KIPS Trans. Comp. and Comm. Sys.
Vol.10, No.11 pp.297~304
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2021.10.11.297

Analysis of Anti-Reversing Functionalities of VMProtect and Bypass Method Using Pin

Seongwoo Park† · Yongsu Park††

ABSTRACT

Commercial obfuscation tools (protectors) aim to create difficulties in analyzing the operation process of software by applying obfuscation techniques and Anti-reversing techniques that delay and interrupt the analysis of programs in software reverse engineering process. In particular, in case of virtualization detection and anti-debugging functions, the analysis tool exits the normal execution flow and terminates the program. In this paper, we analyze Anti-reversing techniques of executables with Debugger Detection and Viralization Tools Detection options through VMProtect 3.5.0, one of the commercial obfuscation tools (protector), and address bypass methods using Pin. In addition, we predicted the location of the applied obfuscation technique by finding out a specific program termination routine through API analysis since there is a problem that the program is terminated by the Anti-VM technology and the Anti-DBI technology and drew up the algorithm flowchart for bypassing the Anti-reversing techniques. Considering compatibility problems and changes in techniques from differences in versions of the software used in experiment, it was confirmed that the bypass was successful by writing the pin automation bypass code in the latest version of the software (VMProtect, Windows, Pin) and conducting the experiment. By improving the proposed analysis method, it is possible to analyze the Anti-reversing method of the obfuscation tool for which the method is not presented so far and find a bypass method.

Keywords : Reverse Engineering, Dynamic Analysis, Protector, Pin

VMProtect의 역공학 방해 기능 분석 및 Pin을 이용한 우회 방안

박 성 우† · 박 용 수††

요 약

상용 난독화 도구(프로텍터)들은 소프트웨어 역공학 과정에서 프로그램의 분석을 지연시키고 방해하는 난독화 기술 및 역공학 방해(안티리버싱) 기법을 적용시킴으로써 소프트웨어의 동작 과정을 분석하는데 어려움을 발생시키는데 목적이 있다. 특히, 가상화 탐지와 안티디버깅 기능 같은 경우 분석 도구가 발견되면 정상적인 실행 흐름을 벗어나 프로그램을 종료시킨다. 본 논문에서는 상용 난독화 도구(프로텍터) 중 하나인 VMProtect 3.5.0을 통해 Debugger Detection, Virualization Tools Detection 옵션을 적용시킨 실행 파일의 안티리버싱 기법을 분석하고 Pin을 이용한 우회 방안을 제안한다. 또한, 적용된 안티리버싱 기법을 분석하는 과정에서 Amti-VM 기술과 Anti-DBI 기술에 의해 프로그램이 종료되는 문제가 발생하기 때문에 API 분석을 통해 특정 프로그램 종료 루틴을 알아내어 적용된 안티리버싱 기법의 위치를 예상하고 위치를 바탕으로 안티리버싱 기법 우회 방안 알고리즘 순서도를 작성하였다. 실험에 사용된 소프트웨어들의 버전의 차이로부터 발생하는 호환성 문제, 기법의 변화 등을 고려하여 최신 버전의 소프트웨어(VMProtect, Windows, Pin)에서 Pin 자동화 우회 코드를 작성하고 실험을 진행하여 성공적으로 우회됨을 확인하였다. 제안된 분석 방안을 개선하여 기법이 제시되지 않은 난독화 도구의 안티리버싱 기법을 분석하고 우회 방안을 찾아낼 수 있다.

키워드 : 안티리버싱, 동적 분석, 프로텍터, 핀

1. 서 론

최근 악성코드를 이용하여 프로그램의 정보 및 중요 기술을 탈취하는데 그치지 않고 컴퓨터의 감염에 이르기까지 다양한 공격 유형을 통한 피해의 규모가 증가하고 있다. 또한, 대처가 이루어지기 전에 이미 악성 행위가 수행된 이후에는 탐지가 효과적이지 못하며 시스템 감염을 통해 시스템이 망가져버릴 수도 있다. 따라서 분석가는 이를 방지하기 위해 클라이언트 시스템이 아닌 가상머신(Virtual Machine)이나 에뮬레이터(Emulator)를 통해 의심되는 파일을 직접 분석하고 실행하며 역공학을 진행하는 과정이 이루어지고 있다.

가상머신을 통해 별도의 하드웨어가 필요 없이 클라이언트

※ 본 연구는 한국연구재단 연구과제(2020R1F1A1048443) 지원으로 수행하였습니다.
† 준 회 원 : 한양대학교 컴퓨터소프트웨어학과 석사과정
†† 비 회 원 : 한양대학교 컴퓨터소프트웨어학부 교수
Manuscript Received : April 1, 2021
First Revision : August 2, 2021
Accepted : September 16, 2021
* Corresponding Author : Yongsu Park(yongsu@hanyang.ac.kr)

시스템과 독립적인 시스템에서 프로그램의 실행이 가능하기 때문에 클라이언트 시스템에는 전혀 피해가 가지 않으며, 가상머신은 시스템이 감염되어도 빠른 초기화를 통해 효율적으로 다양한 프로그램들을 실행할 수 있다는 장점을 가지고 있다.

이러한 장점들을 채택하여, 본 논문에서는 역공학을 진행할 기본적인 실행 환경으로써 VirtualBox 가상머신을 사용하여 가상 환경 내에서 상용 난독화 도구(프로텍터)인 VMProtect의 Debugger Detection, Virtualization Tools Detection 옵션을 통해 안티리버싱 기법이 적용된 프로그램의 동적 분석을 진행한다.

동적 분석을 진행하기 위한 기본적인 분석 도구로써, 문자열을 인식하며 레지스터를 기록하고, API 호출, 라이브러리 루틴 등을 일일이 추적할 수 있는 디버거와 일종의 자동화된 분석 도구인 DBI(Dynamic Binary Instrumentation) [1,2]를 사용한다. DBI 분석 도구는 프로그램 런타임에 실행 코드를 매 명령어 단위로 삽입하여 바이너리 응용 프로그램의 동작을 동적으로 분석 가능하게 한다. 따라서 일일이 바이너리 패치를 통한 반복적인 난독화 우회 작업을 보다 효율적으로 진행할 수 있다는 장점을 가지고 있다.

최근 IEEE Access에 발표된 Y. Lee[3] 등의 연구에서는, VMProtect를 포함한 총 5가지 상용 난독화 도구들에 어떠한 Anti-VM 기술과 Anti-DBI 기술이 사용되고 있는지 분석하였고 DBI를 이용하여 해당 기법들을 우회하는 방안을 제시하였다.

본 논문에서는 VMProtect에 한정해서 [3]의 연구에서 설명한 Anti-VM 기술과 Anti-DBI 기술을 탐지하고 우회하는 알고리즘을 직접 구현해보았다. 또한, 실험에 사용된 소프트웨어들의 버전의 차이로부터 발생하는 호환성 문제, 기법의 변화 등을 고려하여 [3]의 연구에서 사용한 버전보다 최신 버전의 소프트웨어(VMProtect, Windows, Pin)에서 실험을 진행하였다. API 분석을 통해 적용된 안티디버깅 기법을 찾아내고 특정 프로그램 종료 루틴을 알아내어 적용된 난독화 기법의 위치를 예상하였으며 이를 통해 설계된 알고리즘을 바탕으로 Pin 자동화 우회 코드를 작성하여 실험을 진행하고 성공적으로 우회됨을 확인하였다.

논문의 구성은 다음과 같다. 2장에서 배경 지식, 3장에서 관련 연구를 서술하고, 4장에서 VMProtect 안티리버싱 기법, 5장에서 Pin을 이용한 VMProtect 안티리버싱 기법 우회 방안 설계 및 구현, 6장에서 실험 결과, 7장에서 결론을 기술한다.

2. 배경 지식

본 연구에서 사용된 VMProtect 난독화 도구, VirtualBox 가상머신, 그리고 DBI인 Intel Pin에 대해 간략히 소개한다.

2.1 VMProtect

Ekaterinburg 사에서 출시했으며 2020. 10. 6.에 버전 3.5.0이 발표된 VMProtect는 난독화 패킹, 가상화를 하는 도구이며 스크립트 언어와 내장 디스어셈블러를 가지고 있다. 스크립트 언어를 가지고 있기 때문에 플랫폼에 대한 제한이 없고 내장 디스어셈블러를 가지고 있기 때문에 운영 체제에 대해 제한이 없다[4,5].

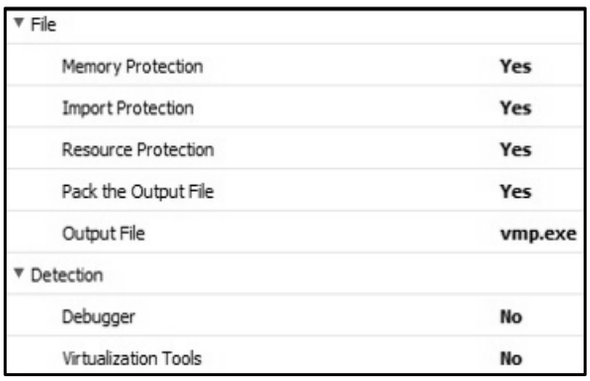

Fig. 1. Options of VMProtect

VMProtect는 Fig. 1과 같이 난독화 옵션으로써 Memory Protection, Import Protection, Resource Protection, Pack the Output File, Debugger Detection, Virutalization Tools Detection 등의 기능이 존재한다.

2.2 VirtualBox

VirtualBox는 상용 소프트웨어로써 현재 오라클이 개발 중이며, 윈도우, 리눅스, macOS, 솔라리스를 guest 운영 체제로 가상 환경을 제공하는 x86 가상화 소프트웨어이다. 현재 최신 버전은 2020. 7.에 발표된 버전 6.0이다. VirtualBox외에도 가상 환경을 제공하는 가상머신에는 여러 가상머신을 다룰 수 있는 VMware, macOS용 Paralles, 동적 변환기를 사용하는 QEMU 등이 존재한다.

2.3 Pin

Intel에서 제공하는 DBI인 Pin[6]은 윈도우/리눅스 상 x64, x86 바이너리를 지원하는 실행파일 분석기이다. Pintool이라는 공유 라이브러리를 통해 분석할 프로세스를 실행시킬 때 Pin에서 제공되는 API를 사용하여 실행한다.

Pin은 Fig. 2와 같이 분석하고자 하는 실행 파일과 Instrumentation API를 JIT(Just-in-time) 컴파일러를 통해 Pin 내부의 가상머신에 입력으로 주고 자동으로 계측하고 분석하는 구조를 가지고 있다. 프로그램 코드 사이에 삽입된 분석 코드를 통해 동적으로 프로그램의 정보를 얻을 수 있고 해당 코드는 C언어와 Pin에서 제공하는 API를 이용하여 사용자에 의해 변경될 수 있다.

3. 관련 연구

현재 다양한 상용 난독화 도구들이 존재하며 처음에는 소프트웨어 저작권과 기술의 보안을 위한 취지로 사용이 되었으나, 백신 프로그램을 무력화하기 위하여 악성코드에 난독화 기술이 적용되기 시작하였다. 이에 따라 적용된 난독화 기

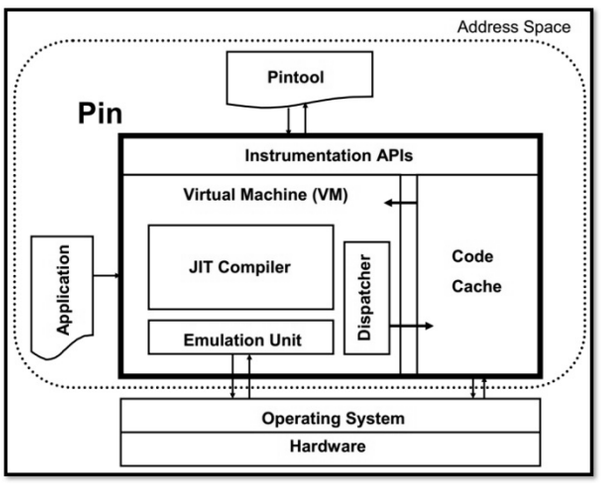

Fig. 2. Intel Pin Architecture

술을 우회하여 악성코드를 분석할 수 있는 방안에 대한 연구가 지속적으로 진행되고 있다. 난독화 기술 및 안티리버싱 기법 관련 연구를 살펴보면 다음과 같다.

P. Chen[7] 등의 연구에서는 악성코드에서 사용하는 Anti-debugging, Anti-VM 기법에 대해 조사하고 16,246개의 일반적인 악성코드와 1037개의 타겟 악성코드를 비교 분석하며 백신에 대한 탐지율 또한 분석한다. 일반적인 악성코드 기준으로 평균 75%에 해당하는 악성코드에 Anti-debugging 기법이 적용되어 있고 평균 60%에 해당하는 악성코드에 Anti-VM 기법이 적용되어 있으며, 타겟 악성코드 기준으로는 각각 68.6%, 84.2%에 해당하는 Anti-debugging, Anti-VM 기법이 적용되어 있다. 또한, R. Branco[8] 등의 연구에 의하면 악성코드 분석을 방해하기 위해 안티리버싱 기법이 이용되며 안티리버싱 기법에는 안티디버깅, 가상머신 탐지, 디스어셈블리 방해를 사용한다. 400만개의 악성코드 샘플 중 6.42%가 안티리버싱 기법을 이용하는데 안티리버싱 기법이 적용된 악성코드 샘플 중 안티디버깅이 43.21%, 가상머신 탐지가 81.40%, 디스어셈블리 방해가 12.13%에 해당된다.

이러한 난독화 기술 및 안티리버싱 기법을 분석하고 우회하는 연구 중 VMProtect 난독화 도구를 이용하여 진행한 C. Bang[9] 등은 VMProtect에서 제공하는 난독화 옵션 중 Pack the Output File, Import Protection을 적용시켜 PE 파일 구조 분석과 함께 난독화 특징을 찾아내어 자동 역난독화 방안을 설계하였다. S. Kim[5] 등은 VMProtect에서 제공하는 난독화 옵션 중 가상화 난독화를 적용시킨 실행 파일에 대해 최적화 및 정적/동적 코드 자동 분석을 통해 역난독화 방안을 연구하였다.

그 외에도 대중적인 상용 난독화 도구인 Themida 난독화 도구를 이용하여 난독화 기술들에 대한 연구가 진행되었다. J. Lee[10] 등은 최신 버전의 Themida 난독화 도구를 이용하여 가상 메모리 할당을 사용하지 않도록 변경되어 정규화가 어려워진 API 난독화 기술을 분석하였고, Y. Kang[11] 등은 Themida의 패킹 동작 방식을 분석하여 OEP(Original Entry Point) 시점에서 원본 프로그램의 정보를 복원하고 원본 코드 영역으로 넘어가는 역난독화 방안을 연구하였다.

특정 난독화 도구에 한정되지 않고 다양한 난독화 도구들의 난독화 기술에 대한 연구 및 실험 또한 진행되었다. J. Park[12] 등은 Pin을 이용하여 Pin이 탐지되는 안티디버깅 기법들을 소개하고 이를 우회하는 Pin 코드를 하나로 합쳐 다양한 상용 난독화 도구들의 안티디버깅 기법을 우회 할 수 있는 Pin 도구를 제시하였다.. 또한, 앞서 기술된 Y. Lee[3] 등의 연구에서는 상용 난독화 도구들에 적용된 Anti-VM 기법과 Anti-DBI 기법들을 소개하고 이를 디버거 명령어 패치를 통해 우회하는 방안과 Pin을 이용하여 Anti-VM 기법을 우회하는 알고리즘을 제시하였다.

4. VMProtect 안티리버싱 기법

VMProtect의 난독화 옵션을 통해 가상머신 실행 환경을 탐지할 수 있을 뿐만 아니라, 안티리버싱 기법을 적용시켜 가상머신 내에서 동적 분석 도구인 디버거와 DBI를 탐지하고 실행 흐름을 바꾸거나 중단시킨다. 따라서 VMProtect에서 제공하는 난독화 옵션 중 Debugger Detection을 적용시켰을 경우에 이용되는 Anti-Debugging 기법과 Virtualization Tools Detection을 적용시켰을 경우에 이용되는 Anti-VM 기법에 대해 기술하고 두 옵션 중 하나라도 적용시켰을 경우에 이용되어 DBI를 탐지하는 Anti-DBI 기법에 대해 기술한다.

4.1에서는 Debugger Detection 옵션에 의해 적용되는 안티디버깅 기법을 기술하며 4.2에서는 Virtualization Tools Detection 옵션에 의해 적용되는 Anti-VM 기법을 설명한다[3]. 4.3에서는 Pin을 탐지하는 DBI Detection 기술을 설명한다[12]. 4.2와 4.3의 각 기법은 API 분석을 통해 특정 프로그램 종료 루틴을 알아내어 위치를 예상하였다.

4.1 Debugger Detection이 적용된 기법

호출된 API를 통해 확인한 결과, VMProtect의 Debugger Detection 옵션에 의해 적용되는 안티디버깅 기법에는 순서대로 IsDebuggerPresent(API), CheckRemoteDebuggerPresent (API), NtQueryInformationProcess(API), NtSetInformationThread (API), CloseHandle(API) 총 5가지가 적용되며 해당 안티디버깅 기법들은 지난 연구들 [13]을 통해 수차례 분석된 바 있다.

4.2 Virtualization Tools Detection이 적용된 기법

[3]에 의하면, VMProtect의 Virtualization Tools Detection 옵션에 의해 적용되는 Anti-VM 기법에는 순서대로 총 2가지가 적용되며 그 결과는 다음과 같다.

1) CPUID(Instruction)

CPUID 명령어는 매개변수를 이용하지 않고 EAX 레지스터를 그대로 사용하여 EAX 값에 따라 다른 정보를 반환한다. 우선 적용된 Anti-VM 기법과 기법의 위치를 예상하기 위해

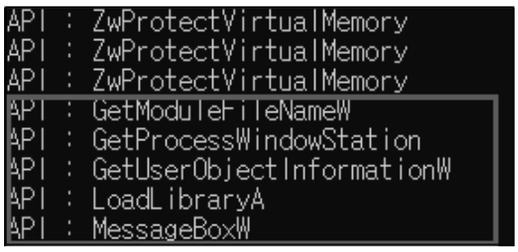

Fig. 3. Termination Routine(API)

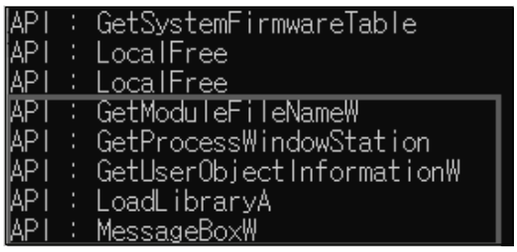

Fig. 4. API Termination Routine (API)

서 DBI를 통해 가상머신이 탐지되지 않는 로컬 시스템에서 호출되는 API와 가상머신이 탐지될 경우 호출되는 API를 비교한다.

가상머신이 탐지되지 않는 로컬 시스템에서는 Fig. 3의 경우에서 ProtectVirtualMemory API가 호출된 이후에 더 이상 출력되는 API 루틴 없이 아무런 메시지를 띄우지 않고 종료된다. 하지만 가상 환경에서 실행하여 가상머신이 탐지된 경우인 Fig. 3에서는 DBI가 탐지되기 전에 가상머신이 탐지되어 빨간 박스 내의 특정 종료 루틴으로 실행 흐름이 진행되는 것을 확인할 수 있다. 이를 통해 가상 머신을 탐지하는 Anti-VM 기법의 예상 위치와 어떤 API에 의해 기법이 적용되는지 알아낼 수 있으며 해당 위치에서 CPUID 명령어를 통해 가상 머신이 탐지된다. AMD 및 Intel의 CPU는 레지스터 ECX의 31번째 비트를 가상머신을 생성하고 실행하는 프로세스인 하이퍼 바이저의 존재 비트로 예약하고 있기 때문에 Anti-VM 기법으로 적용된 CPUID 명령어는 EAX 값에 0x1을 넣고 CPUID를 실행했을 시 반환되는 ECX의 31번째 비트 값을 통해 가상머신을 탐지한다. ECX 값의 31번째 비트는 가상머신이 존재할 경우 1, 아닐 경우 0을 반환하여 0일 경우에 가상머신이 탐지되고 위 그림과 같이 특정 종료 루틴으로 실행 흐름이 변경되어 탐지 메시지를 띄우고 프로그램이 종료된다.

2) GetSystemFirmwareTable(API)

GetSystemFirmwareTable 함수를 통해 SMBIOS(System Management BIOS) 정보를 쿼리하여 가상머신 이름의 문자열 'VirtualBox'에 접근이 가능하다. 적용된 Anti-VM 기법과 기법의 위치를 식별하기 위해서 CPUID 명령어에 의해 가상머신이 탐지되었을 경우 호출되는 API를 통해 프로그램 종료 루틴을 파악한다.

Fig. 4는 5장에서 설명할 우회 방안에 의해 CPUID 기법과 DBI Detection 기법을 우회한 후에 가상머신이 탐지되고 그 후로 진행되는 종료 루틴이다. Fig. 3의 종료 루틴과 같은 루틴이 발생하는 Fig. 4에서는 종료 루틴이 진행되기 전 GetSystemFirmwareTable이 호출되는 것을 확인할 수 있다. 이를 통해 가상 머신을 탐지하는 Anti-VM 기법의 예상 위치와 어떤 API에 의해 기법이 적용되는지 알아낼 수 있으며 해당 위치에서 CMP BYTE PTR DS:[EDX], 0x56 명령어를 통해 EDX 레지스터에 들어있는 메모리 주소에 저장된 가상머신 문자열의 첫 바이트 값을 ASCII 코드에서 0x56이 의미하는 문자인 "V"와 비교한다. 문자열이 같을 경우 반환되는 Flag 값에 의해 가상머신이 탐지되고 CPUID와 같이 특정 종료 루틴으로 실행 흐름이 변경되어 탐지 메시지를 띄우고 프로그램이 종료된다.

4.3 DBI Detection이 적용된 기법

DBI Detection 기법은 VMProtect에 따로 옵션으로 존재하지는 않지만 Debugger Detection 또는 Virtualization Tools Detection 옵션을 적용시켰을 경우 이용되는 기법이다.

1) Single Step Exception

Single Step exception은 Debugger Detection 또는 Virtualization Tools Detection 옵션을 적용시켰을 경우 이용되는 안티리버싱 기법이며 자세한 설명은 [12]에 있다.

로컬 시스템에서 해당 기법이 적용된 실행 파일을 실행할 경우 Fig. 3 또는 Fig. 4와 같이 프로그램 특정 종료 루틴을 거치지 않고 중단된다. 이는 DBI가 탐지되어 실행 흐름이 중단되는 것을 확인할 수 있고 ProtectVirtualMemory API 다음으로 DBI 탐지 기법이 적용되어 있음을 확인할 수 있다.

해당 위치에서 일어나는 Single Step Exception은 EFLAGS의 trap flag 값이 1이 될 경우 CPU를 Single Step 모드로 변경하고 EXCEPTION_SINGLE_STEP을 발생시키는데 이와 같은 예외 기반 안티리버싱 기법은 SEH(Structured Exception Handler) 예외 처리기에 특정 루틴을 수행하게 하고 예외를 발생시켜 SEH 호출 유무에 따라 분석 도구를 탐지한다. 따라서 Single Step Exception은 우선 특정 루틴을 입력한 다음 Fig. 5와 같이 pushfd 명령어를 통해 스택에 EFLAGS 값을 저장한다. EFLAGS를 따로 변경하는 명령어가 존재하지 않기 때문에 pushfd 명령어로 스택에 저장한 trap flag 값을 or 명령어를 통해 1로 변경하고 popfd 명령어로 다시 EFLAGS 값에 저장해야한다. 결과적으로 SEH가 수행되지 않아 프로그램의 실행 흐름이 바뀌거나 중단되면 DBI가 탐지된 경우이다.

```
pushfd
or dword prt ss:[esp], 100
popfd
```

Fig. 5. Single Step Exception

5. Pin을 이용한 VMProtect 안티리버싱 기법 우회 방안 설계 및 구현

VMProtect의 Debugger Detection 옵션에 의해 적용되는 5가지 Anti-Debugging 기법은 DBI 분석 도구 Pin을 이용하여 분석하면 탐지되지 않는다. 따라서, Pin을 통해 자동 우회되는 Debugger Detection 안티리버싱 기법을 제외한 Virtualization Tools Detection 기법과 DBI Detection 기법을 우회하는 방안을 Pin을 통해 설계 및 구현한다. 우선 각 기법의 우회 방안을 알고리즘 순서도를 통해 설계하고 코드를 작성하여 우회를 구현한다.

5.1 Virtualization Tools Detection 우회 방안 설계 및 구현

4절에서 예상된 Anti-VM 기법의 위치에 따른 우회 방안 설계는 Fig. 6과 같다. VMProtect의 Debugger Detection 옵션이 적용된 파일을 Pin으로 실행시키면 실행 도중 우선 CPUID 명령어가 수행되고 가상 머신 환경 여부를 확인하기 때문에 매 명령어가 수행되기 전에 CPUID 인지 확인하고 우회한다. 다음으로 특정 문자열 비교 명령어를 통해 가상 머신 환경 여부를 확인하므로 매 명령어가 특정 문자열 비교 명령어인지 확인하고 우회한다. 그러나 해당 우회 과정은 실행 도중 여러 번 반복될 수 있으며 전부 우회된 후에 정상 종료된다.

Pin을 통한 가상머신 탐지 우회 방안은 해당 레지스터값을 변경함으로써 비교적 쉽게 우회가 가능하다. CPUID기법은

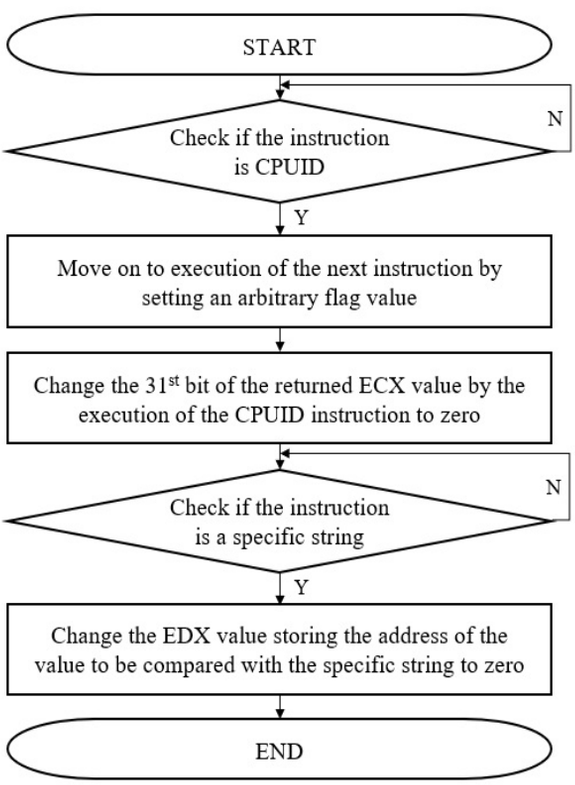

Fig. 6. Flowchart for Bypassing the Anti-VM Technique

CPUID 명령어를 실행했을 시 반환되는 ECX의 31번째 비트 값을 통해 가상머신을 탐지하기 때문에 Fig. 7과 같이 우선 문자열 비교 함수를 통해 CPUID 명령어를 찾아내고 CPUID 명령어가 발견될 때마다 플래그 값을 설정한다. 플래그 값을 설정하는 이유는 CPUID 명령어가 실행되기 전에 EAX 값에 0x1을 넣은 다음 실행을 하기 때문에 CPUID 명령어를 찾은 순간에는 EAX 값에 0x1이 들어있고 아직 명령어가 실행되지 않은 상태이기 때문이다. 다음으로 설정된 플래그를 통해 CPUID 명령어가 실 행되고 ECX 값으로 가상머신이 탐지되는 값으로 반환되면 ECX 레지스터의 31번째 비트 값을 XOR 연산을 통해 1을 0으로 변경시켜서 가상머신이 탐지되지 못하도록 하여 우회를 구현한다.

GetSystemFirmwareTable 함수 호출을 통한 가상머신 탐지 기법 또한 Fig. 8과 같이 문자열 비교 함수를 통해 ASCII 코드에서 0x56을 의미하는 문자 'V'와 EDX 레지스터에 들어있는 메모리 주소에 저장된 가상머신 문자열을 비교하는 명령어인 'cmp byte ptr [edx], 0x56'가 발견될 때마다 EDX 값을 0으로 변경 및 고정시켜서 cmp 명령어에 의해 비교되는 두 값이 같을 경우가 나오지 않게함으로써 우회를 구현한다.

5.2 DBI Detection 안티리버싱 기법 우회 방안 설계 및 구현

4절에서 예상된 Anti-DBI 기법의 위치에 따른 우회 방안 설계는 Fig. 9와 같다. Anti-DBI 기법이 적용된 파일을 Pin으로 실행시키면 실행 도중 Single Step exception이 발생하기 때문에 예외가 발생하기 전에 popfd 명령어를 발견하

```
if (tempcpu == 1)
{
        // change the 31st bit value of ECX to 0
        (UINT32)*ecx = (UINT32)*ecx ^ 0x80000000;
        cout << hex << "ECX:" << (UINT32)*ecx
<< endl;
        tempcpu = 0;
}
else if (strncmp(ins, "cpuid", 5) == 0)
{
        // move on to the next instruction of CPUID
           with a temporary value
        tempcpu = 1;
}
```

Fig. 7. Bypass Code for CPUID

```
if (strncmp(ins, "cmp byte ptr [edx], 0x56", 24) == 0)
{
        // change the EDX value comparing strings to 0
        (UINT32)*edx = (UINT32)*edx & 0x0;
}
```

Fig. 8. Bypass Code for GetSystemFirmwareTable

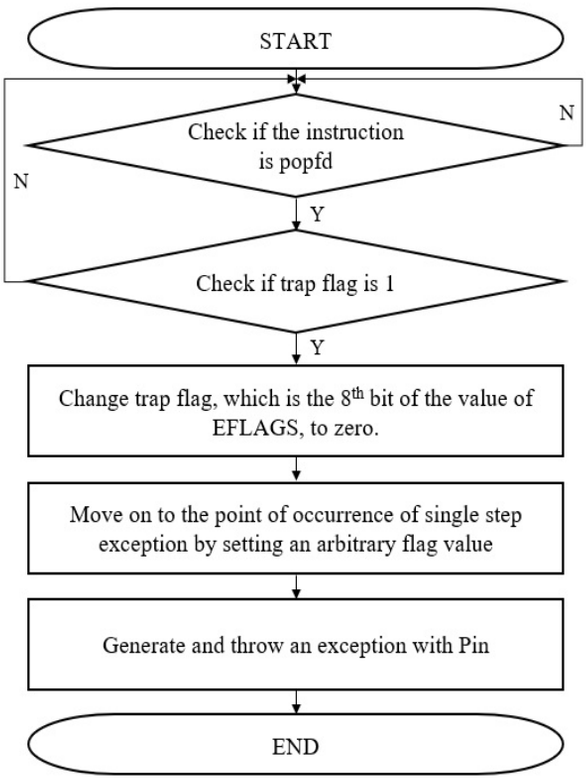

Fig. 9. Flowchart for Bypassing the Anti-DBI Technique

고 우회한다. 그러나 해당 우회 과정은 실행 도중 여러 번 반복될 수 있으며 전부 우회된 후에 정상 종료된다.

Pin은 Single Step exception을 처리하지 못하기 때문에 프로그램이 실행 도중 중단된다. 따라서 Pin을 통한 DBI 탐지 우회는 Single Step exception 발생 전에 Pin에 의해 처리가 가능한 예외를 발생시킴으로써 우회가 가능하다. 우선, Fig. 10과 같이 문자열 비교함수를 통해 스택에 저장된 EFLAGS 값을 레지스터로 옮기는 popfd 명령어가 발견될 때마다 스택에 접근해서 XOR 연산을 통해 EFLAGS의 8번째 비트인 trap flag를 0으로 변경하여 Single Step exception이 발생하지 않게 하고 플래그를 설정한다. 플래그를 설정하는 이유는 강제로 예외를 발생시킬 주소가 popfd 명령어와 그 다음 명령어까지 실행된 후이기 때문이다. 바뀐 EFLAGS 값을 가지고 popfd 명령어를 수행한 다음, 다음 명령어에서 플래그 값을 증가시킴으로써 또 다음 명령어로 넘어간다. 결국 popfd 명령어와 다음 명령어가 실행된 후에 Pin 함수를 통해 강제로 예외를 발생시키면 SEH 호출하여 예외 처리를 진행하게 되고 SEH에 입력된 특정 루트를 수행하게 함으로써 우회를 구현한다.

```
if (tempstep == 2)
{
        // occurence point of Single Step exception
        tempstep = 0;
        EXCEPTION_INFO exceptInfo;
        // throw an exception with Pin
        PIN_InitExceptionInfo(&exceptInfo, EXCEPTCO
    DE_DBG_SINGLE_STEP_TRAP, (ADDRINT)instaddr);
        PIN_RaiseException(context, threadid,
&exceptInfo);
}
if (tempstep == 1)
{
        // execute up to the next instruction of popfd
           with a temporary value
        tempstep++;
}
if (strncmp(ins, "popfd", 5) == 0)
{
        // check if the trap flag, which is the 8th bit of
           the value of EFLAGS in ESP, is 0
        if(((UINT32)*esp & 0x00000100)==0x00000100)
        {
                // change trap flag to 0
                (UINT32)*esp = (UINT32)*esp ^
        0x100;
                tempstep = 1;
        }
}
```

Fig. 10. Bypass Code for Single Step Exception

6. 실험 결과

앞서 제시된 우회 방안 구현에 의해 정상적인 프로그램 실행 흐름으로 진행되는지 여부를 확인하기 위해 특정 문자열('Hey, this actually works.')을 출력하는 실행 파일에 Debugger Detection 옵션과 Virtualization Tools Detection 옵션을 적용시켜 실험을 진행하였다. 실험 환경은 32비트 실행 파일에 VMProtect 3.5.0, VirtualBox 6.0, x64 Windows 10 20H2 버전 가상 환경 그리고 pin 3.0이 사용되었다.

전체적으로 적용된 안티리버싱 기법 순서는 Pin에 의해 우회되는 IsDebuggerPresent, CheckRemoteDebuggerPresent, NtQueryInformationProcess, NtSetInformationThread, CloseHandle 순으로 먼저 적용되어 있고, 그 다음으로 CPUID, Single Step exception, GetSystemFirmwareTable 순으로 적용되어있지만 마지막 세 가지 기법은 프로그램 실행 과정에서 반복적으로 적용되어 있다.

Pin 우회 코드를 삽입하여 프로그램을 실행하면, 자동적으로 Debbuger Detection 기능들이 전부 우회되고 CPUID 명령어가 발견되어 레지스터 ECX 값의 31번째 비트를 연산을 통해 0으로 변경한다. 다음으로 popfd 명령어 발견 시 스택에 들어있는 EFLAGS 값의 8번째 비트를 확인하고 1일 경우 Single Step exception이 발생하므로 0으로 수정 후 예외를 발생시킨다. 이 두 가지 기법은 실행 과정에서 반복적으로 발생하여 우회가 진행되며 마지막으로 GetSystemFirmwareTable 함수가 호출된

Fig. 11. Normal Execution Flow

시점에서 EDX 값으로 0이 저장된 메모리 주소의 값과 ASCII 코드 값을 비교하는 명령어가 수많은 반복 작업을 통해 실행되며 결과적으로 전부 탐지하고 우회함을 확인하였으며 Fig. 11과 같이 기존의 특정 문자열이 출력되었다.

7. 결 론

소프트웨어 역공학 과정에서 프로그램의 분석을 지연시키고 방해하는 난독화 기술 및 안티리버싱 기법을 가지는 난독화 도구(프로텍터)는 프로그램의 정상적인 실행 흐름을 벗어나거나 그대로 종료시키는 문제를 일으킨다. 특히, 난독화 도구(프로텍터)중 하나인 VMProtect는 다양한 옵션을 통해 분석 환경과 분석 도구를 탐지하기 때문에 VMProtect의 옵션이 적용된 프로그램은 분석하는 데 어려움이 있다. 이러한 문제를 해결하고 다양한 분석 환경에서 프로그램을 분석 가능할 수 있도록 분석 도구 탐지 옵션인 Debugger Detection, Virualization Tools Detection 옵션을 적용시킨 실행 파일의 안티리버싱 기법을 분석하고 Pin을 이용한 우회 방안을 제안하였다.

우선 프로그램 내에서 호출되는 API 분석을 통해 특정 프로그램 종료 루틴을 찾아내었다. 이를 통해 해당 프로그램에서 발생할 수 있는 모든 안티리버싱 기법들의 위치를 예상하였고 어떤 API에 의해 기법이 적용되는지 알아낼 수 있었다. 다음으로 예상된 위치를 바탕으로 적용된 Anti-VM 기법과 Anti-DBI 기법에 대한 우회 방안을 알고리즘 순서도로 작성하였다. 마지막으로 해당 기법들에 대한 Pin 자동화 우회 코드를 작성하여 실험을 진행하고 성공적으로 우회됨을 확인하였다.

다양한 난독화 도구들 또는 VMProtect의 모든 난독화 옵션을 분석하지는 못하였으나 본 논문에서 제안된 방안을 통해 가상머신과 Pin에서 VMProtect의 안티리버싱 옵션이 적용된 프로그램을 정상적으로 분석 가능할 것으로 보인다. 향후에는 타 연구로부터 본 논문에서 다루지 않은 VMProtect의 다른 난독화 옵션들을 우회하는 방안을 함께 적용하면 모든 옵션을 우회하고 정상적인 흐름으로 실행 파일을 실행할 수 있을 것으로 보이며 타 연구에서 제시된 다른 난독화 도구의 기법을 알고리즘과 함께 우회 가능할 것으로 보인다. 또한 기존 연구에서 아직까지 기법이 제시되지 않은 난독화 도구의 안티리버싱 기법을 분석하고 우회 방안을 찾아내는 연구를 진행할 예정이다.

References

[1] J. Kirsch, Z. Zhechev, B. Bierbaumer, and T. Kittel, “PwIN - Pwning Intel piN: Why DBI is Unsuitable for Security pplications,” In: J. Lopez, J. Zhou, and M. Soriano (eds), *Computer Security. ESORICS 2018.* Lecture Notes in Computer Science, Vol.11098. Springer, Cham., 2018.

[2] D. C. D’Elia, E. Coppa, S. Nicchi, F. Palmaro, and L. Cavallaro, “SoK: Using dynamic binary instrumentation for security (And how you may get caught red Handed),” *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp.15-27, 2019.

[3] Y. B. Lee, J. H. Suk, and D. H. Lee, “Bypassing Anti-Analysis of Commercial Protector Methods Using DBI Tools,” *IEEE Access*, Vol.9, pp.7655-7673, 2021.

[4] VMSoft. “VMProtect software: VMProtect virtualizes code,” 2018. [Internet], http://vmpsoft.com/products/vmprotect/,

[5] S. Kim. “Code Automatic Analysis Technique for Virtualization-based Obfuscation and Deobfuscation,” *Journal of Korea Institute of Information, Electronics, and Communication Technology*, pp.724-731, 2018.

[6] Chi-Keung Luk, et al., “Pin: building customized program analysis tools with dynamic instrumentation,” In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Vol.40, No.6, pp.190-200, 2015.

[7] P. Chen C. Huygens L. Desmet, and W. Joosen, “Advanced or not? A comparative study of the use of anti-debugging and anti-vm techniques in generic and targeted malware,” *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp.323-336, 2016.

[8] R. R. Branco, G. N. Barbosa, and P. D. Neto, “Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies,” *black hat USA*, 2012. https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_Slides.pdf

[9] C. Bang, J. H. Suk, and S. Lee, “VMProtect Operation Principle Analysis and Automatic Deobfuscation Implementation,” *Journal of the Korea Institute of Information Security & Cryptology*, Vol.30, No.4, pp.605-616, Aug. 2020.

[10] J. Lee, B. Lee, and S. Cho, “A Study on the Analysis Method to API Wrapping that Difficult to Normalize in the Latest Version of Themida,” *Journal of the Korea Institute of Information Security & Cryptology*, Vol.29, No.6, pp.1375-1382, Dec. 2019.

[11] Y. Kang, M. Park, and D. Lee. “Implementation of the Automated De-Obfuscation Tool to Restore Working Executable.” *Journals of the Korea Institute of Information Security And Cryptology*, Vol.27, No.4, pp.785-802, 2017.

[12] J. Park Y. Jang S. Hong, and Y. Park, "Automatic detection and bypassing of anti-debugging techniques for microsoft windows environments," *Advances in Electrical and Computer Engineering*, Vol.19, No.2 pp.23-29, 2019.
[13] Peter Ferrie: The "Ultimate" Anti-Debugging Reference, 2011. http://pferrie.host22.com/papers/antidebug.pdf

박 성 우
https://orcid.org/0000-0002-3988-120X
e-mail : psw921@gmail.com
2019년 University of California, Irvine 수학과(학사)
2020년 ~ 현 재 한양대학교 컴퓨터소프트웨어학과 석사과정
관심분야 : 소프트웨어 보안, 소프트웨어 역공학, 정보 보호

박 용 수
https://orcid.org/0000-0002-7354-4434
e-mail : yongsu@hanyang.ac.kr
1996년 KAIST 전산학과(학사)
1998년 서울대학교 컴퓨터공학과(석사)
2003년 서울대학교 전기컴퓨터공학부(박사)
2003년 ~ 2004년 서울대학교 자동제어특화연구센터 연수연구원
2005년 ~ 현 재 한양대학교 컴퓨터소프트웨어학부 교수
관심분야 : 컴퓨터 보안, 인터넷 보안