지능형 악성코드 분석을 위한 안티리버싱 코드 우회 프레임워크 설계 및 구현*

이선준, 김규호, 신용구, 이정현
숭실대학교 소프트웨어학부
e-mail: {starj1024, krbgh205760, tls09611}@gmail.com, jhyi@ssu.ac.kr

Design and Implementation of Anti-reversing Code Evasion Framework for Intelligent Malware Analysis

SunJun Lee, KyuHoKim, YongGu Shin, and Jeong Hyun Yi
School of Software, Soongsil University

요 약

최근 악성코드의 수가 급격하게 증가하고 있으며 단순히 악성 행위를 하는 것 뿐 아니라 안티디버깅과 같은 다양한 분석 방지 기능을 탑재하여 악성코드의 분석을 어렵게 한다. 역공학 방지 기법이 적용된 지능형 악성코드를 기존 분석 도구를 사용하여 분석하면 악성행위를 하지 않거나 임의로 자기 자신을 종료시키는 방식으로 분석이 용이하지 않다. 이러한 지능형 악성코드들은 분석하기 어려울 뿐만 아니라 기존 백신의 탐지 기능에 전혀 제약을 받지 않는다. 본 논문은 이와 같은 최신 지능형 악성코드에 보다 빠르게 대처하기 위해 역공학 방지 기법이 적용된 악성코드들이 메모리상에서 종료되지 않고 정상 동작하여 악성행위를 자동으로 파악할 수 있는 동적 코드 계측 프레임워크를 제안한다. 또한, 제안한 프레임워크를 개념 검증하기 위해 프로토타입을 설계 및 구현하고, 실험을 통해 그 유효성을 확인한다.

1. 서론

악성 코드의 수는 급격하게 증가하고 있으며 악성코드의 기술력 역시 나날이 발전하고 있다[1]. 많은 정보들이 전산화됨에 따라 악성코드의 발전도 급격하게 이루어 졌으며 악성코드에 의한 피해 사례 또한 나날이 커져가는 것이 현실이다[2]. 때문에 악성코드에 대처하기 위한 연구 또한 활발하게 진행되고 있다.

일반적으로 악성코드에 대처하는 방법은 악성코드를 분석하고, 분석된 데이터를 기반으로 솔루션을 제시하는 것이다. 따라서 악성코드에 대처하기 위해서는 악성코드의 분석이 선행되어야 한다. 하지만 최근의 악성코드 개발자들은 악성코드가 분석당하는 것을 막기 위해 역공학 방지 기법이라는 기법을 악성코드에 적용하여 만든다.

역공학 방지 기법이란 프로그램이 분석환경 위에서 실행되는 경우 본래의 동작과 다른 동작을 하거나 종료되도록 만드는 기법을 의미한다. 이러한 역공학 방지 기법이 적용된 악성코드는 분석환경이 아닐 경우에만 악성행위를 하며 분석환경 위에서는 다른 정상적인 동작을 하거나 종료된다. 따라서 역공학 방지 기법이 적용되어 있는 악성코드를 분석하기 위해서는 경험이 많은 분석가들이 직접 수작업을 통해 해당 기법을 우회하는 것이 일반적이다. 하지만 앞서 언급했듯이 최근 악성코드의 수는 기하급수적으로 증가하고 있기 때문에, 기존의 방법으로는 쏟아져 나오는 악성코드들에 대응하기가 현실적으로 불가능에 가깝다.

따라서 본 논문은 이러한 역공학 방지 기법들이 적용된 악성코드를 보다 빠르게 분석할 수 있는 역공학 방지 코드 우회 시스템을 제안한다.

* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00168, 사이버 위협 대응을 위한 Deep Malware 자동 분석 기술 개발)

2. 배경지식

2.1 역공학 방지 기법

역공학 방지 기법은 코드 도용, 불법복제, 기능 변조 위협 등의 위협으로부터 프로그램을 보호하기 위한 기법들을 의미한다. 대표적인 역공학 방지 기법들에는 난독화, 패킹, 안티 디버깅, 안티 에뮬레이팅 등이 있다.

이중 안티 디버깅을 탐지하는 방법[3]은 TracerPID 검사, 디버거 API 검사, 브레이크 포인트 명령어 검사 등이 있다. TracerPID 검사방법은 리눅스 기반 시스템에서 사용할 수 있는 방법이다. 리눅스 기반의 시스템에서 프로세스 디버깅을 진행할 때 ptrace 시스템 콜을 이용하는데 이

를 이용하면 프로세스 상태 정보 중 디버깅 관련 정보가 설정되어 이를 확인하는 것으로 탐지 할 수 있다. 마찬가지로 디버거 API 검사와 브레이크 포인트 명령어 검사 역시 디버거의 특성 또는 디버거를 사용하기 위한 동작들을 이용하여 디버거의 유무를 검사한다. 일반적인 안티 디버깅 기법은 위와 같은 방법으로 디버거를 검사하며 디버거가 발견된다면 프로그램을 종료시킨다. 다른 역공학 방지 기법들 역시 각각의 특성을 역이용한 방식으로 역공학 기법을 방어한다.

2.2 DBI (Dynamic Binary Instrumentation)

DBI[4] 는 Dynamic Binary Instrumentation의 약자로 Runtime Code Instrumentation이라고도 불리며 분석 대상 프로그램에 *Instrumentation 코드를 삽입하여 프로그램의 행동을 관찰하는 동적 분석 방법이다. DBI 기능을 지원하는 도구에는 인텔에서 제공하는 Pin, MIT와 HP에서 만든 Dynamorio[5] 등이 있다.

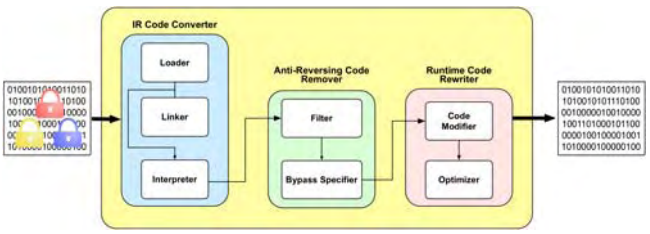
DBI는 프로그램을 분석하는 용도 이외에도 다양하게 활용이 될 수 있는데 대표적으로 인텔에서 제공하는 Pin을 이용하면 단순히 프로그램 분석 뿐 아니라 동적으로 Instrumentation을 수행할 수 있는 자신만의 분석 도구를 만드는 것 또한 가능하다[6].

하지만 이러한 DBI 도구들이 널리 사용됨에 따라 DBI 기능을 막기 위한 방법들이 제시되고 있다. 현재 Pin을 막는 방법이나[4], Dynamorio를 막는 방법[7] 은 활발히 연구되었고 발견된 방법을 피하여 DBI 기능을 수행하도록 하는 연구 역시 진행되고 있다.

3. 시스템 설계 및 구현

제안 시스템은 Dynamorio와 같은 DBI Framework방식으로 설계되었다. 제안 시스템은 선택적으로 코드를 로딩하고, 로딩 된 코드를 실행하면서 분석하여 역공학 방지 기법에 해당되는 코드를 찾아내고 해당 코드를 우회한다.

3.1 시스템 구조



(그림 1) 시스템 구성도

그림 1은 본 논문에서 제안하는 역공학 방지 코드 우회 시스템의 구성도이다. 그림 1에 나타난 바와 같이 전체 시스템은 IR Code Converter, Anti-Reversing Code Remover, Runtime Code Rewriter로 구성된다. 제안 시스템은 역공학 방지 기법이 적용된 악성코드를 입력으로 받아 역공학 방지 기법을 우회하도록 코드를 수정하여, 역공학 방지 기법이 적용되기 이전의 실행흐름을 갖는 코드를 출력한다.

3.2 IR Code Converter

IR Code Converter는 Loader, Linker, Interpreter로 구성되어 있다. Loader와 Linker는 프로그램의 실행 코드를 선별적으로 Code Cache에 할당하는 역할을 한다. Code Cache란 코드의 변조가 가능한 메모리공간을 의미한다. 제안 시스템은 입력 바이너리 파일의 코드 중 시스템 라이브러리에 해당하는 코드는 악성 행위와 상관없이 시스템에서 동작하기 위해 사용하는 코드이기 때문에 분석대상이 아니라고 판단하여 Code Cache에 할당하지 않는다. 따라서 제안 시스템은 시스템 코드를 제외한 사용자 코드만 Code Cache에 할당한다. Code Cache에 할당 되지 않은 코드는 일반적인 프로그램의 실행 과정을 거쳐 실행되며 Code Cache에 할당된 코드는 Interpreter를 거쳐 제안 시스템의 흐름을 따라 실행된다.

```
typedef
   struct _IRStmt {

      struct {
         Addr     addr;   /* instruction address */
         UInt     len;    /* instruction length */
      } IMark;

      struct {
         Bool     isReg    /* check the operand is register or not */
         Int      addr     /* address of the operand */
         Int      data     /* value of the operand */
      } DMark1;

      struct {
         Bool     isReg    /* check the operand is register or not */
         Int      addr     /* address of the operand */
         Int      data     /* value of the operand */
      } DMark2;

} IRStmt;
```

(그림 2) IR 구조

Interpreter에서는 전달받은 코드를 IR 형태로 변환한다. 변환된 IR 코드는 그림 2와 같은 구조를 갖는다. IR구조는 1개의 연산자 IMark와 2개의 피연산자 DMark1, DMark2로 구성되어있다. IMark, DMark1, DMark2를 조합하면 어셈블리 형태의 명령어를 획득할 수 있다. 그림 2와 같은 형태로 변형된 IR은 Basic Bock 단위로 나누어져 관리되는데, Basic Block이란 분기를 기점으로 나눈 IR 코드들의 집합을 의미한다. Basick Block은 그림 3과 같은 구조를 갖는다.

* Instrumentation : 임의의 정보를 수집하기 위해 실행 파일에 삽입하는 코드를 의미한다. 프로그램을 관찰하여 메모리 덤프, 변수 또는 레지스트리 값 출력 등을 의미한다.

```
typedef
    struct {
        IRStmt**        stmts;
        Int             stmts_size;
        Int             stmts_used;
        IRSB*           next;
    } IRSB;
```

(그림 3) Basic Block의 구조체

stmts는 해당 Basic Block에서 가장 먼저 실행되는 IR 코드이다. stmt_size는 Basic Block이 몇 개의 IR 코드를 포함하고 있는지를 의미하며, stmts_used는 해당 Basic Block이 이전에 실행된 적이 있는지를 알려준다. next는 해당 Basic Block 다음에 실행될 Basic block을 가리키고 있다.

Interpreter 모듈은 Basic Blcok 단위나 IR 명령어 1개 단위로 코드를 실행하며, 실행하기 전에 현재 실행하고자 하는 명령어의 IR을 두 번째 모듈인 Anti Reversing Code Remover로 전달한다.

3.3 Anti Reversing Code Remover

Anti Reversing Code Remover는 Filter와 Bypass Specifier로 구성되어있다. Filter는 역공학 방지 기법에서 사용하는 시그니처가 사용되는 Basic Block을 역공학 방지 기법이 적용된 로직이라고 추정한다. 여기서 시그니처란 해당 Basic Block에서 사용하는 문자열, 바이너리 스트링, API, 시스템 속성 값 등의 데이터를 의미한다. Filter는 기존 연구를 통해 분석된 역공학 방지 기법의 시그니처와 패턴 매치를 통해 역공학 방지 기법을 인식한다 [8][9].

Bypass Specifier는 역공학 방지 기법이 포함되어 있다고 추정되는 코드가 실행되지 않도록 코드를 수정한다. 코드의 수정은 역공학 방지 기법이 적용되었을 것으로 추정되는 Basic Block을 실행하지 않도록 한다. 즉, 역공학 방지 기법이 적용되었을 것으로 추정되는 Basic Block으로 가는 분기를 그 다음 Basic Block으로 가도록 바꿈으로써 적용된다.

3.4 Runtime Code Rewriter

Runtime Code Rewriter는 Code Modifier와 Optimizer로 구성되어있다. Code Modifier는 Anti Reversing Code Remover로부터 역공학 방지 기법이 실행되지 않도록 우회된 IR을 수신하고 해당 IR에서 실행되지 않게 된 역공학 방지 기법에 해당하는 데이터와 코드가 포함된 IR을 제거한다. Code Modifier를 지나 역공학 방지 기법이 완전히 제거된 IR을 Optimizer가 수신하고 IR을 다시 바이너리 파일로 변환해 생성한다.

4. 결과

```
004018a8] 55                    push   rbp
004018a9] 48 89 e5              mov    rbp,rsp
004018ac] 48 83 ec 10           sub    rsp,0x10
004018b0] c7 45 f8 03 00 00 00 mov    DWORD PTR [rbp-0x8],0x3
004018b7] c7 45 fc 05 00 00 00 mov    DWORD PTR [rbp-0x4],0x5
004018be] 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
004018c1] 01 45 f8              add    DWORD PTR [rbp-0x8],eax
004018c4] bf b6 20 40 00        mov    edi,0x4020b6
004018c9] e8 e2 f3 ff ff        call   0x0000000000400cb0

00400cb0] ff 25 72 23 20 00     jmp    QWORD PTR [rip+0x202372]

00400cb6] 68 02 00 00 00        push   0x2
00400cbb] e9 c0 ff ff ff        jmp    0x0000000000400c80

00400c80] ff 35 82 23 20 00     push   QWORD PTR [rip+0x202382]

00400c86] ff 25 84 23 20 00     jmp    QWORD PTR [rip+0x202384]

004018ce] bf c3 20 40 00        mov    edi,0x4020c3
004018d3] e8 d8 f3 ff ff        call   0x0000000000400cb0

00400cb0] ff 25 72 23 20 00     jmp    QWORD PTR [rip+0x202372]

004018d8] b8 00 00 00 00        mov    eax,0x0
004018dd] e8 8b ff ff ff        call   0x000000000040186d

0040186d] 55                    push   rbp
0040186e] 48 89 e5              mov    rbp,rsp
00401871] 48 83 ec 10           sub    rsp,0x10
00401875] c7 45 fc 08 00 00 00 mov    DWORD PTR [rbp-0x4],0x8
0040187c] c7 45 f8 00 00 00 00 mov    DWORD PTR [rbp-0x8],0x0
00401883] eb 1b                 jmp    0x00000000004018a0

004018a0] 83 7d f8 09           cmp    DWORD PTR [rbp-0x8],0x9
004018a4] 7e df                 jle    0x0000000000401885

00401885] 8b 55 fc              mov    edx,DWORD PTR [rbp-0x4]
00401888] 8b 45 f8              mov    eax,DWORD PTR [rbp-0x8]
0040188b] 89 c6                 mov    esi,eax
0040188d] bf a3 20 40 00        mov    edi,0x4020a3
00401892] b8 00 00 00 00        mov    eax,0x0
00401897] e8 74 f4 ff ff        call   0x0000000000400d10
```

(그림 4) 출력 화면

제안시스템의 결과 출력 화면은 그림 4와 같다. 출력 화면에는 역공학 방지 코드가 우회된 어셈블리가 출력되며 각각의 어셈블리들은 Basic Block 단위로 출력된다. 화면에 출력되는 정보들은 명령어가 실행되는 주소, 명령어의 기계코드, 기계코드가 디스어셈블 된 어셈블리로 이루어져 있다. 사용자는 이 정보를 직접 이용하거나 해당 명령어로 만들어진 바이너리 출력 파일을 이용하여 분석을 진행할 수 있다.

5. 결론 및 향후 연구

본 제안 시스템을 이용하면 역공학 방지 기법이 적용되지 않은 프로그램을 얻을 수 있기 때문에, 악성코드 분석이 비교적 간단해진다. 또한 기존에 사용하던 순수한 악성코드를 대상으로 하는 분석 프레임워크들을 그대로 이용할 수 있다는 장점이 존재한다.

제안 시스템에서는 역공학 방지 기법을 시그니처를 기반으로 인식한다. 따라서, 악성코드를 배포하는 측에서 같은 방식의 역공학 방지 기법을 쓰더라도 시그니처를 조금만 알아보기 힘들게 조작한다면 탐지 하지 못하는 경우가 발생한다. 때문에 제안 시스템이 가지고 있는 시그니처 데이터를 꾸준히 업데이트 해야만 한다. 향후, 역공학 방지 기법을 인식하는 방법을 문자열 데이터, 바이너리 스트링과 같은 데이터의 패턴 매치가 아닌 ML(Machine Learning)이나 알고리즘을 통해 인식하는 방식으로 개선할 필요가 있다.

참고문헌

[1] ASEC(AhnLab Security Emegency response Center), *ASEC report*, VOL.67, 2015

[2] 심재홍, "*금융 소비자를 위협하는 악성코드 위협사례 분석*", Internet & Security Focus 5월호, 2013

[3] M. Schallner. *Beginners guide to basic linux anti anti debugging techniques*. stonedcoder.org, May 2006. http://www.stonedcoder.org/kd/lib/14-61-1-PB.pdf.

[4] F. Falcon and N. Riva, "*Dynamic Binary Instrumentation Frameworks: I know you're there spying on me*", RECon 2012

[5] MIT and HP, "*Open Source Homepage of Dynamorio*", Dynamic Instrumentation Tool Platform, 2001, http://www.dynamorio.org

[6] D. Bruening, Q. Zhao, and S. "Amarasinghe, *Transparent Dynamic Instrumentation*", Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE'12), pp. 133-144, 2012

[7] C.-K Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "*PIN: Building Customized Program Analysis Tools with Dynamic Instrumentation*", Proceedings of the 2005 ACM SIG PLAN Conference on Programming Language Design and Implementation (PLDI'05), pp.190-200, 2005

[8] 이승환, 신효정, 김형식, "*보안성 시험을 위한 안티 디버깅 API 우회 방법*", Journal of Security Engineering Vol.15, No.1, pp.25-40, 2018

[9] 정제성, 김광조, "*Cuckoo Sandbox를 회피하는 악성코드 탐지 방안 연구*", 한국정보보호학회 하계학술대회, 2015