정 보 보 호 학 회 지
제26권 제1호, 2016. 2

퍼징 기반의 상용 및 공개 소프트웨어에 대한 보안약점 진단 방법 연구

임 기 영*, 강 성 훈**, 김 승 주**

요 약

보안 약점은 소스코드의 공개 여부와는 관계없이 존재하며, 소프트웨어의 취약점으로 이어질 수 있다. 소스코드가 공개된 소프트웨어의 경우 소스 코드 분석을 통해 보안 약점을 제거하지만, 소스코드가 공개되지 않고 바이너리 형태의 실행 파일만 제공되는 소프트웨어의 경우에는 보안 약점을 찾기가 어렵다. 비정상 데이터를 임의로 생성하여 파일 또는 표준입력 형태로 입력하는 퍼징 기법은 위와 같은 소스코드가 공개되지 않은 소프트웨어의 취약점을 찾기 위한 기술이다. 본 논문에서는 소스코드가 없는 상용 및 공개 소프트웨어의 보안 약점을 진단하기 위해 퍼징 기법을 활용하는 방법을 제시하고, 공개된 퍼징 도구 및 프레임워크를 이용하여 설치부터 소프트웨어의 진단 및 발견된 보안 약점을 분석, 제거 등의 퍼징 프로세스를 소개하여 상용 및 공개 소프트웨어 취약점 발견에 도움을 줄 것으로 기대한다.

Ⅰ. 서 론

소프트웨어에는 Microsoft office, Adobe Photoshop과 같이 소스코드를 공개하지 않는 소프트웨어와 Linux, Firefox와 같이 소스코드를 공개하는 소프트웨어가 있다. 보안 약점은 소스코드의 공개 유무와는 관계없이 소프트웨어에 존재할 수 있고 이는 소프트웨어 취약점으로 이어질 수 있다. 소스코드를 공개하는 소프트웨어의 경우 소스코드를 분석하여 보안 약점을 비교적 쉽게 찾을 수 있지만, 소스코드가 공개되지 않고 바이너리 형태의 실행 파일만 제공되는 소프트웨어의 경우에는 전자에 비해 보안 약점을 찾기가 어렵다. 따라서 소스코드가 공개되지 않는 소프트웨어에서도 보안 약점을 찾고 진단할 수 있는 방법론에 대한 연구가 필요하다. 본 논문에서는 소스코드가 없는 상용 및 공개 소프트웨어의 보안 약점을 진단하기 위해 퍼징 기법을 활용하는 방안을 제시하고자 한다. 퍼징은 소프트웨어의 입력 값을 특정 규칙에 따라 자동으로 변형시켜 소프트웨어의 오작동 여부를 탐지하고 그 결과로부터 취약점을 발견한다[1]. 본 논문에서는 퍼징 기법 중 내부 디자인이나 코드에 대한 기반지식 없이 수행하는 블랙박스 테스팅인 덤 퍼징(Dumb Fuzzing)기법을 사용하는 도구와 이 도구를 이용한 보안 약점 분석, 제거 등의 퍼징 프로세스를 소개 하고자 한다.

논문의 구성은 다음과 같다. 2장에서는 관련연구를 통해 퍼징 기법과 이를 사용한 도구에 대해 소개를, 3장에서는 퍼저를 통한 보안 약점 분석과 같은 퍼징 프로세스를, 4장에서는 3장에서 소개한 프로세스를 이용하여 특정 소프트웨어에 대한 퍼징을 수행한 결과를 소개한다.

Ⅱ. 관련연구

2.1. 퍼징

퍼징은 소프트웨어 테스팅 방법 중 하나로 비정상 데이터를 파일 또는 표준 입출력 형태로 대상 소프트웨어에 입력하는 방법으로써 퍼징 기법을 활용하여 소프트웨어 진단 및 분석을 할 경우 네트워크 분석과 디버깅도 수행할 수 있다[2,3]. 퍼징은 대상의 내부 디자인이나 코드에 대한 기반지식 유무에 따라 덤(dumb), 스마트(smart)로 나누어지며, 데이터 처리 방식에 따라 생

* 고려대학교 정보보호대학원 정보보호학과 (morinori89@gmail.com)
** 고려대학교 정보보호대학원 정보보호학과 ({jeremekang, skim71}@korea.ac.kr)

성, 변이로 분류할 수 있다[4]. 각 분류에 대한 설명은 다음과 같다.

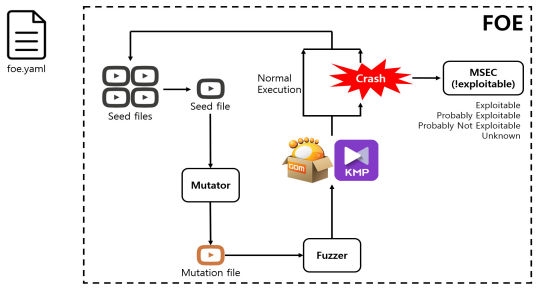
- 덤 퍼징 : 퍼징을 하고자 하는 대상에 어떠한 기반 지식 없이 입력을 주어 테스트를 수행하는 기법이다. 입출력 포맷을 분석하지 않은채로 수행하는 방법으로, 애플리케이션에 임의의 데이터 입력 또는 파일 내 데이터를 무작위 변조해 불러오는 방식으로 퍼징을 수행한다.
- 스마트 퍼징 : 퍼징을 하고자 하는 대상의 소스코드와 구조를 파악하고 있거나, 대상 어플리케이션의입력 방식 또는 불러오는 파일의 파일구조를 파악하고 있을 때 사용하는 기법이다.
- 생성 : 테스트를 위한 입력 데이터를 생성하여 처리하는 기법이다. 파일 포맷과 확장자를 정확히 알고 있는 경우, 해당 파일 포맷에 맞춰 퍼징을 위한 테스트 파일을 생성하는 방법이다.
- 변이 : 퍼징 대상에 사용되던 실제 데이터 또는 파일 샘플을 구해, 그 일부를 무작위 변형시켜 테스트하는 기법을 말한다. 예로, FTP 로그인 아이디 또는 비밀번호를 전송하는 패킷에, 아이디나 비밀번호를 실시간으로 변경하여 테스트하는 것이 이에 해당된다.

퍼징을 효율적으로 수행하기 위해서는 입출력 데이터 또는 대상 애플리케이션이 불러오는 파일 구조를 정확히 알고 있어야 하거나, 자동화된 도구를 이용해야 한다.

일반적으로 알려진 퍼징의 장점으로 입력파일 생성 규칙만 명시해준다면, 테스팅을 자동으로 진행하는 자동화와 입력값을 무작위로 대입하기 때문에 정확하진 않지만 모든 부분에 대해서 테스팅을 수행할수 있는 높은 코드 커버리지를 들 수 있다. 단점으로는 무작위로 입력데이터를 설정하기 때문에, 소프트웨어의 원하는 기능이나 함수의 취약점을 정확히 찾아내기 어렵다는 점과, 무작위로 입력데이터를 설정하기 때문에, 공격 가능한 취약점 식별을 보장할 수 없는 불확실성 및 무작위성이 있다.

2.2. FOE

FOE(Failure Observation Engine) FOE는 Carnegie Mellon University의 Software Engineering Institute에



〔그림 1〕 FOE 내부 구조도

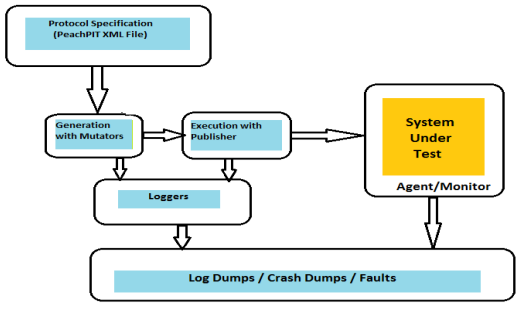
서 개발한 윈도우즈 플랫폼 기반의 퍼징 도구이다[5]. FOE는 파이썬으로 작성되었으며, 파일포맷 기반의 변이 방식으로 동작한다. 윈도우즈 환경에서 실행되는 애플리케이션을 대상으로 취약점 분석을 수행할 수 있다. FOE는 사용자가 원하는 동작방식을 정의한 yaml파일을 기반으로 동작한다.

FOE는 커맨드라인을 이용하여 입력 파일을 실행시킬 수 있는 애플리케이션에 대하여 사용자가 지정한 입력 시드파일들의 데이터 값을 랜덤하거 변이하여 대상 어플리케이션에 전달하고 실행결과로부터 취약점을 찾는 방식의 퍼징을 수행할 수 있다.

2.3. PEACH

peach는 변이를 기반으로 퍼징을 수행하는 도구로써, 피치 핏(pit - XLM 포맷으로 정의된 데이터 모델) 파일을 요구한다. 피치 핏 파일은 피치 플랫폼에서 퍼징을 수행하기 위한 모든 정보를 포함하고 있으며, 피치를 이용하여 퍼징을 수행하기 위해서는 반드시 피치 핏 파일 생성하는 작업을 선행해야 한다.

피치 핏 파일은 peach에서 퍼징을 수행하기 위한 정보가 담겨져 있으며 애플리케이션 퍼징을 위한 파일 구

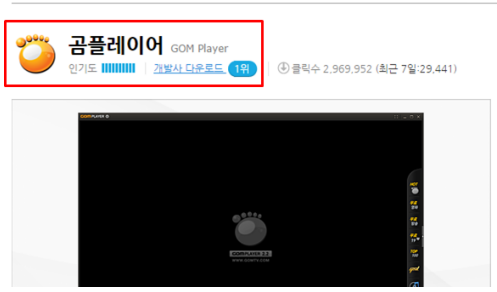


〔그림 2〕 peach 내부구조도

조와 실행경로, 네트워크 퍼징을 위한 서버 주소 및 프로토콜, 패킷 정보 등을 입력할 수 있다. 이를 기반으로 입력 파일을 파싱하고 사용자가 지정한 특정 영역을 변조함으로써 퍼징을 수행한다. 또한 변이 옵션을 설정하여 보안 약점이 발생할 가능성이 높은 영역에 대해서만 데이터 변조를 수행할 수 있다. 따라서 데이터 모델인 피치 핏 파일을 생성하는 과정이 매우 중요하다.

Ⅲ. 퍼저를 통한 보안 약점 분석

본 장에서는 FOE 퍼저의 yaml 파일 작성 방법, 예외처리 리스트 등 구성요소와 분석대상 소프트웨어를 대상으로 퍼징을 수행하는 과정을 소개하고자 한다. 분석대상은 미디어플레이어 중 [그림 3]과 같이 네이버 소프트웨어 다운로드 순위를 기준으로 개발사 다운로드 1위인 곰플레이어로 선정한다.

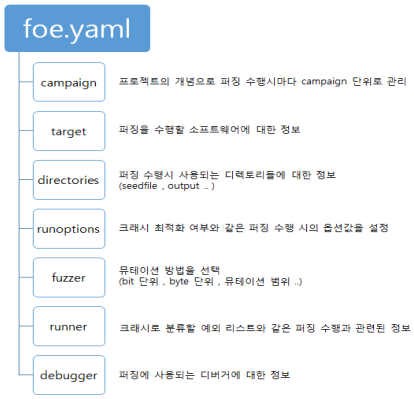


〔그림 3〕 개발사 다운로드 1위 소프트웨어

3.1. FOE를 이용한 퍼징 프로세스

FOE는 2절의 2에서 설명한 구조를 기반으로 동작하며, 사용자는 foe.yaml 파일을 수정함으로써 원하는 옵션값을 설정하여 퍼징을 수행할 수 있다. foe.yaml은 [그림 4]와 같은 구조를 가지고 있으며, 이제 각 영역 내 필드들의 의미를 살펴본다.

campaign 필드는 프로젝트의 개념으로 FOE는 각 퍼징 수행시마다 campaign 단위로 관리하게 된다. 영역 내 속성값인 id는 사용자가 식별할 campaign id를 기술한 부분으로 프로젝트 이름처럼 사용된다. keep_heisenbugs는 디버거를 통해 실행되지 않을 때, hook에 의해 크래시가 검출될 경우, 입력파일의 크래시 유지 여부를 결정하는 속성이다. use_button_clicker



〔그림 4〕 foe.yaml의 구조

```
#################################################################
campaign:
    id: GOMPlayer_v0.1
    keep_heisenbugs: False
    use_buttonclicker: False

#################################################################
```

〔그림 5〕 foe.yaml 내 campaign 영역

```
#################################################################
target:
    program: C:\Program Files (x86)\GRETECH\GomPlayer\GOM.exe
    cmdline_template: $PROGRAM $SEEDFILE NUL
    # NOTE: FOE uses python's shlex.split() method to parse ommand
    # line template after substituting in the program and seedfile values.
    # For this reason, it is required that if any other items in the
    # cmdline_template involve windows paths, you need either use
    # forward slashes or double quotes. For example:
    # cmdline_template: $PROGRAM -in $SEEDFILE -out c:/some/path/to/file
    # cmdline_template: $PROGRAM -in $SEEDFILE -out "c:\some path\to file"
#################################################################
```

〔그림 6〕 foe.yaml 내 target 영역

는 버튼 클릭이 필요한 프로그램인지 설정할 수 있다.

target 영역은 퍼징을 수행할 소프트웨어에 대한 정보가 담겨있다. program 속성은 퍼징을 수행할 대상 애플리케이션의 위치 및 실행경로를 설정하고, cmdline_template은 퍼징 수행시 커맨드라인을 이용하여 대상 어플리케이션에 입력파일을 전달하기 위해, 커맨드라인의 형식을 지정하는 필드다.

directories 영역은 퍼징 수행시 사용되는 seedfile이나 퍼징 수행 결과물이 저장될 디렉토리들에 대한 정보가 담겨있다. seedfile_dir은 대상 애플리케이션에 변이

```
#################################################################
directories:
    seedfile_dir: seedfiles
    working_dir:  C:\FOE2\fuzzdir
    results_dir:  results

#################################################################
```

〔그림 7〕 foe.yaml내 directories 영역

하여 전달할 입력파일들이 담겨있는 디렉토리 경로를 설정하고, working_dir은 시드파일 생성과 같이 동작에 필요한 파일들을 위한 디렉토리를 설정한다. result_dir은 발생한 크래시나 로그와 같은 퍼징 결과값이 저장될 디렉토리 경로를 설정한다.

```
################################################################
fuzzer:
    fuzzer: bytemut
    # fuzzer: swap
    # fuzzer: copy
    # fuzzer: bitmut
    # fuzzer: wave
    # fuzzer: drop
    # fuzzer: insert
    # fuzzer: truncate
    # fuzzer: verify
    # range_list:
    #     - [0x0000, 0x0400]
    #     - [0x1000, 0x100F]

################################################################
```

〔그림 8〕 foe.yaml내 fuzzer 영역

fuzzer 영역은 변이 범위를 바이트 또는 비트 단위로 설정해 줄 수 있다. fuzzer 속성은 파일포맷의 변조 방법을 지정하는 필드로, 복수로 지정 가능하다. range_list 속성은 변조될 바이트 또는 비트 값의 범위를 지정할 수 있다.

```
################################################################
runner:
    # runner: winrun
    runner: null
    hideoutput: False
    runtimeout: 2
    exceptions:
        - 0x80000002    # EXCEPTION_DATATYPE_MISALIGNMENT
        - 0xC0000005    # STATUS_ACCESS_VIOLATION
        - 0xC000001D    # STATUS_ILLEGAL_INSTRUCTION
        - 0xC0000025    # EXCEPTION_NONCONTINUABLE_EXCEPTION
        - 0xC0000026    # EXCEPTION_INVALID_DISPOSITION
        - 0xC000008C    # EXCEPTION_ARRAY_BOUNDS_EXCEEDED
        - 0xC000008E    # EXCEPTION_FLT_DIVIDE_BY_ZERO
        - 0xC0000090    # EXCEPTION_FLT_INVALID_OPERATION
        - 0xC0000091    # EXCEPTION_FLT_OVERFLOW
        - 0xC0000092    # EXCEPTION_FLT_STACK_CHECK
        - 0xC0000093    # EXCEPTION_FLT_UNDERFLOW
        - 0xC0000094    # STATUS_INTEGER_DIVIDE_BY_ZERO
        - 0xC0000095    # EXCEPTION_INT_OVERFLOW
        - 0xC0000096    # STATUS_PRIVILEGED_INSTRUCTION
        - 0xC00000FD    # STATUS_STACK_OVERFLOW

################################################################
```

〔그림 9〕 foe.yaml내 runner 영역

runner영역은 크래시로 분류할 예외 리스트와 같은 퍼징수행 관련된 정보를 설정해 줄 수 있다. runner 속성은 예외를 탐지할 도구를 지정할 수 있으며, winrun으로 지정할 경우 FOE에서 제공하는 hook.dll을 이용하여 예외를 탐지하고, null일 경우 기본 디버거를 이용하여 예외를 탐지한다. hideoutput 속성은 대상 애플리케이션의 표준출력(stdout)을 숨길지 여부에 대해 결정한다. runtimeout 속성은 대상 애플리케이션이 최대로 실행할 수 있는 초(sec)를 지정하며, 대상 애플리케이션이 멈추어서 퍼징이 수행되지 않는 것을 방지한다. exception 속성은 퍼져에서 오류 및 크래시로 처리할 예외들의 종류로, FOE에서는 이러한 예외들이 발생할 경우 크래시로 간주하고 결과물을 생성한다.

```
################################################################
debugger:
    debugger: msec
    runtimeout: 5
    debugheap: False
    max_handled_exceptions: 6
################################################################
```

〔그림 10〕 foe.yaml내 debugger 영역

debugger 영역은 퍼징에 사용되는 디버거에 대한 정보가 담겨있다. debugger 속성은 퍼징 모니터링 및 크래시 분석에 사용할 디버거를 지정하고, runtimeout은 대상 애플리케이션이 디버거에 의해 실행되었을 때 최대로 실행할 수 있는 초(sec)를 지정하는 속성으로, 대상 애플리케이션이 멈추어서 퍼징이 수행되지 않는 것을 방지한다. debugheap은 windbg의 gflags와 같은 역할을 하며 메모리 접근 에러를 유연하게 탐지하는 속성이다. max_handled_exceptions 속성은 몇 번의 예외까지 무시하고 프로그램을 계속 실행할지 명시하는 속성이다.

퍼징 대상인 gomplayer에 대한 퍼징을 수행하기 위해 일부 영역의 데이터를 수정하여 foe.yaml을 작성후 퍼징을 수행한다.

Ⅳ. 퍼저를 통한 보안 약점 분석

4장에서는 3장에서 FOE로 퍼징을 수행한 후 발견된 크래시들에 대해 분석을 하는 방법에 대해 알아본다. 크래시 발생 시의 명령어(instruction), 동작 등을 기반으로 공격 가능 여부를 판단하는 !exploitable을 이용한 방법과 디버거를 이용하여 크래시를 상세히 분석하는 방법에 대해 알아본다.

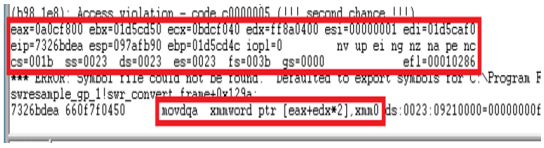
4.1. !exploitable (bang-exploitable)

!exploitable은 MSEC(Microsoft Security Engineering Center)에서 제작한 모듈로, 윈도우즈용 디버거인 windbg의 확장 모듈로써 제공된다[6,7]. 소프트웨어에서 발생한 크래시 덤프의 분석을 자동으로 수행하고, 해

당 크래시의 위험정도를 판단하는 기능을 가진다. 위험정도, 공격 가능성에 따라 Exploitable, Probably Exploitable, Probably Not Exploitable, Unknown으로 분류한다.

메모리 복사 시, 메모리 쓰기 시에 오류가 났을 경우 !exploitable은 공격자가 공격에 관한 데이터를 모두 제어 가능하다고 가정한다. 여기서 말하는 공격에 관한 데이터란 메모리 복사 및 쓰기 시에 목적지 메모리 주소(데이터를 쓸 주소), 출발지 메모리 주소(데이터를 읽을 주소), 길이(쓰거나 읽을 길이)를 말한다. 이러한 가정을 하고 발생한 크래시를 분석하고 위험도를 판단하기 때문에 전적으로 신뢰해서는 안된다.

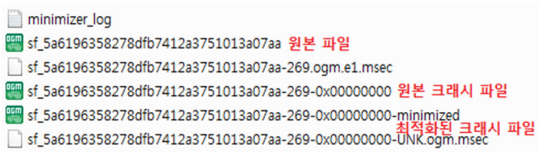
또한 !exploitable은 Microsoft에서 Windows Vista 운영체제에 대해 3억 5천만 개의 퍼징 테스트를 수행한 뒤 발견된 크래시 덤프들을 발생 유형별로 분류하고, 위험도를 판별한 결과 데이터들을 기반으로 분석정책 및 기반 데이터를 만들었다. 이러한 분석정책과 기반데이터들만을 가지고 새로 발생한 크래시의 위험정도를 판단하기 때문에 !exploitable의 분석 및 위험도 판달 결과를 전적으로 신뢰해서는 안 되며, 상세한 분석을 위한 전문가가 필요하다. 이러한 전문가들은 !exploitable의 결과를 참고하여 크래시를 상세 분석하여 실제 exploitable(공격가능) 여부를 판단해야 한다.



〔그림 11〕 곰플레이어에서 발견된 Exploitable Crash

[그림 11]은 실제 곰플레이어에서 퍼징을 수행하여 발견한 크래시이다. !exploitable의 크래시 분석결과를 살펴보면, 접근위반(access violation)예외가 발생했으며 허가되지 않은 사용자 영역 메모리에 쓰기 작업을 수행해 크래시가 발생한 것을 확인할 수 있었다.

크래시 발생 직후의 명령어는 'movdqa xmmword ptr[eax+edx*2], xmm0'이며, !exploitable은 이 명령어에서 메모리 데이터 이동 시에 목적지 주소, 출발지 주소, 길이를 공격자가 마음대로 제어가 가능하다고 가정한다. 즉 메모리 데이터 이동 시에 목적지 주소가 되는 [eax+edx*2] 값을 공격자가 제어 가능하여 원하는 메모리 영역에 접근 가능하다. 또한 출발지 주소가 되는 xmm0 레지스터의 값을 공격자가 제어 가능하여 원하는 값을 이동시킬 수 있다고 가정하고 해당 크래시의 위험도를 판단하고 분류한다.

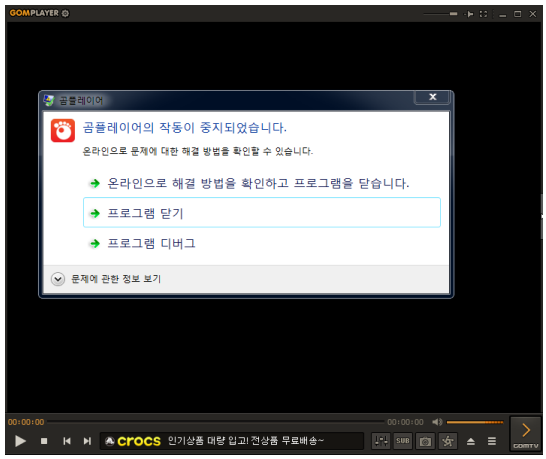


〔그림 12〕 FOE를 이용해 발견된 크래시 결과

[그림 12]는 FOE 퍼저를 이용하여 곰플레이어에서 퍼징을 수행하고 발견된 ogm 확장자에 대한 크래시 결과물이다. 해당 크래시에 대한 간략한 로그와 원본파일, 원본 크래시파일 그리고 최적화된 크래시파일과 각 크래시파일에 대한 로그파일이 있다. 최적화된 크래시 파일은 파일 포맷 내에서 변이된 부분 중에서 크래시에 직접적으로 영향을 끼치는 부분만 남긴 것으로, 이것을 이용해 분석을 시작하면 된다.

발생된 크래시를 기본적으로 실행할 경우, [그림 13]과 같이 크래시가 발생하고 프로그램이 종료된다. WinDBG를 Just In Time Debugger로 설정하여 크래시 및 예외 발생 시 자동적으로 디버거에 연결되게 설정하여 분석을 시작한다.

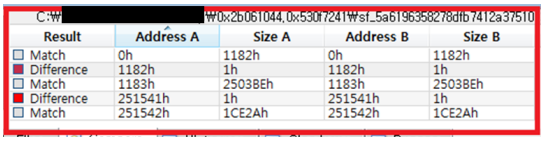
[그림 14]와 같이 크래시파일은 원본파일과 2바이트 크기의 값이 다르다. 파일 포맷 내의 00x1182 영역의 1바이트와 0x251541영역의 1바이트 값이 변경되었고, 변경된 이 값들로 인해 크래시가 발생하였다. 0x1182영



〔그림 13〕 기본적으로 크래시 파일을 입력할 경우

역의 01이 81로 변경되었고, 0x251541영역의 FF가 7F로 변경되었다. 해당 크래시는 곰플레이어 내 MediaSource.ax 모듈 내에 18dd 오프셋 지점에 있는 mov al, byte ptr[ebx] 명령어에서 크래시가 발생했다. 발생 시점의 레지스터 값을 살펴보면, eax가 0x0012e06c로 스택 메모리 주소를 담고 있고, ebx의 값은 0x0이다. 명령어 자체는 ebx가 담고 있는 주소를 참조해 값을 읽어와 al 레지스터에 담는 것인데, ebx가 0x0을 가리키고 있어 크래시가 발생한 것으로 보인다.

크래시가 발생한 명령어 지점 다음의 명령어들은 [그림 15]와 같다. [ebx]의 값을 al 레지스터로 옮기고 결과적으로 [ebp-15h] 지점에 다시 이동한다. 이 경우 공격자가 ebx의 값과 ebp값을 원하는 대로 조작하고, 원하는 값을 이동할 수 있다면, 충분히 공격이 가능한 취약점이라고 볼 수 있지만 ebp가 가리키는 값을 반복문으로 연속해서 덮어쓸 수 있는 구조가 아니고, 한 바이트만 변경할 수 있기 때문에 실제적으로는 공격하기가 어려운 크래시이다.


C:W[redacted]W0x2b061044,0x530f7241Wsf_5a6196358278dfb7412a37510

Result	Address A	Size A	Address B	Size B
Match	0h	1182h	0h	1182h
Difference	1182h	1h	1182h	1h
Match	1183h	25038Eh	1183h	25038Eh
Difference	251541h	1h	251541h	1h
Match	251542h	1CE2Ah	251542h	1CE2Ah

〔그림 14〕 크래시 파일과 원본파일 비교

```
MediaSource+0x18dd:
089018dd 8a03            mov     al,byte ptr [ebx]
089018df 43              inc     ebx
089018e0 c745d401000000  mov     dword ptr [ebp-2Ch],1
089018e7 8845eb          mov     byte ptr [ebp-15h],al
089018ea 84c0            test    al,al
089018ec 0f84fc000000    je      MediaSource+0x19ee (089019ee)
089018f2 33ff            xor     edi,edi
089018f4 8a0b            mov     cl,byte ptr [ebx]
```

〔그림 15〕 크래시 발생 구간의 명령어들

V. 결 론

소프트웨어의 취약점을 점검하고, 발견된 취약점을 제거하기 위한 방안으로 동적 분석 및 퍼징 기법이 널리 사용되고 있다. 이러한 방법들을 통해 실제적으로 많은 취약점들이 발굴되고, 제조사에 전달되어 패치되어 안전한 소프트웨어 환경을 구축하고 있다. 또한, 시큐어 코딩이 강화되고, 일반적인 취약점들이 많이 패치되고 있다. 따라서, 기존의 퍼징 도구들만을 이용해서는 쉽게 취약점을 발견하기가 어려워졌다. 기존의 퍼징 도구들은 파일 포맷 내의 데이터를 랜덤하게 바꾸는 Dumb 퍼징 방식과 파일 포맷 구조체와 같은 내부 데이터를 활용하는 Smart 퍼징 방식을 주로 사용했기 때문이다. 이에 따라 최근에는 Binary Instrumentation을 이용한 취약점 점검 기법이 많이 활용되고 있다. 단순한 변이를 이용한 퍼징 기술이 아닌, Symbolic Execution(기호 실행)[8], Taint Analysis(오염 분석)[9], Binary Instrumentation(바이너리 코드 주입)[11]과 같은 기법을 활용한 퍼징 기법 및 도구가 널리 제작되고 사용되고 있다. 최근에 Blackhat Europe 2015에서 화제가 되었던 Android StageFright 취약점[12]을 발굴하는데 사용된 AFL(American Fuzzy Lop) 퍼징도구 역시 Binary Instrumentation을 활용한 퍼징도구이다. 소프트웨어 취약점 점검을 자동화하고 정확하게 수행하기 위한 노력들이 이루어지고 있는 추세이다. 이러한 추세에 따라, 소프트웨어에서 발생할 수 있는 보안약점 및 취약점의 기준을 명확하게 하여 이러한 도구들이 유용하게 활용될 수 있도록 하는 것이 필요하다.

참 고 문 헌

[1] 이재서, 김종명, 김수용, 윤영태, 김용민, 노봉남 “소프트웨어 취약성 평가를 위한 길이기반 파일 퍼징 테스트 슈트 축약 알고리즘”, *정보보호학회논문지*, 제23권, 제2호, pp. 232-242, 2013.

[2] DeMott,J. “The Evolving Art of Fuzzing”, DEFCON 14, 2006

[3] Pedram Amini, “Blackhat Fuzzing Frameworks“, BlackHat USA, 2007.

[4] 김동진, 조성제, “멀티미디어 플레이어에 대한 퍼징기반 취약점 분석”, *정보과학회논문지*, 제17권, 제2호 pp.98-107, 2011

[5] FOE, https://www.cert.org/vulnerability-analysis/tools/foe.cfm?

[6] swiat, “The History of the !exploitable Crash Analyzer”, http://blogs.technet.com/b/srd/archive/2009/04/08/the-history-of-the-exploitable-crash-analyzer.aspx

[7] 노명선, 노봉남, 나종배, 정광운, 류재철, “MS 크래시 분석도구에 관한 연구”, *정보처리학회논문지*, 제2권, 제9호, pp. 399-404, 2013

[8] James C.King, “Symbolic Execution and

Program Testing", Communications of the ACM, Volume 19, Issue 7, pp. 385-394, 1976

[9] J. Newsome, D. Song, "DynamicTaintAnalysis Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software", NDSS 2005, 2005.

[10] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, Dawn Song, "DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation", NDSS 2011, 2011.

[11] Nicholas Nethercote, "Dynamic Binary Analysis and Instrumentation", *University of Cambridge Technical Report*, Number 606, 2004.

[12] Alexandru Blanda, "Fuzzing Android: a recipe for uncovering vulnerabilities inside system components in Android", BlackHat EU 2015, 2015

〈저 자 소 개〉

임 기 영 (Ki-young Lim)
2012년 2월 : 한국기술교육대학교 인터넷미디어공학부 정보보호 공학 졸업
2016 2월~현재 : 고려대학교 정보보호대학원 정보보호학과 석사과정
관심분야 : 시스템 보안

강 성 훈 (Seong-hoon Kang)
2001년~2010년 : 서원대학교 컴퓨터공학과 졸업
2010년~2013년 : 고려대학교 정보보호학과 석사 졸업
2013년~현재 : 고려대학교 정보보호학과 박사 수료
관심분야 : 정보보호제품 보안성 평가 및 인증

김 승 주 (Seung-joo Kim)
종신회원
1994년~1999년 : 성균관대학교 정보공학과(학사, 석사, 박사)
1998년 12월~2004년 2월 : KISA 팀장
2002년~현재 : 한국정보통신기술협회(TTA) IT 국제표준화 전문가
2004년 3월~2011년 2월 : 성균관대학교 정보통신공학부 조교수, 부교수
2011년 3월~현재 : 고려대학교 사이버국방학과/정보보호대학원 정교수
2004년~현재 : 한국정보보호학회 이사
2005년~2006년 : 교육인적자원부 유해정보 차단 자문위원
2007년 : 국가정보원장 국가사이버안전업무 유공자 표창
2007년~2009년 : 전자 정부 서비스 보안 위원회 사이버 침해사고대응 실무위원회 위원
2010년 : 방송통신위원회 정보통신망 침해사고 민관합동조사단 위원
2012년 3월~2012년 6월 : 선관위 디도스 특별검사팀 자문위원
2013년 4월~2013년 12월 : IT보안인증사무국 자문위원
2013년 9월~현재 : 중앙선거관리위원회 자문위원
2014년 3월~현재 : 헌법재판소 자문위원
관심분야 : 보안공학, 암호이론, 정보보증, 정보보호제품 보안성 평가, Usable Security