情報保護學會論文誌
第19卷 第2號, 2009. 4

윈도우즈에서 제공되는 기본 API에 대한 안전성 고찰

최 영 한,†‡ 김 형 천, 오 형 근, 이 도 훈
ETRI 부설연구소

An Empirical Study of Security for API in Windows Systems

YoungHan Choi,†‡ HyoungChun Kim, HyungGeun Oh, DoHoon Lee
The Attached Institute of ETRI

요 약

본 논문에서는 전세계적으로 90%이상의 사용자층을 보유하고 있는 윈도우즈 OS의 API에 대한 보안 테스팅 중 Fuzz Testing을 적용하여 그 안전성을 검증하였다. 본 논문에서는 윈도우즈의 시스템 폴더 내에 구현된 함수들을 대상으로 테스팅하기 위해 Fuzz Testing 기반 자동화 방법론인 AWAFT를 제안하였다. AWAFT는 보안 취약점 중 버퍼오버플로우와 함수 파라미터 파싱 오류에 초점을 맞추고 있다. AWAFT를 자동화하기 위한 도구를 구현하였으며, Windows XP SP2 시스템 폴더에 적용한 결과 177개의 프로그램 종료 에러를 발견하였으며, 이 중 10개는 프로그램의 실행 흐름을 변경시킬 수 있는 보안상 위험한 취약점이었다. AWAFT는 윈도우즈 기반으로 개발되는 소프트웨어의 라이브러리에 대해 보안 향상을 위해 적용 가능하다.

ABSTRACT

In this paper, we test for security targeting on APIs of Windows OS that is used by many people worldwide. In order to test APIs in DLL fils of Windows OS, we propose Automated Windows API Fuzz Testing(AWAFT) that can execute fuzz testing automatically and implemented the practical tool for AWAFT. AWAFT focuses on buffer overflows and parsing errors of function parameters. Using the tool, we found 177 errors in the system folder of Windows XP SP2. Therefore, AWAFT is useful for security testing of Windows APIs. AWAFT can be applied to libraries of third party software in Windows OS for the security.

Keywords: Software Security Testing, Fuzz Testing

I. 서 론

윈도우즈 Operating System(OS)은 전세계적으로 90%이상의 시장 점유율을 가지고 있는 가장 많은 사용자층을 보유한 운영체제이다[1]. 우리나라 역시 윈도우즈 OS의 사용 편이성으로 인해 대부분의 일반 사용자들이 이용하고 있으며, 이들 사용자들을 위해 윈도우즈용 소프트웨어가 다양하게 개발되고 있다. Microsoft사는 다른 OS와 같이 윈도우즈용 소프트웨어 개발자들을 위해 윈도우즈의 다양한 기능을 사용할 수 있는 Application Programming Interface (API)를 제공하고 있다. 개발자들은 API를 이용하여 시스템 자원을 직접 관리하지 않고 소프트웨어의 기능을 편리하게 구현할 수 있다. 하지만 윈도우즈에서 제공하는 API에 문제가 있으면 해당 API를 이용한 소프트웨어 역시 문제를 내포하게 된다. 게다가 API의 문제가 보안과 관련된 것이면 해당 소프트웨어는 정보 유출이나 시스템 장악이라는 2차적인 피해를 보게 된다.

윈도우즈 관련 보안 취약점은 높은 시장점유율로 인해 매년 발표되고 있으며 꾸준히 늘어나고 있다. Microsoft사는 2006년 78개, 2007년도 69개,

접수일(2008년 5월 29일), 수정일(1차: 2008년 8월 19일, 2차: 2008년 9월 30일), 게재확정일(2009년 1월 22일)
† 주저자, yhch@ensec.re.kr
‡ 교신저자, yhch@ensec.re.kr

2008년 8월 51개의 보안 패치를 배포하여 자사 제품에 대한 보안 취약점을 수정하고 있으며, 이와함께 수정되지 않고 보고만 된 취약점을 합하게 되면 더 많아진다. 이들 취약점은 다양한 소프트웨어에서 발생하지만 궁극적인 원인은 기능을 구현하는 API의 문제로 인해 발생한 것이다. 따라서 소프트웨어의 보안 안전성을 향상시키기 위해선 소프트웨어를 구성하는 API에 대해 보안 취약점을 찾아 수정하는 것이 필요하다.

본 논문은 많은 개발자들이 기본으로 사용하는 윈도우즈 API에 대한 보안 테스팅을 수행하여 안전성을 검증하는 것에 목적이 있다. 윈도우즈에서는 API를 익스포트된 함수와 COM 함수로 기능을 제공하고 있으며 본 논문은 DLL 파일의 익스포트된 함수를 대상으로 안전성을 검증 하였다. 본 논문에서는 윈도우즈 설치시 기본적으로 제공하는 API에 대해 보안 테스팅을 자동으로 수행할 수 있는 방법론을 제안하였으며, 해당 방법론을 Automated Windows API Fuzz Testing(AWAFT)으로 명명하였다. AWAFT는 Fuzz Testing에 기반을 두고 있는 소프트웨어 보안 테스팅 방법론이다. Fuzz testing은 소프트웨어의 보안성을 검사하기 위해 최근에 많이 적용되는 방법으로 버퍼오버플로우(Buffer Overflow)와 같은 보안상 문제점을 찾는데 효과가 있다[2]. 버퍼오버플로우는 프로그램의 실행 흐름을 의도적으로 변경할 수 있는 위험한 취약점으로, 최근 발표되는 대부분의 심각한 보안 취약점은 해당 문제점과 관련이 있다. AWAFT는 DLL 파일 내 익스포트된 함수이름 목록을 추출하여 API에 대한 정보를 가지고 있는 MSDN[3]에서 검색한 후 해당 함수에 대한 프로토타입을 얻는다. 프로토타입을 이용하여 해당 함수를 호출하는 C 소스코드를 생성하며, 파라미터에 다양한 폴트를 삽입함으로써 Fuzz Testing을 수행한다. 본 논문에서는 윈도우즈 시스템 폴더(C:\Windows\System32)에 있는 DLL 파일들을 실험대상으로 하여, 3,179개의 API를 자동으로 컴파일하여 Fuzz Testing을 수행하였으며, 총 10개의 보안 취약점을 찾았다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 API를 대상으로 Fuzz Testing을 수행하는 연구에 대해 살펴본다. 3장에서는 본 논문에서는 제안한 윈도우즈 기본 API에 대한 자동화 Fuzz Testing 방법론을 살펴보며, 4장에서는 해당 방법론을 기반으로 윈도우즈 시스템 폴더에 있는 DLL 파일들에 대해 보안 테스팅을 수행한다. 마지막으로 5장에서는 결론을 맺는다.

II. 관련연구

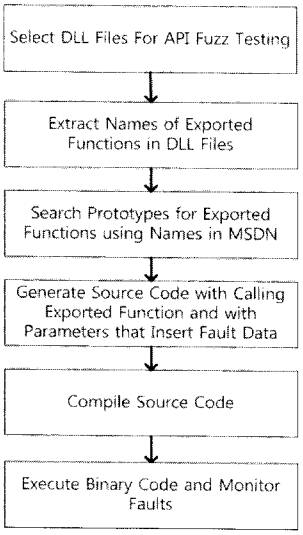
Fuzz Testing은 소프트웨어의 입력에 타당하지 않은 값을 삽입하여 해당 소프트웨어의 이상현상을 찾는 블랙 테스팅의 한 방법이다[4,5]. Fuzz Testing의 입력으로 파일(file), 설정(configuration), 레지스트리(registry), UI(User Interface), 네트워크, 데이터베이스 등 다양하다[6]. 이들 입력 포인트 중 본 논문에서는 API에 초점을 맞춘다.

Forrester는 윈도우즈에서 동작을 하는 어플리케이션에 대해 Fuzz Testing을 수행하였다[7]. 해당 연구는 Unix 시스템에 적용한 Fuzz Testing을 윈도우즈 환경으로 확장한 것이다[8]. 해당 연구는 GUI 기반 어플리케이션에 대해 키보드, 마우스의 이벤트와 Win32 메시지의 2종류 입력 포인트에 대해 fuzz testing을 수행하였다. 따라서 해당 연구는 대상이 되는 입력포인트와 관련이 있는 SendMessage(), PostMessage(), keybd_event(), mouse_event() 함수에 대해서만 테스팅을 수행하였다. Shelton은 본 논문과 같이 윈도우즈 API에 대해 fuzz testing을 수행하였다. 해당 논문에서는 윈도우즈의 다양한 버전에 대해 237개의 함수에 대해서만 수행하였다[9]. Ghosh은 윈도우즈용 COTS (Commercial-Of-The- Shelf) 어플리케이션에 대한 Robustness를 테스팅하였다[10]. 해당 논문은 윈도우즈 시스템에서 동작을 하는 어플리케이션의 에러 및 예외 처리 루틴에 대해 테스팅을 수행하였다. 테스팅을 수행하기 위해 IAT(Import Address Table)을 수정하여 대상 API에 대해 파라미터 및 함수 반환값을 변경하도록 하였다. 하지만 해당 연구에서는 어플리케이션의 실행 파일에 내포된 IAT에 내포된 제한된 함수에 대해서만 Fuzz Testing을 수행할 수 있다.

III. 윈도우즈 제공 API 안전성 검증 방법론

본 장은 윈도우즈에서 제공하는 API의 안전성 검증을 위한 자동화 테스팅 방법에 대해 제안한다. 이는 윈도우즈에서 제공하는 API의 수가 많기 때문에 수동으로 보안 테스팅을 수행하는데 한계가 있기 때문이다. 윈도우즈는 DLL 파일에서의 익스포트된 함수나 COM으로 API들을 개발자에게 제공하고 있다. 본 장에서는 DLL 파일의 익스포트된 함수를 대상으로 하였으며, 해당 방법론을 Automated Windows

API Fuzz Testing(AWAFT)으로 명명하였다. AWAFT를 기반으로 4장에서는 윈도우즈의 시스템 폴더에 대해 보안 테스팅을 수행하였다. AWAFT는 버퍼오버플로우의 취약점을 찾기 위해 Fuzz Testing을 기반으로 소프트웨어 보안 테스팅을 수행한다. Fuzz Testing은 최근 소프트웨어의취약점을 찾는데 많이 사용되고 있는 방법으로, 버퍼오버플로우(Buffer Overflow)나 함수의 파라미터 파싱과 관련된 문제를 찾는데 효과가 있다. 버퍼오버플로우 취약점은 소프트웨어의 실행 흐름을 변경시킬 수 있는 위험한 문제로 최근 발생하는 보안 관련 취약점은 대부분 버퍼오버플로우와 관련이 있다. 다음은 윈도우즈에서 제공하는 API의 보안 테스팅을 자동으로 수행하기 위한 AWAFT의 과정을 나타낸다.



〔그림 1〕 AWAFT 방법론

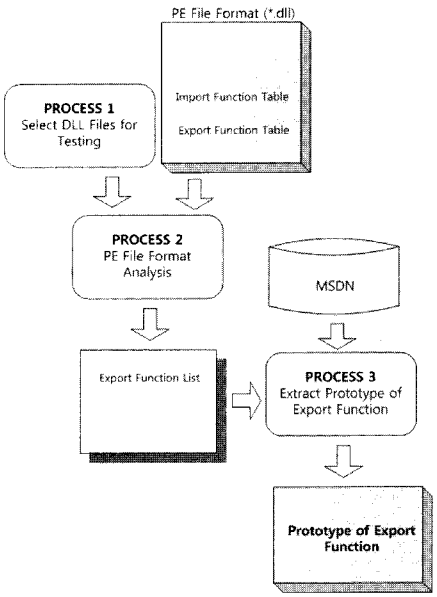
1. AWAFT를 수행할 DLL을 선택한다. AWAFT를 수행할 DLL 파일을 수동으로 선택한다. 해당 과정 이 후의 AWAFT 전과정은 자동으로 수행된다. 본 논문에서는 윈도우즈에서 제공하는 API의 안전성 검증을 위해 시스템 폴더(C:\Windows\system32) 내의 모든 DLL을 대상으로 한다. 하지만 AWAFT는 이 외의 DLL 파일들에 대해서도 확장 가능하다.
2. DLL 파일내에서 익스포트된 API의 함수명을 추출한다. DLL 파일은 PE 파일포맷〔11〕으로 이루어져 있으며, DLL 파일에서 제공하는 익스포트된 함수명 리스트를 파일 내부에 가지고 있다. 익스포트된 함수가 해당 DLL 파일에서 제공하는 API이다. 본 과정에서는 PE 파일포맷을 분석한 후 익스포트된 함수명을 추출한다. 본 논문에서는 PE 파일포맷을 분석하기 위해 파이썬으로 구현되어 있는 pefile〔12〕클래스를 이용하여 익스포트된 함수명 리스트를 추출하였다.
3. MSDN에서 익스포트된 함수에 대한 프로토타입을 찾는다. 익스포트된 함수를 호출하는 C 소스코드를 생성하기 위해 해당 함수의 프로토타입을 알고 있어야 한다. 함수 프로토타입에는 파라미터의 개수와 파라미터의 데이터 타입을 알 수 있다. 윈도우즈에서 기본적으로 제공하는 API의 프로토타입을 알기 위해 AWAFT는 Microsoft 사에서 제공하는 MSDN을 이용하였다. MSDN은 Microsoft의 자사에서 제공하는 API에 대한 설명을 가지고 있다. MSDN은 웹상에서 제공할 뿐만아니라 시스템 내에 설치도 가능하다. AWAFT는 시스템에 저장되는 MSDN에 대해 프로토타입을 검색하는 룰을 작성하여 익스포트된 함수에 대한 프로토타입을 자동으로 찾았다. MSDN은 하드디스크에 압축된 HTML 파일로 저장이 된다. AWAFT는 검색 룰을 regular expression〔13〕을 이용하여 작성하였기 때문에 HTML 원본파일이 필요하다. 이를 위해 AWAFT는 7-Zip〔14〕을 이용하여 압축된 HTML 설명 파일을 압축 해제하였다. 그 후 HTML 설명 파일에 대해 함수의 프로토타입을 찾았다.
4. 익스포트된 함수에 폴트 데이터를 삽입하는 C 소스 코드를 생성한다. 3의 과정에서 찾은 익스포트된 함수의 프로토타입을 이용하여 해당 함수를 호출하는 C 소스 코드를 생성한다. API Fuzz Testing은 API의 파라미터에 폴트를 삽입하는 방법이기 때문에 함수를 호출할 때 파라미터에 다양한 길이의 폴트 데이터를 삽입한다. 파라미터에 삽입되는 폴트 데이터는 파라미터의 데이터 타입에 따라 다양한 값들이 사용된다. 해당 C 소스코드의 컴파일된 코드를 실행 시키면 Fuzz Testing을 수행하게 되는 것이다.
5. C 소스 코드를 컴파일한다. 생성된 C 소스코드를 컴파일한다. AWAFT는 윈도우즈 대상이기 때문에 Microsoft사에서 제공하는 컴파일러인 Microsoft C/C++ Compiler(cl.exe)을 이용하여 실행 파일을 생성한다.

6. 생성된 실행코드를 실행시킨 후 폴트를 모니터링한다. 생성된 실행 코드를 윈도우즈 환경에서 자동으로 실행 시킨다. AWAFT는 CRASH scale[9] 중 ABORT 폴트에 초점을 맞춘다. ABORT 폴트는 프로세스가 비정상적으로 종료되는 상황이다. 윈도우즈 환경에서는 ABORT 폴트가 Access Violation(AV)으로 나타나며 윈도우즈에서 제공하는 에러창이나 디버거로 발견할 수 있다. ABORT 폴트가 발생하면 소프트웨어의 실행 흐름에 영향을 미치는지 분석한다. AV는 보안 취약점을 유발시킬 수 있는 가능성을 가지고 있기 때문에 수동으로 AV가 소프트웨어에 미치는 영향을 분석해야 한다.

IV. 윈도우즈에서 제공하는 API 안전성 검증실험

본 장에서는 윈도우즈에서 제공하는 API의 안전성 검증을 위해 시스템 폴더(C:\Windows\System32)에 설치된 DLL 파일들에 대해 AWAFT를 적용하였다. 보안 테스팅의 대상이 되는 OS는 Windows XP SP2이다. 시스템 폴더에는 총 1,182개의 DLL 파일들이 존재한다. 본 논문에서는 AWAFT을 위한 자동화 도구를 구현하였으며, 총 2개의 모듈로 이루어져 있다.

- Exported Function Extractor(EFE): EFE는 DLL 파일내의 익스포트된 함수명을 추출한 후 MSDN을 이용하여 해당 함수명에 대한 프로토타입을 검색한다. EFE는 AWAFT의 제1에서 제3의 과정을 자동으로 수행한다. 즉 EFE에서는 보안 테스팅의 대상이 되는 API를 호출할 수 있는 소스코드 작성을 위한 정보를 추출하는 단계이다.
- API Fault Inserter(AFI): AFI는 EFE에서 찾은 함수의 프로토타입을 이용하여 해당 함수를 호출하는 C 소스코드를 생성한다. 소스코드 작성시 함수의 파라미터에 폴트 데이터를 삽입한다. 폴트 데이터는 가변적인 길이의 문자열이나 다양한 타입의 값들이 이용된다. 작성한 소스코드를 컴파일하여 실행파일을 생성한다. 생성된 실행코드들을 자동으로 실행하여 함수에 의해 발생되는 폴트에 대해 디버거를 이용하여 모니터링한다. AFI는 AWAFT 중 제4에서 제6의 과정인 Fuzz Testing을 수행한다.



[그림 2] EFE 동작 메커니즘

[그림 2]는 EFE의 전체 동작 메커니즘을 나타낸다. 처음 과정으로 보안 테스팅을 수행할 DLL 파일 목록을 조사하였다. 본 장에서는 윈도우즈 시스템 폴더에 있는 전체 DLL 파일을 대상으로 하였다. 우선 해당 DLL 파일들에 대해 PE 파일포맷을 분석하여 DLL 파일내에 저장되어 있는 익스포트된 함수명의 리스트를 추출하였다. 이들 함수명만으로 해당 API를 호출할 수 있는 실행코드를 만들 수 없기 때문에 다음 과정으로 EFE는 함수의 프로토타입을 찾는다. 윈도우즈에서 제공하는 API들은 MSDN에 그 설명을 찾을 수 있으며 또한 프로토타입을 찾을 수 있다. MSDN에 설명이 없는 API들을 Undocumented API라 하며, 이들 API는 호출할 수 없다. 따라서 Undocumented API의 경우 특수한 경우를 제외하고는 개발자들이 사용하지 않기 때문에 본 논문의 보안 테스팅 대상에서 제외하였다. MSDN의 자동 검색을 위해 로컬 시스템에 Visual Studio 2005에서 함께 제공하는 MSDN을 설치한 후, 설치된 파일을 검색하였다. MSDN은 압축된 HTML 파일로 저장이 되어 있어 자동으로 찾기 위해선 압축이 풀려져야 하며 본 논문에서는 7-Zip을 이용하였다. MSDN의 압축 파일을 풀게 되면 총 640,054개의 파일들이 생성되었다.

[표 1]은 시스템 폴더 내의 DLL의 총 수, 익스포트된 함수의 총 수, 그리고 보안 테스팅 대상이 되는 함수

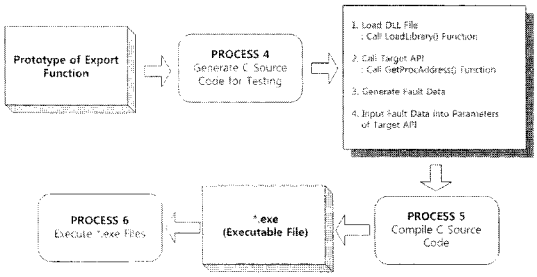
〔표 1〕 윈도우즈 시스템 폴더에 대한 EFE 분석 결과

	파일의 종류		전체비율
시스템 폴더내의 DLL의 총수	PE 파일 포맷 분석이 가능한 파일들	1,084 (92%)	1,182
	PE 파일포맷 분석이 불가능한 파일들	29 (2%)	
	익스포트된 함수가 없는 파일들	69 (6%)	
익스포트된 함수의 총 수	분석 대상이 되는 함수들	30,430 (33%)	92,044
	분석 대상에서 제외된 함수들	61,614 (67%)	
분석 대상이 되는 함수의 총 수	프로토타입을 가지는 함수들	6,117 (20%)	30,430
	프로토타입이 없는 함수들	24,313 (80%)	

의 총 수를 나타낸다. EFE는 시스템 폴더 내 DLL 파일 중에 익스포트된 함수 리스트를 추출하고, 해당 리스트를 기반으로 MSDN에서 프로토타입을 검색한다. Windows XP SP2의 시스템 폴더에는 총 1,182개의 DLL 파일들이 존재한다. 이들 중 정상적인 PE 파일 포맷을 따르지 않은 파일인 98개의 함수를 제외한 총 1,084개의 DLL 파일들이 테스팅의 대상이 된다. 이들 파일들에 대해 익스포트된 함수의 목록을 추출하였다. EFE는 1,084개의 파일들에 대해 92,044개의 익스포트된 함수명을 추출하였다. 이들 함수 중 *DllMain, ServiceMain, DllCanUnloadNow, DllGetClassObject, DllRegisterServer, DllUnregisterServer, None, ?FunctionName(? 다음에 함수이름이 정의됨)* 의 API는 제외하였다. DllMain과 ServiceMain 함수는 프로그램을 시작하는 엔트리 포인터라 제외하였다. DllCanUnloadNow, DllGetClassObject, DllRegisterServer, DllUnregisterServer 함수는 COM과 관련이 있는 API들로 해당 API들을 이용하면 수많은 COM API들을 이용할 수 있다. 마지막인 None과 ?FuncName은 함수명이 정확하지 않아 제외하였다. 이들 함수들을 제외하면 MSDN에서 프로토타입을 검색하기 위한 대상이 되는 함수는 총 30,430개가 된다. MSDN은 다양한 HTML 방식으로 구현되어 있기 때문에 프로토타입을 검색하기 위해선 Regular Expression을 위한 다양한 패턴을 구현해야 한다. 2개의 패턴을 적용한 결과 최종적으로 6,117개의 프로토타입을 검색하였다. 패턴의 수가 많아지면 이 이상의 프로토타입을 찾을 수 있다. 다음은 적용한 2개의 패턴이며, FUNCTIONNAME에 함수명을 삽입하여 검색을 수행한다.

```
1) 〈PRE class="syntax" xml:space="preserver"〉
.*FUNCTIONNAME.*:〈/B〉
2) 〈PRE class=syntax〉〈B〉
.*FUNCTIONNAME.*:〈/B〉
```

결론적으로 본 논문의 보안 테스팅 대상은 시스템 폴더에 있는 MSDN의 검색이 가능한 총 30,430개의 API에 대해 20.1%에 해당하는 6,117개의 API가 된다.



〔그림 3〕 AFI의 동작 메커니즘

〔그림 3〕은 AFI의 전체 동작 메커니즘을 나타낸다. AFI는 EFE에서 추출한 함수 프로토타입을 이용하여 해당 API를 호출하는 C 소스코드를 생성한 다음 컴파일한다. 그 후 생성된 실행파일을 자동 실행하여 API의 이상현상을 검사한다. 본 논문에서는 다음과 같이 폴트 룰(Rule)을 적용하여 실행파일을 생성하였으며, 추후 확장도 가능하다.

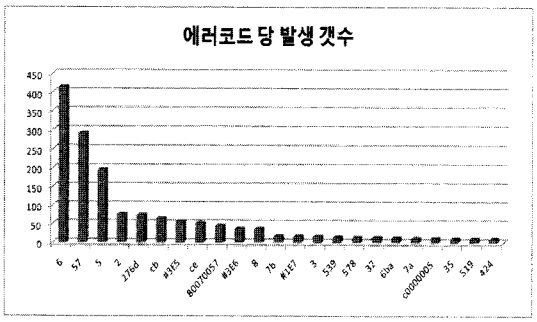
- 하나의 소스코드는 하나의 함수만 호출한다: 2개 이상의 함수를 호출하는 경우 함수 사이의 상관관계를 고려해야 하기 때문에 추가 분석이 요구되어 하나의 함수만 대상으로 하였다.
- 함수의 파라미터에는 2KB의 버퍼 주소가 삽입된다: 파라미터에 삽입되는 값에 따라 생성파일이 생성되어야 하기 때문에 본 논문에서는 하나의 경우만 고려하였다. 예를 들어 GDI32.DLL 내 AbortPath 함수에 대해 폴트를 삽입한 데이터인 char* str를 파라미터로 삽입하는 경우 해당 변수를 AbortPath((HDC)str)로 형변환을 해야 한다. 이렇게 함으로써 긴 문자열을 파라미터에 삽입하였을 때 파라미터의 타입이 수와 관련된 값인 경우 문자열의 주소 자체가 전달되며,

주소가 전달되어야 하는 포인터인 경우 문자열의 주소가 전달된다.

- 〈windows.h〉만 생성되는 소스파일에 INCLUDE 된다: 함수에 따라 적합한 헤더 파일을 삽입하기 위해선 추후 분석이 요구되기 때문에 본 논문에서는 자동화를 위해 보편적으로 사용되는 헤더 파일인 〈windows.h〉의 하나만 삽입하였다.

본 논문에서는 EFE에서 추출한 6,117개의 함수에 대해 C 소스코드를 생성하였다. 위의 룰을 적용하여 생성된 소스파일을 컴파일 시킨 결과 3,179개의 실행파일을 생성하였으며, 나머지는 컴파일 오류를 발생하였다. 룰의 개수가 많아지면 오류를 발생시키는 비율이 낮아질 것이다. 컴파일 오류를 분석한 결과 상당수가 파라미터의 데이터 타입을 더블포인터형이나 void 형을 사용한 경우이거나, 적절한 헤더 파일을 참조하지 못했기 때문이다.

생성된 3,179개의 실행파일을 AFI에서 자동 실행시켰다. 실행결과시 발생되는 에러를 분석하기 위해 익스포트된 함수를 호출한 후 GetLastError() 함수를 삽입하였다. 해당 함수는 프로그램 실행시 가장 최근에 발생한 에러 코드를 반환한다. 이와 함께 ABORT 폴트를 모니터링하기 위해서 Microsoft사에서 제공하는 WinDbg를 이용하였다. 실행파일을 자동 실행한 결과, 총 3,179개의 함수에 대해 1,650개에서 총 81종의 에러코드를 반환하였다. 발생한 에러 중 177개는 ABORT 폴트였다. 177개의 ABORT 폴트 중 10개는 추가 분석 결과 프로그램의 실행흐름을 변경시킬 수 있는 가능성을 내포한 보안상 위험한 폴트였다.



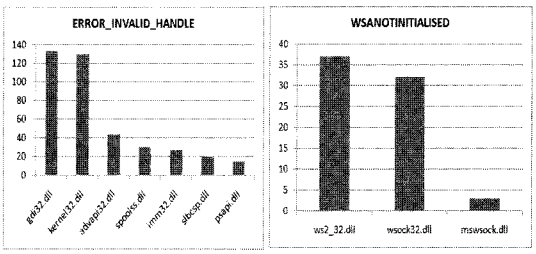
〔그림 4〕 에러코드 당 발생갯수

〔그림 4〕는 각각의 에러코드를 발생시키는 익스포트된 함수의 개수를 보여주며, 에러 발생횟수가 10개 이상의 경우만 나타내었다. X축은 Microsoft사에서 정의한 에러코드를 나타내며, Y축은 해당 에러코드를 발생시키는 함수의 개수를 나타낸다. 〔그림 4〕에서 상위 5개의 에러코드는 〔표 2〕와 같다.

〔표 2〕 실험 중 발생한 상위 5개의 에러코드 설명

에러코드	설 명
0x6	파라미터에 사용되는 핸들이 적절하지 않음
0x57	파라미터로 전달된 값이 적절하지 않음
0x5	접근이 허용되지 않음
0x2	시스템이 지정한 파일을 찾지 못함
0x276d	해당 응용프로그램이 WSAStartup() 함수를 호출하지 않거나, WSAStartup() 함수가 호출 실패함

〔표 2〕에서의 에러코드 중 0x6과 0x276d는 함수 하나만 호출하여 해당 함수를 호출하기 전 필요한 함수를 호출하지 않아 발생하는 에러들이다. 즉 해당 익스포트된 함수는 독자적으로 사용되지 못하는 함수들이다. 함수의 연관관계를 고려하지 않고 하나의 함수만 호출하였기 때문에 폴트 데이터가 함수 내부에 삽입되기 전 반환된 것이다. 0x6은 ERROR_INVALID_HANDLE로 정상적인 핸들을 파라미터로 삽입하지 않았을 때 발생한다. 0x276d는 WSANOTINTIALISED로 네트워크 통신을 하기 위해 초기화를 적절히 수행하지 않아 발생한다. 함수의 연관관계로 인해 발생하는 에러들은 총 1,650개의 에러들 중 29.6%를 차지한다. 〔그림 5〕는 해당하는 2개의 에러코드를 발생시키는 DLL 파일내 함수의 수를 나타낸다. 이들 DLL 파일은 윈도우즈 시스템에서 기본적으로 사용되는 파일들로 이들 파일들에 대해 보안 안전성을 테스팅하기 위해선 추가적으로 함수들간의 호출관계를 고려한 C 소스코드 작성이 필요함을 알 수 있다.



〔그림 5〕 함수의 연관관계를 고려하지 않아 발생하는 에러들

V. 결 론

본 논문에서는 윈도우즈에서 제공하는 DLL 파일 내 API에 대해 안전성을 검증하였다. 이를 위해 윈도우즈 익스포트된 함수의 보안 테스팅을 수행할 수 있는 방법론인 AWAFT를 제안하고, AWAFT를 자동으로 수행할 수 있는 도구를 구현하였다. 본 논문에서는 AWAFT를 윈도우즈 중 많은 사용자층을 보유하고 있는 Windows XP SP2의 시스템 폴더에 적용하여 보안 테스팅을 수행하였다. 실험결과 총 3,179개의 함수에 대해 총 177개의 ABORT 폴트를 찾아냈으며, 이 중 10개는 프로그램의 실행흐름을 변경시킬 수 있는 폴트였다. 따라서 윈도우즈에서 제공하는 API의 경우 윈도우즈용 프로그램을 개발하기 위해 사용되어야 하지만, 이와 함께 주의 깊게 사용되어야 함을 알 수 있다. 이는 윈도우즈에서 제공하는 API의 경우 전적인 신뢰를 바탕으로 소프트웨어를 구현하는 것이 아니라 함수를 호출하기 전 적절히 API의 파라미터 값 검사를 수행해서 소프트웨어의 보안성을 높여야 함을 의미한다.

소프트웨어 보안 테스팅은 보안을 고려한 소프트웨어 개발만큼 중요하다. AWAFT는 윈도우즈에서 사용되는 API에 대해 보안 테스팅을 수행하는 방법으로 Thirty Party에서 구현하는 소프트웨어에서도 적용가능하다. 이들 소프트웨어에서 사용하기 위해 개발된 라이브러리의 경우 API의 프로토타입을 알기 때문에 AWAFT의 함수 프로토타입 검색과정 없이 Fuzz Testing을 수행할 수 있다. 이렇게 함으로써 소프트웨어의 신뢰성을 높일 수 있으며 취약점으로 인해 발생할 수 있는 보안 사고를 사전에 막을 수 있다.

AWAFT를 적용한 실험 결과 1,650개의 에러 중 29.6%가 함수간의 연관관계를 고려하지 않아 발생하였다. 즉 상당수의 익스포트된 함수들은 하나만 독자적으로 사용되지 않고 2개 이상의 함수가 유기적으로 사용된다. 따라서 추후 이들 함수들에 대해 함수의 연관관계를 고려하여 보안 테스팅을 할 수 있는 방법론을 연구할 계획이다.

참 고 문 헌

[1] Top Operating System Share Trend, http://marketshare.hitslink.com/report.aspx?gprid=9

[2] K.R. Van, "Integrating tool into the SDLC," 2007 FIRST Conference, June 2007.

[3] MSDN, Microsoft Developer Network, http://msdn.microsoft.com/ko-kr/default.apx

[4] N.P. Kropp, P.J. Koopman, and D.P. Siewiorek, "Automated robustness testing of off-the-shelf software components," 28th Fault Tolerant Computing Symposium, pp. 230-239, June 1998.

[5] M. Sutton, A. Greene, and P. Amini, Fuzzing: Brute Force Vulnerability Discovery, Addison-Wesly, pp. 21-32, June 2007.

[6] P. Oehlert, "Violating assumptions with fuzzing," IEEE Security & Privacy Magazine, vol. 3, no. 2, pp. 58-62, Mar. 2005.

[7] J.E. Forrester and B.P. Miller, "An empirical study of the robustness of Windows NT applications using random testing," 4th USENIX Windows System Symposium, pp. 59-68 , Aug. 2000.

[8] B.P. Miller, L. Fredrikson, and B. So, "An empirical study of the reliability of Unix utilities," Communication of the ACM, vol. 33, no. 12, pp. 32-44, Dec. 1990.

[9] C.P. Shelton, P. Koopman, and K. Devale, "Robustness testing of the Microsoft Win32 API," International Conference on Dependable Systems and Network, pp. 261-270, June 2000.

[10] A.K. Ghosh and M. Schmid, "An approach to testing COTS software for robustness to operating system exception and errors," International Symposium on Software Reliablity Engineering, pp. 166-174, Nov. 1999.

[11] Portable Executable(PE) File Format, http://en.wikipedia.org/wiki/Portable_Executable

[12] Pefile python class, http://www.dkbza.org/pefile.html

[13] J.E.F. Friedl, Mastering Regular Expression, O'REILLY, pp. 399-432, Jan. 1997.
[14] 7-Zip, http://www.7-Zip.org

〈著者紹介〉

사 진

최 영 한 (YoungHan Choi) 정회원
2002년 2월: 한양대학교 전자전기학과 졸업
2004년 2월: 한국과학기술원 전자전산학과 석사
2004년 2월~현재: ETRI 부설연구소
〈관심분야〉 운영체제, 정보보호, 소프트웨어 테스팅

사 진

김 형 천 (HyoungChun Kim) 정회원
1999년 8월: 고려대학교 전산학과 졸업
2001년 8월: 고려대학교 전산학과 석사
2008년 8월: 고려대학교 정보경영공학 박사 수료
2001년 5월~현재: ETRI 부설연구소 선임연구원
〈관심분야〉 네트워크보안, 소프트웨어 보안 테스팅

사 진

오 형 근 (HyungGeun Oh) 정회원
2000년 2월: 순천향대학교 전산학과 석사
2000년 2월~8월: 한국사이버페이먼트 기술개발부 선임연구원
2000년 8월~현재: ETRI 부설연구소
2008년 9월~현재: ETRI 부설연구소 팀장
〈관심분야〉 소프트웨어 취약점 분석, 악성코드, 네트워크 보안

사 진

이 도 훈 (Do Hoon Lee) 정회원
1989년 2월: 한양대학교 전자계산학과 졸업
1991년 2월: 한양대학교 전자전산학과 석사
1991년 3월~2000년 1월: 국방과학연구소
2000년 2월~현재: ETRI 부설연구소
〈관심분야〉 운영체제, 정보보호, 소프트웨어 테스팅