2006 한국컴퓨터종합학술대회 논문집 Vol. 33, No. 1(C)

C++ 언어를 위한 데이터 obfuscation 도구 구현 및 테스트

조병민*O, 노진욱*, 오현수*, 장혜영*, 정민규**, 이승원**, 박용수+, 우제학++, 조성제*
*단국대학교, **서울대학교, +한양대학교, ++㈜코어트러스트
onlyhr98@gmail.comO, {neocastle, pyxis3, chenil}@dankook.ac.kr, april18.mg@gmail.com,
{leesw, yspark}@ssrnet.snu.ac.kr, jhwoo@coretrust.com, sjcho@dankook.ac.kr

An Implementation of Data Obfuscation Tool for C++ Language

ByoungMin Cho*O, JinUk Noh*, Hyunsoo Oh*, Hye-Young Chang*, Mingyu Jung**,
Seungwon Lee**, Yongsoo Park+, Jehak Woo++, Seongje Cho*
*Dankook University, **Seoul National University, +Hanyang University, ++Coretrust, Inc.

요 약

상용 응용 소프트웨어들은 주요 정보 및 알고리즘들을 내포하고 있어, 악의적인 역공학자들에 의해 특정 소프트웨어를 decompile하여 자료구조 및 제어 흐름을 분석하려는 공격이 이루어 지고 있다. 본 논문에서는 MS 윈도우 XP 상의 Visual C++ (MFC 포함) 프로그램을 역공학 공격으로부터 보호하기 위한 데이터 obfuscator를 구현하고 그 성능을 평가한다. 구현한 obfuscator는 C++ 소스코드를 입력 받아 3가지의 데이터 obfuscation 알고리즘을 적용한 후, 이를 다시 소스코드로 재생성하는 도구로 큰 성능저하 없이 구현 가능함을 알 수 있다.

1. 서론

한국소프트웨어저작권협회의 SW 저작권 침해 현황 자료에 의하면 2005년도에 침해건수가 1,311건에 피해규모가 약 300억 원에 달하여, 2006년도 현재까지는 침해 건수가 191건, 침해금액 약 48억 원에 달할 정도로 심각하다[1]. 특히, 국내 패키지 게임은 해킹과 불법유포로 인해 더 이상 개발이 이루어지지 않고 있다. 정부도 이의 심각성을 깨닫고, SW 산업을 육성시키고자 SW 지적재산권 보호기반을 강화시키려는 노력을 기울이고 있다.

본 논문에서는 소프트웨어의 핵심 알고리즘, 주요 모듈, 자료구조에 저장된 값 등을 역공학(reverse engineering) 분석 공격 및 탬퍼링(tampering 또는 변조) 공격들로부터 보호할 수 있는 기법에 대해 연구한다. 즉, MS 윈도우 XP 상에서 C++ 프로그램 소스의 여러 자료구조들을 제 3자에 의한 소스 코드 분석 및 역공학 분석 공격으로부터 보호하기 위한 데이터 obfuscation 도구를 구현하고 그 성능을 평가한다. Obfuscation의 사전적 의미는 ' ~을 혼란시키다', ' ~을 알기 어렵게 만들다' 이고 유사한 맥락으로 컴퓨터 과학에서의 Obfuscation의 의미는 기존코드의 기능은 그대로 유지하면서 역공학을 통한 분석이 어렵도록 코드를 재구성하는 작업을 말한다. 완전한 역공학 분석 공격을 방어한다는 의미 보다는 암호화 기법처럼 분석에 필요한 비용과 시간을 증가시켜 난해하게 만드는데 그 의미가 있다.

본 논문에서는 주로 데이터 obfuscation 도구의 구현에 초점을 맞춘다. 2장에서는 현재까지 연구된 obfuscation기법들에 대한 간략한 설명과 본 논문에서 구현한 도구의 차이점을 소개하고 3장에서는 구체적인 obfuscator 구현과 구현 알고리즘에 대한 내용을 살펴본다. 4장에서는 실험 및 성능평가에 대한 결과를 다룬다. 마지막 5장에서는 결론 및 향후 과제에 대해서 논의한다.

2. 관련 연구 및 본 연구의 특징

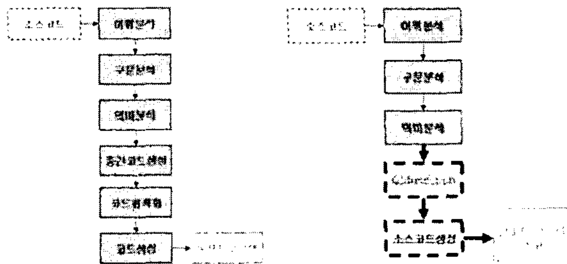
Obfuscation에 대한 연구는 주로 컴파일 과정에 생성되는 중간언어인 어셈블리어[2]나 Java의 바이트 코드[3]를 대상으로 이루어 졌으며 일부 상용화가 이루어지기도 했다. 이는, 상용소프트웨어가 대부분 소스가 없는 바이너리 형태로 제공되므로 역공학 분석 공격을 방어하기 위해 어셈블리어 코드를 대상으로 obfuscation을 적용하는 것이 효과적이기 때문이며, Java의 바이트 코드의 경우에도 플랫폼 독립성으로 인해 널리 유통되기 때문에 바이트 코드의 해킹 및 분석 가능성을 감소시키는 것이 타당하기 때문이다. 이와 유사하게 현재 MS는 .NET 기반의 Microsoft Intermediary Language (MSIL)에 대한 Obfuscation 적용 방안을 연구하고 있다.

그러나 C++ 언어에 대한 Obfuscation 도구의 구현은 C++ 문법의 복잡성으로 인해 연구가 거의 이루어 지지 않았으며, 특히 본 논문에서와 같이 MS Visual C++ 6.0으로 생성된 C/C++ (MFC 포함) 소스 코드를 대상으로 연구 및 구현한 예를 현재까지 찾을 수가 없었다. 본 논문은 C++ 소스코드에 obfuscation을 적용하는 도구를 개발하여, 역공학 공격이나 decompile을 통한 소스 코드의 외부 유출을 방지하는 기법을 제안한다.

3. Obfuscation 도구 개요

3.1 소스코드 대 소스코드 변환

컴파일러는 특정 언어의 프로그램을 입력으로 받아들여 같은 기능을 가진 다른 언어의 프로그램으로 변환하여 주며, 사용자에게 입력 프로그램 내의 오류를 알려주기도 한다[4]. 본 논문에서는 소스코드에 대해 Obfuscation을 적용한 후, 원래 소스와 동등한 기능을 하는 또 다른 소스코드를 생성하는 것이 목적이므로 컴파일 과정에서 의미분석 단계까지를 사용한다. 그 다음, 의미 분석된 프로그램을 다시 소스코드로 재생성해주는 과정을 추가한다. 컴파일 과정 및 본 연구에서 제안한 방법을 도식화한 것이 그림 1에 나타나 있다.



(a) 일반적인 컴파일 과정　　(b) Obfuscation을 적용한 소스 코드 변환 과정

그림 1. 컴파일 과정 및 Obfuscation 과정

그림 1에서 (b)는 (a)의 ' 중간코드생성' 과정과 그 이후 과정들 대신에 Obfuscation을 적용하는 부분과 소스코드를 재생성 하는 과정을 포함하고 있다. 다음은 source to source 변환을 실제 구현하기 위해 사용한 도구에 대해 설명한다.

3.2 ANTLR (ANother Tool for Language Recognition)

ANTLR은 ANTLR Meta Language라는 특화된 언어로 작성된 lex나 yacc와 유사한 문법 명세서를 통해 언어인식기(프로그램)를 생성해주는 도구이다[5]. 언어인식기는 C, C++, Java, C#등의 다양한 언어로 생성될 수 있으며 문법 명세서에 기술된 문법으로 작성된 문서들을 인식한다. 생성된 언어 인식기는 크게 Lexer와 Parser 등 두 부분으로 나눌 수 있으며 각각은 컴파일러의 어휘 분석과 구문분석, 의미분석을 담당하게 된다. 그림 2는 " User.g" 라는 사용자 정의 문법을 통해 C++문법의 언어 인식기를 생성하는 과정을 보여준다.

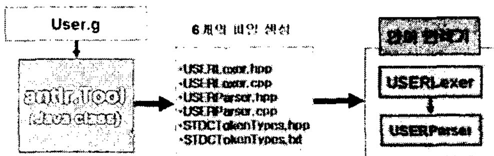


그림 2. 언어인식기 생성

본 논문의 an original source to the obfuscated source 변환 도구는 언어인식 과정 후에 코드를 재생성하는 모듈을 추가함으로써 구현된다. 파서(parser)를 통해 최종적으로 의미 분석 단계를 거치게 되면 AST(Abstract Syntax Tree)가 생성되며 소스 코드 수준에서 Obfuscation 알고리즘의 적용은 AST의 조작을 통해 이루어 진다.

3.3 AST (Abstract Syntax Tree)

AST는 입력 스트림(stream)의 내부적인 표현을 위해 ANTLR에서 정의한 트리구조이며 컴파일러 이론에서 구문트리와 유사하다. 트리를 구성하는 노드는 어휘분석을 통해 나뉘는 토큰들의 이름, 타입, 행 번호 등의 정보를 저장하게 된다. 이후 Obfuscation 알고리즘들은 AST 노드들의 정보를 이용하여 해당 정보의 검색 및 구분 작업을 수행하고 노드 삽입, 수정 및 재구성을 통해 개발된다. 추가적으로 심볼 정보를 파일로 저장하여 사용하게 되는데, 전체적인 흐름도는 그림 3과 같다.

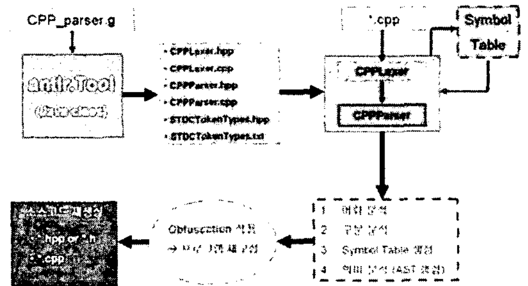


그림 3. Obfuscation 도구 전체 흐름도

4. Data Obfuscation 알고리즘의 구현

데이터 Obfuscation은 프로그램이 사용하는 자료구조를 대상으로 한다. 관련 기법으로는 메모리상의 저장 방법을 바꾸거나 지역 변수를 글로벌 변수로 변경하는 storage obfuscation, 데이터의 해석 방법을 변경하는 encoding obfuscation, 데이터의 그룹을 변경하는 aggregation obfuscation, 데이터의 순서를 변경하는 ordering obfuscation 등이 제안되어 있다[6]. 본 논문에서는 데이터 obfuscation을 수행하기 위하여 다음과 같이 세 가지 알고리즘을 구현하였다.

4.1 Variable split

Variable split 알고리즘은 (1) 기본 타입의 변수들을 여러 개의 변수로 나누거나 (2) 변수를 사용자 정의 구조체 또는 값을 반환하는 함수로 대체함으로써 해석을 어렵게 만드는 기법이다. 본 연구에서는 특정 변수들은 함수 또는 간단한 수식을 통해 분할하여 분할되기 전의 변수와 동일한 값을 반환하도록 코드를 수정하였다. 그림4는 정수형 변수를 분할하는 예이다.

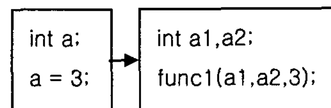


그림 4. Variable split

4.2 Fold array

Fold array 알고리즘은 그림 5와 같이 1차원 배열을 다차원의 배열로 변환함으로써 배열의 값에 대한 분석을 어렵게 한다.

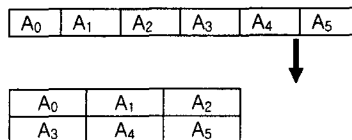


그림 5. Fold array

4.3 Insert Class

Insert Class 알고리즘은 그림6과 같이 C++등의 객체지향 프로그램에서 각각의 객체를 구성하는 클래스들에 의미 없는 상위 클래스를 추가하거나 하나의 클래스를 둘로 나누어 상속의 개념을 추가함으로써 복잡도를 증가시킨다.

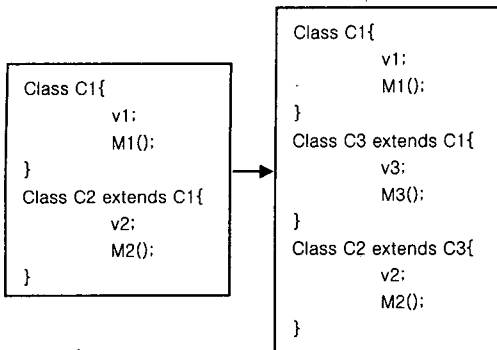


그림 6. Insert Class

4.4 개발환경 및 사용 도구

개발 시스템은 Intel 펜티엄IV 1.7GHz, SDRAM 576MB, Windows XP 운영체제 상에서 구현되었으며 사용 도구로는 MS Visual Studio C++ 6.0 과 ' Source Insight' 텍스트 문서 편집기를 사용하였다.

5. 실험 및 성능평가

C++와 C는 문법에서 많은 차이가 나지 않고 실험결과를 명확히 하기 위해 간단한 C프로그램을 이용한 테스트를 수행하였다. 그림 7은 정수형 변수를 variable split 알고리즘을 이용하여 2개의 short 형 변수로 분할하고 값을 할당한 결과이며, 그림 8은 Fold Array 알고리즘을 적용하여 1차원 배열을 2차원배열로 변환한 결과이다.

```
#include "stdafx.h"

using std::cout;

int main(int argc, char* argv[])
{
    int a,b=1;
    a=0;
    a=b+2;
```

```
#include "stdafx.h"
int _4598(short x, short y)
{
int p = 1;
for(int i=0 ; i < x ; i++) p *= 3;
p += y;
return p;
}

using std :: cout ;
int main ( int argc , char * argv [ ] )
{
    int _1160 , b = 1 ;
short _1670, _26078;

int _6183=0;
int _7162,_5418,_6087;
_26078=0;
for(_1670 = 0 ;_1670<_6183;_1670++)
{
_7162 = 1;
for(_5418=0 ; _5418 < _1670 ; _5418++) _7162 *= 3;
_6087 = _7162/3;if( _7162 >= _6183) break;
}
```

그림 7. Variable Split 수행 결과

```
#include "stdafx.h"

using std::cout;

int main(int argc, char* argv[])
{
    int a[6];
    int i = 0;

    for(i=0; i<6 ; i++) a[i] = i;

    for(i=0; i<6 ; i++) cout << a[i] << " ";

    return 0;
}
```

```
#include "stdafx.h"
using std :: cout ;
int amain ( int argc , char * argv [ ] )
{
    int a [ 3 ][ 2 ] ;
    int i = 0 ;
    for ( i = 0 ; i < 6 ; i ++ ) a [ i%3 ][ i/3 ] = i ;
    for ( i = 0 ; i < 6 ; i ++ ) cout << a [ i%3 ][ i/3 ] << " "
    return 0 ;
}
```

그림 8. Fold Array 수행 결과

Obfuscation 도구에 대한 성능평가는 MFC 프로그램의 소스를 대상으로 이루어 졌다. 그림 9는 하나의 소스코드를 대상으로 전처리 파일로부터 심볼정보를 가져와서 알고리즘을 적용시키는 일련의 과정에 대한 시간과 메모리 사용량을 측정한 것이다. 그림 9의 왼쪽은 심볼정보를 전처리 파일로부터 획득하는 과정이 포함된 것이고 오른쪽은 심볼정보가 있을 경우 Obfuscation 알고리즘만 적용시킨 결과이다. 심볼정보를 얻기 위해 전처리 파일을 사용할 경우 많은 시간과 메모리가 소요되지만 심볼정보가 있을 경우 수행시간과 메모리 사용량은 현저히 감소한다.

CPU	(왼쪽)	(오른쪽)
Priority	8	8
Kernel Time	0:01:27.828	0:00:00.703
User Time	0:01:53.578	0:00:01.343
Total Time	0:03:21.406	0:00:02.046
Virtual Memory		
Private Bytes	0 K	0 K
Peak Private Bytes	182,908 K	4,264 K
Virtual Size	196,744 K	18,808 K
Page Faults	46,277	1,568
Page Fault Delta	0	0

그림 9. Obfuscation 적용 후의 성능측정

6. 결론 및 향후 과제

본 논문에서 구현한 Obfuscation 도구는 전처리 파일을 통해 심볼정보를 획득하기 때문에 수행속도가 느리다는 문제점을 안고 있으며 C++문법의 난해함으로 인해 다양한 알고리즘의 적용이 어려웠다. 따라서 심볼 정보획득 과정을 최적화시키고 새로운 데이터 obfuscation 알고리즘을 추가하는 것이 우선과제이며, 나아가 potency, resilience 등의 관점에서 좀 더 체계적으로 구현된 알고리즘의 성능을 평가하고자 한다.

참고문헌

[1] http://www.spc.or.kr
[2] Chenxi Weng, "A Security Architecture for Survivability Mechanisms", October 2000, University of Virginia
[3] Levent Ertaul, Suma Venkatesh, "JHide-A 도구 Kit for Code Obfuscation", November 2004, LNCS
[4] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "Compiler Design, Techniques and 도구s", January 1986, Addison-Wesley
[5] http://www.antlr.org
[6] http://palisade.plynt.com/issues/2005Aug/code-Obfuscation/