Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.1, Feb. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.1.129

ConfuserEx의 난독화 복구 자동화 시스템 구축 연구

이 재 휘,† 박 영 석, 김 동 현, 허 규‡
NAVER Corporation (직원)

A Study on Implementing an Automated Tool for De-Obfuscating a ConfuserEx

Jae-hwi Lee,† Young-seok Park, Dong-hyeon Kim, Gyu Heo‡
NAVER Corporation (Staff)

요 약

매크로 프로그램을 이용하여 포털 사이트의 컨텐츠를 생산하거나 변경하려는 시도는 계속해서 발생하고 있으며, 국내 포털 사이트인 네이버 또한 이러한 시도에 대응하기 위해 프로그램을 확보하고 분석할 수 있도록 노력하고 있다. 그러나 확보한 프로그램에 난독화가 적용된 경우가 있어, 난독화에 대응하는 방안 또한 필요한 것으로 보인다. 본 논문에서는 확보한 프로그램에 적용된 난독화 도구 중 높은 비율을 차지한 ConfuserEx에 대해 분석하고, 실행 파일에 적용된 분석방해 기능을 자동으로 해제하는 시스템을 제안해 난독화 해제에 소비되는 시간을 단축할 수 있도록 한다.

ABSTRACT

According to a continuous attempts to manipulate content on portal sites using automated programs, a Naver, one of a portal site from Korea, is also trying to secure and analyze the programs to respond to the attempts. However, since some of the programs are secured by obfuscation tools, it is necessary to develop de-obfuscation technique. In this paper, we analyze a ConfuserEx, which occupied high percentage from obfuscation tools that applied to obtained programs, and propose an automated tool for de-obfuscating to save time for unpacking.

Keywords: ConfuserEx, .NET obfuscation, Unpacking

I. 서 론

일반적인 사용자가 브라우저나 모바일 앱을 통해 접근하는 것과는 다르게, 매크로 프로그램을 이용해 포털 사이트에 컨텐츠를 생산하거나 변경하려는 시도가 계속해서 발생하고 있다. 이는 다른 사용자들의 정상적인 서비스 이용을 방해하므로, 포털 사이트의 서비스 신뢰도를 떨어뜨리는 피해를 유발할 수 있다. 국내 인터넷 포털 사이트인 네이버 또한 이러한 시도에 대응하기 위해 모니터링을 지속하고 있으며, 다양한 프로그램을 확보하고 분석해 차단하고 있다.

최근 3년간 네이버에서 확보한 프로그램 중 난독화가 적용된 비율은 약 30%였으며, 일부 프로그램의 경우 사용자가 직접 개발한 것으로 보이는 난독화가 적용되어 있기도 했다. 따라서 매크로 프로그램을 이용하는 컨텐츠 생산 또는 변경 시도에 대처하기 위해서 프로그램들이 적용하고 있는 난독화에 대한 연구가 필요하다.

네이버에서 확보한 프로그램에 적용된 난독화 도구의 비율은 Fig. 1.과 같으며, 그중 높은 비율을 차지한 ConfuserEx[1]에 대해 알아보고, 실행 파일에 적용된 분석방해 기능을 자동으로 해제하는 시스템을 제안해 난독화 해제에 소비되는 시간을 단축하고자 한다.

Received(10. 17. 2022), Modified(12. 12. 2022), Accepted(12. 13. 2022)
† 주저자, jaehwi.lee@navercorp.com
‡ 교신저자, hg.bg@navercorp.com(Corresponding author)

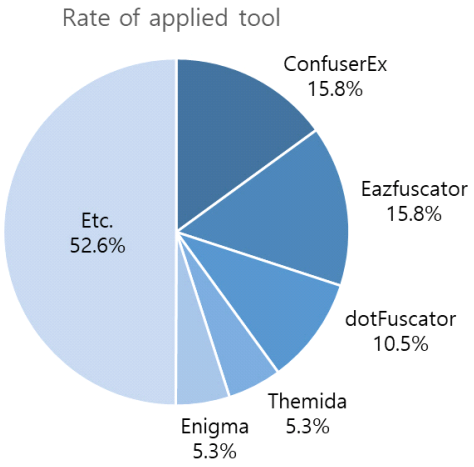

Fig. 1. Rate for applied obfuscation tools

논문의 구성은 다음과 같다. 2장에서는 닷넷 실행 파일과 관련된 기존 난독화 해제 연구를 알아보고, 3장에서는 ConfuserEx가 지원하는 분석방해 기능에 대해 알아본다. 4장에서는 ConfuserEx가 적용된 실행 파일에서 분석방해 기능을 해제하는 방안을 설명하고, 본 연구에서 제안하는 해제 작업을 자동으로 진행하는 시스템의 적용 결과를 확인한다. 마지막 5장에서는 본 연구의 결론 및 향후 연구 방향을 기술한다.

II. 관련 연구

2.1 CIL 레벨 난독화 해제

닷넷(.NET) 실행 파일은 Microsoft 사의 Windows 운영체제 기반 프로그램 개발 및 실행 환경인 닷넷 프레임워크(framework)의 CLR (Common Language Runtime) 가상 머신 위에서 작동한다. 닷넷 실행 파일은 바이트코드(bytecode) 명령어인 CIL(Common Interface Language)과 실행에 필요한 정보인 메타데이터(metadata)를 포함하고 있으며, 실행 시 CLR의 JIT(Just In Time) 컴파일러가 실행 환경에 맞는 native 코드를 생성할 때 이 정보들을 사용한다[2]. 메타데이터에는 함수와 파라미터(parameter) 등 다양한 정보를 포함하고 있으며, 개발자가 지정한 이름까지도 존재한다. 그러므로 CIL을 해석하면 C#과 같은 고급 언어로 디컴파일(decompile)이 가능하다. Fig. 2.는 메타데이터의 Name에 개발자가 지정한 'label1'이라는 이름이 저장된 것을 보여준다.

고급 언어로 디컴파일이 가능하다는 점 때문에 닷넷으로 개발된 프로그램의 코드를 식별하기 어렵도록 만들기 위해 CIL 레벨의 난독화를 적용하기도 한다. 현재 다양한 CIL 레벨의 난독화 도구가 존재하며, 이 도구들은 프로그램 분석에 활용할 수 있는 문자열을 암호화해 은닉하거나, 클래스나 함수의 이름을 ASCII 범위 밖의 값으로 변경해 육안으로 서로 다른 함수를 구분하기 어렵게 만드는 등 다양한 분석방해 기능을 적용한다. 그러나 메타데이터와 CIL이 노출된 것은 변함없으므로, 이를 이용해 어느 정도 복구가 가능하다.

de4dot은 CIL 레벨의 난독화를 해제하는 오픈소스 도구이다[3]. 해당 도구를 이용해 해제할 수 있는 CIL 레벨의 난독화 도구는 Table 1.과 같으며, 난독화가 적용된 파일을 입력하면, 자동으로 난독화가 해제된 실행 파일을 생성해 준다. 만약 지원하지 않는 난독화 도구라 하더라도, 사용자가 직접 난독화 관련 함수의 토큰(token)을 입력하면 난독화가 풀리기도 한다. 그러나 본 논문의 연구 대상인 ConfuserEx의 경우 CIL 레벨의 난독화임에도 난독화가 정상적으로 풀리지 않았으며, Fig. 3.과 같이 코드 식별이 여전히 어려운 것을 알 수 있다.

Member	Offset	Size	Value	Meaning
Flags	00000A12	Word	0001	Click here
Name	00000A14	Word	0001	label1
Signature	00000A16	Word	00C9	Blob Index

Fig. 2. User defined name stored in Metadata

Table 1. Obfuscators that de4dot supports

Obfuscators	
Agile.NET	Babel.NET
CodeVeil	CodeWall
CryptoObfuscator	DeepSea
Dotfuscator	Eazfuscator.NET
Goliath.NET	ILProtector
MPRESS	MaxtoCode
Rummage	Skater.NET
SmartAssembly	Spices.NET
Xenocode	.NET Reactor

```
switch ((num2 = (num ^ 3616983770U)) % 44U)
{
case 0U:
    GForm0.smethod_1(this.textBox_1, <Module>.smethod_2<str
    num = 3185892484U;
    continue;
case 1U:
    GForm0.smethod_10(this.textBox_2, new Point(271, 55));
    num = (num2 * 736603130U ^ 4018230153U);
    continue;
case 2U:
    GForm0.smethod_11(this.label_2, <Module>.smethod_5<stri
    num = 3329742001U;
    continue;
```

Fig. 3. Unavailable to deobfuscate ConfuserEx by de4dot

2.2 ConfuserEx 난독화 해제

ConfuserEx로 보호된 실행 파일을 복구하는 오픈소스 도구가 Table 2.와 같이 존재한다. 범용적인 CIL 난독화 해제 도구들과 달리 ConfuserEx의 알고리즘을 분석해 그 특징에 맞게 난독화를 해제한다. 그러나 모든 도구의 최종 업데이트 날짜가 수년 전으로, 현재에도 코드가 계속해서 업데이트되고 있는 ConfuserEx에는 정상적인 대응이 어려울 것으로 보이며, 실제로 테스트한 결과 ConfuserEx의 최신 버전인 1.6 버전으로 난독화 한 실행 파일은 해제가 되지 않았다.

Table 2. deobfuscating tools for ConfuserEx

ConfuserEx deobfuscators	
NoFuserEx[4]	ConfuserExSwitchKiller [12]
Netguard-Unpacker-Public[5]	ConfuserEx-Anti-Debug-Remover[13]
ClarifierEx[6]	ConfuserEx-Static-String-Decryptor[14]
Rzy-Protector-V2-Unpacker[7]	ConfuserEx-Resources-Decryptor[15]
ConfuserEx-Dynamic-Unpacker[8]	ConfuserExResourceReplace[16]
ConfuserEx-Unpacker-Mod-By-Bed[9]	ConfuserExTools[17]
ConfuserEx-Unpacker-2[10]	Unscrambler[18]
Krawk-Unpacker[11]	EasyPredicateKiller[19]

III. ConfuserEx의 분석방해 기능

ConfuserEx의 최신 버전인 1.6 버전에서는 Table 3.과 같이 다양한 분석방해 기능을 지원한다. 사용자가 선택한 옵션에 따라 분석방해 기능을 실행 파일에 추가하며, 기본적으로 분석방해 기능은 실행 파일의 C# 모듈 initializers[20]인 '<Module>' 내부에 생성된다. 그러므로 '<Module>' 내부의 코드를 분석하면, 각 분석방해 기능을 찾을 수 있다.

분석방해 기능은 실행 과정에서 관찰할 수 있는 유형과 실행 파일 내부 정보가 변하는 유형이 있다. 실행 과정에서 관찰할 수 있는 보호 기능은 'Anti-debug', 'Anti-dump', 'Anti-tamper', 'Constants', 'Resources' 옵션이다. 'Anti-debug' 옵션은 현재 실행 중인 프로세스에 디버거(debugger)가 연결되어 있는지를 검사하는 쓰레드(thread)를 생성한다. 'Anti-dump' 옵션은 실행 중인 프로세스에서 실행 파일을 덤프(dump)할 수 없도록, 실행 중 메모리에 직접 접근해 PE 헤더[21]의 일부 정보를 지운다. 'Anti-tamper' 옵션은 바이트코드를 인코딩해 실행 파일 내부에 저장하여 바이트코드가 노출되지 않도록 숨긴다. 따라서 코드가 나타나지 않으므로, 필요에 따라 코드의 흐름을 수정할 수 없게 된다. 숨겨진 바이트코드는 실행 시점에 디코딩해 복구한 후 실행한다. 'Constants' 옵션은 기존 실행 파일의 #US 스트림에 존재하는 문자열 정보를 Fig. 4.와 같이 문자열 길이, 문자열 형태의 배열로 생성하고, 이를 인코딩해 실행 파일 내부에 저장한다. 그리고 Fig. 5.와 같이 생성한 배열에서 문자열을 읽어오는 함수를 생성하고, 이를 이용해 문자열을 읽어오도록 기존 코드를 수정한다. 'Resources' 옵션은 리소스 정보를 저장하고 있는 모듈을 새로 생성해 실행 파일 내부에 인코딩해 저장하고, 실행 시점에 디코딩해 모듈을 로드 한 다음 사용한다. 기존 실행 파일에 존재하던 리소스 정보는 모두 삭제되므로, 인코딩된 모듈을 복구하지 않으면 리소스 정보를 파악할 수 없다.

실행 파일의 내부 정보가 변하는 보호 기능은 'Anti-IL disassemble', 'Control flow', 'Protection hardening', 'Invalid metadata', 'Reference proxy', 'Rename', 'Type scramble', 'Watermark' 옵션이다. 'Anti-IL disassemble' 옵션은 Windows SDK(Software Development Kits)에서 기본 제공하는 바이트코

Table 3. Protection options for ConfuserEx v1.6

Protection option		Description
Observable during execurtion	Anti-debug	Check debugger is running
	Anti-dump	Damage header within memory
	Anti-tamper	Conceal bytecode
	Constants	Conceal #US stream data
	Resources	Conceal resource
File modified	Anti-IL disassemble	Set SuppressIldasmAttribute
	Control flow	Shuffle code order
	Protection hardening	Check other protection works fine
	Invalid metadata	Add invalid metadata
	Reference proxy	Add wrapper function
	Rename	Rename into inexpressible character set
	Type scramble	Set type to generic(T)
	Watermark	Add watermark

드 명령어 해석 도구인 ILDasm이 디스어셈블(disassemble)을 하지 않도록 설정하는 'SuppressIldasmAttribute' 클래스[22]를 모듈에 추가한다. 'Control flow' 옵션은 순차적으로 나타나던 원본 코드를 뒤섞어 실행 순서를 파악하기 어렵게 만든다. Fig. 6.과 같이 결과를 식별하기 어려운 산술식을 참조해 분기하도록 switch나 goto 등의 명령어를 생성하고, 반복문으로 감싼다. 'Protection hardening' 옵션은 다른 옵션이 설정된 상태에서만 사용할 수 있으며, 다른 보호 기능들이 정상적으로 작동하고 있는지 확인해 분석을 더 어렵게 만든다. 'Invalid metadata' 옵션은 손상된

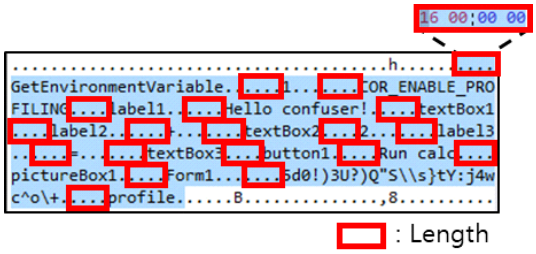

Fig. 4. Decoded constants stream on memory

```
int count = (int)<Module>.byte_0[id] |
   (int)<Module>.byte_0[id + 1] << 8 |
   (int)<Module>.byte_0[id + 2] << 16 |
   (int)<Module>.byte_0[id + 3] << 24;
string result = (T)((object)string.Intern(
   Encoding.UTF8.GetString(<Module>.byte_0, id + 4, count)));
```

Fig. 5. Generated code to read decoded constants

가짜 metadata 정보를 파일에 추가해 분석 도구가 오작동하도록 유도한다. Metadata 헤더에서 스트림의 개수를 나타내는 'NumberOfStreams' 값이 늘어나고, Fig. 7.과 같이 크기가 작은 가짜 스트림을 생성한다. 'Reference proxy' 옵션은 외부 모듈의 함수를 호출하는 래퍼(wrapper) 함수를 생성하고, 기존 코드에서 래퍼 함수를 거쳐 외부 모듈의 함수를 호출하도록 만든다. 코드가 호출하는 외부 모듈의 함수 정보가 바로 나타나지 않으므로, 코드가 호출하는 함수를 파악하기 어렵게 만든다. 'Rename' 옵션은 metadata 상에 존재하는 모든 name 정보를 ASCII 범위 밖의 표시할 수 없는 문자로 변경한다. 'Type scramble' 옵션은 코드에서 나타나는 모든 타입을 generic(T)으로 바꾸어 각 변수의 역할을 유추하기 어렵게 만든다. 'Watermark' 옵션은

```
uint num = 2093939763U;
for (;;)
{
    uint num2;
    switch ((num2 = (num ^ 1512496066U)) % 4U)
    {
    case 0U:
        ₩u200E₩u206E₩u206C₩u206E₩u202B₩u206C₩u200D₩u202A₩u20
          ₩u206F₩u200D₩u202D₩u202E₩u202E₩u206A₩u202E₩u206C₩u
          ₩u202A₩u200D₩u202B₩u206E₩u206A₩u200B₩u202E₩u202D₩u
        num = (num2 * 2367808897U ^ 3588603384U);
        continue;
    case 1U:
        ₩u200E₩u206E₩u206C₩u206E₩u202B₩u206C₩u200D₩u202A₩u20
          ₩u206F₩u200D₩u202D₩u202E₩u202E₩u206A₩u202E₩u206C₩u
          ₩u202D₩u206A₩u202D₩u206F₩u202C₩u206B₩u200F₩u206D₩u
        num = (num2 * 1127686882U ^ 3916289400U);
        continue;
    case 3U:
        goto IL_05;
```

Fig. 6. Control flow protection applied code

Offset	Size	Name
Dword	Dword	szAlignedAnsi
000000A8	0000108C	#~
00001134	00005CB8	#Strings
00006DEC	00000010	#GUID
00006DFC	00000688	#Blob
00007484	00000010	#GUID
00007494	00000004	#Strings
00007498	00000004	#Blob
0000749C	00000004	#Schema

Fig. 7. Abnormal stream lengths for invalid metadata

ConfuserEx를 이용해 실행 파일을 난독화 하였음을 나타내는 워터마크를 추가한다.

IV. ConfuserEx의 분석방해 기능 해제

4.1 분석방해 기능 해제 방안

앞서 분석한 ConfuserEx의 최신 버전인 1.6 버전의 분석방해 기능별 알고리즘에 따라 CIL 레벨에서 복구를 진행하였다. 보호된 실행 파일의 원본을 복구하는 방안은 Table 4.와 같다. 'Type scramble'은 데이터가 복구되면 generic 타입이어도 실제로 어떤 타입을 가졌는지 식별이 가능하고, 'Watermark'는 어떤 난독화가 적용되었는지를 나타내는 정보만 추가한 것이므로 제외하였다.

먼저 보호된 실행 파일을 실행해 메모리를 덤프한다. 실행 파일의 헤더가 손상되었지만, 메모리상의 주소는 변하지 않았으므로 초기 실행 파일이 매핑(mapping)된 주소를 그대로 덤프한다. 덤프한 파일에는 디코딩된 바이트코드가 남아있고, 실행 이전의 보호된 파일에는 헤더 정보가 남아있다. 따라서 두 파일에 남아있는 데이터를 합쳐 하나의 실행 파일을 생성할 수 있으며, 이때 헤더 정보에서 불필요한 가짜 메타데이터 헤더를 제거한다. 다음으로 생성한 실행 파일의 메타데이터를 식별할 수 있도록 내부 이름 정보를 복구하고, 래퍼 함수를 제거해 코드의 흐름을 식별할 수 있는 형태로 복구한다.

이후 과정에서는 생성한 실행 파일을 직접 실행하며 정보를 습득해 추가로 실행 파일을 복구해야 한다. 만약 수작업으로 분석방해 기능을 해제하는 상황

Table 4. Process for unpacking ConfuserEx v1.6

No.	Process	Object options
1	Dump memory	Anti-tamper
2	Restore header	Anti-dump
		Invalid metadata
3	De-mangle name	Rename
4	Remove wrapper	Reference proxy
5	Conceal debugger	Anti-debug
		Anti-IL disassemble
6	Restore resource	Resources
7	Restore string	Constants
8	Restore control flow	Control flow
9	Remove obfuscator	Protection hardening

이라면, 생성한 실행 파일을 디버거 탐지 우회 기능을 기본으로 지원하는 dnSpy[23]를 이용해 열거나, 디버거 탐지 쓰레드 실행 코드를 제거해야 한다. 디버거 탐지 쓰레드 실행 코드는 모듈 initializers 생성자(constructor) 함수 내부에 위치한다. 디버거가 탐지되지 않고 정상적으로 실행할 수 있는 상태가 되면, 프로세스의 메모리에서 필요한 정보를 추출해 남은 복구 과정을 진행해야 한다.

우선 데이터와 관련된 복구를 진행해야 한다. 모듈 initializers 생성자가 호출하는 함수를 따라가다 보면 Fig. 8.과 같이 'Assembly.Load' 함수를 통해 모듈을 로드하는 코드가 나타난다. 코드를 실행하면 임의의 난수 문자열을 이름으로 가진 모듈이 메모리에 올라가며, 모듈 내부에는 기존 실행 파일의 리소스 정보가 존재한다. 해당 모듈에서 리소스 정보를 추출해 실행 파일에 저장해 복구한다. 다음으로 모듈 initializers 내부에는 int 타입의 파라미터를 받아서 generic 타입의 결과를 반환하는 함수가 다수 존재하는데, 해당 함수들은 전달받은 파라미터에 따라 기존 실행 파일의 #US 스트림에 존재했던 문자열을 반환한다. 각 함수에서 필요한 정보를 추출할 수 있도록 Fig. 9.와 같이 IL 레벨에서 후킹(hooking) 코드를 삽입해 전달받은 파라미터와 반환하는 문자열을 기록하고, 함수를 호출하는 코드를 문자열로 치환해 복구한다.

```
byte[] array;
<Module>.assembly_0 = Assembly.Load(<Module>.smethod_1(array));
num2 = (num3 * 4226155924U ^ 4292012836U);
```

Fig. 8. Loading resource module via Assembly.Load

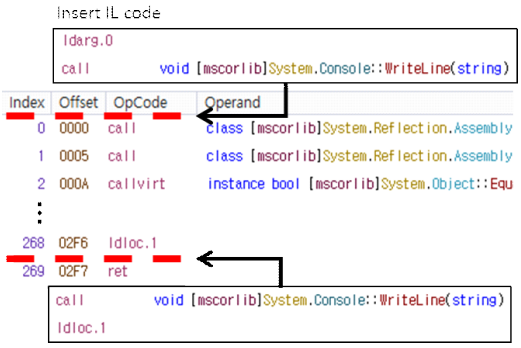

Fig. 9. Insert IL code to log parameter and result

데이터 복구가 완료되면, 실행하는 코드의 복구가 필요하다. 기존의 순차적으로 실행되는 코드는 난독화로 인해 분기문과 반복문을 사용해 뒤섞이게 된다. 그러나 결국에는 기존 코드의 순서대로 실행되어야 하므로, 각 분기문 내의 코드가 실행되는 순서를 기록해 코드의 순서를 복구할 수 있다.

모든 복구 과정이 끝나면 모듈 initializers 내부의 코드는 더 이상 실행 과정에 직접적으로 영향을 미치는 작업을 하지 않으므로, Fig. 10.과 같이 모듈 initializers 내부의 모든 클래스와 함수를 삭제해 난독화 모듈을 제거할 수 있다.

Fig. 10. Components removed in Module initializers

4.2 분석방해 기능 해제 시스템 구조

본 연구에서 제안하는 ConfuserEx로 보호된 실행 파일을 복구하는 자동화된 시스템의 전체적인 구조는 Fig. 11.과 같다. 시스템은 C#으로 개발하였으며, 닷넷 실행 파일의 정보를 해석해주는 dnlib 라이브러리[24]를 사용했다. 시스템의 구조는 크게 2개의 단계로 구분되며, 모든 단계마다 파일을 실행해 프로세스 메모리에서 정보를 획득하고, 이를 토대로 작업을 진행한다.

우선 첫 번째 단계에서는 프로세스에서 현재 실행 중인 실행 파일의 영역 메모리를 덤프한다. 덤프한 메모리는 난독화로 인해 헤더 정보가 손상된 상태이므로, 보호된 실행 파일의 헤더와 조합해 덤프 파일을 실행할 수 있는 형태로 복구한다. 그리고 복구한 실행 파일 내부의 이름 정보를 인식할 수 있는 ASCII 범위 내의 문자열 형태로 복구하고, 작업이 다른 함수를 호출하고 종료하기만 하는 함수를 래퍼 함수로 간주해 제거한다. 마지막으로 generic 타입으로 숫자를 입력받는 함수의 시작 지점에서는 파라미터 정보를 기록하고, 종료 지점에서는 결과 정보를 기록하는 후킹 코드를 삽입한다. 삽입된 코드는 추후 실행되면서 Fig. 12.와 같이 각 함수가 입력받은 파라미터와 반환한 문자열을 기록한다. 첫 번째 단계의

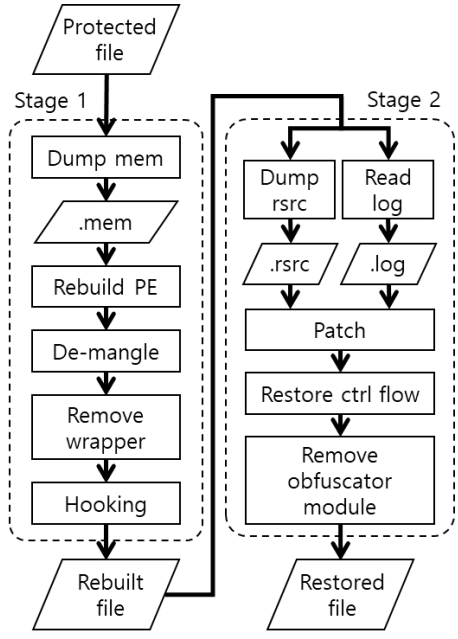

Fig. 11. Architecture of unpacking system

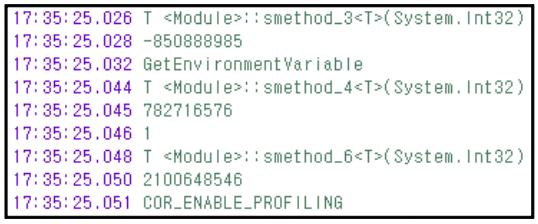

Fig. 12. Logs left while running hooked code

모든 과정이 마무리되면, 생성한 실행 파일을 이용해 다음 단계를 진행한다.

두 번째 단계에서는 실행 중에 'Assembly.Load' 함수를 관찰해 모듈이 로드되면, 모듈 내의 리소스 정보를 실행 파일의 리소스 영역에 저장한다. 그리고 후킹 코드에서 나온 정보에 따라 특정 함수의 반환 값이 문자열 형태라면, 해당 함수를 호출한 지점에서 바로 문자열을 사용하도록 수정한다. 다음으로 분기 조건이 고정값을 가지고 있는 경우, 분기문 내 코드의 실행 순서에 따라 뒤섞여 있는 코드의 순서를 정렬한다. 앞의 모든 과정이 정상적으로 진행되었다면, 실행 파일과 난독화 모듈이 연결된 지점이 더 이상 존재하지 않는다. 따라서 모듈 initializers 내부의 모든 클래스와 함수를 삭제하면, 모든 분석방해 기능이 해제된 실행 파일을 획득하게 된다.

4.3 분석방해 기능 해제 결과

최신 버전인 ConfuserEx 1.6 버전이 적용된 실행 파일을 본 연구에서 제안하는 시스템에 입력하여 분석방해 기능을 해제하였으며, 복구한 실행 파일 모두 정상적으로 실행할 수 있었다. 실험에 사용한 최신 버전의 ConfuserEx가 적용된 실행 파일은 총 3개이며, 직접 개발한 테스트용 프로그램 1개(Test)와 네이버에서 습득한 매크로 프로그램 2개(Acq 1, Acq 2)를 사용했다. 실험에 사용한 실행 파일 모두 2장의 관련 연구에 기재된 도구로는 정상적으로 복구되지 않는 것을 확인하였다.

Table 5.는 각 실행 파일에서 복구한 리소스의 수(Rsrc)와 수정한 코드의 수(Patch), 시스템의 단계별로 소요된 시간(Stg 1, Stg 2), 그리고 수작업으로 실행 파일을 복구하는데 소요된 시간(Man)을 보여준다. 제안하는 시스템의 효율을 보이기 위해 테스트용 프로그램의 분석방해 기능을 수작업으로 해제하였으며, 습득한 프로그램에 비해 상대적으로 매우 적은 양의 작업이 필요함에도 41분이 소요되었다. 따라서

Table 5. Result from de-obfuscation system

File	Rsrc	Patch	Stg 1	Stg 2	Man
Test	1	21	6s	1s	41m
Acq 1	291	23,191	12s	57s	-
Acq 2	125	265	9s	25s	-

제안하는 시스템을 활용하지 않고 수작업으로 대응해야 하는 상황이라면, 습득한 프로그램을 복구하는 작업은 상당한 시간이 소요될 것을 예상할 수 있다.

Fig. 13.은 후킹 코드가 기록한 로그를 이용해 난독화 된 코드를 복구한 결과와 원본 코드를 보여준다. 난독화 된 코드에서는 어떤 값을 할당하는지 식별할 수 없었으나, 복구된 코드에서는 'Hello confuser!' 문자열을 쉽게 확인할 수 있다. 그리고 변수의 이름을 복구하는 과정으로 인해 'label1'이 'label_0'으로 다르게 나타나지만, 이는 실행 흐름에는 영향을 미치지 않는다.

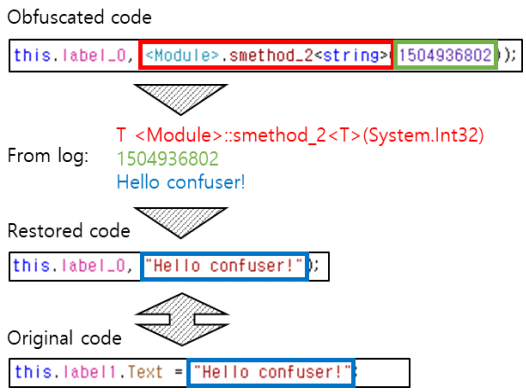

Fig. 13. Restored code in comparison to original one

V. 결 론

본 연구에서는 포털 사이트의 컨텐츠를 생산하거나 변경하려고 시도하는 매크로 프로그램이 사용하는 난독화 도구 중 하나인 ConfuserEx의 분석방해 기능에 대해 알아보고, 각 분석방해 기능을 해제하는 방안을 기술하였다. 또한 ConfuserEx가 적용된 실행 파일의 난독화를 자동으로 해제하는 시스템을 제안하고, 실험을 통해 실행 파일의 난독화를 해제하는 것이 가능함을 보였다. 제안한 시스템을 활용한다면, 난독화를 해제하기 위해 수작업으로 ConfuserEx가

적용된 실행 파일을 분석하는 시간을 단축할 수 있을 것으로 보인다. 다만 본 연구는 확보한 프로그램이 사용하는 다양한 난독화 도구 중 ConfuserEx의 최신 버전에 한해 적용이 가능하므로, 앞으로는 확보한 다른 유형의 난독화 도구를 해제하는 연구를 진행할 예정이다.

References

[1] ConfuserEx, "An open-source, free protector for .NET applications", https://github.com/mkaring/ConfuserEx, 12. 12. 2022

[2] Microsoft, "What's New in Windows 10, build 19041, Jun. 2022",https://docs.microsoft.com/en-us/windows/uwp/dotnet-native/net-native-and-compilation), 12. 12. 2022

[3] de4dot, ".NET deobfuscator and unpacker", https://github.com/de4dot/de4dot, 12. 12. 2022

[4] NoFuserEx, "Free deobfuscator for ConfuserEx", https://github.com/CodeShark-Dev/NoFuserEx, 12. 12. 2022

[5] Netguard-Unpacker-Public, "Public NetGuard Deobfuscator", https://github.com/Tanasittx/NetGuard-Unpacker-Public), 12. 12. 2022

[6] ClarifierEx, "Deobfuscator for ConfuserEx", https://github.com/chaplin89/ClarifierEx), 12. 12. 2022

[7] Rzy-Protector-V2-unpacker, "An unpacker (deobfuscator) for the protector (obfuscator) Rzy Protector V2", https://github.com/illuZion9999/Rzy-Protector-V2-unpacker), 12. 12. 2022

[8] ConfuserEx-Dynamic-Unpacker, "A dynamic confuserex unpacker that relies on invoke for most things", https://github.com/uvbs/ConfuserEx-Unpacker), 12. 12. 2022

[9] ConfuserEx-Unpacker-Mod-By-Bed, "Edited copy of cawks confuserex unpacker, support more than your averagep rogram", https://github.com/BedTheGod/ConfuserEx-Unpacker-Mod-by-Bed), 12. 12. 2022

[10] ConfuserEx-Unpacker-2, "ConfuserEx-Unpacker-2", https://github.com/hackovh/ConfuserEx-Unpacker-2, 12. 12. 2022

[11] Krawk-Unpacker, "Krawk-Unpacker", https://github.com/exploitdev/Krawk-Unpacker, 12. 12. 2022

[12] ConfuserExSwitchKiller, "ConfuserExSwitchKiller", https://github.com/VAllens/ConfuserExSwitchKiller, 12. 12. 2022

[13] ConfuserEx-Anti-Debug-Remover, "ConfuserEx-Anti-Debug-Remover", https://github.com/ALEHACKsp/ConfuserEx-Anti-Debug-Remover, 12. 12. 2022

[14] ConfuserEx-Static-String-Decryptor, "ConfuserEx-Static-String-Decryptor", https://github.com/Techlord-RCE/ConfuserEx-Static-String-Decryptor, 12. 12. 2022

[15] ConfuserEx-Resources-Decryptor, "This tool can decrypt encrypted resources from ConfuserEx and replace them", https://github.com/MindSystemm/ConfuserEx-Resources-Decryptor), 12. 12. 2022

[16] ConfuserExResourceReplace, "ConfuserExResourceReplace", https://github.com/guzelyuksel/ConfuserExResourceReplace, 12. 12. 2022

[17] ConfuserExTools, "ConfuserEx unpacking tools", https://github.com/wwh1004/ConfuserExTools), 12. 12. 2022

[18] Unscrambler, "Universal unpacker and fixer for a number of modded ConfuseREx protections", https://github.com/dr4k0nia/Unscrambler), 12. 12. 2022

[19] EasyPredicateKiller, "Replacing and Calling ConfuserEx x86 Predicates", https://github.com/ZeroPlusBlog/EasyPredicateKiller), 12. 12. 2022

〔20〕 Microsoft, "What's new in C# 9.0, Support for code generators, Jun. 2022", https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9#support-for-code-generators), 12. 12. 2022
〔21〕 Microsoft, "PE Format, File Headers, Jun. 2022", https://docs.microsoft.com/en-us/windows/win32/debug/pe-format), 12. 12. 2022
〔22〕 Microsoft, "SuppressIldasmAttribute Class", https://docs.microsoft.com/en-us/dotnet/api/system.runtime.compilerservices.suppressildasmattribute?redirectedfrom=MSDN&view=net-6.0), 12. 12. 2022
〔23〕 dnSpy, ".NET debugger and assembly editor", https://github.com/dnSpy/dnSpy), 12. 12. 2022
〔24〕 dnlib, "Reads and writes .NET assemblies and modules", https://github.com/0xd4d/dnlib), 12. 12. 2022

〈저자소개〉

이 재 휘 (Jae-hwi Lee) 정회원
2015년 2월: 경북대학교 컴퓨터학부 공학사 졸업
2017년 8월: 고려대학교 정보보호대학원 석사 졸업
2018년 3월~현재: NAVER Corp. 보안팀 연구원
〈관심분야〉 정보보호, 역공학, 어뷰즈 탐지

박 영 석 (Young-seok Park) 정회원
2014년 8월: 성균관대학교 전자전기공학부 학사 졸업
2016년 8월: 한국과학기술원 정보보호대학원 석사 졸업
2016년 7월~현재: NAVER Corp. 보안팀 연구원
〈관심분야〉 정보보호, 어뷰즈 탐지

김 동 현 (Dong-hyeon Kim) 정회원
2018년 2월: 동아대학교 컴퓨터공학과 졸업
2018년 5월~현재: NAVER Corp. 보안팀 연구원
〈관심분야〉 정보보호, 어뷰즈 탐지

허 규 (Gyu Heo) 정회원
2008년 2월: 순천향대학교 정보보호학과 졸업
2007년 12월~현재: NAVER Corp. 보안팀 연구원
〈관심분야〉 정보보호, IoT 보안, 서비스 보안