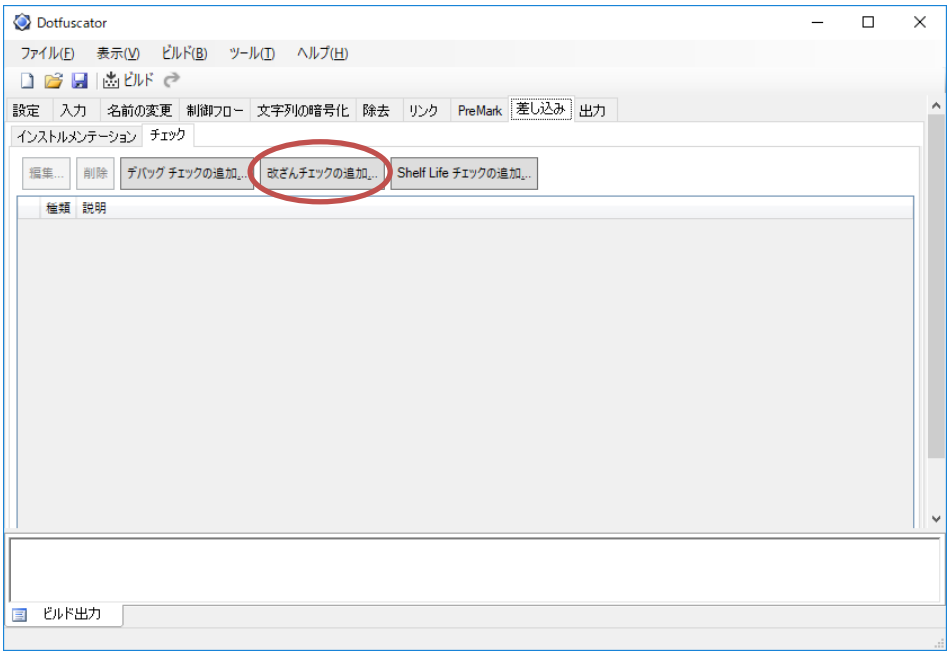
※ソースプログラムは書き換えませんので、ご安心ください。

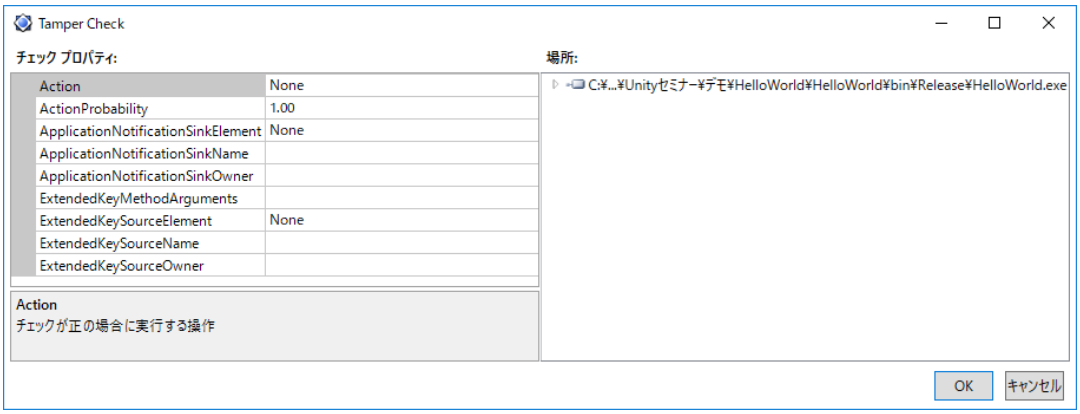
（１）チェックを構成

Dotfuscator ウィザードのメニューより、[差し込み]タブをクリックします。



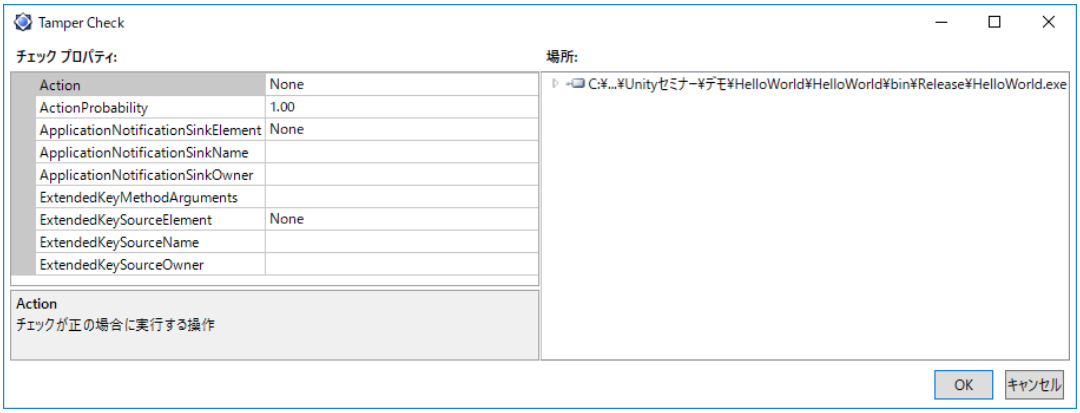
改ざんチェックの追加ボタンをクリックします。





（２）チェックの動作を決めるさまざまなプロパティを指定

チェックの動作を決めるさまざまなプロパティを指定できます。



主なプロパティは下記になります。

Action：チェックが改ざんされたアプリケーションを検出したときに実行するチェック操作です。

ActionProbability：改ざんが検出されたときにチェック操作が行われる確率（0.00 ～ 1.00）を指定します。

今回は Form1 のボタンをクリックした場合、毎回アプリケーションを hang させるように設定します。

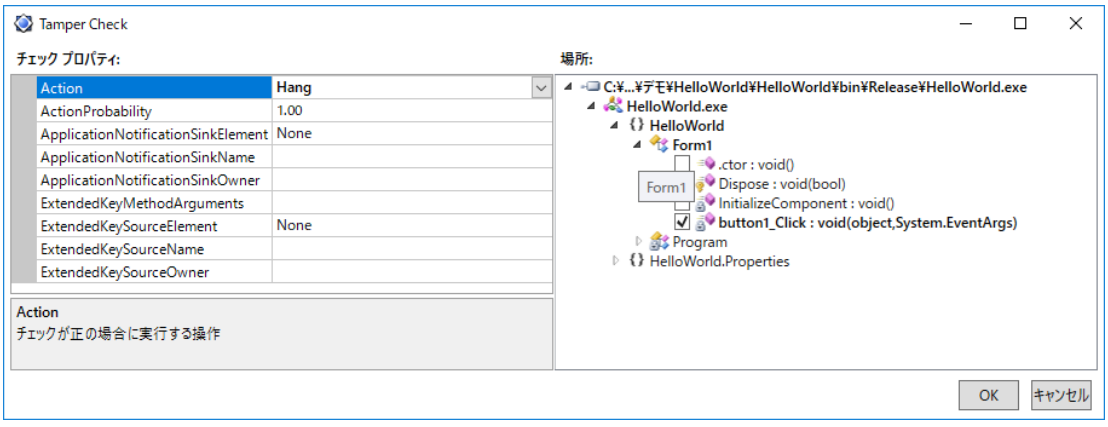
場所:

button1_Click にチェック

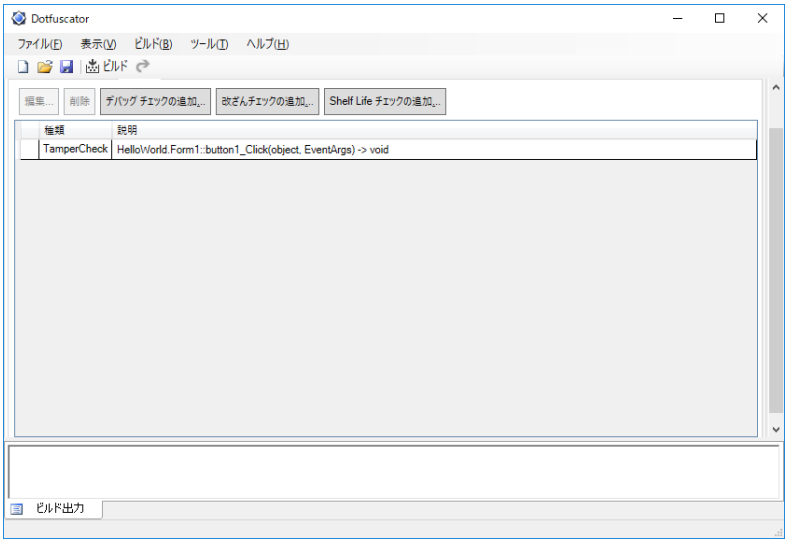
チェック プロパティ:

Action：Hang

ActionProbability:1.00



TamperCheck が追加されました。



アプリケーションをビルドします。

ビルド出力画面にて改ざんチェックが差し込まれたことがわかります。

TamperCheck: HelloWorld.Form1::button1_Click

（3）チェックの検証

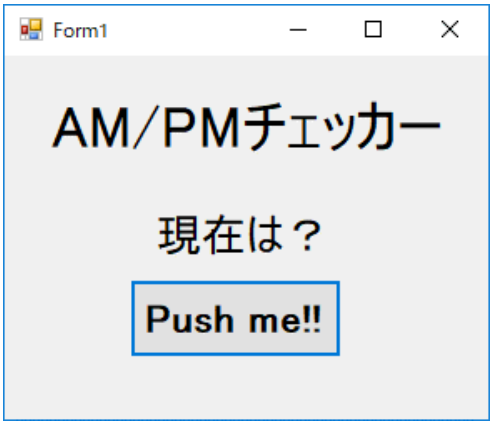
ビルドされたアプリケーションの逆コンパイルをします。今回は改ざん検知の効果を分かりやすくするために、難読化は行っていません。

```
bool flag3 = TimeSpan.Compare(DateTime.Now.TimeOfDay, new TimeSpan(12, 0, 0)) == -1;
if (flag3)
{
    this.label1.Text = "午前です";
}
else
{
    this.label1.Text = "午後です";
}
```

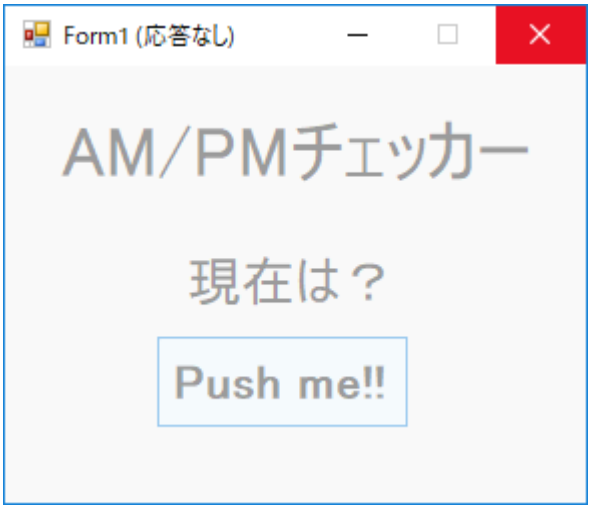
Label1.text に表示されるメッセージを「午前です」から「テスト」に変更し、リコンパイルを行います。

```
if (flag3)
{
    this.label1.Text = "テスト";
}
else
{
    this.label1.Text = "午後です";
}
```

バイナリを実行します



ボタン押下すると、「応答なし」(Hang)されたことが分かります。



IV. まとめ

難読化が、「静的解析」（つまり、アセンブリ ファイルの解析）からアプリケーションを保護するのに対し、チェックは、実行されている間の「動的解析」からアプリケーションを保護します。二つの機能を組み合わせることで、攻撃者が目標を達成することをますます難しくさせます。今回はデバッグチェックを紹介しましたが、その他のチェック機能も簡単に導入ができ、強力なプロテクションを提供します。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