PaoZip Studio

コンパイル済みバイナリを暗号化して、あなたの知的財産を守る

ユーザーズマニュアル

バージョン 1.0.0

2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

1. はじめに

1.1 PaoZip Studio とは

PaoZip Studio は、コンパイル済みのバイナリファイル（.exe, .dll, .jar など）を丸ごと暗号化して保護するツールです。

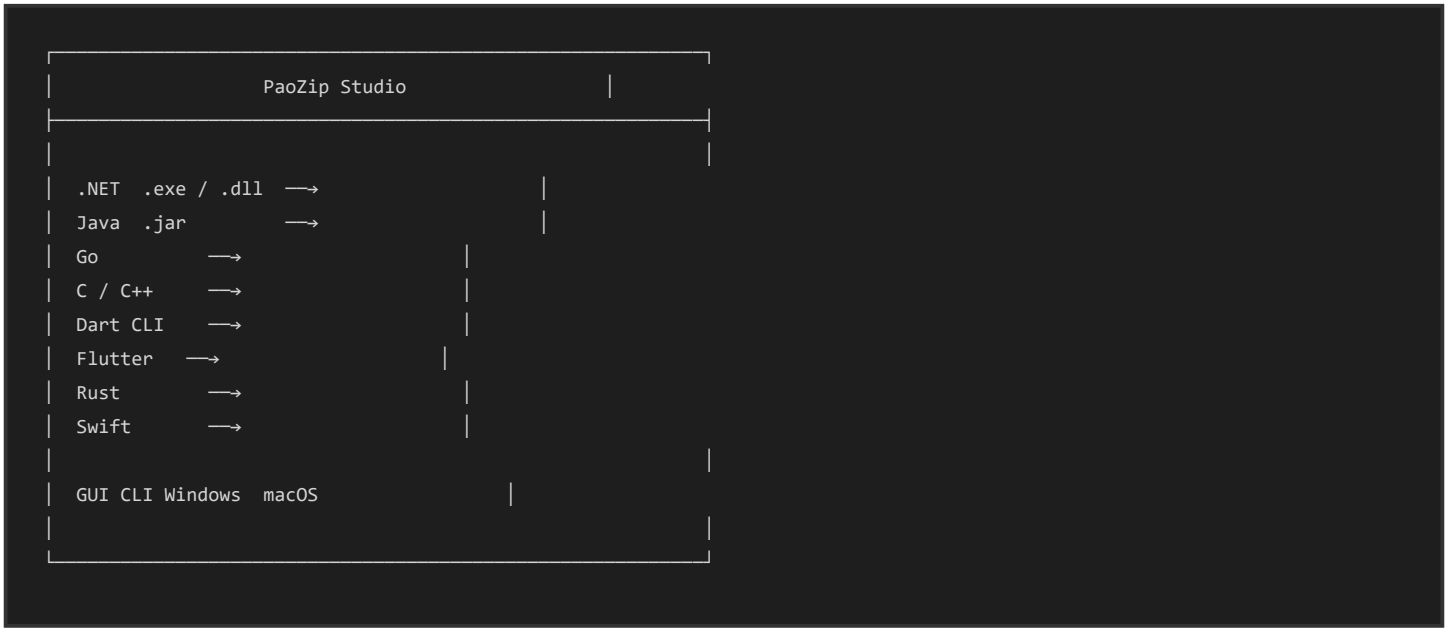
普通のソフトウェアは、リバースエンジニアリングツールを使えば、中のロジックをかなり読み取れてしまいます。特に .NET や Java はバイトコードなので、元のソースコードに近い形まで復元できてしまうことも。

PaoZip Studio を使えば——

```
.exe  →    →  .exe
```

これだけです。
保護されたアプリは、エンドユーザーからは今まで通り普通に使えます。でも中身を覗こうとしても、暗号化されたデータしか見えません。

1.2 何ができるの？



しかも——

- 保護されたアプリは そのまま実行できます（使い方は変わりません）
- 引数もそのままパススルー
- 終了コードもそのまま返却
- ファイルサイズは多くの場合 小さくなります（圧縮効果）

1.3 対応言語一覧

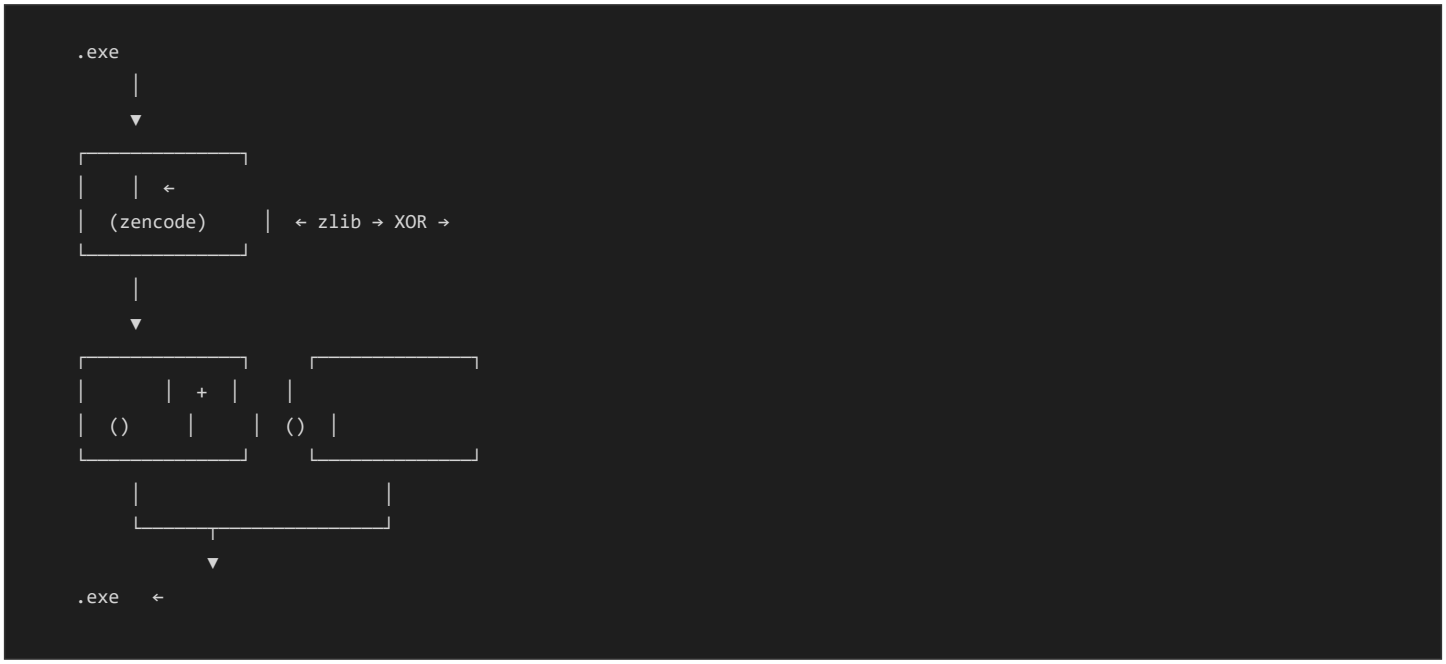
言語	入力	保護方式	特記事項
.NET	.exe / .dll	ネイティブランチャー	Framework / .NET 5+ 両対応
Java	.jar	ネイティブランチャー	Main-Class 必須。出力は .exe
Go	.exe / バイナリ	ネイティブランチャー	
C	.exe / バイナリ	ネイティブランチャー	
C++	.exe / バイナリ	ネイティブランチャー	
Dart CLI	.exe / バイナリ	ネイティブランチャー	dart compile exe の出力
Flutter	デスクトップアプリ	Flutter専用ランチャー	data/app.so を暗号化
Rust	.exe / バイナリ	ネイティブランチャー	
Swift	.exe / バイナリ	ネイティブランチャー	

> 全言語統一アーキテクチャ: すべてのコンパイル型言語で、同一のネイティブ C ランチャー方式を採用しています。
> 暗号化されたペイロードをランチャー内に埋め込み、実行時にテンプディレクトリに展開→実行→削除します。
> 詳しくは 8.2 保護の仕組み をご覧ください。

1.4 仕組み — こうやって守ります

PaoZip Studio の保護は、こんな感じで動きます。

保護するとき（あなたの PC で）：



エンドユーザーが実行するとき：

```
  .exe
      |
      ▼

      |
      ▼

      |
      ▼
    ←
      |
      ▼

      |
      ▼
    ←
```

ポイント：

- 暗号鍵はランチャーの中に難読化して埋め込まれます
- 復号されたデータは実行後に自動削除されます
- エンドユーザーは保護されていることに気づきません

1.5 paozip 言語別ツールとの違い

PaoZip Studio の他に、言語ごとの「paozip for ○○」というツールもあります。何が違うのでしょう？

```
paozip for Python / PHP / Dart / ...

  →
  → import / include

PaoZip Studio

  →
  →
```

比較	言語別ツール	PaoZip Studio
保護対象	ソースコード	コンパイル済みバイナリ
使うタイミング	開発中〜配布前	ビルド後〜納品
特に有効な場面	コードの秘匿	リバースエンジニアリング対策
併用	可能（二重保護！）	可能（二重保護！）

> 両方使えば、ソースもバイナリも保護できます。最強の組み合わせです。

2. 動作環境

2.1 対応 OS

OS	GUI	CLI	状態
Windows 10/11 (x64)	PaoZip Studio (WPF / MAUI)	paozip protect	対応済み
macOS 14+ (Apple Silicon / Intel)	PaoZip Studio (MAUI)	paozip protect	対応済み

2.2 必要な環境

Windows（GUI + CLI）：

- .NET ランタイム不要（自己完結型バイナリとして配布）
- 追加のコンパイラやツールは一切不要

macOS：

- macOS 14 (Sonoma) 以降
- Xcode Command Line Tools（clang コンパイラが必要）

> Windows 版 は、事前コンパイル済みランチャーテンプレートを使用するため、GCC 等の外部コンパイラは不要です。
> macOS 版 では、Xcode Command Line Tools をインストールしてください（xcode-select --install）。
> .NET や Java のアプリを保護する際にも、保護ツール側に追加の SDK は不要です。
> （もちろん、保護する前のビルドには各言語の開発ツールが必要です）

3. インストール

3.1 Windows

Step 1: ダウンロードした ZIP を解凍します。

```
PaoZip Studio/
├── Paozip.Studio.exe      ← GUI
├── paozip.exe             ← CLI
└── Templates/
    ├── launcher_generic_gui.exe    ←
    ├── launcher_dotnetcore.exe
    ├── launcher_java.exe
    └── launcher_flutter.exe
```

Step 2: 好きな場所に配置します。

インストーラーはありません。ZIP を解凍するだけです。

C:¥PaoZip Studio¥ や C:¥Tools¥PaoZip Studio¥ など、お好みの場所に置いてください。

Step 3: （任意）CLI を PATH に追加します。

コマンドラインから paozip をどこでも使いたい場合は、解凍先を PATH 環境変数に追加してください。

> 体験版には回数制限がありますが、機能制限はありません。すべての機能を試せます。

3.2 macOS

Mac でも PaoZip Studio が使えます！

PKG インストーラーでかんたんインストール

Step 1: PaoZipStudio-1.0.0.pkg をダブルクリック

Step 2: 画面の指示に従ってインストール

Step 3: Applications フォルダに「PaoZip Studio」が登場！

```
/Applications/
└── Paozip.Studio.Maui.app    ←  PaoZip Studio
```

初回起動時の Gatekeeper 許可

初回起動時、macOS が「開発元が未確認のため開けません」と表示することがあります。

方法 1: 右クリック →「開く」→「開く」をクリック

方法 2: システム設定 → プライバシーとセキュリティ →「このまま開く」をクリック

> これは macOS のセキュリティ機能（Gatekeeper）です。一度許可すれば、次回からはダブルクリックで起動できます。

Xcode Command Line Tools のインストール

バイナリ保護にはコンパイラ（clang）が必要です。まだインストールしていない場合：

```
xcode-select --install
```

ダイアログが表示されたら「インストール」をクリック。数分で完了します。

> PaoZip Studio は .NET MAUI で開発されたネイティブ macOS アプリです。
> Windows 版と同じ機能が Mac でも使えます。

4. クイックスタート

ここでは、実際にアプリを保護してみましょう。驚くほど簡単です。

4.1 GUI で保護してみよう

Step 1: PaoZip Studio を起動します。

・Windows: Paozip.Studio.exe をダブルクリック
・macOS: Applications フォルダから PaoZip Studio を起動（または Spotlight で検索）

Step 2: 保護したいファイルを選びます。

「Browse」ボタンをクリックして、保護したいファイルを選択します。

（ドラッグ＆ドロップでもOK！）

```
:
  Windows:
    .exe   → .NET / Go / C / C++ / Dart / Rust / Swift
    .dll   → .NET 5+
    .jar   → Java
  macOS:
     → Go / C / C++ / Dart / Rust / Swift
    .jar    → Java
```

Step 3: ファイル情報を確認します。

ファイルを選ぶと、自動的にフレームワーク検出が行われます。

```
  Framework:  .NET 8.0        ←
  Type:       Console Application
  Size:       12,345 bytes
  Types:      42
```

ネイティブバイナリの場合は：

```
  Framework:  Native          ← .NET  Java
  Type:       Windows x64     ← macOS  "macOS ARM64"  "macOS x86_64"
  Size:       2,363,392 bytes
```

Step 4: 「Protect」ボタンをクリック！

あとは待つだけ。保護ログがリアルタイムで表示されます。

```
  Inspecting assembly...
  Framework: .NET 8.0  Type: Console Application
  Size: 12,345 bytes  Types: 42
  Reading: MyApp.dll
  Generating encryption key...
  Encrypting... 12,345 bytes
  Obfuscating key...
  Generating native launcher...
  Using template: launcher_generic_gui.exe
  Launcher created
  Writing: MyApp.protected.exe
  ---
  Protection Complete!
  Original: 12,345 bytes
  Protected: 8,192 bytes                  ←
```

4.2 CLI で保護してみよう

GUI を使わずに、コマンドラインからも保護できます。

Windows:

```
paozip protect MyApp.exe
```

macOS:

```
paozip protect myapp
```

たったこれだけ。出力ファイル名を指定することもできます：

```
paozip protect MyApp.exe -o MyApp.secured.exe       # Windows
paozip protect myapp -o myapp.secured                # macOS
```

4.3 保護したバイナリを実行してみよう

保護されたバイナリは、元のバイナリと同じように実行できます。

Windows:

```
#
MyApp.exe --input data.csv --output result.json

#
MyApp.protected.exe --input data.csv --output result.json
```

macOS:

```
#
./myapp --input data.csv --output result.json

#
./myapp.protected --input data.csv --output result.json
```

引数も、終了コードも、標準出力も——全部そのまま。

エンドユーザーには、保護されていることがわかりません。

> 保護済みバイナリを再度保護しようとすると、ちゃんとエラーになります。二重保護はできません（安全設計！）。

5. 対応言語ガイド

PaoZip Studio は、入力ファイルを自動判別します。

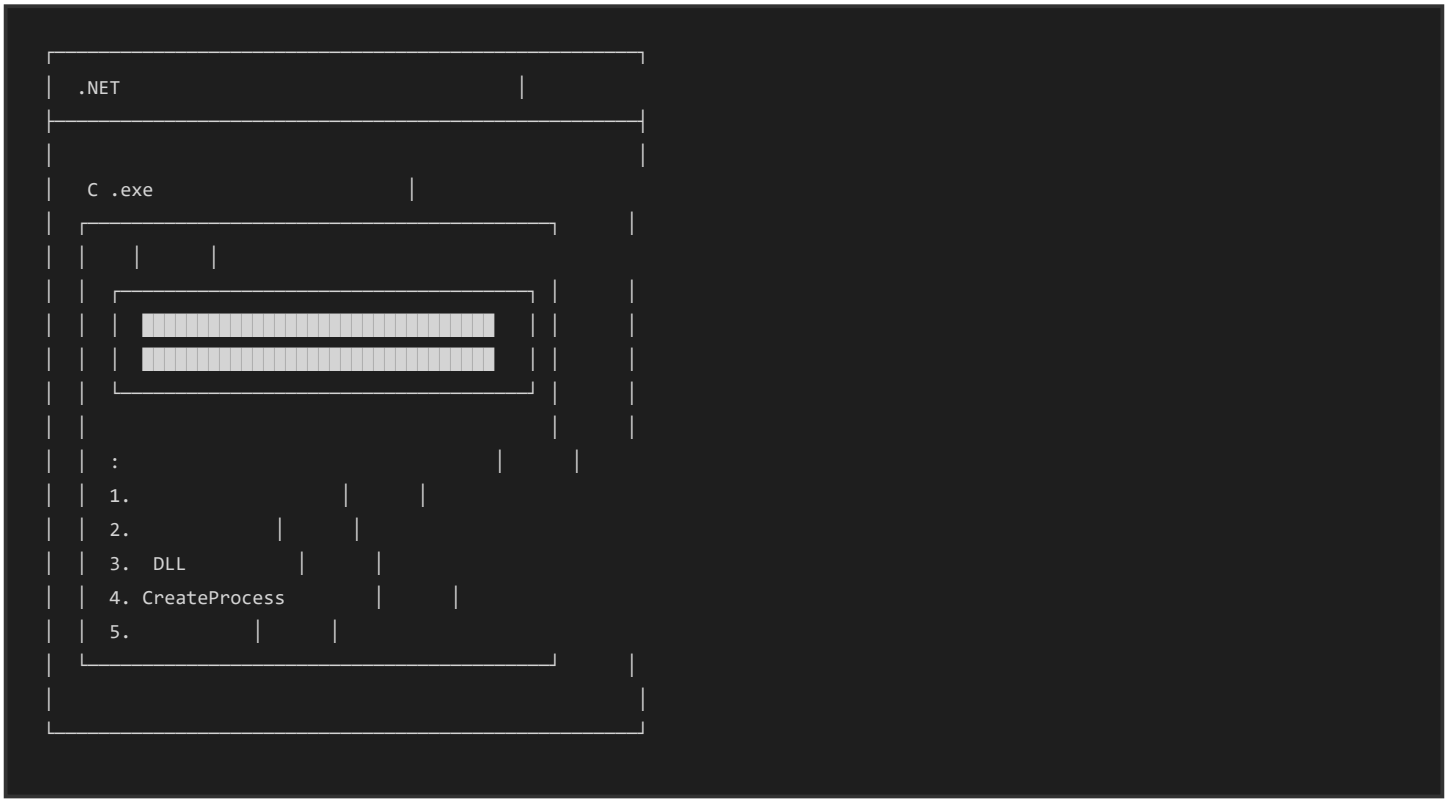
.NET なのか Java なのかネイティブバイナリなのか——あなたが選ぶ必要はありません。

5.1 .NET アセンブリ

対応フレームワーク:

・.NET Framework 4.x（.exe 直接保護）
・.NET 5 / 6 / 7 / 8 / 9 / 10+（.exe を入れれば自動で .dll を検出）

保護方式: ネイティブランチャー（C / C++ と同一方式）



使い方:

```
# .NET Framework
paozip protect MyApp.exe

# .NET 5+.exe  .dll
paozip protect MyApp.exe

# macOS .app MAUI /
paozip protect MyApp.app
```

ポイント:

・出力はネイティブ C ランチャー（.exe）です

- .NET 5+ は .exe（apphost）をドラッグ&ドロップするだけで、同名の .dll を自動検出して保護します（.dll を直接指定する必要はありません）
- .NET Framework: 依存 DLL と .config をランチャーの隣に配置してください
- .NET 5+: 依存 DLL と .json（runtimeconfig, deps）をランチャーの隣に配置してください
- macOS: .app バンドルをそのままドラッグ&ドロップ可能（MAUI / ネイティブ自動判別）
- ランチャーは実行時にこれらをテンプディレクトリにコピーして実行します

逆コンパイル対策として最も効果的：

.NET アセンブリは IL（中間言語）で構成されているため、ILSpy や dnSpy などで簡単に逆コンパイルできてしまいます。PaoZip Studio で保護すれば、ネイティブ C ランチャーの中に暗号化された IL が埋め込まれるため、逆コンパイルツールでは中身を一切読めません。ランチャー自体はネイティブコードなので、.NET の逆コンパイルツールはそもそも認識しません。

5.2 Java JAR

対応: Main-Class が MANIFEST.MF に記述された実行可能 JAR

保護方式: ネイティブランチャー（C / C++ と同一方式）

```
paozip protect MyApp.jar
# → MyApp.exe
```

仕組み:

- 暗号化された JAR がネイティブ C ランチャーの中にリソースとして埋め込まれます
- 起動時にテンプディレクトリに復号 → java -jar で実行 → 終了後に削除
- JAR の中身は一切見えません

```
#
java -jar MyApp.jar --port 8080

# .exe
MyApp.exe --port 8080
```

> 出力が .jar → .exe に変わります。java -jar で実行する代わりに、ダブルクリックで直接実行できるようになります。Java がインストールされている必要はありますが、ユーザーにとってより自然な起動方法になります。

> Java の JAR ファイルは .class ファイルの集合体で、jd-gui などで簡単にデコンパイルできます。保護すれば、ネイティブ C ランチャーの中に暗号化データとして埋め込まれるため、Java のデコンパイルツールでは全く認識できません。

5.3 Go

対応: go build で生成された実行可能バイナリ

保護方式: ネイティブランチャー

```
# Go  →
go build -o myapp.exe main.go              # Windows
paozip protect myapp.exe                    # → myapp.protected.exe

go build -o myapp main.go                   # macOS
paozip protect myapp                        # → myapp.protected
```

テスト結果（実績）：

項目	値
元のサイズ	2.3 MB
保護後	1.4 MB（38% 圧縮！）
引数パススルー	OK
終了コード	OK

> Go バイナリはランタイムを含むため大きくなりがちですが、PaoZip Studio の圧縮効果で保護後の方が小さくなることが多いです。

5.4 C / C++

対応: GCC / Clang / MSVC でコンパイルされた実行可能バイナリ

保護方式: ネイティブランチャー

```
# Windows
gcc -O2 -o myapp.exe myapp.c           # C
g++ -O2 -o myapp.exe myapp.cpp         # C++
paozip protect myapp.exe

# macOS
clang -O2 -o myapp myapp.c             # C
clang++ -O2 -o myapp myapp.cpp         # C++
paozip protect myapp
```

テスト結果：

言語	元	保護後	圧縮率
C	64 KB	44 KB	31% 減
C++	69 KB	47 KB	32% 減

5.5 Dart CLI

対応: dart compile exe で生成された実行可能バイナリ

保護方式: ネイティブランチャー

```
# Windows
dart compile exe main.dart -o myapp.exe
paozip protect myapp.exe

# macOS
dart compile exe main.dart -o myapp
paozip protect myapp
```

> Dart CLI バイナリは AOT（事前コンパイル）で生成されるネイティブバイナリです。Go や Rust と同様の方式で保護できます。

5.6 Flutter Desktop

対応: flutter build windows / flutter build macos で生成されたデスクトップアプリ

保護方式: ネイティブランチャー（Flutter 専用）

Flutter デスクトップアプリは、他のバイナリとはちょっと構造が違います。

```
build/windows/x64/runner/Release/
├── myapp.exe              ← C++ 90KB
├── flutter_windows.dll    ← Flutter
└── data/
    ├── app.so             ← Dart
    └── flutter_assets/    ←
```

大事なポイント： Dart のコードは .exe ではなく data/app.so に入っています。

PaoZip Studio はこれを自動検出して、app.so だけを暗号化します。

```
# Release  .exe
paozip protect myapp.exe
```

保護の流れ：

```
:                          :
├── myapp.exe              ├── myapp.exe          ←
├── flutter_windows.dll    ├── myapp_original.exe ←
└── data/                  ├── flutter_windows.dll
    ├── app.so  ← 5.5 MB   └── data/
    └── flutter_assets/        ├── app.so.enc ← 2.4 MB
                               └── flutter_assets/
```

仕組み：

1. ランチャー（新しい myapp.exe）が起動
2. data/app.so.enc を復号 → data/app.so を生成
3. 元の Flutter ランナー（myapp_original.exe）を実行
4. Flutter アプリが普通に動作
5. アプリ終了後、data/app.so を削除

> Flutter アプリの見た目や動作は一切変わりません。ユーザーは保護されていることに気づきません。

テスト結果：

項目	値
app.so 元サイズ	5.5 MB
app.so.enc 保護後	2.4 MB（56% 圧縮！）
動作	完全に正常
app.so 自動削除	OK

5.7 Rust

対応: rustc / cargo build で生成された実行可能バイナリ

保護方式: ネイティブランチャー

```
# Windows
cargo build --release
paozip protect target/release/myapp.exe

# macOS
cargo build --release
paozip protect target/release/myapp
```

テスト結果：

項目	値
元のサイズ	158 KB
保護後	108 KB（32% 圧縮）
引数パススルー	OK
終了コード	OK

5.8 Swift

対応: swiftc でコンパイルされた実行可能バイナリ

保護方式: ネイティブランチャー

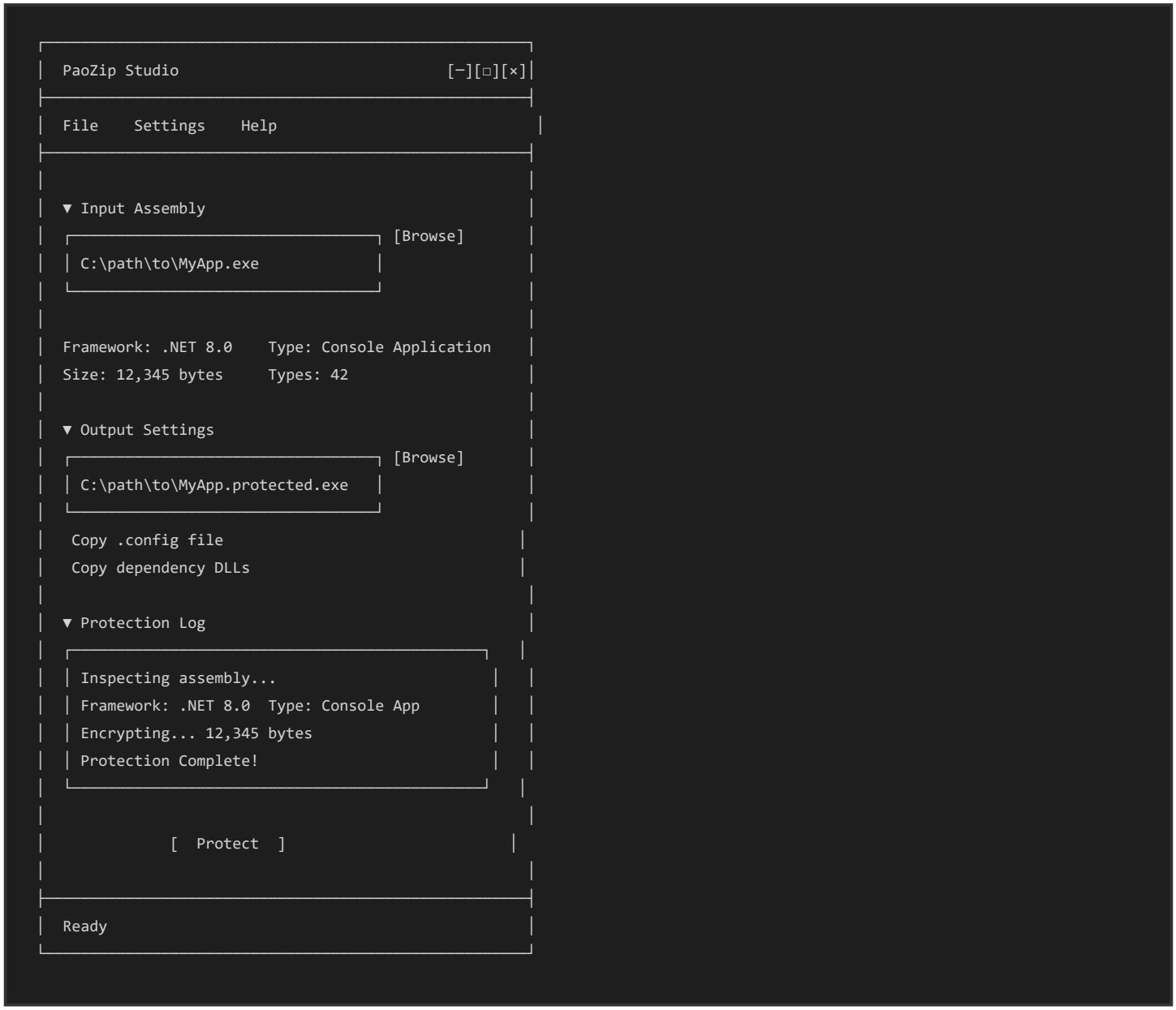
```
# Swift
swiftc -O -o myapp main.swift

#
paozip protect myapp.exe    # Windows
paozip protect myapp        # macOS
```

> Swift on Windows と macOS の両方に対応しています。

6. GUI の使い方

6.1 メイン画面

PaoZip Studio の GUI はシンプルで直感的です。



各セクションの説明：

セクション	説明
Input Assembly	保護したいファイルを選択。ドラッグ＆ドロップにも対応。macOS では .app バンドルもドロップ可能
Output Settings	出力先パス。自動的に .protected.exe が設定される
Protection Log	保護処理のリアルタイムログ
Protect ボタン	クリックで保護開始！

6.2 ダーク/ライトテーマ

Settings → Theme で切り替えられます。

- ダークテーマ: 暗い背景。目に優しい。夜の作業に最適
- ライトテーマ: 明るい背景。日中の作業に最適

設定はアプリ内で即座に反映されます。

6.3 日本語/英語切り替え

Settings → Language で切り替えられます。

- 日本語: すべてのメニュー、ラベル、ログが日本語に
- English: Everything switches to English

ログメッセージも含めて、完全にバイリンガル対応です。

6.4 プロジェクトファイル

よく使う設定を .paoproj ファイルとして保存・読み込みできます。

- File → Save Project: 現在の設定（入力パス、出力パス、オプション）を保存
- File → Open Project: 保存した設定を読み込み

CI/CD や定型的な保護作業に便利です。

7. CLI の使い方

GUI を起動せずに、コマンドラインから直接保護できます。CI/CD パイプラインに組み込む場合に特に便利です。

7.1 protect コマンド

```
Usage: paozip protect <input> [options]

Arguments:
  <input>

Options:
  -o <path>   : *.protected.*
```

7.2 使用例

Windows:

```
paozip protect MyApp.exe                    # → MyApp.protected.exe
paozip protect MyApp.exe -o dist/MyApp.exe  #
paozip protect MyApp.dll                    # .NET 5+ → MyApp.protected.dll
paozip protect MyApp.jar                    # Java → MyApp.protected.exe
```

macOS:

```
paozip protect myapp                        # → myapp.protected
paozip protect myapp -o dist/myapp          #
paozip protect MyApp.jar                    # Java → MyApp.protected
```

7.3 CI/CD での活用

GitHub Actions の例：

```
steps:
  - name: Build
    run: dotnet publish -c Release -o dist/

  - name: Protect
    run: paozip protect dist/MyApp.dll -o dist/MyApp.dll

  - name: Upload
    uses: actions/upload-artifact@v4
    with:
      name: protected-app
      path: dist/
```

Jenkins の例：

```
stage('Protect') {
    steps {
        bat 'paozip protect build\\MyApp.exe -o release\\MyApp.exe'
    }
}
```

> CLI の終了コード: 0 = 成功、1 = 失敗。CI/CD のステップ制御に使えます。

8. 技術仕様 — もっと詳しく知りたい人へ

ここからは技術的な詳細です。PaoZip Studio がどう動いているのか、もっと知りたい人向け。

8.1 暗号化方式

PaoZip Studio は zencode という独自の暗号化エンジンを使用しています。

暗号化の流れ：

鍵の保護：

- 暗号化鍵は 64 文字のランダム hex 文字列（256 ビット相当）
- ランチャーに埋め込む際は XOR マスク方式で難読化
- mask[] と maskedKey[] の2つの配列に分割
- key[i] = mask[i] ^ maskedKey[i] で復元
- 鍵が平文でバイナリに含まれることはありません

8.2 保護の仕組み

全コンパイル型言語で統一されたネイティブランチャー方式を採用しています。

.NET、Java、Go、C/C++、Dart、Rust、Swift — すべて同じアーキテクチャです。

ネイティブランチャー方式

対象: 全コンパイル型言語（.NET, Java, Go, C, C++, Dart CLI, Rust, Swift）

```
C  .exe
 |
 ▼

 |
 ▼
    ← Windows: %TEMP%\paozip_{PID}\ / macOS: /tmp/paozip_{PID}/
 |
 ▼

 |
 ▼
      ← .dll, .config, .json
 |
 ▼
           ← CreateProcess / fork+exec
 |
 ▼

 |
 ▼
  ←
```

特長：

- ランチャー自体がネイティブ C コードなので、.NET / Java の逆コンパイルツールで解析不可
- 実行後に自動削除（リトライ付き）
- 引数と終了コードは完全にパススルー
- 全言語で同じ仕組みなので、信頼性が高い

言語別の動作の違い：

言語	テンプに書き出すも	実行方法 (Windows)	実行方法 (macOS)
.NET Framework	.exe	CreateProcess	—
.NET 5+	.dll	dotnet exec	dotnet exec
Java	.jar	java -jar	java -jar
Go / C / C++ / Rust / Swift / Dart CLI	バイナリ	CreateProcess	fork + exec

Flutter Desktop: 専用方式

Flutter デスクトップは構造が特殊なため、専用の方式を使います。

```
  |
  ▼
data/app.so.enc
  |
  ▼
 data/app.so
  |
  ▼
 Flutter _original.exe
```

```
  |
  ▼
Flutter  app.so
  |
  ▼

  |
  ▼
data/app.so
```

8.3 二重保護防止

保護済みバイナリには PAOZIPLAUNCHER というマーカー文字列が埋め込まれます。

再度保護しようとすると、このマーカーを検出してエラーになります。

```
$ paozip protect MyApp.protected.exe
Error: This binary is already protected by PaoZip.
```

これは意図的な安全設計です。二重保護は意味がなく、問題を起こす可能性があるため。

8.4 パフォーマンス

保護処理にかかる時間：

入力サイズ	処理時間（目安）
~100 KB	1〜2 秒
~1 MB	2〜3 秒
~5 MB	3〜5 秒
~50 MB	10〜15 秒

事前コンパイル済みテンプレートを使用するため、保護処理は高速です。

サイズ変化：

多くの場合、保護後のファイルは 元より小さく なります。これは zlib 圧縮の効果です。

言語	圧縮率（実測）
Go	38% 減
Flutter (app.so)	56% 減
C / C++	31〜32% 減
Rust	32% 減

起動時のオーバーヘッド：

・復号 + テンプファイル書き出しの時間（通常 0.1〜0.5 秒）

・ユーザーが体感できるレベルではありません

9. トラブルシューティング

「Launcher template not found」と表示される

保護に必要なランチャーテンプレートが見つからない場合に表示されます。

対処法：

- Templates/ フォルダが PaoZip Studio と同じ場所にあるか確認してください
- テンプレートファイル（launcher_*.exe）が含まれていることを確認してください

「This binary is already protected by PaoZip」

保護済みのバイナリをもう一度保護しようとしています。

元の（保護前の）バイナリを使ってください。

「Not a recognized executable format」

PaoZip Studio が認識できないファイル形式です。

PE（Windows .exe）または Mach-O（macOS バイナリ）のみ対応しています。

「No Main-Class found in MANIFEST.MF」

Java の JAR ファイルに Main-Class が設定されていません。

ライブラリ JAR は保護できません。実行可能 JAR のみ対応しています。

.NET 5+ で「managed assembly was not found」

.NET 5+ の .exe（apphost）をドラッグ&ドロップすると、同名の .dll を自動検出して保護します。

.dll が見つからない場合にこのエラーが出ます。.exe と同じフォルダに同名の .dll があることを確認してください。

保護後のアプリが起動しない

- アンチウイルスソフト: 下記「アンチウイルスソフトとの互換性」を参照してください。
- 依存ファイル: 保護後のバイナリを別のフォルダに移動した場合、依存する DLL や設定ファイルも一緒に移動してください。
- Flutter Desktop: flutter_windows.dll や data/flutter_assets/ が同じフォルダにあることを確認してください。

アンチウイルスソフトとの互換性

PaoZip
で保護したバイナリは、実行時に暗号化されたペイロードを復号して起動する仕組みです。この動作パターンを一部のアンチウイルスソフトが疑わしいと判断する場合があります。

VirusTotal（72製品一斉スキャン）での検証結果:

結果	製品数
安全と判定（Undetected）	66 / 72
誤検知（False Positive）	6 / 72

安全と判定した主要製品（一部）:

Windows Defender、Norton、ESET-NOD32、Kaspersky、Avast、AVG、BitDefender、McAfee (Fortinet)、Avira、DrWeb、Emsisoft、GData、ClamAV、Google、Baidu、Huorong 他

誤検知した製品（6社）:

Arctic Wolf、Bkav Pro、CrowdStrike Falcon、DeepInstinct、Elastic、SecureAge — いずれも AI/機械学習ベースのヒューリスティック検出です。

誤検知された場合の対処法:

1. アンチウイルスソフトの設定で、保護済みバイナリのフォルダを除外（ホワイトリスト）に追加してください
2. 企業環境でエンドポイント製品（CrowdStrike 等）を使用している場合は、IT管理者に除外申請を行ってください
3. 保護済みバイナリをお客様に配布する際は、コード署名証明書で署名することで誤検知率を大幅に低減できます

macOS:「開発元が未確認のため開けません」

Gatekeeper による警告です。右クリック →「開く」→「開く」で起動できます。

または、システム設定 → プライバシーとセキュリティ →「このまま開く」をクリック。

macOS:「clang: command not found」

Xcode Command Line Tools がインストールされていません。

```
xcode-select --install
```

macOS: 保護したバイナリに実行権限がない

保護後のバイナリに実行権限を付与してください。

```
chmod +x myapp.protected
```

CLI の終了コードが 1 になる

保護処理が失敗しています。エラーメッセージを確認してください。

よくある原因：

・入力ファイルが存在しない

・出力先に書き込み権限がない
・ランチャーテンプレートが見つからない（Windows）、またはコンパイラが見つからない（macOS: clang）

10. ライセンス・お問い合わせ

ライセンスモデル

PaoZip Studio は 言語ごとのライセンス です。

ライセンス	保護できる言語
.NET ライセンス	.NET Framework / .NET 5+
Java ライセンス	Java JAR
Go ライセンス	Go バイナリ
C/C++ ライセンス	C / C++ バイナリ
Dart ライセンス	Dart CLI / Flutter Desktop
Rust ライセンス	Rust バイナリ
Swift ライセンス	Swift バイナリ

体験版について：

- すべての言語・機能を試せます（機能制限なし）
- 回数制限があります
- 体験版で保護したバイナリは製品版と同じ品質です

購入・ダウンロード

https://www.pao.ac/paozip/buy.html

11. 使用許諾

PaoZip Studio の使用について、PaoZip Studio の使用者（以下「利用者様」と称します）と有限会社パオ・アット・オフィス（以下「弊社」と称します）は、以下の各項目についての内容に同意するものとします。

1. 使用許諾書

この使用許諾書は、利用者様が PaoZip Studio を使用する場合に同意しなければならない契約書です。

2. 使用許諾書の同意

利用者様が PaoZip Studio を使用する時点で、本使用許諾書に同意されたものとします。同意されない場合は、PaoZip Studio を使用する事はできません。

3. ライセンス（使用権）の購入

利用者様が PaoZip Studio の製品版を使用して開発を行う場合には、1台の開発用コンピュータで PaoZip Studio を使用するにあたり、言語ごとに1ライセンスを購入する必要があります。

保護されたバイナリを配布・実行する環境にはライセンスは必要ありません。ランタイムライセンスフリーでございます。

4. 著作権

PaoZip Studio の著作権は、いかなる場合においても弊社に帰属いたします。

5. 免責

PaoZip Studio の使用によって、直接的、あるいは、間接的に生じた、いかなる損害に対しても、弊社は補償賠償の責任を負わないものとします。

6. 禁止事項

PaoZip Studio 及びその複製物を第三者に譲渡・貸与する事は出来ません。PaoZip Studio を開発ツールとして再販/再配布することを禁止します。なお、保護されたバイナリを配布することは問題ございません。

7. 保証の範囲

弊社は PaoZip Studio の仕様を予告無しに変更することがあります。その場合の利用者様に対する情報提供は、弊社 WEB サイトにて行う事とします。

8. 適用期間

本使用許諾条件は利用者様が PaoZip Studio を使用した日より有効です。

12. 代金支払い方法

PaoZip Studio の製品版をご利用頂ける場合は、言語ごとにライセンスを購入して頂く必要があります。

体験版について:
すべての言語・機能を試せます（機能制限なし、回数制限あり）。体験版で保護したバイナリは製品版と同じ品質です。

必要なライセンス数の数え方

PaoZip Studio で開発を行うパソコンの台数 × 使用する言語数

1ライセンス当たりの価格（言語ごと）

11,000円（税込）

バグフィックス等のバージョンアップは原則として無償とさせて頂きます。

お支払方法

11,000円 × ライセンス数 を下記口座へ銀行振込、または、郵便振替による送金をして下さい。

銀行名	支店名（コード）	口座番号	名義
三菱UFJ銀行	新宿支店 (341)	普通 3831891	ユ）パオアットオフィス
PayPay銀行	すずめ支店 (002)	普通 6461359	ユ）パオアットオフィス

郵便口座番号	名義
00150-0-576845	有限会社 パオ・アット・オフィス

※ 振込手数料は利用者様負担でお願い致します。

お支払いの通知と製品の送付

1. 振り込みが完了した時点で、必ず弊社 WEB サイトの入金連絡フォームから入金のご連絡をお願いいたします。
2. 弊社では上記連絡を受けて入金確認を行い、PaoZip Studio の製品版ライセンスキーを利用者様へ電子メールにてお送りさせて頂きます。

見積書/納品書/請求書/領収証の発行

見積書/納品書/請求書/領収証の発行は可能でございます。製品サイトでの手続きにより発行いたします。

お問い合わせ

ご質問・ご要望・バグ報告など、お気軽にどうぞ。

有限会社 パオ・アット・オフィス

- 製品サイト: https://www.pao.ac/paozip/
- 購入ページ: https://www.pao.ac/paozip/buy.html
- メール: info@pao.ac

PaoZip Studio — あなたのコードを、あなたの知的財産を、守ります。