**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN TỐT NGHIỆP**

**ĐỀ TÀI:**

**NGHIÊN CỨU, XÂY DỰNG VÀ PHÁT TRIỂN ỨNG DỤNG BẢO VỆ MÃ NGUỒN OBFUSCATTOR NET CHO NGÔN NGỮ C#**

Giảng viên hướng dẫn: THS.TRẦN PHONG NHÃ

Sinh viên thực hiện: TRẦN ANH DUY

Lớp: CÔNG NGHỆ THÔNG TIN

Khóa:K59

Mã số sinh viên: 5951071010

TP. Hồ Chí Minh, tháng 07 năm 2022

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI **CỘNG HÒA XÃ HỘI CHỦ NGHIÃ VIỆT NAM**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH** Độc lập – Tự do – Hạnh phúc

**NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP**

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

-------\*\*\*-------

**Mã sinh viên:** 5951071010 **Họ tên SV:** Trần Anh Duy

**Khóa:** 59 **Lớp:** Công nghệ thông tin

1. **Tên đề tài**

Nghiên cứu, xây dựng và phát triển ứng dụng bảo vệ mã nguồn Obfuscattor NET cho ngôn ngữ C#.

1. **Mục đích, yêu cầu**
   1. **Mục đích:**

Các ngôn ngữ đã biên dịch như Rust, C++ được chuyển đổi trực tiếp sang mã máy (machine code). Cách duy nhất để dịch ngược lại cách chúng hoạt động là sử dụng một disassembler, đây là một quá trình gian khổ và phức tạp. Điều đó không phải là không thể, nhưng việc cố gắng suy ra logic ứng dụng cấp cao từ một luồng hợp ngữ là một việc cực kỳ cực kỳ khó.

Mặt khác, các ngôn ngữ như C# và Java không được biên dịch cho bất kỳ hệ điều hành cụ thể nào. Thay vào đó, chúng được biên dịch sang một ngôn ngữ trung gian, chẳng hạn như .NET có ngôn ngữ trung gian gọi là MSIL. Ngôn ngữ trung gian tương tự như hợp ngữ, nhưng nó có thể dễ dàng chuyển đổi lại thành mã nguồn. Điều này có nghĩa là nếu bạn có DLL công khai hoặc tệp thực thi mà doanh nghiệp của bạn đang phân phối, bất kỳ ai có bản sao tệp thực thi của bạn đều có thể mở tệp đó trong trình dịch ngược .NET như ILSpy và có thể trực tiếp đọc (và sao chép) mã nguồn của bạn.

Và đó chính là đề tài của em, mục đích nhằm ngăn chặn hoặc giảm thiểu quá trình lấy được mã nguồn của phần mềm, cụ thể là phần mềm viết bằng C#.

* 1. **Yêu cầu:**
* Yêu cầu công nghệ:
  + Sử dụng ngôn ngữ lập trình C#.
  + Sử dụng Net Framework, .Net.
  + Sử dụng Visual studio.
* Yêu cầu:
  + Nhận diện phần mềm viết bằng ngôn ngữ C#.
  + Nhận diện công nghệ của C#.
  + Cho phép tùy chỉnh chức năng mã hóa.
  + Mã hóa mã nguồn.
  + Xây dựng lại phần mềm bằng mã nguồn đã mã hóa.
  + Chức năng bypass mã hóa.
* Yêu cầu phi chức năng :
  + Tốc độ truy cập nhanh, chính xác.
  + Giao diện cực kỳ đơn giản, thân thiện.
  + Thao tác đơn giản, dễ dàng sử dụng.

1. **Nội dung và phạm vi đề tài**
   1. **Nội dung:**

* Tổng quan về vấn đề.
* Phân tích và nghiên cứu các kỹ thuật mã hóa.
* Phân tích và thiết kế hệ thống theo yêu cầu.
* Thiết kế giao diện đơn giản và dễ sử dụng.
* Xây dựng phần mềm bảo mật và mã hóa mã nguồn cho ngôn ngữ C#.
* Kiểm thử và ứng dụng thực nghiệm.
  1. **Phạm vi đề tài:**
* Nghiên cứu vận dụng kiến thức đã được học và kinh nghiệm tham gia phát triển ứng dụng theo đề tài.
* Phát triển ứng dụng phù hợp với khả năng bên cạnh sử dụng nhiều công nghệ mới bên ngoài trường lớp.
* Nghiên cứu ngôn ngữ C#, cách hoạt động, IL,… và một số thư viện hỗ trợ liên quan.

1. **Công nghệ, công cụ và ngôn ngữ lập trình**
   1. **Công nghệ:**

* Encryption
* Dnlib
  1. **Công cụ:**
* Visual Studio
* Các công cụ hỗ trợ Reverse Engeering .NET như CFF Explorer, .NET Reflector, ILSPY,...
* Các công cụ debugger dành cho .NET
  1. **Ngôn ngữ lập trình:**
* C#

1. **Các kết quả chính dự kiến sẽ đạt được và ứng dụng**
   1. **Kết quả dự kiến đạt được:**

* Người dùng dễ dàng thực hiện được mọi chức năng của phần mềm mà không cần kiến thức chuyên sâu về mã hóa.
* Nhận biết được phần mềm có phải thuộc .NET hay không.
* Đọc thông tin của phần mềm .NET và các dependency của nó.
* Cho phép chọn chỗ lưu phần mềm đã mã hóa vào một nơi khác tùy thích.
* Dễ dàng chọn những phương thức bảo mật để áp dụng vào phần mềm.
* Mã hóa phần mềm theo cấu hình.
* Khả năng lưu lại cấu hình của từng project.
* Khả năng load cấu hình đã lưu của project một cách nhanh chóng.
* Hiển thị lỗi và các exception một cách trực quan.
* Khả năng xem lại log của quá trình mã hóa.
* Khả năng lưu trữ stack-trace để trace back quá trình mã hóa.
* Khả năng lọc ra những bộ phận của phần mềm khỏi quá trình mã hóa.
  1. **Ứng dụng:**
* Trong tương lai Obfuscattor NET mong muốn trở thành ứng dụng, công cụ hiệu quả trong việc bảo mật cho mọi sản phẩm của .NET.

1. **Cán bộ, giảng viên hướng dẫn**

Họ và tên : ThS. Trần Phong Nhã

Đơn vị công tác : Bộ môn Công Nghệ Thông Tin

Điện thoại : 0961170638

Emai: : [nhatran.sv@gmail.com](mailto:nhatran.sv@gmail.com)

|  |  |
| --- | --- |
| **Ngày tháng 03 năm 2022**  **Trưởng BM Công nghệ Thông tin** | **Đã giao nhiệm vụ TKTN Giáo viên hướng dẫn** |
| **ThS. Trần Phong Nhã** | **ThS. Trần Phong Nhã** |

Đã nhận nhiệm vụ TKTN

Sinh viên: Trần Anh Duy Ký tên:

Điện thoại: 0976718029 Email: [takatran29@gmail.com](mailto:takatran29@gmail.com)

# **LỜI CẢM ƠN**

Lời đầu tiên em xin cảm ơn thầy Trần Phong Nhã đã giúp đỡ và hướng dẫn nhiệt tình cho em trong suốt thời gian em làm Báo Cáo Tốt Nghiệp này. Tiếp theo em xin cảm ơn quý Thầy Cô trường Đại Học Giao Thông Vận Tải phân hiệu tại TP.HCM đã giảng dạy và truyền cảm hứng đến tất cả những sinh viên như em, để chúng em có thêm nhiều kiến thức, mở mang thêm cách tư duy, sáng tạo trong cuộc sống.

Em rất mong nhận được lời khuyên, lời góp ý chân thành từ quý Thầy Cô để em tự hoàn thành hơn nữa vốn kiến thức của mình.

Với những kiến thức còn thiếu sót, những kỹ năng còn hạn hẹp, cũng như em đã dành hết thời gian và tâm huyết của mình để hoàn thành Báo Cáo Tốt Nghiệp, em không thể tránh được những sai lầm, em mong Thầy Cô có thể thông cảm và bỏ qua cho em.

Lời cuối cùng em xin chúc quý Thầy Cô phân hiệu cũng như Bộ Môn Công Nghệ Thông Tin có thật nhiều niềm vui, sức khỏe và luôn thành công trong công việc.

Em xin chân thành cảm ơn!

# **NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

........................................................................................................................................... ...........................................................................................................................................

**Tp. Hồ Chí Minh, ngày .… tháng …… năm ............**

**Giáo viên hướng dẫn**

**Trần Phong Nhã**

# **MỤC LỤC**

[LỜI CẢM ƠN 6](#_Toc107057460)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 7](#_Toc107057461)

[MỤC LỤC 8](#_Toc107057462)

[DANH MỤC CHỮ VIẾT TẮT 10](#_Toc107057463)

[MỞ ĐẦU 1](#_Toc107057464)

[1 Lý do chọn đề tài 1](#_Toc107057465)

[2 Mục tiêu nghiên cứu 2](#_Toc107057466)

[3 Phạm vi đề tài 2](#_Toc107057467)

[4 Cấu trúc bài báo cáo 3](#_Toc107057468)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 4](#_Toc107057469)

[1.1 Tổng quan về Ngôn Ngữ Lập Trình C# [1] 4](#_Toc107057470)

[1.1.1 Giới thiệu về ngôn ngữ 4](#_Toc107057471)

[1.1.2 Lý do chọn ngôn ngữ lập trình C# 5](#_Toc107057472)

[1.1.2 Các tính năng của C# 5](#_Toc107057473)

[1.1.3 Đặc trưng của ngôn ngữ lập trình C# 7](#_Toc107057474)

[1.1.4 Ưu điểm của C# 8](#_Toc107057475)

[1.1.5 Nhược Điểm của C# 8](#_Toc107057476)

[1.2 Tổng quan về .NET [2] 9](#_Toc107057477)

[1.2.1 Khái niệm .NET 9](#_Toc107057478)

[1.2.2 Đặc điểm nổi bật của .NET 10](#_Toc107057479)

[1.3 Tổng quan về mã hóa và trộn mã (obfuscation) [3] 10](#_Toc107057480)

[1.3.1 Khái niệm 10](#_Toc107057481)

[1.3.2 Lý do nên mã hóa 11](#_Toc107057482)

[1.3.3 Cách hoạt động của trộn mã hay mã hóa 11](#_Toc107057483)

[1.3.4 Các kỹ thuật xáo trộn cơ bản 12](#_Toc107057484)

[1.3.5 Ưu điểm của mã hóa mã nguồn 14](#_Toc107057485)

[1.3.6 Nhược điểm 15](#_Toc107057486)

[CHƯƠNG 2: PHÂN TÍCH BÀI TOÁN 16](#_Toc107057487)

[2.1 Lý do nên mã hóa .NET 16](#_Toc107057488)

[2.2 Một số phần mềm hỗ trợ việc dịch ngược mã nguồn .NET [4] 18](#_Toc107057489)

[2.2.1 CFF Explorer 19](#_Toc107057490)

[2.2.2 .NET Reflector 20](#_Toc107057491)

[2.2.3 ILDASM 21](#_Toc107057492)

[2.2.4 ILSPY 22](#_Toc107057493)

[2.3 Một số phần mềm hỗ trợ việc mã hóa mã nguồn .NET 23](#_Toc107057494)

[2.4 Kết luận 24](#_Toc107057495)

[CHƯƠNG 3: PHÂN TÍCH, THIẾT KẾ VÀ NGHIÊN CỨU CÁC KỸ THUẬT VÀ THUẬT TOÁN MÃ HÓA 25](#_Toc107057496)

[3.1 Các kỹ thuật mã hóa sẽ áp dụng 25](#_Toc107057497)

[3.1.1 Kỹ thuật đổi tên (Entity Renaming) [5] 25](#_Toc107057498)

[3.1.2 Kỹ thuật mã hóa string (String Encryption) [6] 27](#_Toc107057499)

[**3.1.2.1**  **Mật mã Caesar (Caesar Cipher)** 29](#_Toc107057500)

[**3.1.2.2**  **Repeating-key XOR Cipher** 31](#_Toc107057501)

[3.1.3 Kỹ thuật kiểm soát luồng chảy (Control Flow Obfuscation) [7] 34](#_Toc107057502)

[3.1.4 Kỹ thuật đột biến (Mutation Obfuscation) 38](#_Toc107057503)

[3.1.5 Kỹ thuật mã giả (Dummy Code Insertion) [8] 40](#_Toc107057504)

[3.2 Các kỹ thuật bảo vệ thêm cho .NET 42](#_Toc107057505)

[3.2.1 Liên kết các thư viện vào Assembly (Library Linking) [9] 42](#_Toc107057506)

[3.2.2 Proxy Reference [10] 44](#_Toc107057507)

[3.2.3 MSIL Dissasembler Protection 44](#_Toc107057508)

[3.2.4 Anti Tamper [11] 45](#_Toc107057509)

[CHƯƠNG 4: PHÂN TÍCH, THIẾT KẾ HỆ THỐNG PHẦN MỀM VÀ THỰC NGHIỆM 46](#_Toc107057510)

[4.1 Phân tích cách hoạt động của phần mềm 46](#_Toc107057511)

[4.1.1 Thư viện dnlib 46](#_Toc107057512)

[4.1.2 Cách hoạt động của phần mềm 46](#_Toc107057513)

[4.2 Sơ đồ phân rã chức năng 47](#_Toc107057514)

[4.2 Sơ đồ Use Case tổng quát 48](#_Toc107057515)

[4.3 Giao diện phần mềm 50](#_Toc107057516)

[4.3.1 Giao diện màn hình chính khi mở phần mềm 50](#_Toc107057517)

[4.3.2 Giao diện chọn project 51](#_Toc107057518)

[4.3.3 Giao diện lỗi khi chọn sai assembly 52](#_Toc107057519)

[4.3.4 Giao diện assembly hợp lệ 53](#_Toc107057520)

[4.3.5 Giao diện chọn đường dẫn lưu 54](#_Toc107057521)

[4.3.6 Giao diện chọn đường dẫn lưu thành công 55](#_Toc107057522)

[4.3.7 Giao diện chọn config mã hóa 56](#_Toc107057523)

[4.3.8 Giao diện build 57](#_Toc107057524)

[4.3.9 Giao diện build thành công 58](#_Toc107057525)

[4.3.10 Giao diện build thất bại 59](#_Toc107057526)

[4.4 Thực nghiệm và kết quả 59](#_Toc107057527)

[KẾT LUẬN 64](#_Toc107057528)

[1 Kết quả đạt được 64](#_Toc107057529)

[1.1 Về mặt kiến thức 64](#_Toc107057530)

[1.2 Về mặt ứng dụng 64](#_Toc107057531)

[1.3 Về mặt con người 64](#_Toc107057532)

[2 Tồn tại hạn chế 65](#_Toc107057533)

[3 Hướng phát triển 65](#_Toc107057534)

[TÀI LIỆU THAM KHẢO VÀ PHỤ LỤC 66](#_Toc107057535)

# **DANH MỤC CHỮ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Mô tả** | **Ý nghĩa** | **Ghi chú** |
| 1 | IL | Intermediate Language |  |
| 2 | CIL | Common Intermediate Language |  |
| 3 | MSIL | Microsoft Intermediate Language |  |
| 4 | OOP | Object Oriented Programming |  |
| 5 | PE | Portable Executable |  |
| 6 | JIT | Just-in-Time |  |
| 7 | GAC | Global Assembly Cache |  |

# **MỞ ĐẦU**

## 1 Lý do chọn đề tài

Việc dẫn em đến chọn đề tài này xuất phát vào năm ngoái. Bản thân em đã viết nên một chương trình .NET sử dụng để chỉnh sửa một số game. Nó miễn phí sử dụng nhưng một số chức năng cao cấp thì phải buộc người dùng bỏ tiền túi để mua. Trong 2 tuần đầu xuất bản sản phẩm, em đã thấy được phiên bản “cracked” của mình được share nhiều nơi. Những phiên bản này làm cho phần mềm cho phép sử dụng những chức năng phải bỏ tiền một cách miễn phí. Sau khi tìm hiểu nhanh thì em nhận ra lý do cơ bản đã khiến cho việc này xảy ra: dịch ngược mã nguồn.

Với công nghệ phát triển theo từng ngày, việc hoạt động của những kẻ tấn công, hacker,... ngày càng phổ biến. Họ nhắm vào nhiều phần mềm với nhiều thủ thuật dùng để lấy mã nguồn, hoặc phá phần mềm bằng nhiều cách như reverse engineering, cracking,...

Reverse engineering là kỹ thuật dịch ngược một chương trình thành mã nguồn có thể đọc được, và điều này rất nguy hiểm. Kẻ tấn công có thể sử dụng mã nguồn đoạt được để trục lợi cho bản thân hoặc gây tiềm năng thiệt hại đối với phần mềm và nhiều cá nhân.

Đối với những ngôn ngữ bậc thấp như C++, Rust,... những phần mềm viết bằng ngôn ngữ này sẽ được chuyển đổi trực tiếp sang mã native hay gọi là mã máy tính (machine code). Những kẻ tấn công nếu muốn lấy được mã nguồn thì họ phải sử dụng trình disassembler, trình dịch này dịch mã máy thành mã hợp ngữ (assembly), nhưng làm việc với hợp ngữ mà từ nó rút ra được cấu trúc, logic hoạt động của phần mềm là một việc cực kỳ khó nếu không nói bất khả thi. Vì thế những phần mềm viết bằng C++, Rust,... nó đã được bảo vệ từ khi thành sản phẩm.

Mặc khác, đối với ngôn ngữ như C# và Java cụ thể ở đây là C#, nó không hoạt động như C++. Khi những phần mềm viết bằng C# được biên dịch, chúng sẽ được dịch sang một ngôn ngữ trung gian, gọi là Intermediate Language, đối với C# là Microsoft Intermediate Language. Phần mềm .NET được biên dịch thành ngôn ngữ MSIL ở quá trình compile và nó chỉ được dịch sang ngôn ngữ máy khi ở quá trình run-time. Vì thế kẻ tấn công hoặc người dùng có thể dễ dàng lấy được mã nguồn bằng cách dịch ngược ngôn ngữ MSIL thành C#. Hiện nay có rất nhiều phần mềm hỗ trợ cho quá trình này. Một người dùng không có kiến thức chuyên môn vẫn có thể lấy được mã nguồn của phần mềm .NET của bạn.

Điều này có thể hoặc không ảnh hưởng đến bạn nếu bạn không quan tâm hoặc phần mềm bạn viết thuộc dạng mã nguồn mở. Nhưng nếu phần mềm bạn viết có nhiều thông tin bảo mật, hoặc nhiều chức năng nâng cao, hoặc liên quan đến kinh doanh,... Việc bảo mật cho mã nguồn của bạn mà điều cần thiết. Vậy làm sao để bảo mật được mã nguồn nếu ngôn ngữ MSIL có thể dễ dàng dịch ngược lại C#? Chính là mã hóa mã nguồn của bạn trước khi chúng được dịch thành MSIL. Vâng, đấy là đề tài của em, phần mềm có tên khá dễ thương là Obfus**cat**tor NET, có logo là một con mèo.



## 2 Mục tiêu nghiên cứu

* Nghiên cứu sử dụng ngôn ngữ C# và công nghệ .NET
* Nghiên cứu kỹ thuật và thuật toán trong mã hóa mã nguồn
* Nghiên cứu quy trình hoạt động của ngôn ngữ trung gian
* Nghiên cứu các kỹ thuật bảo vệ phần mềm chuyên sâu
* Nghiên cứu quy trình triển khai xây dựng phần mềm
* Cài đặt và thực nghiệm phần mềm
* Tương lai xa: Nghiên cứu và phát triển kỹ thuật mã hóa

## 3 Phạm vi đề tài

* Nghiên cứu vận dụng kiến thức đã được học và kinh nghiệm tham gia phát triển phần mềm .NET
* Phát triển phần mềm và thuật toán cao hơn khả năng của bản thân bên cạnh sử dụng nhiều công nghệ mới bên ngoài trường lớp

## 4 Cấu trúc bài báo cáo

* **Chương 1**: Cơ sở lý thuyết
* **Chương 2**: Phân tích bài toán
* **Chương 3**: Phân tích, nghiên cứu và thiết kế các kỹ thuật và thuật toán mã hóa
* **Chương 4**: Phân tích, thiết kế hệ thống phần mềm và thực nghiệm

# **CHƯƠNG 1: CƠ SỞ LÝ THUYẾT**

## 1.1 Tổng quan về Ngôn Ngữ Lập Trình C# [1]

### 1.1.1 Giới thiệu về ngôn ngữ

C# (hay C Sharp, đọc là “xi-thăng” hoặc “xi-sáp”) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000 (được phát triển bởi Anders Hejlsberg và team của ông trong khi phát triển .NET Framework). C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java.

Trong các ứng dụng Windows truyền thống, mã nguồn chương trình được biên dịch trực tiếp thành mã thực thi của hệ điều hành. Trong các ứng dụng sử dụng .NET Framework, mã nguồn chương trình (C#, VB.NET) được biên dịch thành mã ngôn ngữ trung gian MSIL (Microsoft intermediate language). Sau đó mã này được biên dịch bởi Common Language Runtime (CLR) để trở thành mã thực thi của hệ điều hành. Hình bên dưới thể hiện quá trình chuyển đổi MSIL code thành native code.

Diagram

Description automatically generated

**Hình 1.1** Cách hoạt động của C#

C# với sự hỗ trợ mạnh mẽ của .NET Framework giúp cho việc tạo một ứng dụng Windows Forms hay WPF (Windows Presentation Foundation), phát triển game, ứng dụng Web, ứng dụng Mobile trở nên rất dễ dàng.

### 1.1.2 Lý do chọn ngôn ngữ lập trình C#

C# là một ngôn ngữ lập trình hướng đối tượng đơn giản, hiện đại. Mục đích của C# là phát triển ngôn ngữ lập trình không chỉ dễ học mà còn hỗ trợ chức năng hiện đại cho mọi loại hình phát triển phần mềm. Ngoài ra C# còn có:

* Một ngôn ngữ được tối ưu hóa cho Windows
* Framework mạnh mẽ
* Thư viện nugget rất lớn
* Cộng đồng lớn và phát triển rất nhanh
* Công cụ linh hoạt
* Hỗ trợ đa nền tảng

### 1.1.2 Các tính năng của C#

Mặc dù các cấu trúc của C# trực tiếp tuân theo các ngôn ngữ bậc cao khác, C và C++ và là một ngôn ngữ lập trình hướng đối tượng. Nó có rất nhiều điểm tương đồng với Java, có nhiều tính năng lập trình khiến nó trở nên hấp dẫn đối với các lập trình viên khác nhau trên thế giới.

Dưới đây là danh sách một số tính năng quan trọng của C#:

* + Boolean
  + Thu gom rác tự động (garbage collection)
  + Thư viện tiêu chuẩn (standard library)
  + Gắn các phiên bản (mouting versions)
  + Thuộc tính và sự kiện
  + Quản lý các đại biểu và sự kiện
  + Dễ sử dụng
  + Indexers
  + Biên soạn có điều kiện (conditional compilation)
  + Đa luồng đơn giản (simple multi-threading)
  + Biểu thức LINQ và Lambda

Là một ngôn ngữ hướng đối tượng, C# hỗ trợ thuyết đóng gói, kế thừa và đa hình. Tất cả các biến và phương thức, cùng với Main process, entry point của ứng dụng, được đóng gói trong định nghĩa của lớp. Một lớp có thể kế thừa đơn giản từ lớp đầu tiên nhưng có thể triển khai một số giao diện. Các phương thức ghi đè các phương thức ảo trong lớp chính (cha) yêu cầu từ khóa ghi đè như một kỹ thuật để tránh định nghĩa lại ngoài kế hoạch. Trong C#, một cấu trúc giống như một lớp nhẹ; là một kiểu được gán cho ngăn xếp có thể thực thi các giao diện nhưng không tương thích với tính kế thừa.

Tích lũy các nguyên tắc hướng đối tượng chung này, C# tạo điều kiện thuận lợi cho việc phát triển các thành phần phần mềm thông qua một số cấu trúc ngôn ngữ sáng tạo, bao gồm những điều sau:

* + Chữ ký (signatures) của các phương thức được đóng gói được gọi là đại biểu (delegates), cho phép thông báo về các sự kiện thuộc loại an toàn (safe type).
  + Thuộc tính, cung cấp như là người đánh giá cho các biến thành viên riêng.
  + Các thuộc tính, cung cấp siêu dữ liệu khai báo về các loại trong thời gian chạy.
  + Nhận xét tài liệu XML trực tuyến.
  + Truy vấn tích hợp ngôn ngữ (LINQ) cung cấp khả năng truy vấn tích hợp trong nhiều nguồn dữ liệu khác nhau.

Nếu một lập trình viên phải tương tác với các phần mềm Windows khác nhau, như các đối tượng COM hoặc các DLL Win32 bản địa (native), thì lập trình viên có thể thực hiện việc này trong C# thông qua một thủ tục được gọi là “Interoperation”. Interop cho phép các chương trình C# thực hiện gần như mọi thứ mà một ứng dụng C++ gốc có thể thực hiện. C# thậm chí còn xử lý các con trỏ và khái niệm mã “không an toàn” cho những phần mà việc truy cập trực tiếp vào bộ nhớ là hoàn toàn quan trọng.

Quá trình xây dựng C# rất đơn giản so với C và C++ và rất linh hoạt hơn trong Java. Không có tệp tiêu đề bổ sung và các phương thức và kiểu không cần thiết phải khai báo theo một định dạng cụ thể hoặc một thứ tự cụ thể. Tệp nguồn C# có thể mô tả bất kỳ lượng lớp, cấu trúc, giao diện và sự kiện nào.

### 1.1.3 Đặc trưng của ngôn ngữ lập trình C#

**Đơn giản:** C# loại bỏ một vài sự phức tạp và rối rắm của những ngôn ngữ như Java và C++, bao gồm việc loại bỏ những macro, những template, đa kế thừa, và lớp cơ sở ảo (virtual base class).

Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++. Nếu chúng ta thân thiện với C và C++ hoặc thậm chí là Java, chúng ta sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và những chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để làm cho ngôn ngữ đơn giản hơn.

**Hiện đại:** Lập trình hướng đối tượng (OOP: Object-oriented programming) là một phương pháp lập trình có 4 tính chất. Đó là tính trừu tượng (abstraction), tính đóng gói (encapsulation), tính đa hình (polymorphism) và tính kế thừa (inheritance). C# hỗ trợ cho chúng ta tất cả những đặc tính trên.

**Là ngôn ngữ ít từ khóa:** C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì sẽ mạnh hơn. Điều này không phải sự thật, ít nhất là trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào.

**Mã nguồn mở và nhanh:** Tính C# là ngôn ngữ lập trình mã nguồn mở .NET Foundation, được quản lý và điều hành độc lập với Microsoft. Thông số kỹ thuật ngôn ngữ C#, trình biên dịch và các công cụ liên quan là các dự án mã nguồn mở trên Github. Trong khi thiết kế các tính năng của ngôn ngữ C# được dẫn dắt bởi Microsoft, cộng đồng mã nguồn mở rất tích cực trong việc phát triển và cải tiến ngôn ngữ.

C# nhanh so với một số ngôn ngữ lập trình cấp cao khác. Các phiên bản C# sau này có nhiều cải tiến về hiệu suất, tại thời điểm này là phiên bản 11.

**Đa nền tảng:** C# là ngôn ngữ lập trình đa nền tảng. Bạn có thể xây dựng các ứng dụng .NET có thể được triển khai trên các nền tảng Windows, Linux và Mac.

**An toàn và hiệu quả:** C# là một ngôn ngữ an toàn. C# không cho phép chuyển đổi kiểu dữ liệu có thể dẫn đến mất dữ liệu hoặc các vấn đề khác. C# cho phép các nhà phát triển viết code an toàn. C# cũng tập trung vào việc viết code hiệu quả.

**Đa năng:** Chúng ta có thể sử dụng C# để xây dựng các ứng dụng phần mềm hiện đại ngày nay. C# có thể được sử dụng để phát triển tất cả các loại ứng dụng bao gồm ứng dụng Windows, các thành phần và thư viện, Service và API, ứng dụng Web, ứng dụng di động, ứng dụng đám mây và video game.

**Tiến hóa:** C# 11.0 là phiên bản mới nhất của C#. Nếu bạn nhìn vào lịch sử ngôn ngữ C#, C# đang phát triển nhanh hơn bất kỳ ngôn ngữ nào khác. Dưới sự quản lý của Microsoft và sự hỗ trợ mạnh mẽ từ cộng đồng. C# ban đầu được thiết kế để viết các ứng dụng máy khách Windows, nhưng ngày nay, C# có thể làm được khá nhiều thứ như ứng dụng console, ứng dụng đám mây và phần mềm học máy hiện đại.

### 1.1.4 Ưu điểm của C#

Dựa vào những tính năng va đặc trưng nổi bật trên, ta rút ra C# còn có nhiều ưu điểm như:

* C# là kiểu an toàn.
* Được tối ưu hóa về mặt bộ nhớ và có thể tự giải phóng một cách thông minh dựa vào khả năng automatic garbage collection của C#.
* Concept của assembly giải quyết rất tốt được vấn để kiểm soát phiên bản (versioning control).
* Dễ đàng phát triển.
* Có lượng lớn thư viện và nugget packages để tăng cao hiệu xuất.
* Đa nền tảng, miễn máy có cài đặt .NET Framework.
* Hỗ trợ cho hệ thống phân tán (distributed system).
* Just In Time (JIT): Ngoài việc biên dịch ngôn ngữ trung gian sang ngôn ngữ máy, JIT còn cho phép hot reloading công việc, giúp phát triển code nhanh và tiện lợi hơn (được hiểu như việc debug trong ngôn ngứ lập trình khác là debug hàm nào chạy hàm đấy thì ở đây nó sẽ viết đến đây thì sẽ biên dịch đến đấy).

### Nhược Điểm của C#

* Đây không phải là ngôn ngữ bậc thấp, nên việc tương tác với những tính năng “low-level” không thể thực thi một cách trực tiếp thông qua phần cứng hoặc trình điều khiển (driver). Phải thực thi một cách gián tiếp.
* C# không đi kèm với một trình biên dịch (compiler) độc lập có thể diễn giải một cách đơn giản các cấp độ ngôn ngữ tối đa cho kiến trúc phần cứng cơ bản của một trình hợp dịch (assembly) thuần túy. Nó sử dụng mã byte của nó và trình biên dịch JIT được kết hợp chặt chẽ vào .NET Framework và là xương sống của cấu trúc .NET như một phần đi giữa mã máy ở vị trí giao tiếp trực tiếp với phần cứng.
* Cần phải cài đặt .NET Framework trên máy để có thể sử dụng.

## Tổng quan về .NET [2]

### Khái niệm .NET

.NET Framework được Microsoft đưa ra chính thức từ năm 2002. .NET Framework chỉ hoạt động trên Windows. Những nền tảng ứng dụng như WPF, Winforms, ASP.NET(1-4) hoạt động dựa trên .NET Framework.

Mono là phiên bản cộng đồng nhằm mang .NET đến những nền tảng ngoài Windows. Mono được phát triển chủ yếu nhằm xây dựng những ứng dụng với giao diện người dùng và được sử dụng rất rộng rãi: Unity Game, Xamarin…

Cho đến năm 2013, Microsoft định hướng đi đa nền tảng và phát triển .NET core. .NET core hiện được sử dụng trong các ứng dụng Universal Windows platform và ASP.NET Core. Từ đây, C# có thể được sử dụng để phát triển các loại ứng dụng đa nền tảng trên các hệ điều hành khác nhau (Windows, Linux, MacOS,…)

Graphical user interface

Description automatically generated

**Hình 1.2** Sơ đồ .NET

### 1.2.2 Đặc điểm nổi bật của .NET

**Thư viện lập trình lớn:** .NET sở hữu những thư viện lập trình rất lớn, có khả năng hỗ trợ tối đa cho việc tạo lập, xây dựng các ứng dụng web; Truy cập, kết nối các CSDL, cấu trúc dữ liệu; Lập trình giao diện,… Những đặc trưng này khiến nó trở thành nền tảng chính được rất nhiều lập trình viên lựa chọn.

**Năng suất làm việc cao:** Lập trình, thiết kế ứng dụng với .NET tiết kiệm rất nhiều thời gian bởi nó cung cấp sẵn khá nhiều thành phần dùng trong thiết kế. Chỉ cần bạn biết cách sử dụng cũng như tùy biến những đoạn code có sẵn này cho phù hợp với dự án của mình, việc lập trình sẽ trở nên đơn giản hơn bao giờ hết.

**Biến đổi linh hoạt nhờ kiến trúc “ghép nối lỏng”:** .NET được thiết kế, xây dựng với khả năng biến đổi linh hoạt nhờ cấu trúc ‘ghép nối lỏng’. Điều này mang lại nhiều lợi thế về năng suất.

**Đa ngôn ngữ:** .NET là nền tảng hỗ trợ cho đa ngôn ngữ. Lập trình viên có thể sử dụng nền tảng này để tạo lập các ứng dụng web bằng nhiều ngôn ngữ lập trình khác nhau mà vẫn đảm bảo khả năng tích hợp của nó.

**Bảo mật cao:** .NET có phần kiến trúc bảo mật được thiết kế theo dạng từ dưới lên. Điều này giúp bảo vệ dữ liệu cũng như các ứng dụng khỏi sự đe dọa của hacker thông qua mô hình bảo mật tinh vi là evidence-based.

**Tận dụng mọi thứ có sẵn trong hệ điều hành Windows:** Windows sở hữu rất nhiều dịch vụ có khả năng hoạt động trên mọi nền tảng như truy cập dữ liệu, mô hình dạng đối tượng thành phần, giao diện người dùng tương tác, bảo mật tích hợp và cả giám sát giao dịch. .NET tận dụng những dịch vụ này để đơn giản hóa cách sử dụng, giúp lập trình trên nền tảng này trở nên dễ dàng hơn.

## 1.3 Tổng quan về mã hóa và trộn mã (obfuscation) [3]

### Khái niệm

Obfuscation là quá trình xáo trộn và mã hóa mã nguồn phần mềm của bạn để không thể dễ dàng bị dịch ngược. Mục tiêu là ngăn chặn tất cả các tin tặc bình thường và càng nhiều tin tặc nghiêm trọng càng tốt cố gắng kiểm tra và bẻ khóa mã của bạn. Do đó, các tác vụ đều được xử lý và thực thi ở chế độ nền (background processing).

Trong phát triển phần mềm, obfuscation là hành động tạo ra mã nguồn hoặc mã máy mà con người khó hiểu. Giống như sự xáo trộn trong ngôn ngữ tự nhiên, nó có thể sử dụng các biểu thức vòng vo không cần thiết để soạn các câu lệnh. Các nhà lập trình có thể cố tình làm xáo trộn mã để che giấu mục đích của nó (bảo mật thông qua sự che đậy) hoặc logic của nó hoặc các giá trị tiềm ẩn được nhúng trong đó, chủ yếu, để ngăn chặn việc giả mạo, ngăn chặn kỹ thuật đảo ngược hoặc thậm chí để tạo ra một câu đố hoặc thách thức giải trí cho ai đó đang đọc mã nguồn mã số. Điều này có thể được thực hiện thủ công hoặc bằng cách sử dụng một công cụ tự động, công cụ sau này là kỹ thuật được ưa chuộng trong công nghiệp.

### Lý do nên mã hóa

Nếu bạn đang triển khai mã trong các môi trường không đáng tin cậy, nơi bạn muốn bảo vệ mã nguồn của mình, thì bạn hầu như luôn phải sử dụng ít nhất một trình mã hóa cơ bản để đổi tên các hàm, phương thức và thuộc tính nhằm làm cho việc dịch ngược tốn nhiều công sức hơn.

Nếu bạn thực sự không cần ai có thể dịch ngược ứng dụng của mình, bạn có thể sử dụng một trình làm xáo trộn khó xâm nhập hơn, nhưng thực sự bạn nên cân nhắc xem vấn đề có được giải quyết tốt hơn hay không bằng cách chuyển sang một ngôn ngữ không có vấn đề này, chẳng hạn như C++ hoặc Rust.

### Cách hoạt động của trộn mã hay mã hóa

Sự xáo trộn trong mã máy tính sử dụng các cụm từ vòng vo phức tạp và logic dư thừa để làm cho mã khó hiểu đối với người đọc. Mục đích là để đánh lạc hướng người đọc với cú pháp phức tạp của những gì họ đang đọc và khiến họ khó xác định nội dung thực sự của tin nhắn.

Với mã máy tính, người đọc có thể là một người, một thiết bị tính toán hoặc một chương trình khác. Obfuscation cũng được sử dụng để đánh lừa các công cụ chống vi-rút và các chương trình khác dựa nhiều vào chữ ký số để diễn giải mã. Các trình biên dịch có sẵn cho các ngôn ngữ như Java, hệ điều hành như Android và iOS, và các nền tảng phát triển như .NET. Họ có thể tự động đảo ngược mã nguồn; obfuscation nhằm mục đích gây khó khăn cho các chương trình này trong việc dịch ngược của chúng.

Sau đây là đoạn mã ví dụ về mã JavaScript bình thường:

var greeting = 'Hello World';

greeting = 10;

var product = greeting \* greeting;

Đoạn mã tương tự ở dạng xáo trộn trông như thế này:

var \_0x154f=['98303fgKsLC','9koptJz','1LFqeWV','13XCjYtB','6990QlzuJn','87260lXoUxl','2HvrLBZ','15619aDPIAh','1kfyliT','80232AOCrXj','2jZAgwY','182593oBiMFy','1lNvUId','131791JfrpUY'];var \_0x52df=function(\_0x159d61,\_0x12b953){\_0x159d61=\_0x159d61-0x122;var \_0x154f4b=\_0x154f[\_0x159d61];return \_0x154f4b;};(function(\_0x19e682,\_0x2b7215){var \_0x5e377c=\_0x52df;while(!![]){try{var \_0x2d3a87=-parseInt(\_0x5e377c(0x129))\*parseInt(\_0x5e377c(0x123))+-parseInt(\_0x5e377c(0x125))\*parseInt(\_0x5e377c(0x12e))+parseInt(\_0x5e377c(0x127))\*-parseInt(\_0x5e377c(0x126))+-parseInt(\_0x5e377c(0x124))\*-parseInt(\_0x5e377c(0x12f))+-parseInt(\_0x5e377c(0x128))\*-parseInt(\_0x5e377c(0x12b))+parseInt(\_0x5e377c(0x12a))\*parseInt(\_0x5e377c(0x12d))+parseInt(\_0x5e377c(0x12c))\*parseInt(\_0x5e377c(0x122));if(\_0x2d3a87===\_0x2b7215)break;else \_0x19e682['push'](\_0x19e682['shift']());}catch(\_0x22c179){\_0x19e682['push'](\_0x19e682['shift']());}}}(\_0x154f,0x1918c));var greeting='Hello\x20World';greeting=0xa;var product=greeting\*greeting;

Phiên bản bị xáo trộn gần như không thể theo dõi bằng mắt người.

### Các kỹ thuật xáo trộn cơ bản

Obfuscation liên quan đến một số phương pháp khác nhau. Thông thường, nhiều kỹ thuật được sử dụng để tạo hiệu ứng phân lớp.

Các chương trình được viết bằng ngôn ngữ phần mềm được biên dịch, chẳng hạn như C# và Java, dễ làm xáo trộn hơn. Điều này là do họ tạo ra các hướng dẫn cấp trung gian thường dễ đọc hơn. Ngược lại, C++ khó bị xáo trộn hơn, vì nó biên dịch thành mã máy, điều này khó hơn đối với mọi người.

Một số kỹ thuật xáo trộn phổ biến bao gồm:

* + **Đổi tên (Renaming):** Trình obfuscator thay đổi các phương thức và tên của các biến. Các tên mới có thể bao gồm các ký tự không in được hoặc không nhìn thấy.
  + **Đóng gói (Packaging):** Điều này sẽ nén toàn bộ chương trình để làm cho mã không thể đọc được.
  + **Kiểm soát dòng chảy (Control Flow):** Mã dịch ngược được tạo ra để trông giống như logic spaghetti, không có cấu trúc và khó duy trì mã khi dòng suy nghĩ bị che khuất. Kết quả từ mã này không rõ ràng và thật khó để biết điểm của mã là gì bằng cách nhìn vào nó.
  + **Chuyển đổi mẫu hướng dẫn:** Cách tiếp cận này sử dụng các hướng dẫn phổ biến do trình biên dịch tạo ra và hoán đổi chúng để lấy các hướng dẫn phức tạp hơn, ít phổ biến hơn để thực hiện hiệu quả điều tương tự.
  + **Chèn mã giả:** Mã giả có thể được thêm vào chương trình để làm cho chương trình khó đọc hơn và thiết kế ngược, nhưng nó không ảnh hưởng đến logic hoặc kết quả của chương trình.
  + **Metadata hoặc loại bỏ mã không sử dụng:** Mã và metadata không sử dụng cung cấp cho người đọc thêm thông tin về chương trình, giống như chú thích trên tài liệu Word, có thể giúp họ đọc và gỡ lỗi chương trình. Việc xóa metadata và mã không sử dụng khiến người đọc có ít thông tin hơn về chương trình và mã của nó.
  + **Chèn vị ngữ đục:** Vị từ trong mã là một biểu thức logic đúng hoặc sai. Các vị từ mờ là các nhánh có điều kiện - hoặc các câu lệnh if-then - trong đó kết quả không thể dễ dàng xác định được bằng phân tích thống kê. Việc chèn một vị từ không rõ ràng sẽ giới thiệu mã không cần thiết không bao giờ được thực thi nhưng gây khó hiểu cho người đọc đang cố gắng hiểu đầu ra được dịch ngược.
  + **Chống debugger:** Các kỹ sư phần mềm hợp pháp và tin tặc sử dụng các công cụ gỡ lỗi để kiểm tra từng dòng mã. Với những công cụ này, các kỹ sư phần mềm có thể phát hiện ra các vấn đề với mã và tin tặc có thể sử dụng chúng để thiết kế ngược mã. Các chuyên gia bảo mật CNTT có thể sử dụng các công cụ chống gỡ lỗi để xác định khi nào một tin tặc đang chạy chương trình gỡ lỗi như một phần của cuộc tấn công. Tin tặc có thể chạy các công cụ chống gỡ lỗi để xác định khi nào một công cụ gỡ lỗi đang được sử dụng để xác định những thay đổi mà chúng đang thực hiện đối với mã.
  + **Chống làm giả:** Các công cụ này phát hiện mã đã bị giả mạo và nếu nó đã được sửa đổi, nó sẽ dừng chương trình.
  + **Mã hóa string:** Phương pháp này sử dụng mã hóa để ẩn các chuỗi trong tệp thực thi và chỉ khôi phục các giá trị khi chúng cần thiết để chạy chương trình. Điều này gây khó khăn cho việc duyệt qua một chương trình và tìm kiếm các chuỗi cụ thể.
  + **Chuyển vị mã:** Đây là sự sắp xếp lại các quy trình và các nhánh trong mã mà không có ảnh hưởng rõ ràng đến hành vi của nó.

### Ưu điểm của mã hóa mã nguồn

**Chạy nhanh hơn:** Code càng ít, thì ứng dụng chạy càng nhanh. Nghe hơi vô lý đối với mã hoá, nhưng trong nhiều trường hợp nhất định, một số thuật toán như pruning hoặc minification có thể giảm tải được lượng lớn code của chương trình.

**Giảm bộ nhớ phải sử dụng:** Trong các ngôn ngữ cổ thông dịch tại run-time (thường được gọi là script), giống như các phiên bản cũ của BASIC, các chương trình được thực thi nhanh hơn và tốn ít RAM hơn nếu chúng sử dụng các tên biến ký tự đơn, tránh nhận xét và chỉ chứa các ký tự trống cần thiết (tóm lại, càng ngắn càng nhanh).

**Bảo vệ bí mật thương mại:** Trong trường hợp mã nguồn của chương trình phải được gửi đến người dùng, ví dụ như JavaScript trong một trang web, thì bất kỳ bí mật thương mại, cơ chế cấp phép hoặc tài sản trí tuệ nào khác có trong chương trình đều có thể truy cập được đối với người dùng. Sự xáo trộn làm cho việc hiểu mã và sửa đổi nó trở nên khó khăn hơn.

Các chương trình máy tính để bàn đôi khi bao gồm các tính năng giúp làm xáo trộn mã của chúng. Một số chương trình có thể không lưu trữ toàn bộ mã của chúng trên đĩa và có thể kéo một phần mã nhị phân của chúng qua web trong thời gian chạy. Họ cũng có thể sử dụng tính năng nén và / hoặc mã hóa, thêm các bước bổ sung vào quá trình tháo gỡ.

**Chống thuốc (cracking):** Trong những trường hợp như vậy, việc xáo trộn chương trình có thể khiến người dùng khó vượt qua các cơ chế cấp phép hoặc lấy thông tin mà nhà cung cấp chương trình muốn che giấu. Nó cũng có thể được sử dụng để làm cho việc hack các trò chơi nhiều người chơi khó hơn.

**Chống phát hiện bởi phần mềm chống vi-rút:** Đây là một ưu điểm đáng sợ của mã hóa mã. Các chương trình độc hại có thể sử dụng phương pháp gây nhiễu để che giấu những gì chúng thực sự đang làm. Hầu hết người dùng thậm chí không đọc các chương trình như vậy; và những công cụ thường có quyền truy cập vào các công cụ phần mềm có thể giúp họ hoàn tác sự xáo trộn, do đó, chiến lược này có hiệu quả hạn chế.

### Nhược điểm

* Mặc dù mã hóa có thể làm cho việc đọc, viết và dịch ngược một chương trình trở nên khó khăn và tốn thời gian, nhưng nó không nhất thiết sẽ khiến nó không thể thực hiện được.
* Nó tăng độ phức tạp và thời gian để build.
* Nó có thể gây debugging trở nên khó khăn nếu phần mềm được mã hóa một cách tối đa và phức tạp.
* Một khi mã trở thành phần mềm discontinued và không còn được duy trì, những người có sở thích có thể muốn duy trì chương trình, thêm mod hoặc hiểu rõ hơn về nó. Nếu assembly đã được mã hóa thì nó khiến họ khó làm những việc hữu ích với phần mềm đấy.
* Một số loại mã hóa nhất định (tức là mã không chỉ là mã nhị phân ở local và yêu cầu tải xuống các tệp nhị phân nhỏ từ máy chủ web nếu cần) có thể làm giảm hiệu suất và / hoặc yêu cầu Internet để sử dụng phần mềm.

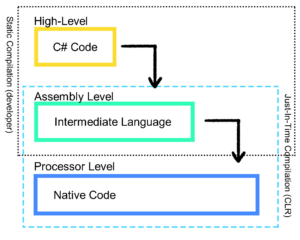
# **CHƯƠNG 2: PHÂN TÍCH BÀI TOÁN**

## Lý do nên mã hóa .NET

Như đã nêu ở phần lý do chọn đề tài. Mã hóa hoặc xáo trộn là một chủ đề quan trọng trong lĩnh vực phát triển .NET. Trong khi một số người xung quanh các trang web trên mạng có thể cho rằng việc mã hóa là một việc làm vô ích, không cần thiết thì họ chả biết gì cả.

Tóm lại, bất kỳ mã nào được viết trên .NET Framework sẽ được biên dịch sang ngôn ngữ trung gian của IL hoặc .NET, là ngôn ngữ nằm giữa mã máy và các ngôn ngữ cấp cao hơn như VB.NET hoặc C# hoặc thậm chí là Managed C++. Sau đó, mã IL chạy với CLR hoặc Common Language Runtime, về cơ bản đây là thứ cho phép các nhà phát triển sử dụng ngôn ngữ phù hợp nhất với họ (VB.NET, C#, Managed C++, trong số các ngôn ngữ khác) và vẫn dựa trên thực tế là nó sẽ được chuyển đổi thành gần giống mã IL và hoạt động giống hệt nhau khi đến thời điểm thực thi mã.

Nhu cầu về mã hóa có liên quan đến bản chất của .NET Framework. Khuôn khổ .NET được xây dựng với một bộ lớp cực kỳ ấn tượng cho phép mã xem xét các hợp ngữ trong thời gian chạy, về cơ bản cho phép một chương trình tự truy vấn các kiểu dữ liệu và thông tin. Khả năng này hầu hết được cung cấp bởi các lớp Reflection trong .NET. Thông qua các lớp này, một lập trình viên có thể tạo các kiểu mới trong thời gian chạy, truy vấn các đối tượng hiện có để tìm thông tin và thực hiện tất cả các việc cực kỳ mạnh mẽ, chẳng hạn như tự động tạo mã thực thi một cách nhanh chóng. Tuy nhiên, một lưu ý lớn đối với điều này là nó cũng cho phép lập trình viên viết một chương trình có thể ngang hàng với nội bộ của hợp ngữ .NET đã biên dịch và chuyển đổi mã IL đã biên dịch thành bất kỳ ngôn ngữ .NET nào mà anh ta có thể quan tâm. Ví dụ: Chương trình .NET Reflector của Lutz Roeder cho phép bạn xem xét mã vật lý của bất kỳ tổ hợp .NET nào. Vấn đề với điều này là về cơ bản, nó tiết lộ các công ty hoặc tài sản trí tuệ cá nhân của bạn với thế giới, hoặc ít nhất là với các nhà phát triển khác, những người có công cụ như .Net Reflector (nhiều công cụ trong số đó được cung cấp miễn phí). Nếu bạn chưa quen với mã hóa, mình khuyên bạn nên lấy chương trình này và mở một trong các tổ hợp .NET của riêng bạn mà bạn hoặc công ty của bạn có thể đã viết. Bạn có thể hơi ngạc nhiên về những gì bạn nhìn thấy đấy!



**Hình 2.1** Cách .NET được biên dịch

Mã hóa khá đơn giản (về mặt khái niệm), ý tưởng là thay đổi mã của bạn theo cách làm cho mã khó hiểu hơn đối với con người nếu họ nhìn vào nó bằng cách sử dụng phản chiếu .NET. Cách rõ ràng nhất để làm điều này là thay đổi tên của các biến lớp, biến cục bộ, tên phương thức riêng và được bảo vệ, v.v. sao cho hàm "GetName()" có lẽ trở thành "BWh157q()". Chương trình sẽ vẫn hoạt động theo cùng một cách, chỉ là một người có thể nhìn thấy tên hàm đó sẽ gặp khó khăn hơn trong việc tìm ra chính xác chức năng đó hoạt động như thế nào mà không cần xem mã nội bộ, điều này có lẽ cũng bị xáo trộn.

Một vài lưu ý chung về việc mã hóa một .NET assembly:

* + Assembly này có thể sẽ tăng hoặc thu nhỏ kích thước, điều này là do tất cả việc đổi tên sẽ xảy ra. Thông thường đây không phải là một vấn đề, nhưng nó có thể đưa ra các vấn đề của riêng mình trong những tình huống đặc biệt.
  + Những phương thức public khi mã hóa nói chung sẽ phá vỡ ứng dụng. Điều này xảy ra khi một số ứng dụng khác có thể gọi phương thức là "GetName()", nhưng bây giờ nó đã được thay đổi thành một thứ khác, liên kết đó hiện đã bị hỏng và chương trình gọi hiện không thể tìm thấy phương thức công khai đó.

Nếu bạn không gặp vấn đề với khả năng người khác biết mã của bạn, bạn không cần phải lo lắng về việc mã hóa nó. Nhưng nếu bạn đang bán một ứng dụng vì lợi nhuận, trừ khi nó được bán theo một cách độc đáo nào đó hoặc thu được lợi nhuận từ một số phương thức khác, bạn có thể muốn nghĩ về mã hóa, sớm hơn trong một dự án hơn là muộn hơn. Cách thức mà một ứng dụng được triển khai cũng sẽ giúp trả lời bạn có hay không.

Ví dụ, một người tình cờ thấy một ứng dụng rất hữu ích được viết bằng .NET và giống như nhiều sản phẩm khác, nó yêu cầu số sê-ri để đăng ký và mở khóa hết tiềm năng của nó. Anh ấy đã sử dụng NET Reflector đối với assemmbly đã tải xuống từ trang web của công ty (chưa được kích hoạt) và trong vòng 5-10 phút đã có thể tách phần nơi nó xác thực số sê-ri và tạo ra một license để mở khóa nhanh phần mềm dùng thử và đã đưa cho anh ta bản sao của chính mình về ứng dụng đã đăng ký. Mã chỉ đơn giản là không được mã hóa và thật không may, các nhà phát triển tại công ty đó không nhận thức được các vấn đề xung quanh nó, khi họ gặp khó khăn khi tạo ra một kế hoạch cấp phép khóa nối tiếp nhưng không bảo vệ mã của họ (hoặc ít nhất là phần xử lý với xác thực khóa nối tiếp) ở vị trí đầu tiên.

Điều quan trọng là phải hiểu rằng mã hóa và vô số các kỹ thuật khác có sẵn để thay đổi mã ứng dụng của bạn và bảo mật nó tốt hơn, không phải là điều dễ hiểu. Nếu một sản phẩm tuyên bố rằng nó có thể mã hóa mã của bạn đến mức không ai có thể hack nó, thì họ đang nói dối đấy. Mặc dù trong nhiều trường hợp, nó có thể thực hiện đúng lời hứa, nhưng đó là một tuyên bố rất ngây thơ, bởi vì như với bất cứ điều gì, nếu một người đủ quyết tâm, anh ta có thể bẻ khóa, hack hoặc đột nhập bất cứ thứ gì. Điểm thực sự của sự mã hóa là tạo ra đủ phiền phức cho những kẻ tấn công tương lai rằng nó sẽ tiêu tốn quá nhiều thời gian và năng lượng để làm phiền. Nếu mất 5-10 phút để hack một thứ gì đó, nhiều người có khả năng làm như vậy, nhưng nếu mất nhiều tháng hoặc nhiều năm để làm sáng tỏ một ứng dụng bị mã hóa nào đó, thì rất ít người sẽ bận tâm đến việc thiết kế ngược nó.

## Một số phần mềm hỗ trợ việc dịch ngược mã nguồn .NET [4]

Các chương trình viết bằng .NET có cấu trúc khác với các native PE file, cho nên các công cụ debug/disassembler thông thường không thể phát huy hết sức mạnh, mà muốn dịch ngược được các chương trình .NET và phân tích cấu trúc và mã nguồn của chúng, không gì tốt hơn bằng việc sử dụng các chương trình chuyên dụng dành riêng cho .NET, và mình sẽ giới thiệu trong phần này.

### 2.2.1 CFF Explorer

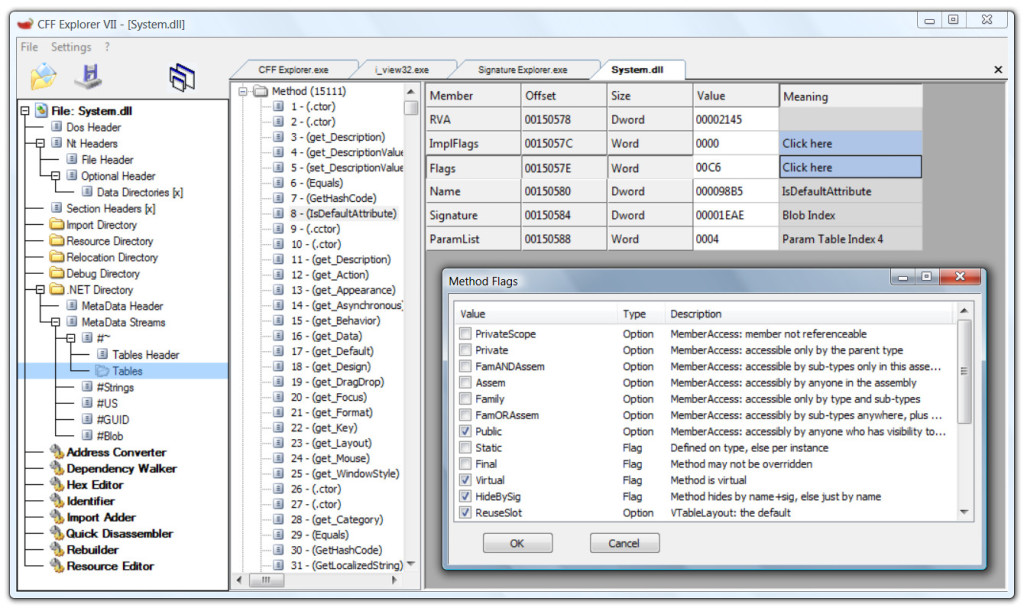
Đây là 1 chương trình rất đa năng nằm trong bộ Explore Suite của NTCore phát triển. Mặc dù đây chỉ là dự án ngoài lề được viết bởi Daniel Pistelli, tuy nhiên các tính năng của nó được đánh giá rất cao và hữu ích. Chương trình là một PE Editor đúng nghĩa với các tính năng sau (vì đây đa phần là các thuật ngữ chuyên môn nên khi dịch ra tiếng Việt có thể không sát nghĩa, nên để nguyên tiếng Anh):

* Special fields description and modification (.NET supported)
* PE Rebuilder (with Realigner, IT Binder, Reloc Remover, Strong Name Signature Remover, Image Base Changer)
* View and modification of .NET internal structures
* Support in the Resource Editor for .NET resources (dumpable as well)
* Hex Editor
* Quick Disassembler (x86, x64, MSIL)
* Dependency Walker

Đây cũng là chương trình đầu tiên hỗ trợ việc xem và thay đổi cấu trúc, thông số của .NET PE file, có thể thực hiện ngay cả khi trong máy không cài .NET Framework. Ta cũng có thể mở rộng khả năng hoạt động của chương trình bằng cachs viết các script cho chương trình tự động chạy, đó cũng là điểm mạnh rất đáng giá. Vậy nên, nếu bạn muốn thực hiện Reverse Engineering .NET, đây là công cụ bạn cần phải có trong bộ “đồ nghề” của mình.

+ **Điểm cộng**: Nhẹ, miễn phí, nhiều tính năng rất hữu ích, hỗ trợ hiệu quả .NET, khả năng mở rộng tốt.

+ **Điểm trừ**: Không.



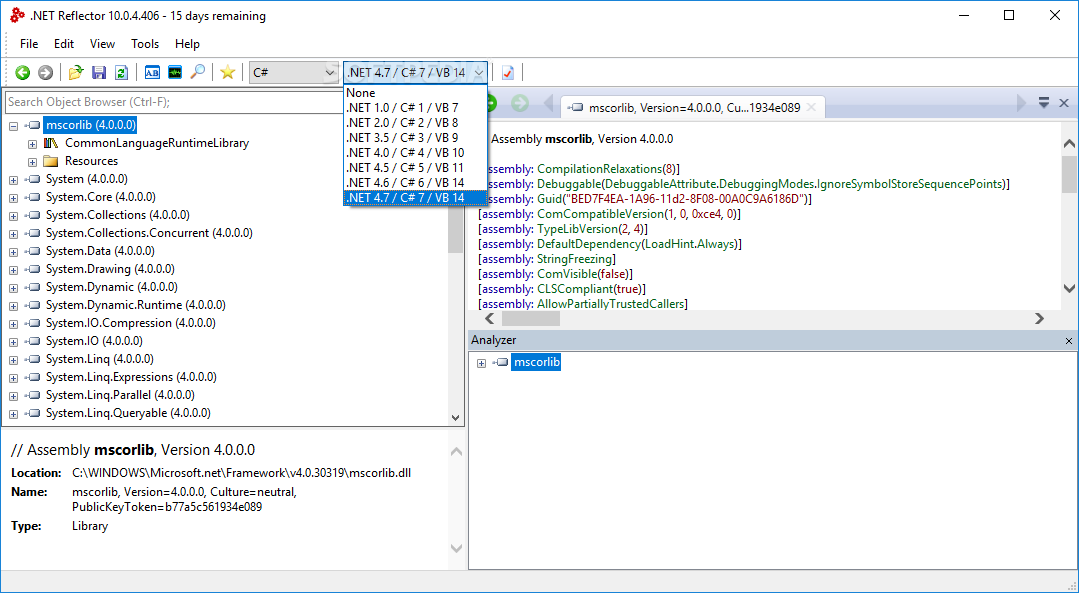
**Hình 2.1** Giao diện CFF Explorer

### 2.2.2 .NET Reflector

Nhắc đến .NET Reflector thì không phải là cái tên quá xa lạ đối với những ai đang nghiên cứu về .NET. Đây là công cụ phổ biến nhất trong việc dịch ngược .NET. Có một decompile engine chất lượng, có nhiều plugin/addon hỗ trợ, tính năng hữu ích, giao diện trực quan dễ sử dụng, có thể tích hợp vào trong Visual Studio, kèm theo đó là 1 “ông trùm” chống lung phía sau là Red-Gate, không lạ lắm khi .NET Reflector càng ngày càng trở lên mạnh mẽ và phổ biến hơn. Tính năng dịch ngược của .NET Reflector theo quan điểm cá nhân của mình là rất tốt, có thể đưa về 90-95% code gốc, và có thể browse code không khác lắm so với việc chúng ta đọc code trong Visual Studio. Tuy nhiên mình vẫn thích các phiên bản cũ của .NET Reflector khi Lulz Roeiier còn phát triển độc lập. Lúc đó .NET Reflector chạy rất nhẹ nhàng và chính xác, mặc dù không có nhiều tính năng hay ho như .NET Reflector hiện hành, và quan trọng nhất là nó “miễn phí”. Red-Gate đã mua lại .NET Reflector và biến nó thành một công cụ độc quyền của hãng, và bán với giá cao. Tất nhiên là cracker/reverser không thích điều này, và họ đã phát triển các công cụ thay thế khác với tính năng giống .NET Reflector và chạy nhẹ nhàng hơn rất nhiều.

+ **Điểm cộng:** Mạnh, phổ biến, dễ sử dụng, nhiều addon, khả năng tích hợp tốt với Visual Studio, nhiều tính năng hữu ích.

+ **Điểm trừ:** Nặng, mất phí, thiếu các tính năng chuyên dụng và nâng cao.



**Hình 2.2** Giao diện .NET Reflector

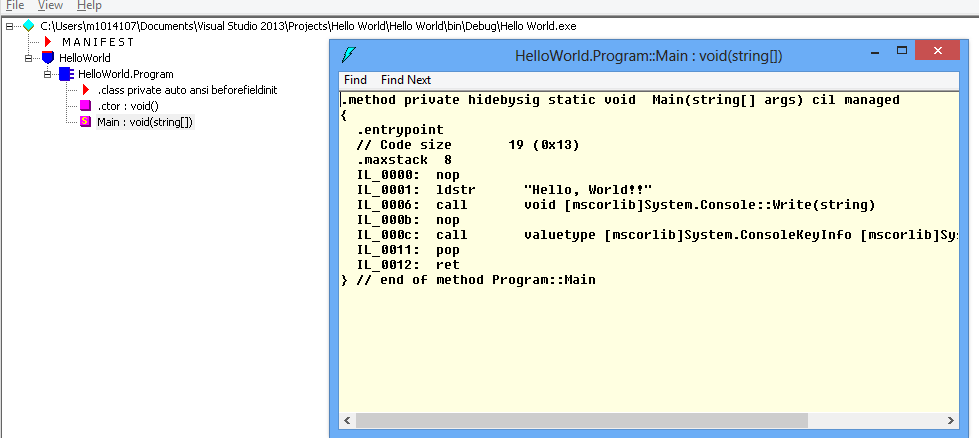
### 2.2.3 ILDASM

ILDASM là tên viết tắt của IL DisASseMbler. Đây là 1 công cụ decompile các file .NET của chính Microsoft tạo ra, nằm cùng trong bộ Visual Studio và Window SDK, có thể được gọi từ Visual Studio Command Prompt (nguồn: MSDN). Chức năng chính là dịch ngược chương trình .NET và có thể chuyển về dạng IL, sau đó save lại dưới dạng txt file. Chúng ta có thể sửa code từ file text này, và sau đó compile lại bằng một công cụ đi cùng với ILDASM là ILASM. ILDASM và ILASM (cũng nằm trong bộ Visual Studio).

Đúng như tên gọi của nó, ILDASM chỉ đơn thuần là 1 công cụ Disassemble không hơn không kém, và nhiệm vụ của nó chỉ là dịch ngược về mã IL, cho nên nó thiếu đi rất nhiều tính năng cần thiết như: code analysis, code search, multi-file disassemble,…

+ **Điểm cộng:** Nhẹ, có sẵn (nếu như có cài Visual Studio và Windows SDK), hiện thị code tốt, export code khá tốt

+ **Điểm trừ:** Quá đơn điệu, thiếu nhiều tính năng cơ bản cũng như các tính năng mở rông, nâng cao.



**Hình 2.3** Giao diện ILDASM

### 2.2.4 ILSPY

ILSpy là trình decompiler/disassembler mã nguồn mở, được tạo ra nhằm mục đích thay thế cho .NET Reflector sau khi Red-Gate mua lại .NET Reflector và biến nó thành phần mềm thương mại. ILSpy là 1 công cụ rất nhẹ, chỉ bao gồm các tính năng cơ bản đủ dùng, đã có thể thay thế một phần nào .NET Reflector. Cá nhân mình thích nhất ở phần mềm này là sự nhanh và nhẹ của nó. Code sau khi dịch ngược khá chuẩn xác.

Dưới là một số tính năng nổi trội của ILSPY:

• Assembly browsing

• IL Disassembly

• Support C# 5.0 async

• Decompilation to C#

o Supports lambdas and 'yield return'

o Shows XML documentation

• Decompilation to VB

• Saving of resources

• Save decompiled assembly as .csproj

• Search for types/methods/properties (substring)

• Hyperlink-based type/method/property navigation

• Base/Derived types navigation

• Navigation history

• BAML to XAML decompiler

• Save Assembly as C# Project

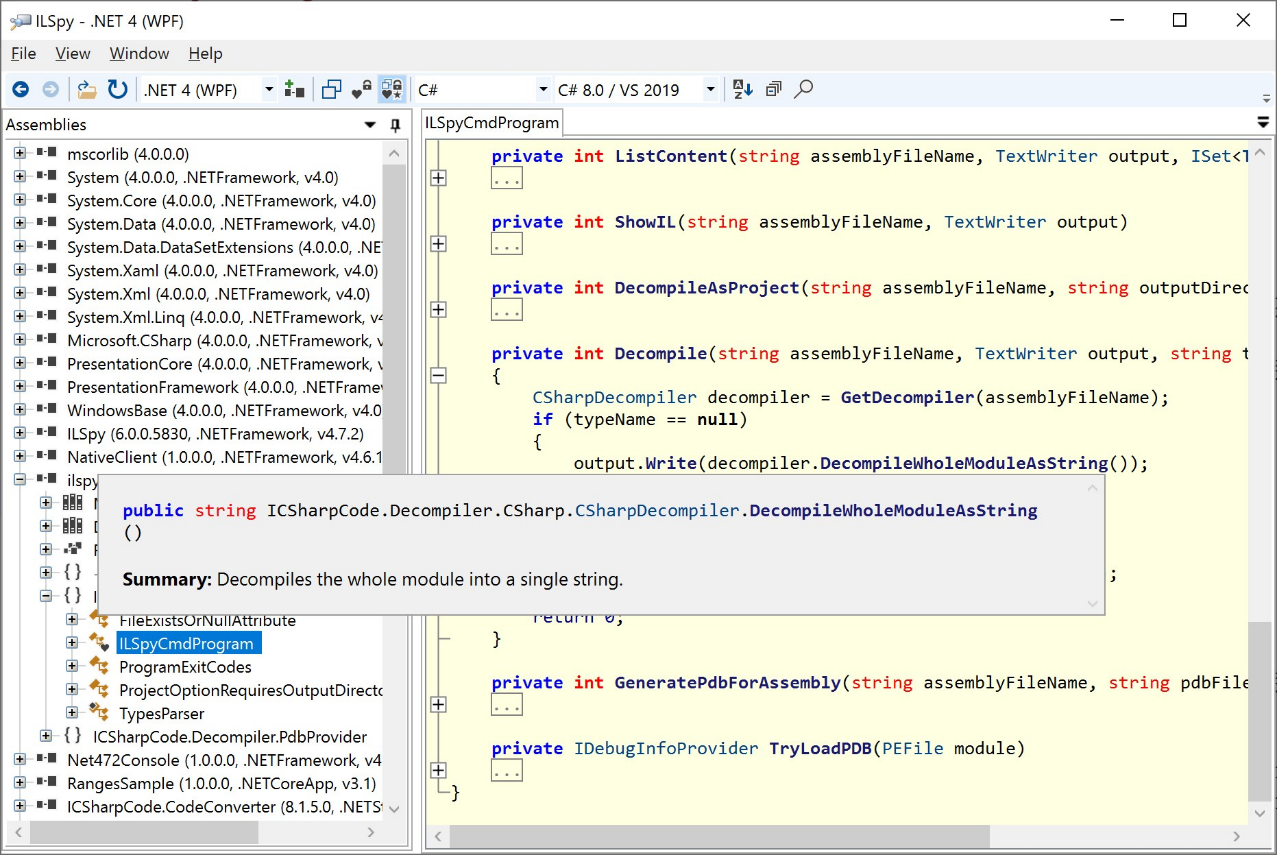
• Find usage of field/method

• Extensible via plugins (MEF)

• Assembly Lists

+ **Điểm cộng:** Rất nhẹ, nhanh, đầy đủ tính năng cơ bản, mã nguồn mở, miễn phí.

+ **Điểm trừ:** Ít plugin, thiếu tính năng mở rộng nâng cao.



**Hình 2.4** Giao diện ILSPY

## 2.3 Một số phần mềm hỗ trợ việc mã hóa mã nguồn .NET

Như bạn đã thấy, mã nguồn của C# có thể dễ dàng dịch ngược bởi những phần mềm hỗ trợ việc dịch ngược mã nguồn đã nêu ở trên, cho nên ta có những phần mềm giúp bảo mật và mã hóa mã nguồn. Đề tài này không phải là hiếm và hiện nay đã có một số phần mềm hướng về nội dung này. Nhưng đa số đều là bản thương mại và phải trả một số tiền rất cao (từ 400 USD trở lên cho cá nhân) để có thể sử dụng. Điển hình là một số phần mềm như sau:

* **Eazfuscator.NET**
* **SmartAssembly**
* **Agile.NET**
* **Babel**
* **Themida**
* **Obfuscar**

## 2.4 Kết luận

Những phần mềm nêu trên đều có khả năng bảo mật tốt và hỗ trợ đầy đủ những kỹ thuật bảo mật mã nguồn, nhưng để sử dụng thì số tiền bỏ ra rất là đắt cho một cá nhân hay một team nhỏ. Đó là lý do đề tài này ra đời, nhằm mục đích viết ra một phần mềm có khả năng như những gì mà một trình bảo vệ có thể làm, nhưng miễn phí, và có khả năng phát triển kỹ thuật bảo mật mạnh mẽ hơn nhiều.

# **CHƯƠNG 3: PHÂN TÍCH, THIẾT KẾ VÀ NGHIÊN CỨU CÁC KỸ THUẬT VÀ THUẬT TOÁN MÃ HÓA**

## 3.1 Các kỹ thuật mã hóa sẽ áp dụng

Những thuật toán dưới đây có thể sẽ không bao gồm tính phức tạp và thời gian thực thi (trừ mã hóa string). Bởi vì bản chất của chúng là bảo mật, độc lập cho mỗi cá nhân sử dụng và dĩ nhiên không có sự cạnh tranh nào cả.

### 3.1.1 Kỹ thuật đổi tên (Entity Renaming) [5]

Kỹ thuật này đổi tên tất cả các cấu trúc siêu dữ liệu (metadata), điều này bao gồm không gian tên (namespace), tên lớp, chữ ký phương thức (method signatures) và trường (fields) cũng như triển khai phương thức và giá trị string của hợp ngữ của bạn. Kỹ thuật đổi tên là biến đổi chúng thành những ký tự không thể đọc được hoặc không thể hiểu được, phương thức này sẽ chuyển đổi các lớp, phương thức và tên trường thành các ký tự unicode không thể in được, hoặc những đoạn chữ không có ý nghĩa hoặc khác nghĩa hoàn toàn. Khi dịch ngược, kết quả là một mã nguồn cực kỳ khó hiểu. Vì các ký tự không in được được sử dụng nên sẽ không thể biên dịch các nguồn được tạo ra sau khi dịch ngược.

Như chúng ta đã biết, việc đặt tên rõ ràng cho một biến, một lớp là điều hết sức quan trọng. Bỡi lẽ nó giúp chúng ta hiểu được bản chất và mục đích của chúng. Chính vì thế, kỹ thuật đổi tên lợi dụng vào vấn đề này, mục đích cuối cùng vẫn khiến cho kẻ tấn công không hiểu rõ được mục đích của mã nguồn của mình.

Dưới đây là một ví dụ đơn giản, đây là một đoạn code gốc:

public class Data

{

public int age;

public Data(int age)

{

this.age = age;

}

}

Và dưới đây là đoạn code bên trên, nhưng đã áp dụng kỹ thuật đổi tên:

public class OMGLOOKAVeryNewClass

{

public int b;

public a(int b)

{

this.b = b;

}

}

Dưới đây là một số ví dụ của options kỹ thuật đổi tên lợi hại:

Giữ lại namespace:

|  |  |
| --- | --- |
| **Tên chưa bảo mật** | **Tên đã mã hóa** |
| ClassA.ClassB.Main | ClassA.ClassB.a |
| ClassA.AnotherThing.LinkedList() | ClassA.AnotherThing.b() |

Giữ lại hệ thống cấp bậc (hierarchy):

|  |  |
| --- | --- |
| **Tên chưa bảo mật** | **Tên đã mã hóa** |
| ClassA.ClassB.Main | a.a.a |
| ClassA.AnotherThing.LinkedList() | a.a.b() |

Ẩn hết namespace:

|  |  |
| --- | --- |
| **Tên chưa bảo mật** | **Tên đã mã hóa** |
| ClassA.ClassB.Main | a |
| ClassA.AnotherThing.LinkedList() | b() |

Play puzzle game, shall we?

|  |  |
| --- | --- |
| **Tên chưa bảo mật** | **Tên đã mã hóa** |
| ClassA.ClassB.Main | There.IsSomeThing.I.Have.To.Tell.You |
| ClassA.AnotherThing.LinkedList() | There.MustBe.A.Secret.LMAO() |

Đổi tên là một kỹ thuật cơ bản nhất, nhưng nó lại có tầm quan trọng nhất và rất hiệu quả trong việc giảm tải đáng kể khả năng và hiệu quả dịch ngược của kẻ tấn công.

Pseudocode (mã giả):

Get name of variable, field, class,...

if name is valid (non Chinese, non base64)

Initialize new\_name to obfuscated\_name(name)

Find and replace all name with new\_name

else

Do something for that name or skip

Pseudocode cho “hidden namespace”:

Get name of namespace

Replace name as “”

### 3.1.2 Kỹ thuật mã hóa string (String Encryption) [6]

Các string có thể được nhận dạng ngay lập tức như là một phần của well-managed, well-executabled code. Ngay cả khi các phương thức và biến được đổi tên, các tham chiếu vào string có thể giúp kẻ tấn công tìm thấy các đoạn mã quan trọng liên quan bằng cách kiểm tra các tham chiếu string trong hệ nhị phân.

Điều này chứa bất kỳ thông báo nào mà người dùng có thể xem, ví dụ như “ConnectionString”. Việc mã hóa sẽ cung cấp một rào cản chống lại tấn công này.

Mã hóa string mã hóa các string trong tệp thực thi nhưng chỉ trả về giá trị thực của chúng khi cần thiết ở quá trình run-time.

Một ví dụ đơn giản. Một phần mềm của .NET yêu cầu phải nhập mã hoặc giấy phép để có thể sử dụng. Kẻ tấn công có thể dựa vào manh mối của những string xảy ra cho những sự kiện liên quan đến phần này. Chẳng hạn như khi nhập sai mã hoặc giấy phép, phần mềm sẽ hiện lên “Invalid license”. Kẻ tấn công sẽ ngay lập tức tìm mọi thứ liên quan đến các từ khóa như “license” hoặc “Invalid license” trong hệ nhị phân. Bằng cách xác định các string như vậy, tin tặc có thể lần ra manh mối và cố gắng phá vỡ bảo vệ giấy phép được nhúng trong sản phẩm mà chúng đang tấn công. Vì thế, kỹ thuật mã hóa string này sẽ ẩn dấu toàn bộ những string có trong assembly và thay bằng bởi những chuỗi ký tự khó hiểu.

Dưới đây là một ví dụ đơn giản, đây là một đoạn code gốc:

Console.WriteLine("You are the most beautifullest person I have never seen!");

Và dưới đây là đoạn code bên trên, nhưng đã áp dụng kỹ thuật mã hóa string:

Console.WriteLine("bazj,؟Y^^'"zwe@e2z‑z");

Chúng ta dĩ nhiên có vô số các kỹ thuật cũng như thuật toán để mã hóa string, nên phần này khá là đơn giản. Nhưng để tìm ra một thuật toán mã hóa string mà vừa nhanh vừa nhẹ thì chỉ có thể liên quan đến **Base64**.

Base64 là một thuật toán mã hóa cho phép bạn chuyển đổi bất kỳ ký tự nào thành một bảng chữ cái bao gồm các chữ cái Latinh, chữ số, dấu cộng và dấu gạch chéo. Nhờ nó, bạn có thể chuyển đổi các ký tự Trung Quốc, biểu tượng cảm xúc và thậm chí cả hình ảnh thành một chuỗi "có thể đọc được", có thể được lưu hoặc chuyển đi bất cứ đâu.

Để hiểu một cách hình tượng tại sao Base64 được phát minh, hãy tưởng tượng rằng trong một cuộc gọi điện thoại, Alice muốn gửi một hình ảnh cho Bob. Vấn đề đầu tiên là cô ấy không thể mô tả hình ảnh một cách đơn giản như thế nào, bởi vì Bob cần một bản sao chính xác. Trong trường hợp này, Alice có thể chuyển đổi hình ảnh sang hệ nhị phân và ra lệnh cho Bob các chữ số nhị phân (bit), sau đó anh ta sẽ có thể chuyển đổi chúng trở lại hình ảnh ban đầu. Vấn đề thứ hai là giá cước cho các cuộc gọi điện thoại quá đắt và quy định mỗi byte vì 8 chữ số nhị phân sẽ tồn tại quá lâu. Để giảm chi phí, Alice và Bob đồng ý sử dụng phương pháp truyền dữ liệu hiệu quả hơn bằng cách sử dụng một bảng chữ cái đặc biệt, thay thế mỗi “sáu chữ số” bằng một “chữ cái”.

Để thấy được sự khác biệt, đây là ví dụ về một dãy nhị phân gốc:

010001 110100 100101 000110 001110 000011 011101 100001 000000 010000 000000 000001 000000 001111 000000 000000 000000 001111 111100 000000 000000 000000 000000 000000 000000 000010 110000 000000 000000 000000 000000 000000 000000 010000 000000 000001 000000 000000 000000 000010 000000 100100 010000 000001 000000 000011 001011

Và dưới đây là đoạn mã bên trên, nhưng đã áp dụng Base64 Encoding:

R0lGODdhAQABAPAAAP8AAAAAACwAAAAAAQABAAACAkQBADs

Ta có thể thấy sự khác biệt là rõ rệt. Cho dù có xóa đi toàn bộ khoảng trống cho dãy nhị phân gốc, độ dài của chúng vẫn ít hơn so với đoạn string đã được mã hóa sang Base64. Vì thế, Base64 là một thuật toán rất hiệu quả trong việc bảo mật cũng như tốc độ và dung lượng của nó, một thuật toán hoàn mỹ.

Nhưng vì tính phổ biến và thông dụng của Base64, dĩ nhiên chúng có thể dễ dàng dịch ngược lại được. Vậy thì làm sao để bảo mật mà vẫn áp dụng Base64? Đây là một bài toán khá thú vị đối với em. Dĩ nhiên cái này sẽ là bí mật, nhưng đây là mục đích học tập nên thuật toán sẽ được đưa ra ở dưới.

#### **3.1.2.1 Mật mã Caesar (Caesar Cipher)**

Trong mật mã học, mật mã Caesar, còn được gọi là mật mã Caesar, mật mã dịch chuyển, mã Caesar hoặc dịch chuyển Caesar, là một trong những kỹ thuật mã hóa đơn giản và được biết đến rộng rãi nhất. Nó là một loại mật mã thay thế, trong đó mỗi chữ cái trong bản rõ được thay thế bằng một chữ cái với một số vị trí cố định trong bảng chữ cái. Ví dụ, với sự dịch chuyển sang trái là 3, D sẽ được thay thế bằng A, E sẽ trở thành B, v.v. Phương pháp này được đặt theo tên của Julius Caesar, người đã sử dụng nó trong thư từ riêng tư của mình.

Bước mã hóa được thực hiện bởi mật mã Caesar thường được kết hợp như một phần của các sơ đồ phức tạp hơn, chẳng hạn như mật mã Vigenère, và vẫn có ứng dụng hiện đại trong hệ thống ROT13. Như với tất cả các mật mã thay thế một bảng chữ cái, mật mã Caesar dễ bị phá vỡ và trong thực tế hiện đại về cơ bản không cung cấp bảo mật thông tin liên lạc.

Sự biến đổi có thể được biểu diễn bằng cách sắp xếp hai bảng chữ cái; bảng chữ cái mật mã là bảng chữ cái đơn giản được xoay sang trái hoặc phải theo một số vị trí. Ví dụ, đây là một mật mã Caesar sử dụng phép quay trái ba vị trí, tương đương với sự dịch chuyển sang phải của 23 (tham số shift được sử dụng làm khóa):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Plain** | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| **Cipher** | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |

Khi mã hóa, một người tra cứu từng ký tự của tin nhắn trong dòng "đơn giản" và viết ra ký tự tương ứng trong dòng "mật mã".

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Nếu đơn giản như thế thì vậy tại sao nó được em áp dụng? Đó là vì chúng được hết hợp với một thuật toán XOR ở dưới, và dĩ nhiên bản thân của Caesar này sẽ được shift đến một con số khủng khiếp đủ để đánh lừa được vô số kẻ tin tặc muốn giải mã.

Dưới đây là một ví dụ mẫu cho việc áp dụng Ceasar vào máy:

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Ciphertext with **290220** times of shifting: 힛흡흹힂힂힣흼흲힃흽흺흳힁힛흺흳힁힛흺흳힁힛흡흹힂힂힂힣흼흲힃흽흺흳힁힛흺흳힁힛흺흳힁

Khi một character được shift quá nhiều, chúng sẽ được đẩy lùi sang unicode rất thấp hoặc rất cao mà máy không thể hiển thị được, trong nhiều trường hợp nó sẽ hiển thị những ký tự kỳ lạ, đối với trường hợp này, nó hiển thị một đoạn tiếng Hàn tương đương với nhau. Nhưng thực chất, những ký tự “tương đương” này lại mang một giá trị unicode khác nhau.

Thuật toán đã viết:

// this is an example of the shifting algorithm



Bên dưới là tốc độ thực thi của thuật toán trên với đầu vào là một string văn bản có **2664** ký tự và số lần shift là **290220**, chỉ mất 10 millisecond để thực thi. Việc giải mã chúng thì chỉ cần đảo chiều cho biến **numberOfShifting.**

Graphical user interface

Description automatically generated with low confidence

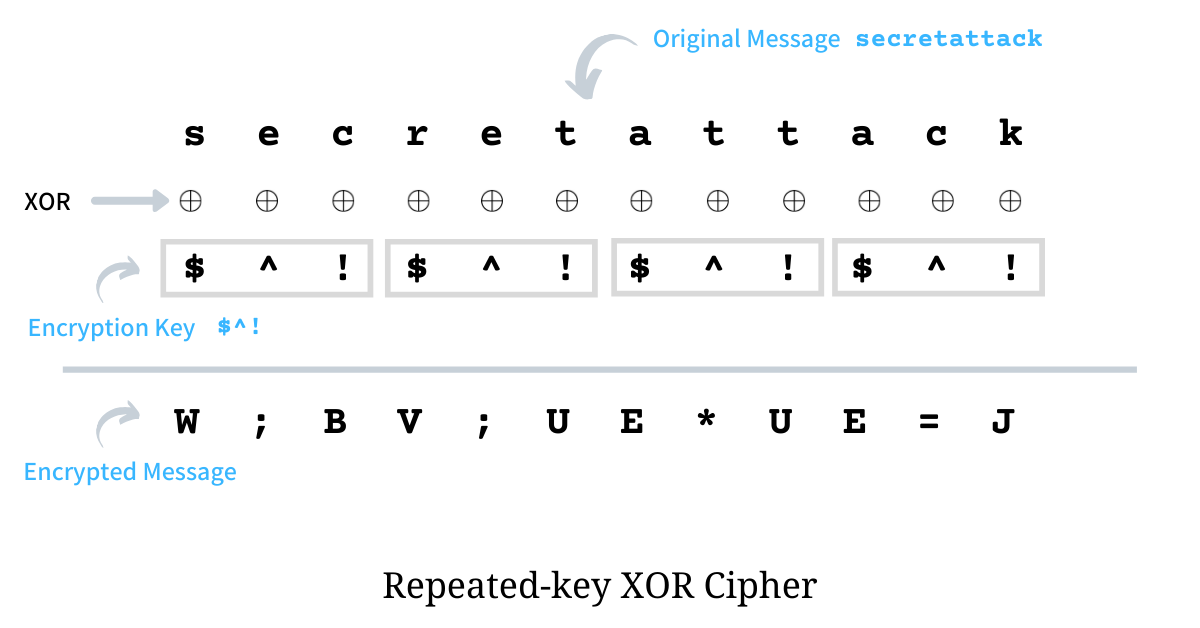
**Hình 3.1** Tốc độ thực thi của Casear

Thuật toán này có độ phức tạp là O(n).

#### **3.1.2.2 Repeating-key XOR Cipher**

Văn bản thuần túy được mã hóa bằng khóa mã hóa bằng cách thực hiện thao tác XOR bitwise trên mọi ký tự. Khóa mã hóa được lặp lại cho đến khi nó XOR mọi ký tự đơn của văn bản thuần túy và luồng byte kết quả một lần nữa được dịch lại thành ký tự và được gửi cho bên kia. Các byte mã hóa này không cần phải nằm trong số các ký tự in được thông thường và lý tưởng nên được hiểu là một dòng byte.

Ví dụ, chúng ta sẽ mã hóa đoạn string này- secretattack - bằng khóa mã hóa $ ^! và theo thuật toán, trước tiên chúng ta lặp lại khóa mã hóa cho đến khi nó khớp với độ dài của văn bản thuần túy và sau đó XOAY nó với văn bản thuần túy. Hình minh họa bên dưới cho thấy toàn bộ quá trình mã hóa.



**Hình 3.2** Mô phỏng cách hoạt động của Repeated-key XOR Cipher

Đối với ký tự đầu tiên trong văn bản thuần túy - **s** -, tức là giá trị ASCII là 115 mà khi XORed với **$** cho kết quả là **87** có ký tự tương đương là **W**, tương tự đối với ký tự thứ hai **e** là được mã hóa thành **;,** đối với **c** là **B**, đối với ký tự thứ tư **r**, vì khóa lặp lại, XOR được lấy với **$** để lấy **V** và quá trình tiếp tục. Văn bản được mã hóa kết quả bằng cách sử dụng khóa lặp lại XOR trên túi bí mật văn bản thuần túy với khóa **$ ^!** là **W; BV; UE \* UE = J.**

Dưới đây là một ví dụ mẫu cho việc áp dụng Repeated-key XOR Cipher vào:

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: ▬☻♠→►↑s◄☺∟♦↔s§∟s↓♠▲♥s∟♣▬☺s▬▼↕s↨∟¶

Pseudocode (mã giả):

internal void string XOREncryption(string text, string key) {

Convert text to byte

Convert key to byte

Initialize repetitions = 1 + (text.Length() // key.Length())

key = key \* repetitions

Loop each character on text and XOR using text and generated key above

Convert the XORed bytes into readable string UTF8

return the converted text

Bên dưới là tốc độ thực thi của thuật toán trên với đầu vào là một string văn bản có **2664** ký tự và key là **0x53**, chỉ mất 8 millisecond để thực thi.



**Hình 3.2** Tốc độ thực thi của Repeated-Key XOR Cipher

Bây giờ, chỉ cần kết hợp 2 kỹ thuật trên với nhau với đầu cuối là Base64 là ta có một thuật toán mã hóa vừa nhanh vừa ổn về tính bảo mật.

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: xpaUxpaU4ZfGlpTGlpTGlpTGlpTGlpT4l8aWlPiXxpaUxpaUxpaUxpaUxpaUxpaUxpaU+ZfGlpTGlpTGlpTGlpTGlpTGlpTGlpT7l8aWlOGXxpaUxpaUxpaUxpaUxpaU+5fGlpThl8aWlMaWlMaWlMaWlMaWlPuXxpaU4ZfGlpTGlpTGlpTGlpTGlpT4l8aWlPiXxpaU+JfGlpTGlpTGlpTGlpTGlpTGlpTGlpT5l8aWlMaWlMaWlMaWlMaWlMaWlMaWlPuXxpaU4ZfGlpTGlpTGlpTGlpTGlpT7l8aWlOGXxpaUxpaUxpaUxpaUxpaUxpaU

Pseudocode (mã giả):

internal void string FinalEncryption(string text) {

Encrypt text using Ceasar Cipher

Encrypt Casear’s encryppted text using XOR Cipher

Convert the final result into Base64

return the final encrypted text

### 3.1.3 Kỹ thuật kiểm soát luồng chảy (Control Flow Obfuscation) [7]

Kỹ thuật này cung cấp khả năng kiểm soát flow, mã hóa control flow để ẩn logic chương trình bằng cách chuyển đổi các mẫu luồng mã thoát thành các cấu trúc tương đương về mặt ngữ nghĩa, tuy nhiên khác với mã được viết ban đầu. Thuật toán mã hóa luồng điều khiển chuyển đổi việc triển khai ban đầu thành mã spaghetti, do đó làm cho việc suy luận logic của chương trình trở nên cực kỳ khó khăn hơn. Đảm bảo rằng dòng mã ứng dụng của lắp ráp bị xáo trộn vẫn còn nguyên vẹn.

Có thể hiểu nôm ra là thay vì đi từ A đến B, bản phải đi đến CDLGTVPO rồi B.

Trong nhiều trường hợp, sử dụng một số cấu trúc misleading hoặc sử dụng những fail condition có thể gây rối và làm hỏng những trình dịch ngược. Kỹ thuật này rất thú vị và nắm phần rất quan trọng trong kỹ thuật bảo vệ mã nguồn.

Dưới đây là một ví dụ khá “đơn giản”:

num = 2;

System.Console.WriteLine(num);

return 0;

v = 2

switch(v)

{

case 1:

System.Console.WriteLine(num);

v = 3;

case 2:

num = 2;

v = 1;

case 3:

return 0;

}

Áp dụng kỹ thuật Control Flow Obfuscation (ta có rất nhiều kỹ thuật và thuật toán cho nó), đoạn code như trên sau khi compile sẽ trông như thế này:

private static int Main(string[] args)

{

int result;

for (;;)

{

IL\_01:

uint num = 2254568472u;

for (;;)

{

uint num2;

switch ((num2 = (num ^ 3037644272u)) % 6u)

{

case 1u:

num = (num2 \* 223771410u ^ 648192441u);

continue;

case 2u:

num = (num2 \* 909441097u ^ 4218481382u);

continue;

case 3u:

goto IL\_01;

case 4u:

{

int value = 2;

Console.WriteLine(value);

num = (num2 \* 2137906813u ^ 2826615287u);

continue;

}

case 5u:

result = 0;

num = (num2 \* 3850219098u ^ 1166622248u);

continue;

}

return result;

}

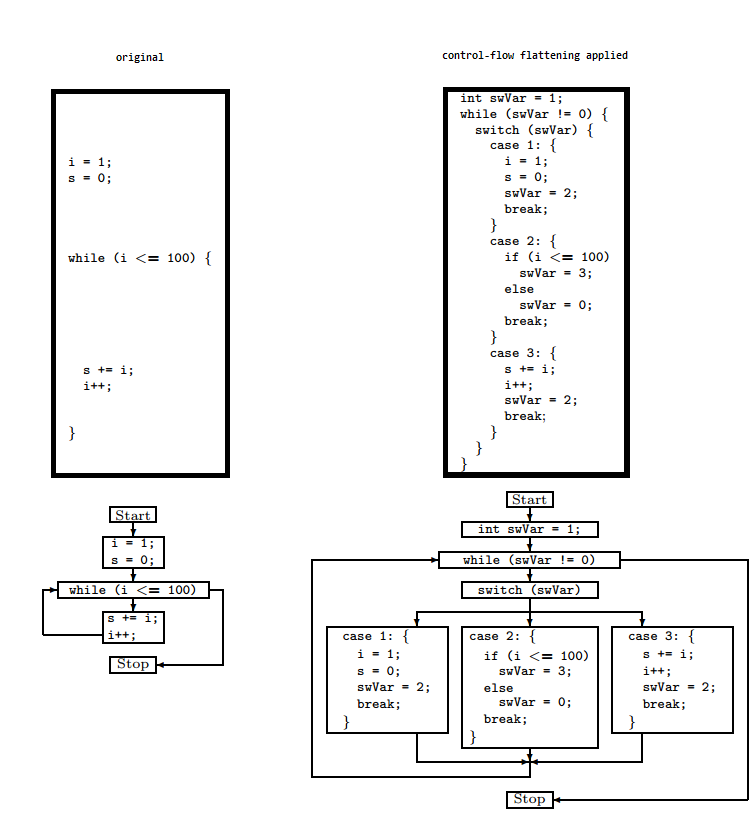
}

}

Dĩ nhiên rồi, đoạn code bên dưới nếu chạy thì nó sẽ cho kết quả không khác gì đoạn code ban đầu. Tại sao? Bởi vì nó đã được biến hóa thành những luồng khó hiểu, dead-end và không liên quan. Mục đích nhằm gây rối cho kẻ tấn công, khiến họ không đọc được logic của mã nguồn, nhưng vẫn giữ lại tính toàn vẹn của nó. Bạn có thể hiểu nôm na là từ if... else condition sẽ biến thành switch... case và vice versa.

Phương pháp Control Flow Obfuscation được thực thi như sau:

1. Đầu tiên, chúng ta chia nhỏ phần thân của hàm thành các block cơ bản, sau đó chúng ta đặt tất cả các block này (ban đầu ở các cấp độ lồng nhau khác nhau) bên cạnh nhau.
2. Các block cơ bản được cấp bằng nhau hiện nay được đóng gói trong một cấu trúc chọn lọc (một câu lệnh switch trong ngôn ngữ C#) với mỗi block trong một trường hợp riêng biệt và lựa chọn được đóng gói lần lượt trong một vòng lặp.
3. Cuối cùng, logic của control flow được đảm bảo bởi một biến điều khiển đại diện cho trạng thái của chương trình, được đặt ở cuối mỗi khối cơ bản và được sử dụng trong các vị từ của vòng lặp và selection.



**Hình 3.3** Ví dụ đơn giản về cách hoạt động của Control Flow Obfuscation

Pseudocode (mã giả):

Parse codes into multiple blocks

foreach block in blocks

Apply control flow obfuscation on block

Add jump on block

Return the obfuscated blocks

### 3.1.4 Kỹ thuật đột biến (Mutation Obfuscation)

Kỹ thuật đột biến hay Mutation Obfuscation là một kỹ thuật em tự thiết kế ra, dựa trên Control Flow Obfuscation. Về bản chất nó cũng không khác gì mấy so với tiền đề của nó. Nhưng ở đây, thay vì thay đổi luồng đi của code mà kỹ thuật Control Flow Obfuscation đã đảm nhiệm, thì nó khiến mọi thứ nhánh nhỏ trong block trở thành một thứ xáo trộn khó hiểu, và có rất nhiều tạp nham không có lý do nào để tồn tại cả nhưng nó lại cần thiết để chạy. Cái tên đột biến là cái tên rất phù hợp cho kỹ thuật trộn mã này.

Để cho dễ hình dung hơn, thay vì int number = 3 thì number sẽ bằng một nhiều đoạn code con khác bổ trợ nhau và dĩ nhiên đầu cuối sẽ là 3.

Dưới đây là một ví dụ rất “đơn giản”:

private void ClosingForm\_Load(object sender, EventArgs e)

{

try {

if (File.Exists(Methods.GetFolder("temp\_", Strings.FileName.StageData)))

File.Delete(Methods.GetFolder("temp\_" Strings.FileName.StageData));

}

catch { }

}

Áp dụng kỹ thuật Mutation Obfuscation, đoạn code như trên sau khi compile sẽ trông như thế này:

private void ClosingForm\_Load(object sender, EventArgs e)

{

int num = 1446769112;

int arg = 4;

int num2 = 185710028;

try {

num3 = Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(c2V0X1RyYWRlQmFuU3RhdGU=('㙌'));

int num4 = 116233085;

num3 = ((num3 + num4 == 1513209625) ? num2 : Math.Abs(2));

string[] array = new string[num3];

array[0] = "temp\_";

array[1] = "stagedata.pak";

if (File.Exists(Methods.GetFolder(array))) {

int num5 = Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(1814938916);

int num6 = c2V0X1RyYWRlQmFuU3RhdGU=('墁');

num5 = ((num5 + num6 == Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(1620691528)) ? c2V0X1RyYWRlQmFuU3RhdGU=('ሸ') : Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(Math.Abs(2)));

string[] array2 = new string[num5];

array2[0] = "temp\_";

array2[1] = "stagedata.pak";

File.Delete(Methods.GetFolder(array2));

}

}

catch { }

}

Dựa vào đoạn mẫu trên, bạn có thể thấy được sự độc đáo của thuật toán này. Có rất nhiều đoạn không hiểu kiểu gì hoặc trông không liên quan chèn lên nhau, nhưng thực chất chúng tất cả đều có giá trị sử dụng và không có đoạn code nào là dư thừa! Khiến chung ta cảm giác khó chịu khi nhìn vào. Đó chỉ là mỗi Mutation Obfuscation thôi nhé! Nếu kết hợp toàn bộ những thuật toán nêu trên thì không phải là một cảm giác vui vẻ chút nào đâu.

Để ý đoạn này:

try {

**num3** = Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(c2V0X1RyYWRlQmFuU3RhdGU=('㙌'));

int **num4** = 116233085;

**num3** = ((**num3** + **num4** == 1513209625) ? **num2** : Math.Abs(2));

string[] **array** = new string[num3];

array[0] = "temp\_";

array[1] = "stagedata.pak";

if (File.Exists(Methods.GetFolder(**array**))) {

int **num5** = Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(1814938916);

int **num6** = c2V0X1RyYWRlQmFuU3RhdGU=('墁');

**num5** = ((**num5** + **num6** == Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(1620691528)) ? c2V0X1RyYWRlQmFuU3RhdGU=('ሸ') : Z2V0X05hbWU=YWRkX09uUGx1Z2luVW5sb2FkaW5nZ2V0X0hlYWRsaW5l(Math.Abs(2)));

string[] **array2** = new string[**num5**];

array2[0] = "temp\_";

array2[1] = "stagedata.pak";

File.Delete(Methods.GetFolder(**array2**));

}

}

catch { }

Để ý vào những đoạn được tô màu xanh, nhìn vào chúng ta sẽ thấy rằng nó có lẽ là dư thừa để đánh lừa người dịch mã. Nhưng không, nó hoàn toàn có bổn phận cho những đoạn mã bên dưới, như đã nêu ở trên là không có một đoạn nào là dư thừa. Từ một dòng code đơn giản đã biến thành một đoạn code rối xeng, đó chính thuật toán đột biến gen.

Pseudocode (mã giả):

Get code module tables

Initializes code pattern

foreach TypeDef in Tasks

loop in number of methods TypeDef has

if TypeDef doesn’t have body || TypeDef is constructor

continue;

loop in number of instructions TypeDef has

if supported then apply mutation

### 3.1.5 Kỹ thuật mã giả (Dummy Code Insertion) [8]

Đây là một kỹ thuật trong nhánh Control Flow Obfuscation. Dễ hình dung hơn nó như là một phiên bản lite của thuật toán đột biến. Cách thức hoạt động của nó không khác gì đột biến, nhưng những code nó thêm vào hoàn toàn không có một giá trị gì cả. Như tên của nó suggest, thì những dòng code đó là những dòng code “rác” mang tính “bù nhìn” để đánh lạc hướng trình dịch và người dịch mã.

Chèn mã bù nhìn vào tệp thực thi không ảnh hưởng đến logic của chương trình nhưng phá vỡ trình dịch ngược hoặc làm cho mã được dịch ngược trở nên khó hiểu hơn nhiều.

Dưới đây là một ví dụ:

private void ClosingForm\_Load(object sender, EventArgs e)

{

MessageBox.Show(“Hello World”);

}

Sau khi áp dụng kỹ thuật Dummy Code Insertion:

private void ClosingForm\_Load(object sender, EventArgs e)

{

int num = 701354730;

int num2 = 11878314;

int num3 = 1101901486;

int num4 = 98750199;

int num5 = 465574568;

int num6 = 331959123;

int num7 = 1537535021;

int num8 = 1867625223;

int num9 = 1129533415;

int num10 = 264072666;

int num11 = 1186421193;

int num12 = 1069753182;

int num13 = 821776663;

MessageBox.Show(“Hello World”);

}

Dựa vào ví dụ đấy, hình dung được mục đích của nó rồi chứ?

Ngoài thêm những đoạn code “rác” vào bên trong những block của code, nó còn tạo ra nhiều lớp và phương thức tương thực mã không có một liên hệ gì với luồng code chính.

internal class c2V0X1N1bW1hcnk=SGFzUGVybWlzc2lvbg==Z2V0X1N5bnRheA==

{

public static void UmVnaXN0ZXI=UmVhZA==()

{

int num = 1824476050;

int num2 = 442001172;

int num3 = 1340244797;

int num4 = 1952459591;

int num5 = 1149289608;

int num6 = 1968876624;

}

public static void =SGFzUGVybWlzc2lvbg==VHJ5SW52b2tl()

{

int num = 172045030;

int num2 = 1316629148;

int num3 = 968821846;

int num4 = 961467482;

int num5 = 1583340568;

}

}

Pseudocode (mã giả) cực đơn giản:

Insert dummy code if applicable

## 3.2 Các kỹ thuật bảo vệ thêm cho .NET

Ngoài những kỹ thuật làm xáo trộn mã nguồn bên trên, ta còn nhiều hình thức bảo mật khác nhằm tăng cao khả năng bảo mật của phần mềm .NET của bạn.

### 3.2.1 Liên kết các thư viện vào Assembly (Library Linking) [9]

Khi một phần mềm .NET được build, tùy những gì nó sử dụng và cách nó được bao gồm thì nó thường sẽ đi kèm rất nhiều thư viện cùng level với nó. Hình ở dưới sẽ giúp bạn dễ hình dung hơn những gì đi kèm theo cho một sản phầm của .NET sau khi được biên dịch.

Text

Description automatically generated with medium confidence

**Hình 3.4** Một phần mềm .NET sẽ có nhiều thư viện đi kèm

Kẻ tấn công và các trình dịch ngược có thể dễ dàng lợi dụng những thư viện đi kèm để lần ra manh mối mà dịch ngược chúng. Vì thế, liên kết toàn bộ những thư viện này vào assembly của .NET là một điều thông minh. Dĩ nhiên phần mềm của bạn sẽ trở thành một PE (Portable Exetuable) và khiến cho assembly của bạn trở nên nhỏ gọn và thân thiện hơn so với một phần mềm sử dụng những trình cài đặt. Ngoài ra ta còn có thêm nhiều ưu điểm của việc liên kết các thư viện như:

* + Đơn giản hóa quá trình deployment cho sản phẩm hoàn thiện
  + Giảm thiểu dung lượng của phần mềm
  + Chắc chắn rằng những thư viện lúc nào cũng tồn tại (trong nhiều trường hợp, máy người dùng có thể thiếu hoặc mất đi những thư viện của .NET khiến phần mềm không thể chạy được những đoạn yêu cầu thư viện đó. Việc liên kết sẽ khiến cho thư viện này lúc nào cũng tồn tại và khiến việc thiếu thư viện sẽ không bao giờ xảy ra)

Phần mềm như trên, sau khi áp dụng Library Linking sẽ trông như thế này:

Text

Description automatically generated with medium confidence

**Hình 3.5** Một phần mềm .NET sau khi liên kết thư viện

Bạn thấy là bây giờ phần mềm .NET bên trên chỉ còn là một file executable đơn giản. Bây giờ ta có thể đưa file nó bất kỳ đâu mà không cần phải lo lắng về thư viện yêu cầu của nó.

### 3.2.2 Proxy Reference [10]

Nếu bạn thực hiện call bên ngoài assembly của mình, những lệnh call này sẽ vẫn hiển thị trong mã của bạn sau khi mã hóa string và áp dụng Control Flow Obfuscsation. Ví dụ: nếu bạn thực hiện call đến một MessageBox, bạn vẫn có thể tìm thấy call đó trong ứng dụng đã được tháo rời, ngay cả khi đã bật mã hóa string và áp dụng Control Flow Obfuscsation. Bằng cách thực hiện theo các lệnh call bên ngoài như thế này, người khác vẫn có thể hiểu các phần mã của bạn.

Để giải quyết vấn đề này, chúng ta tạo proxy động trong giai đoạn run-time. Proxy này làm cho việc hiểu mã của bạn trở nên khó khăn hơn vì tất cả các lệnh gọi bên ngoài dường như được gửi đến proxy chứ không phải method, thuộc tính hoặc trường bên ngoài mà nhận được nó.

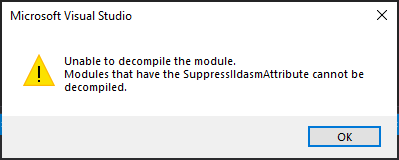
Nếu kẻ tấn công chỉnh sửa phần mềm của mình, thì proxy sẽ bị hỏng và khiến cho phần mềm .NET hỏng theo và không thể hoạt động.

### 3.2.3 MSIL Dissasembler Protection

IL Disassembler là một công cụ đồng hành với IL Assembler (Ilasm.exe). Ildasm.exe lấy một tệp thực thi di động (PE) có chứa mã ngôn ngữ trung gian (IL) và tạo một tệp văn bản phù hợp làm đầu vào cho Ilasm.exe. Công cụ này được cài đặt tự động với Visual Studio. 15, 2021.

Chúng ta có thể khiến cho trình dịch này không thể đọc được phần mềm .NET của mình bằng cách thêm một attribute SuppressIldasmAttribute vào tag của .NET.

Dưới đây là điều sẽ xảy ra nếu chúng ta cố tình import những phần mềm đã được bảo vệ vào MSIL Decompiler:



**Hình 3.6** Bảng thông báo lỗi của MSIL Dissaembler

### 3.2.4 Anti Tamper [11]

Đây là một kỹ thuật dùng để ngăn chặn kẻ tấn công chỉnh sửa assembly của bạn. Bằng cách kiểm tra những checksum của phương thức đã được ghi trong mã nguồn. Điều này sẽ tăng khá nhiều dung lượng code nhưng kết quả lại khá hiệu quả.

Cách hoạt động:

new IntegrityValidator(typeof(Program)).ValidateIntegrity(); // anti-tamper inject

[Checksum("8163cb50e7332f915355e94cfe324fb5")] //hash of method body bytes

public void DontTamper()

{

Console.WriteLine("hi");

}

Ở đây sẽ có một class gọi là IntergrityValidator nhằm mục đích kiểm tra checksum của những method ở run-time. Nếu có sự khác biệt, class này sẽ ngay lập thức khiến hệ thống phần mềm bị dừng lại.

# **CHƯƠNG 4: PHÂN TÍCH, THIẾT KẾ HỆ THỐNG PHẦN MỀM VÀ THỰC NGHIỆM**

## 4.1 Phân tích cách hoạt động của phần mềm

### 4.1.1 Thư viện dnlib

Dnlib là một thư viện mã nguồn mở viết bằng C#, được viết bởi một thành viên có userid là **0xd4d** trên GitHub.

Đường dẫn đến repo: https://github.com/0xd4d/dnlib

Đây là một thư viện hoàn hảo dùng để đọc và viết những module, thư viện và code trong assembly của .NET:

* + Đọc được metadata của assembly
  + Không can thiệm vào GAC
  + Có thể build lại phần mềm .NET mà không cần sử dụng một trình dịch hay IDE nào cả
  + Biên dịch theo chuẩn .NET!

Đây là thư viện cứu rỗi cho đề tài này, bởi lẽ nó giảm thiểu được phạm vi nghiên cứu của em đi rất nhiều. Bỏ đi công đoạn tách mã, lấy mã assembly và viết mã. Thư viện này làm hết những gì mà em yêu cầu. Giúp làm hẹp đi phạm vi xuống chỉ còn nghiên cứu thuật toán bảo mật và áp dụng.

### 4.1.2 Cách hoạt động của phần mềm

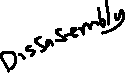
Nhờ thư viện dnlib, nó khiến quy trình hoạt động trở nên rất đơn giản:

1. Phần mềm .NET sẽ được đọc thông qua dnlib, sau khi chúng được đọc và được phân tích bởi dnlib.
2. Tiến hành thực hiện mã hóa.
3. Sử dụng dnlib để build lại phần mềm hoàn chỉnh!

Bên dưới là một sơ đồ đơn giản gợi ý cho cách hoạt động của nó:

Diagram

Description automatically generated

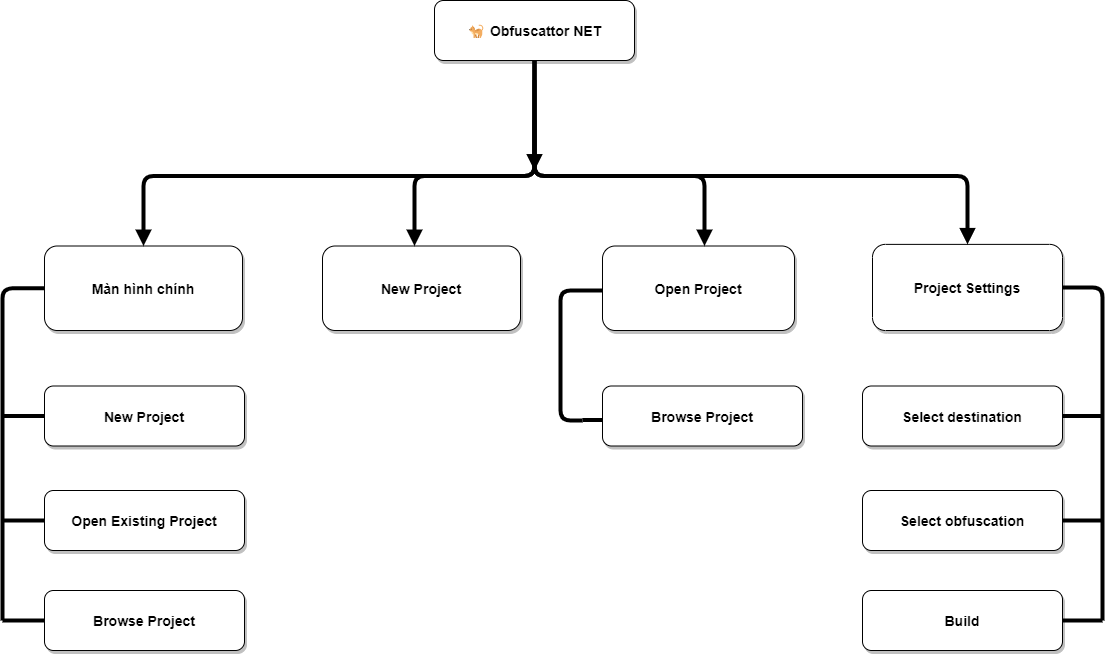


**Hình 4.1** Cách hoạt động của phần mềm Obfuscattor NET

## 4.2 Sơ đồ phân rã chức năng

Với những yêu cầu và chức năng đã được chọn, Obfuscsattor NET sẽ được xây dựng và ứng dụng vào thực tế. Để hiểu rõ hơn Obfuscattor NET sẽ làm gì thì chúng ta sẽ thông qua một số chức năng.

Với châm ngôn “back-end khủng bố như sao thì kệ, nhưng front-end phải càng đơn giản càng tốt”, thì Obfuscattor NET đã được thiết kế cho client sử dụng một cách đơn giản nhất mà không yêu cầu nhiều kiến thức chuyên sâu. Obfuscattor NET có 4 mục chức năng chính được thể hiện với sơ đồ phân rã chức năng bên dưới:



**Hình 4.2** Sơ đồ phân rã chức năng của Obfuscattor NET

## 4.2 Sơ đồ Use Case tổng quát

Để mô tả rõ hơn về các đối tượng và chức năng trong hệ thống, em đã vẽ sơ đổ Use Case tổng quát cho các chức năng chính.

Vì bản chất của phần mềm này không áp dụng cơ sở dữ liệu, việc Use Case có hay không thì em không chắc chắn. Em nghĩ thì củng chả sao và nên việc thiết kế Use Case khá là đơn điệu. Dưới đây là sơ đồ Use Case mà đã được áp dụng:

Diagram

Description automatically generated

**Hình 4.3** Sơ đồ Use Case của Obfuscattor NET

## 4.3 Giao diện phần mềm

Obfuscattor NET được thiết kế rất đơn giản.

### 4.3.1 Giao diện màn hình chính khi mở phần mềm

Màn hình chính sẽ hiện 2 options để chọn. Tạo project mới hoặc chọn một project có sẵn. Project ở đây chính là một file cấu hình của một assembly đặc trưng cho ứng dụng Obfuscattor NET. Nó lưu trữ thông tin của Obfuscattor NET đối với assembly của bạn.

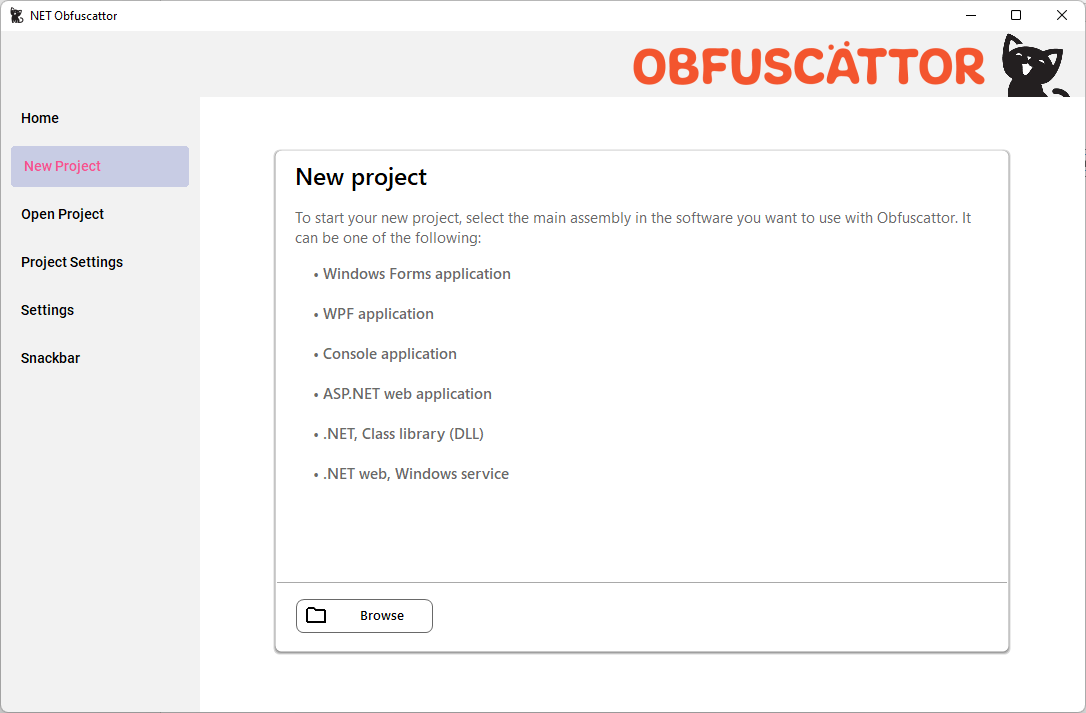
Graphical user interface

Description automatically generated

**Hình 4.4** Màn hình chính

### 4.3.2 Giao diện chọn project

Ở màn hình này nó sẽ hiển thị một đoạn text hướng dẫn người dùng nên chọn những assembly nào phù hợp của .NET.



**Hình 4.5** Màn hình chọn project

### 4.3.3 Giao diện lỗi khi chọn sai assembly

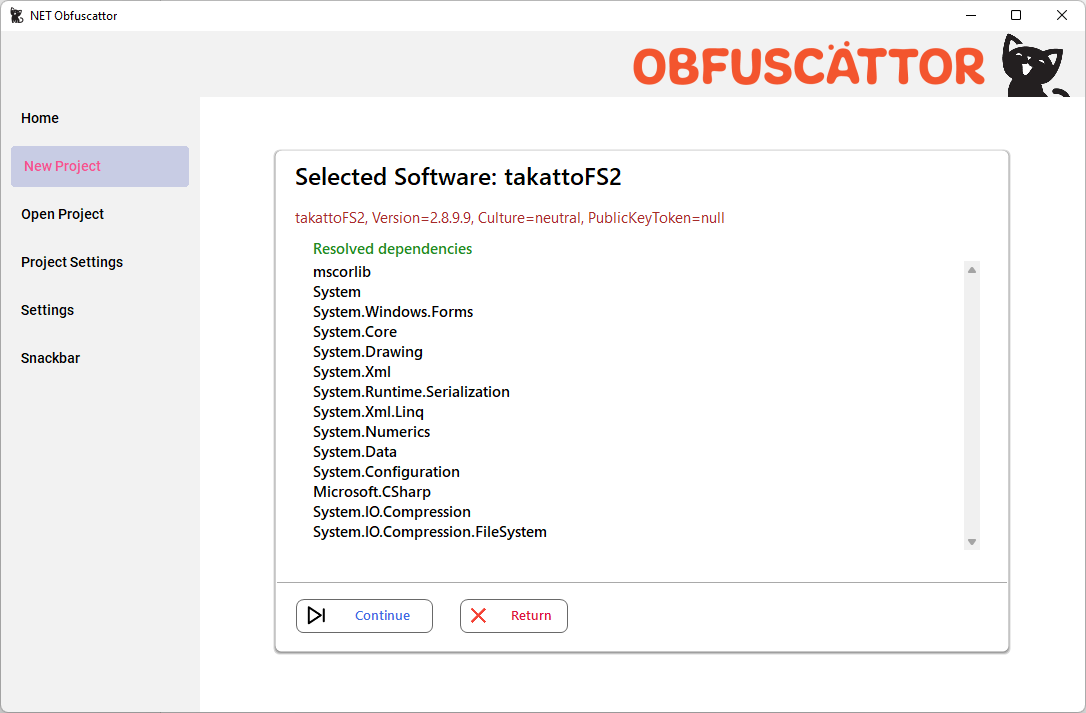
Obfuscattor NET dĩ nhiên được thiết kế để tránh việc nhập vào những assembly không thuộc nền tảng .NET.



**Hình 4.6** Màn hình báo lỗi

### 4.3.4 Giao diện assembly hợp lệ

Nếu assembly đã chọn là một assembly hợp lệ của nền tảng .NET, Obfuscattor sẽ dẫn bạn đến màn hình thứ 2. Ở đây nó sẽ nêu lên thông tin cơ bản của assembly của bạn và một số dependencies đi kèm với nó.

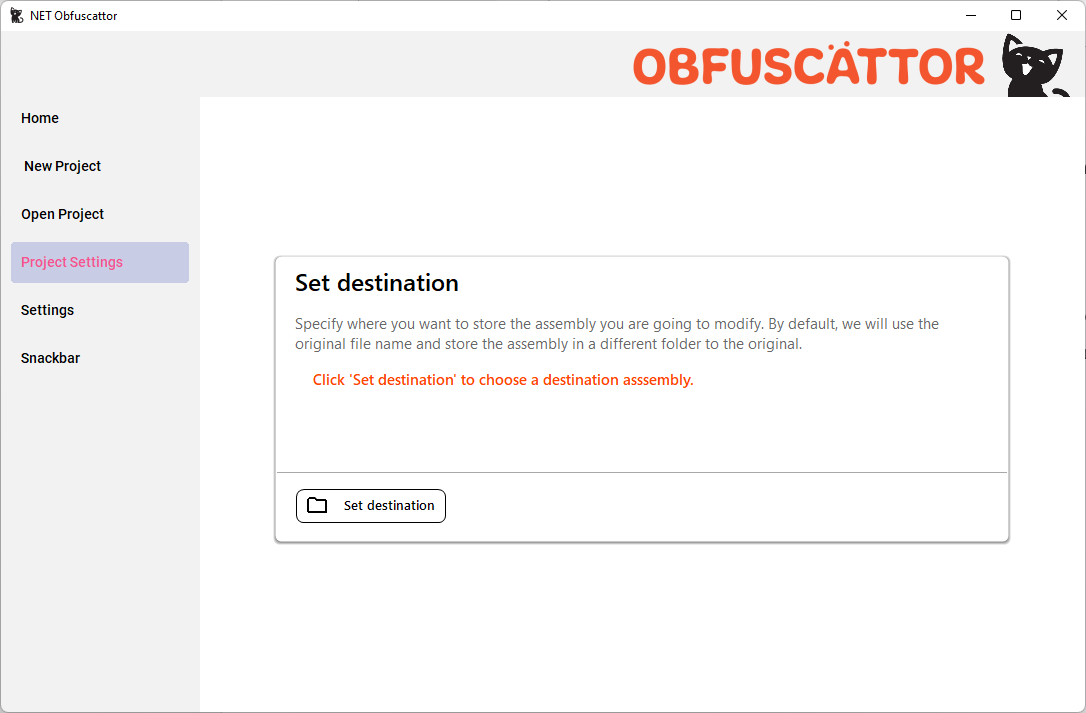


**Hình 4.7** Màn hình sau khi chọn đúng assembly

Bạn có hai quyết định ở màn hình này. Một là quay lại chọn assembly khác, hai là tiếp tục qua bước tiếp theo.

### 4.3.5 Giao diện chọn đường dẫn lưu

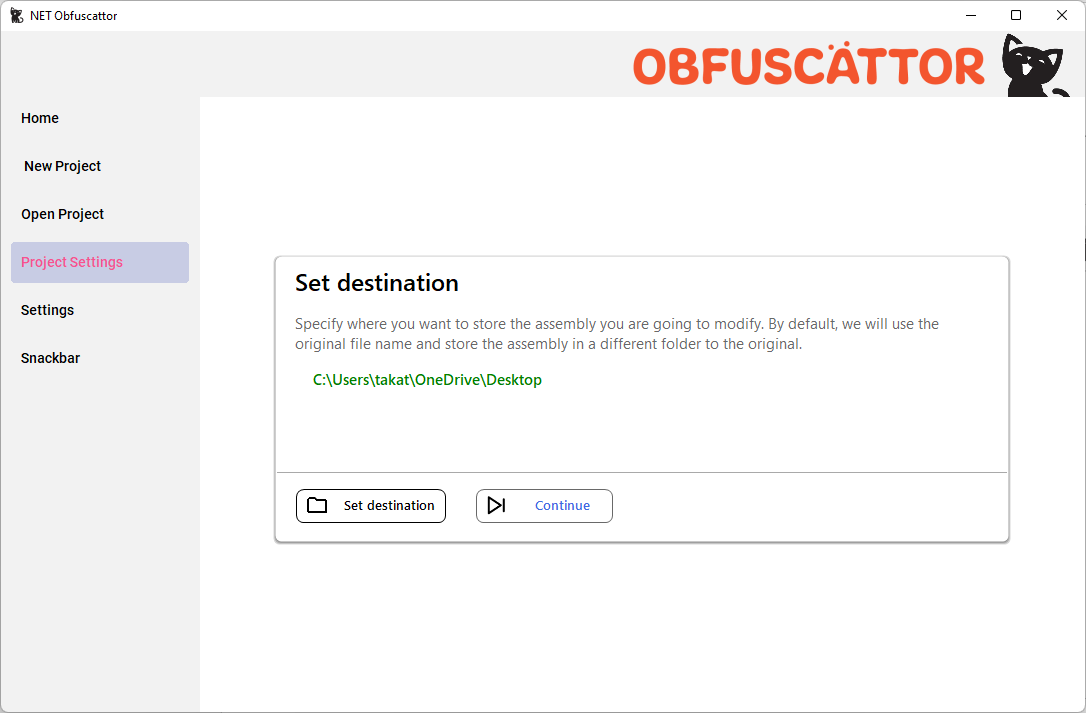
Nếu bạn chọn tiếp tục, Obfuscattor NET sẽ yêu cầu đường dẫn lưu của assembly đã mã hóa. Có nghĩa là đây sẽ là nơi lưu trữ assembly đã mã hóa của bạn sau khi thực hiện build thành công.



**Hình 4.8** Màn hình chọn đường dẫn lưu

### 4.3.6 Giao diện chọn đường dẫn lưu thành công

Nếu đường dẫn lưu bạn chọn là hợp lệ, Obfuscattor NET hiển thị nút continue để bạn có thể tiếp tục qua mục chọn mã hóa.

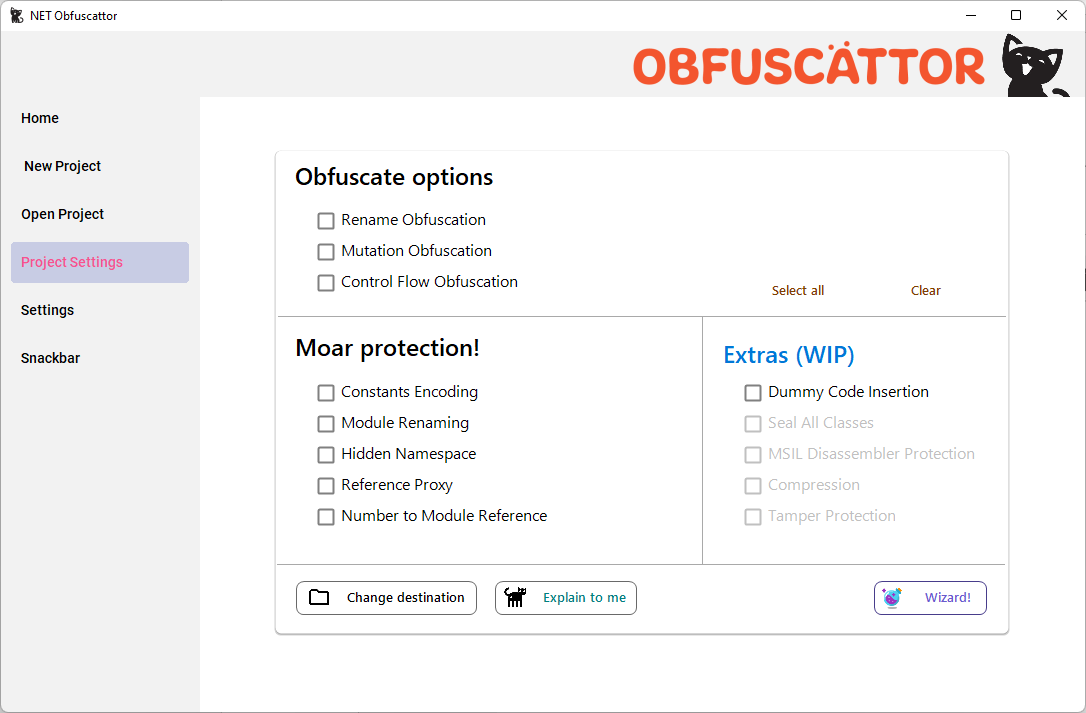


**Hình 4.9** Màn hình chọn đường dẫn lưu thành công

### 4.3.7 Giao diện chọn config mã hóa

Đây là bước quan trọng nhất, ở đây bạn chỉ việc chọn những phương pháp mà bạn muốn áp dụng bảo vệ cho assembly của bạn. Dĩ nhiên những gì dưới đây đã được giải thích ở bên trên rồi. Nhưng nếu bạn vẫn không rõ, có thể nhấn vào “Explain to me” để hiển thị một hộp thoại giải thích những option bên trên nhé.

Sau khi chọn những option, bạn có thể quay lại để chọn đường dẫn lưu thông qua button “Change destination”, hoặc chúng ta có thể bắt đầu bảo vệ phần mềm thông qua nút “Wizard”!



**Hình 4.10** Màn hình chọn config bảo vệ

### 4.3.8 Giao diện build

Cứ bình tĩnh, Obfuscattor NET đang áp dụng phép thuật vào assembly của bạn.

Graphical user interface, text, application, email

Description automatically generated

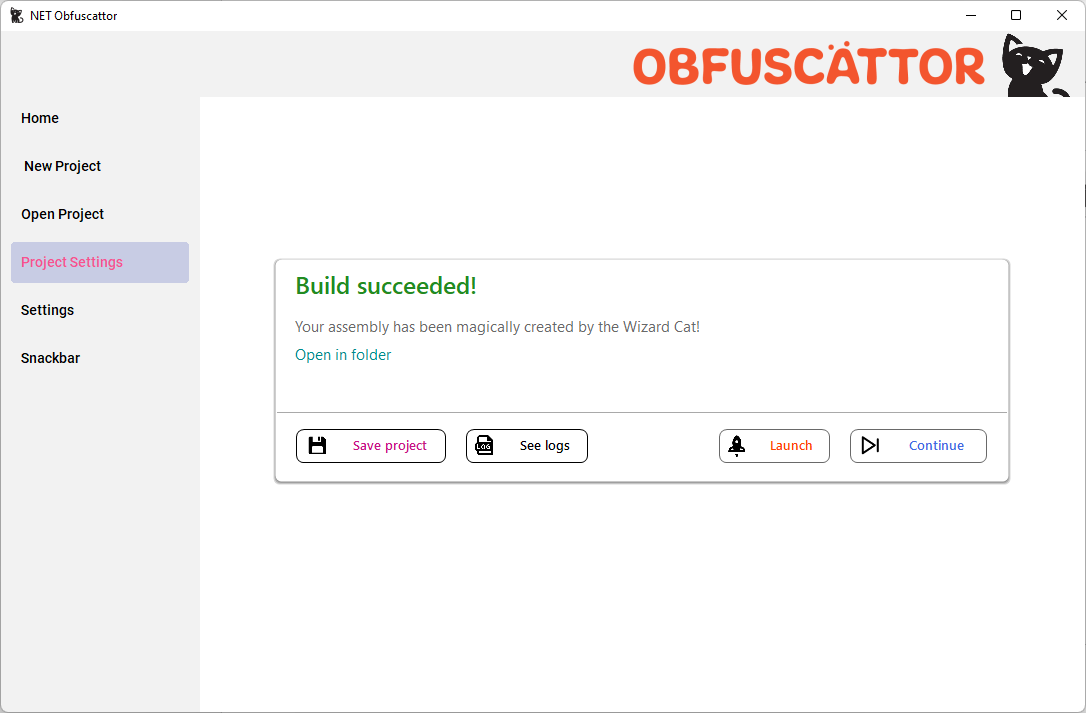
**Hình 4.11** Màn hình đang build

### 4.3.9 Giao diện build thành công

Nếu assembly được build thành công, màn hình bên dưới sẽ hiển thị. Chúc mừng bạn đã có một assembly đã được bảo vệ rồi đấy!

Ở dưới mình sẽ có nhiều option đi kèm.

* + “Save project” sẽ lưu project của bạn vào danh sách project. Một khi đã lưu, lần sau khi mở ứng dụng lên bạn có thể dễ dàng build lại assembly đó một cách nhanh chống thông qua 3 cú click chuột! Rất tiện phải không?
  + “See logs” sẽ hiển thị log của quá trình build. Thông thường nó không có gì nhiều nhưng số ít trường hợp có thể gặp lỗi. Nhưng nếu vẫn build thành công thì những lỗi này không đáng kể và Obfuscattor NET đã chọn skip chúng.
  + “Launch” để chạy assembly của bạn.
  + “Continue” để quay lại quá trình build ban đầu.

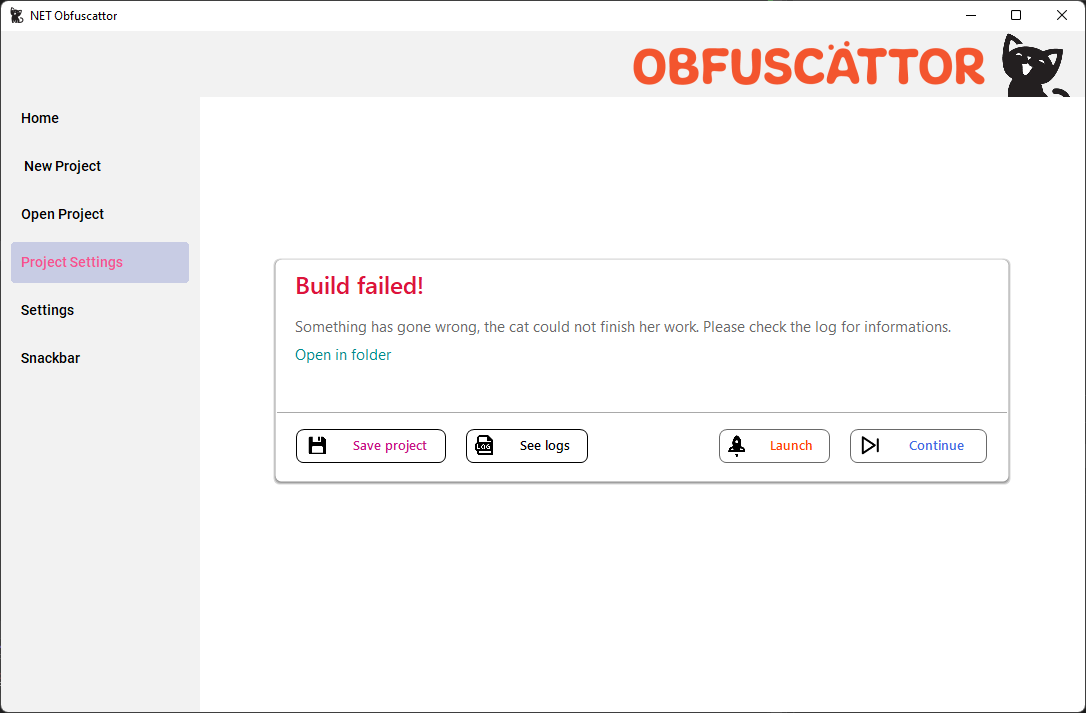


**Hình 4.12** Màn hình build thành công

### 4.3.10 Giao diện build thất bại

Dĩ nhiên sẽ có trường hợp build thất bại. Để xem lý do tại sao thì hãy nhấn vào nút “See logs” để kiểm tra đoạn nào đã khiến assembly build không thành công.

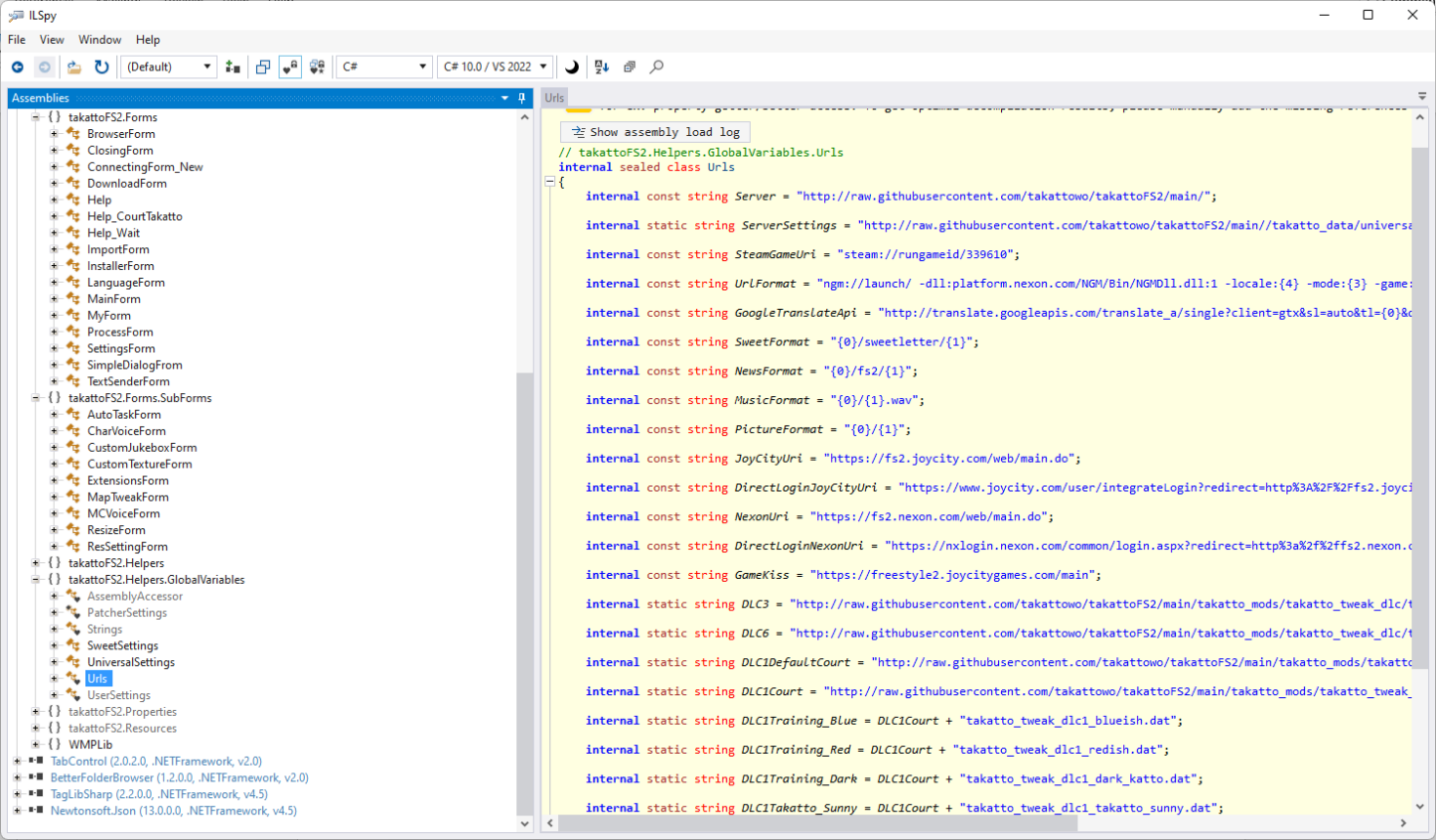
Trong một số trường hợp, re-build có thể thành công đấy.



**Hình 4.13** Màn hình build thất bại

## 4.4 Thực nghiệm và kết quả

Dưới đây là một assembly **chưa được sử dụng** Obfuscattor NET để build. Bằng cách sử dụng phần mềm dịch ngược là ILSpy, chúng ta có thể dễ thàng thấy được mã nguồn của phần mềm này.



**Hình 4.14** Một phần mềm .NET chưa được bảo vệ được dịch ngược bởi ILSpy

Có thể thấy là nó dịch ngược hoàn toàn được phần mềm đấy mà không hề sai sót gì cả. Để củng cố cho câu nói này, em sẽ so sánh với mà nguồn gốc của nó.

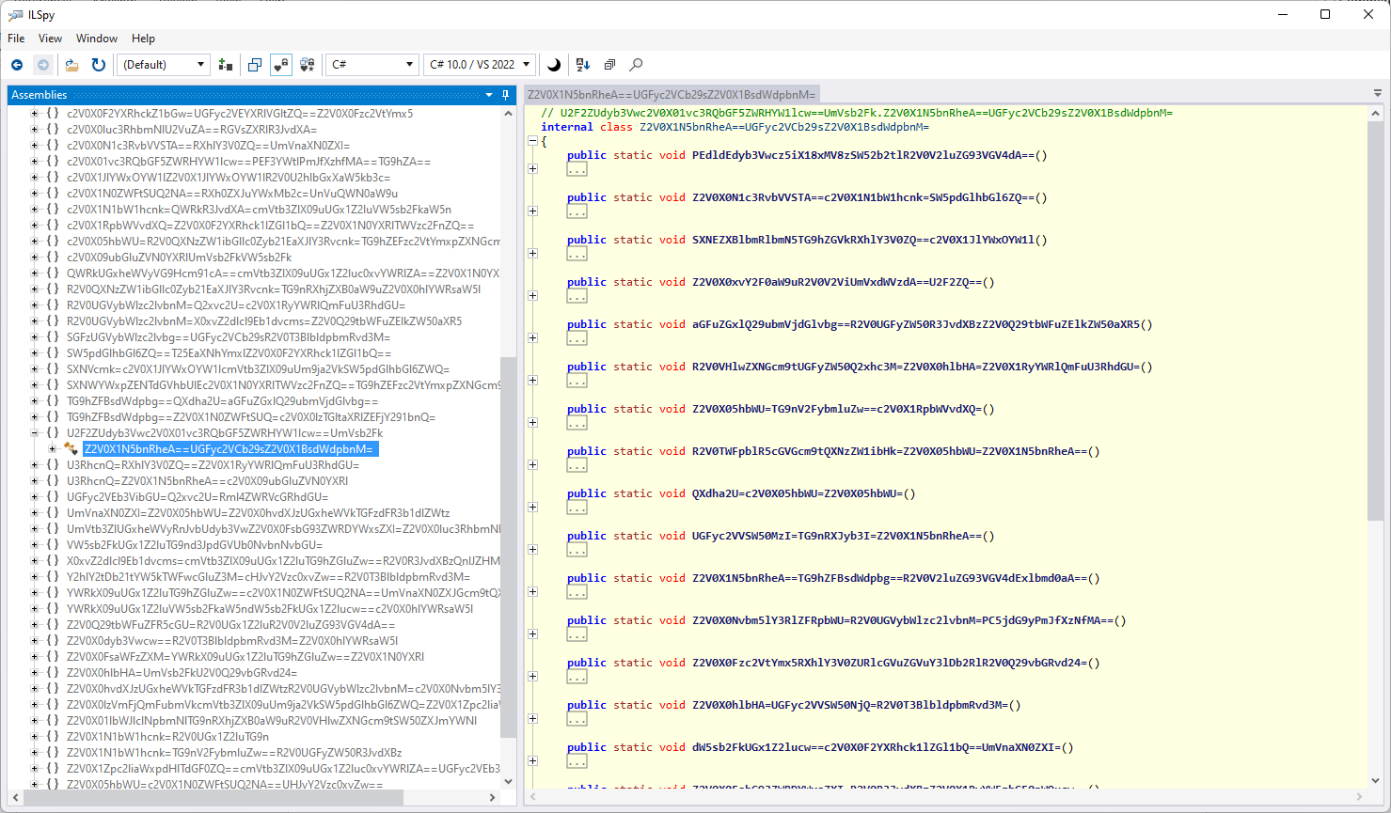
Hình ảnh bên dưới là hình ảnh chụp của mã nguồn phần mềm trên. Như bạn đã thấy, nó hoàn toàn chính xác với những gì trình dịch ngược ILSpy đã hiển thị. Đúng hết mọi thứ từ cấu trúc các thư mục, lớp cho đến nội dung bên trong. Điều này rất nguy hiểm nếu sản phẩm của bạn là một sản phẩm có nhiều bí mật thương mại hoặc đơn giản là muốn bảo vệ tài sản trí tuệ của mình.

A computer screen capture

Description automatically generated with medium confidence

**Hình 4.15** Hình ảnh mã nguồn của phần mềm trên

Dưới đây là kết quả của một assembly sau khi sử dụng Obfuscattor NET để build. Với options là everything có trong đấy.



**Hình 4.16** Hình ảnh mã nguồn sau khi sử dụng Obfuscattor NET

Như một phép màu, đúng chưa? Mã nguồn của mình bây giờ là một thứ gì đó không dành cho mắt người nữa rồi!

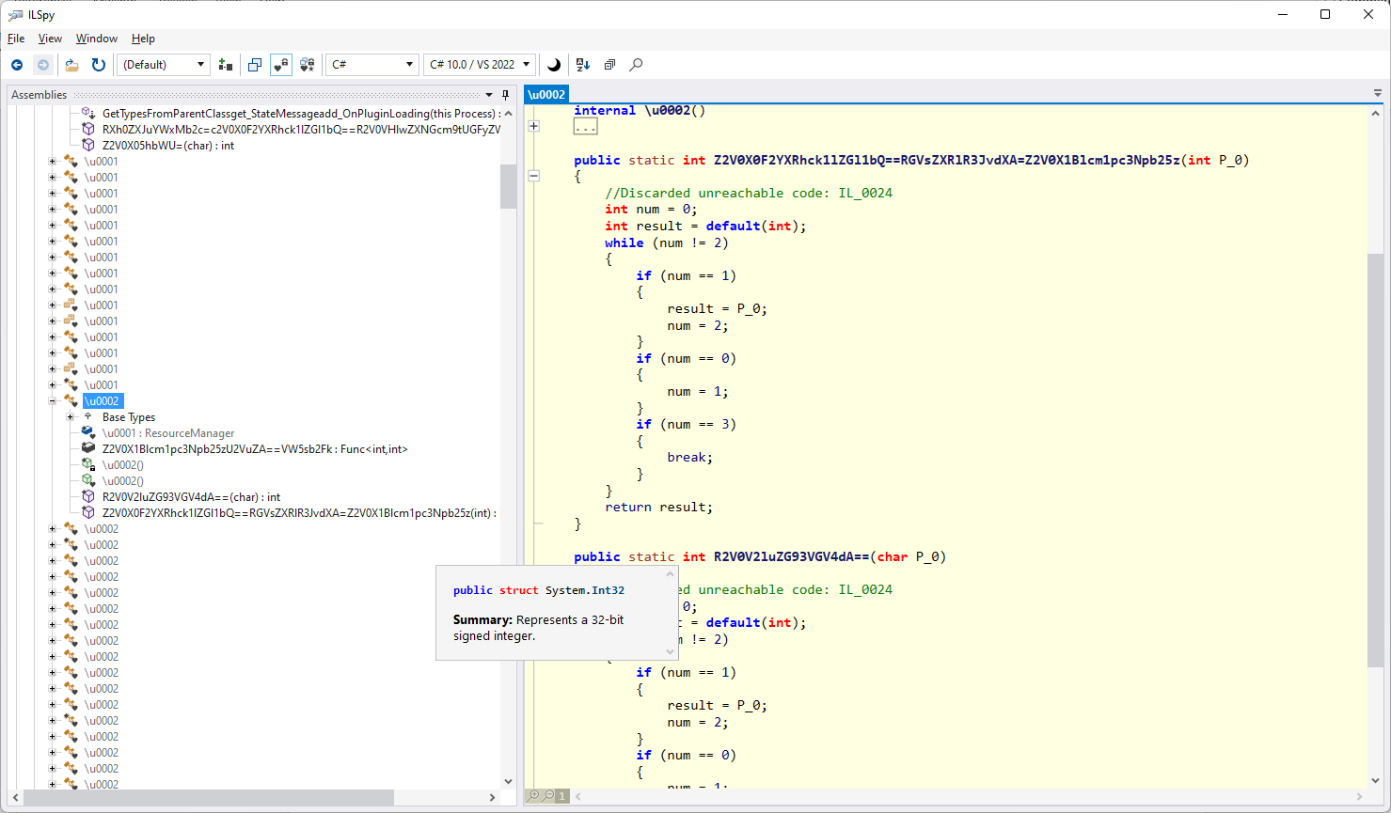
Graphical user interface, text, application

Description automatically generated

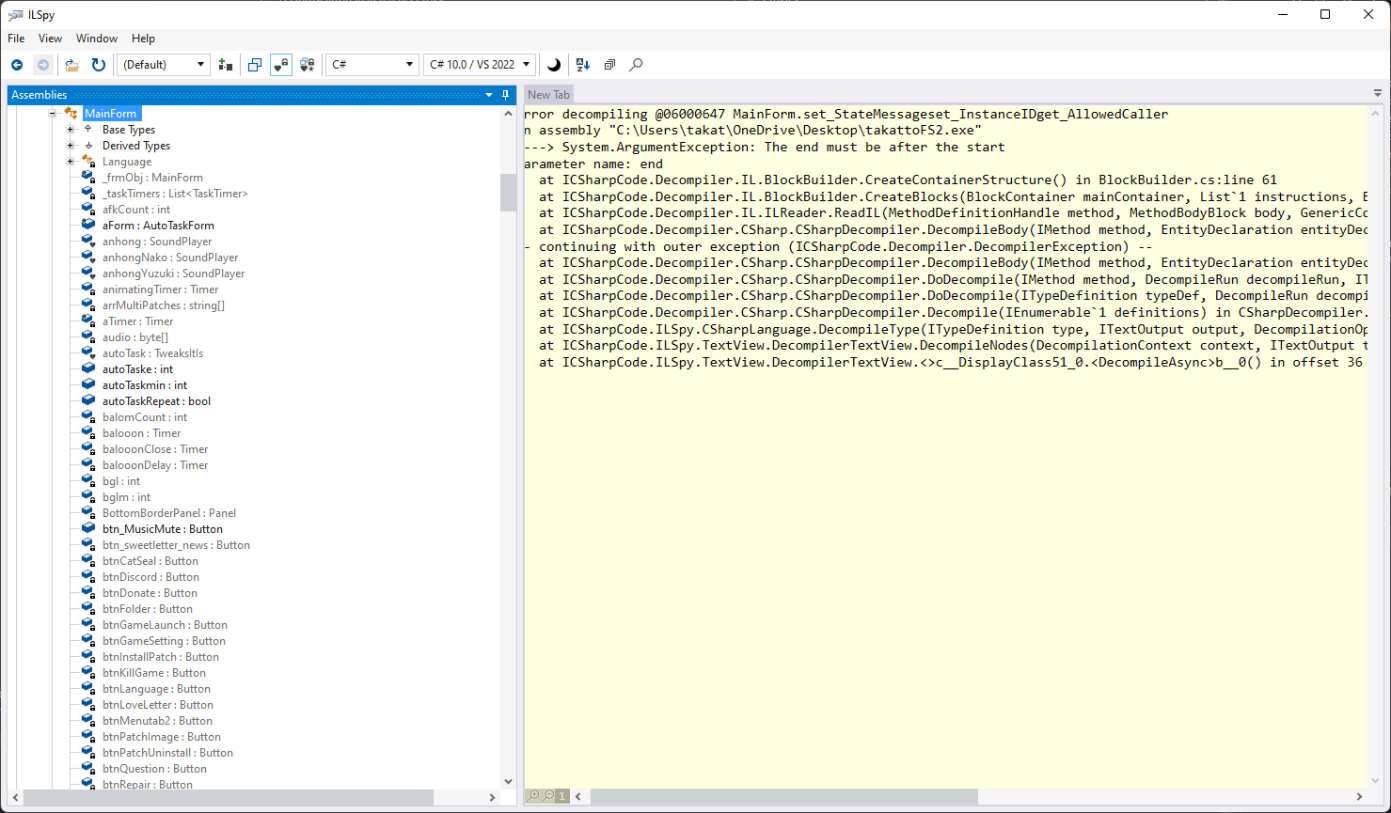
**Hình 4.17** Hình ảnh mã nguồn sau khi sử dụng Obfuscattor NET

Mọi thứ đều bị biến đổi và ẩn dấu trong một khu rừng của những lớp không rõ mục đích. Hãy thử mở rộng một số class xem nó có gì và bingo! Chả có gỉ cả!

Ngay cả bản thân em, người thiết kế ra chúng, cũng đành bó tay nếu muốn “dịch” bọn này thành code ban đầu. Và, nhiệm vụ đã thành công.



**Hình 4.18** Hình ảnh mã nguồn sau khi sử dụng Obfuscattor NET



**Hình 4.19** ILSpy không thể resolve được MainForm

# **KẾT LUẬN**

## 1 Kết quả đạt được

Trong quá trình nghiên cứu và hoàn thành đồ án tốt nghiệp với đề tài “Nghiên cứu, xây dựng và phát triển ứng dụng bảo vệ mã nguồn Obfuscattor NET cho ngôn ngữ C#”, em đã đạt được các kết quả như sau:

### 1.1 Về mặt kiến thức

* Hiểu được ngôn ngữ C# và .NET.
* Tiếp thu được nhiều kiến thức về mật mã học trong Cyber Security.
* Hiểu được cách hoạt động của IL, compiler, và mọi thứ liên quan đến .NET.
* Cải thiện kỹ năng viết thuật toán và pseudocode.
* Cải thiện khả năng và trình độ lập trình đáng kể.

### 1.2 Về mặt ứng dụng

* Hoàn thành được hầu hết mọi chức năng đã được dự kiến làm.
* Có khá nhiều thuật toán bảo vệ mang tính bảo mật khá cao.
* Phần mềm được thiết kế cực kỳ đơn giản, người không có kiến thức chuyên sâu gì cả vẫn có thể sử dụng một cách dễ dàng.
* Sản phẩm assembly đầu ra được bảo vệ như mong muốn.
* Miễn phí và open-source.

### 1.3 Về mặt con người

Trong quá trình làm đồ án thời gian làm còn khá là hạn chế, cũng như áp lực từ công việc full-time đang làm, mặc dù khá là khó khăn nhưng đó là một khoảng thời gian khá là thú vị đối với em. Từ một thằng sinh viên trẻ trung nhưng chỉ sau 3 tháng em đã mọc khá nhiểu tóc bạc cũng như thêm nhiều bệnh lý đi kèm như đau đầu, đau lưng và suy nhược cơ thể. Từ một thằng xem đồ án này là việc khó mà hoàn thành bởi vì trình độ có hạn, nhưng nhờ sự nỗ lực và giúp đỡ từ kiến thức đã được truyền dạy từ thầy cô và lượng lớn cộng đồng những người bạn, đã giúp em hoàn thành được đề tài được xem là “ao chình” đối với em.

## 2 Tồn tại hạn chế

Bên cạnh những khía cạnh đạt được, do thời gian thực hiện có hạn cùng với trình độ kiến thức còn nhiều hạn chế nên đã còn thiếu sót một số chức năng đã được mong đợi như:

* Khả năng lưu trữ stack-trace để trace-back lại phần mềm đã mã hóa.
* Khã năng bypass mã hóa cho những code trong filter.
* Khả năng bulk-obfuscation, cho phép mã hóa nhiều assembly một lúc.

Ngoài ra, việc bảo mật bằng Obfuscattor NET còn đi thêm nhiều khuyết điểm và lỗi có thể đi kèm như:

* Làm cho assembly nặng hơn so với ban đầu.
* Bị những chương trình diệt vi-rút gán nhầm là độc hại.
* Có khả năng làm hư assembly đối với những project cấu trúc lớn và phức tạp.

## 3 Hướng phát triển

Trong tương lai, nếu có điều kiện, Obfuscattor NET sẽ được:

* Phát triển khả năng bảo mật.
* Thêm nhiều loại hình thức bảo mật.
* Thêm nhiều tính năng bổ trợ.
* Mở rộng ngôn ngữ hỗ trợ đến Java.
* Giảm thiểu khuyết điểm của phần mềm gây ra được đề cập ở trên.

## TÀI LIỆU THAM KHẢO VÀ PHỤ LỤC

[1]. Được trích từ sách **The C# Player’s Guide, 3rd Edition**

- Tác giả: RB Whitaker

- Nhà xuất bản: Starbound Software

- Ngày phát hành: Ngày 13, tháng 3, năm 2016

- Nội dung được dẫn và dịch sang tiếng Việt từ **Part 1: Getting Started**

[2]. Được trích từ trang chủ chủ Microsoft, **What is Dotnet**

- Đường dẫn: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>

- Ngày truy cập: Ngày 12, tháng 5, năm 2022

[3]. Được tham khảo từ cuốn giáo trình **Code obfuscation theory and practices**

- Tác giả: Fang Hui

- Ngày phát hành: Ngày 18, tháng 3, năm 2014

[4]. Được trích từ bài viết **Các công cụ cần thiết cho quá trình Reverse Engineering .NET (bản đầy đủ)**

- Đường dẫn: <https://www.slideshare.net/Levintaeyeon/cc-cng-c-cn-thit-cho-qu-trnh-reverse-engineering-net-bn-y>

- Tác giả: Levis Nickaster

- Ngày truy cập: Ngày 13, tháng 5, năm 2022

[5]. Được tham khảo từ website của Dotfuscator **User Guide**

- Đường dẫn: <https://www.preemptive.com/dotfuscator/pro/userguide/en/protection_obfuscation_renaming.html>

- Tác giả: Dotfuscator

- Ngày truy cập: Ngày 17, tháng 5, năm 2022

[6]. Được tham khảo từ bài viết **String Encryption Tutorial**

- Đường dẫn: <https://www.zelix.com/klassmaster/docs/tutorials/stringEncryptionTutorial.html>

- Tác giả: Zelix KlassMaster

- Ngày truy cập: Ngày 18, tháng 5, năm 2022

[7]. Được tham khảo từ một câu hỏi đăng bởi perror và những câu trả lời từ nhiều user được viết trên StackExchange **What is a "control-flow flattening" obfuscation technique?**

- Đường dẫn: <https://reverseengineering.stackexchange.com/questions/2221/what-is-a-control-flow-flattening-obfuscation-technique>

- Ngày truy cập: Ngày 22, tháng 5, năm 2022

[8]. Được tham khảo từ cuốn giáo trình **PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), ISSN 0033-2097, R. 88 NR 10b/2012**

- Tác giả: Jun KAWANO, Hiroshi KAI, Yoshinobu HIGAMI, Shinya KOBAYASHI

- Ngày phát hành: năm 2012

- Được tham khảo từ phần **Dummy code insertion and its efforts on concealment for secure processing**

[9]. Được tham khảo từ bài viết **Merging dependencies**

- Đường dẫn: <https://documentation.red-gate.com/sa6/obfuscating-your-code-with-smartassembly/merging-dependencies>

- Tác giả: SmartAssembly

- Ngày đăng: ngày 20, tháng 5, năm 2013

- Ngày truy cập: Ngày 25, tháng 5, năm 2022

[10]. Được tham khảo từ bài viết **Creating a proxy for external calls**

- Đường dẫn: <https://documentation.red-gate.com/sa6/obfuscating-your-code-with-smartassembly/creating-a-proxy-for-external-calls>

- Tác giả: SmartAssembly

- Ngày đăng: ngày 16, tháng 10, năm 2014

- Ngày truy cập: Ngày 25, tháng 5, năm 2022

[11]. Được tham khảo từ một ý tưởng của user Rottweiler trên Github, issue, **Anti-tampering solution**

- Đường dẫn: <https://github.com/XenocodeRCE/Noisette-Obfuscator/issues/13>

- Tác giả: Rottweiler

- Ngày đăng: ngày 1, tháng 3, năm 2017

- Ngày truy cập: Ngày 29, tháng 5, năm 2022

Các tài liệu tham khảo thêm:

1. Bài viết **A Study on Control Flow Flattening Technique**

- Đường dẫn: <https://zongyuwu.medium.com/a-study-on-confuserex-control-flow-flattening-technique-a93030f05acc>

- Tác giả: ZYWU

- Ngày đăng: ngày 16, tháng 1, năm 2019

- Ngày truy cập: Ngày 3, tháng 5, năm 2022

2. Bài viết **Tóm tắt các phương pháp giải mã ngôn ngữ C#** (tiêu đề được dịch sang tiếng Việt từ tiếng Trung Quốc)

- Đường dẫn: <https://www.cxyzjd.com/article/chenniangu7653/100915547>

- Tác giả: chenniangu7653

3. Giáo trình **C Code and the Art of Obfuscation**

- Tác giả: Lloyd Markle

- Ngày phát hành: ngày 2, tháng 2, năm 2007

4. Giáo trình **Algorithm Design & Implementation**

- Tác giả: Sumaira Hussain

- Ngày phát hành: ngày 11, tháng 1, năm 2016

Đường dẫn đến repository của đồ án này: <https://github.com/takattowo/ObfuscattorNET>