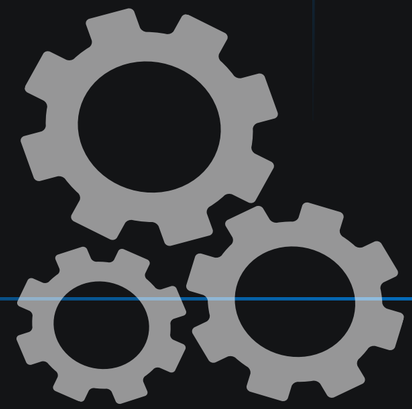
10109이민호 , 10105박성찬

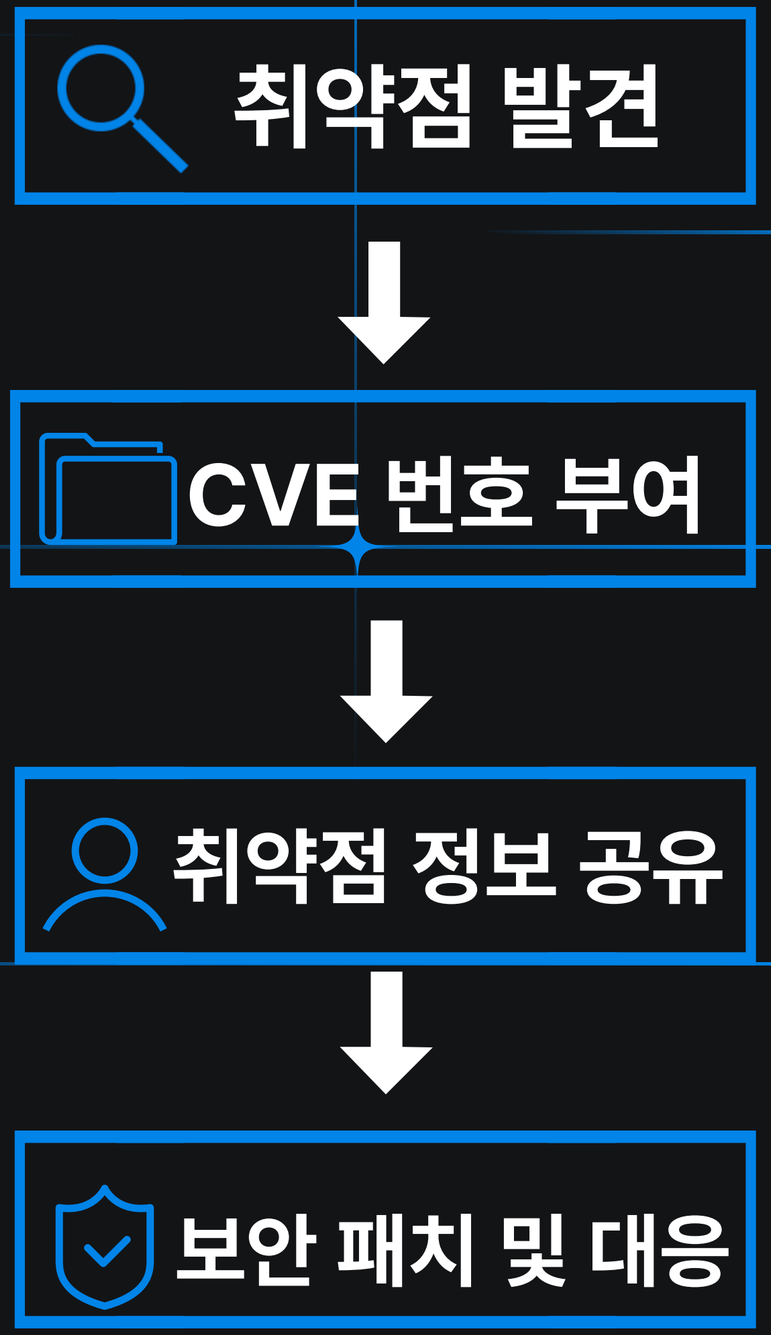
CVE-2009-1329(BOF)
취약점 분석



목차

CVE이란? :

CVE(Common Vulnerabilities and Exposures)는 전 세계에서 발견된 보안 취약점에 부여하는 고유 식별 번호입니다.



버퍼 오버플로우(Buffer OverFlow)

정의

프로그램이 데이터를 저장하기 위해 할당한 메모리 공간(버퍼)보다 더 많은 양의 데이터를 입력받아, 메모리의 인접한 영역을 침범하고 덮어쓰게 만드는 현상이나 취약점

원인

버퍼 오버플로우가 발생하는 근본적인 이유는 프로그램이 메모리 크기(경계)를 확인하지 않고 데이터를 입력받기 때문입니다.

위험성

공격자가 메모리 값을 조작하여 실행 흐름을 변경할 수 있고
그리고 임의 코드 실행(RCE)이 가능해져 시스템이 장악될 수 있고
관리자 권한 탈취와 같은 권한 상승 공격으로 이어질 수 있다

동작 과정

사용자가 버퍼 크기보다 많은 데이터를 입력하면
데이터가 할당된 메모리 경계를 넘어 다른 영역까지 덮어쓴다
저장된 반환 주소(Return Address)까지 영향을 준다

CVE-2009-1329
취약점 발생 원인

입력값 길이 검증 부족

프로그램이 .m3u 파일의 데이터의 입력 길이를 제대로 확인 안함
긴 문자열이 입력될 수 있음(버퍼 크기 초과)

버퍼 오버플로우 발생

제한된 크기의 버퍼에 과도한 데이터가 저장되면서 버퍼를 초과한 데이터가 인접한 메모리 영역을 덮어씀

프로그램 흐름 변경 가능

- 덮어쓴 데이터로 인해 프로그램의 실행 흐름이 변경될 수 있다
- 공격자는 자신이 원하는 코드가 실행되도록 유도할 수 있음

CVE-2009-1329
영향 및 위험성

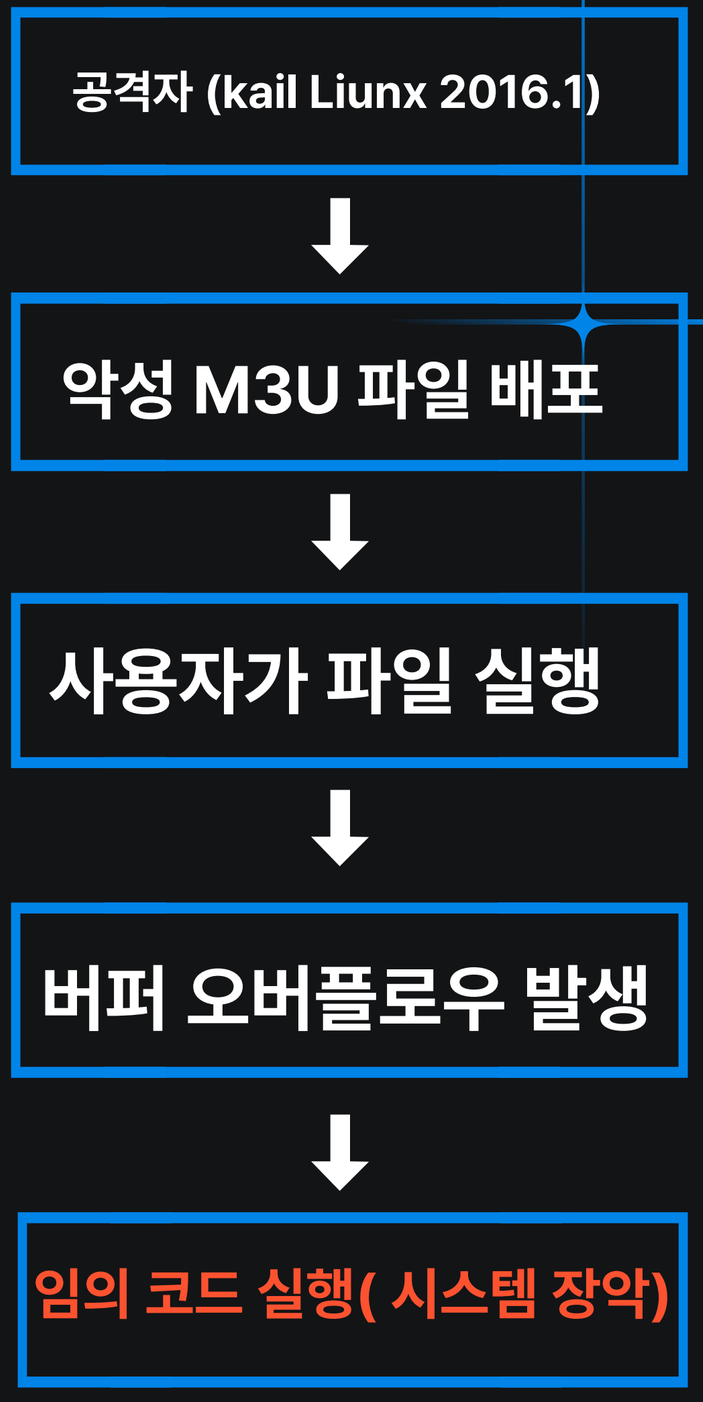
● 임의 코드 실행 (RCE)
- 공격자가 원하는 코드를 피해자 시스템에서 실행할 수 있음

● 악성코드 감염
- 백도어, 랜섬웨어 등 악성 프로그램 설치 가능

● 시스템 권한 획득
- 프로그램 권한 범위 내에서 시스템 제어 가능

● 정보 유출
- 저장된 파일이나 개인정보가 노출될 수 있음

● 시스템 안정성 저하
- 프로그램 비정상 종료(Crash) 또는 오작동 발생



CVE-2009-1329
취약점 분석

버퍼 오버플로우 확인

```
# example.txt 파일 생성
f = open("example.txt", "w"):
text = "A" * 30000
f.write(text)
```

- "A"를 30,000번 반복한 문자열을 만든다.
- 프로그램이 처리하기 어려울 정도로 매우 긴 입력을 만든다
- 긴 입력으로 프로그램이 비정상 종료(Crash)될 수 있다.

프로그램이 갑자기 종료됨
디버거에서 EIP = 41414141 확인

왜 EIP가 41414141일까?

- **문자 'A'의 ASCII 값은 0×41**
- **따라서 AAAA → 41 41 41 41**
- **입력한 데이터가 EIP까지 덮어썼다는 의미이다.**

CVE-2009-1329
취약점 분석

Offset 찾기

Offset이란?

- 입력 시작부터 EIP까지의 거리(Byte)를 의미한다.
- 이 거리를 알아야 EIP를 정확하게 제어할 수 있다.

왜 Offset을 찾을까?

처음에는 "A"만 계속 입력한다. AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

하지만 이렇게 하면 EIP가 어디에서 덮어쓰였는지 알 수 없다.

CVE-2009-1329
취약점 분석

Offset 찾기 (Pattern 활용)

- Pattern을 사용하는 이유

- **Pattern 예시**

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3

(중복되지 않는 문자열)

CVE-2009-1329
취약점 분석

리틀엔디안(Little Endian) ESP 주소

왜 리틀엔디안을 사용할까?

- x86(Windows) 시스템은 Little Endian 방식을 사용한다.
- 메모리에 주소를 저장할 때 바이트 순서를 거꾸로 저장한다.
- 따라서 원하는 주소를 입력하려면 역순으로 입력해야 한다.

예시: 원하는 주소: 0×12345678

메모리에 저장되는 순서: 78 56 34 12

CVE-2009-1329
취약점 분석

ESP(Stack Pointer) 확인

ESP란?

- ESP(Stack Pointer)는 현재 스택(Stack)의 위치를 가리키는 레지스터이다
- 프로그램 실행 중 스택의 최상단 주소를 저장한다.

왜 중요할까?

- 버퍼 오버플로우에서 EIP 뒤에는 ESP가 위치한다.
- ESP가 가리키는 위치에는 사용자가 입력한 데이터(Payload)가 저장된다.
- 따라서 실행 흐름을 ESP로 이동시키면 입력한 데이터를 실행할 수 있다.

CVE-2009-1329
취약점 분석

NULL Byte와 JMP ESP

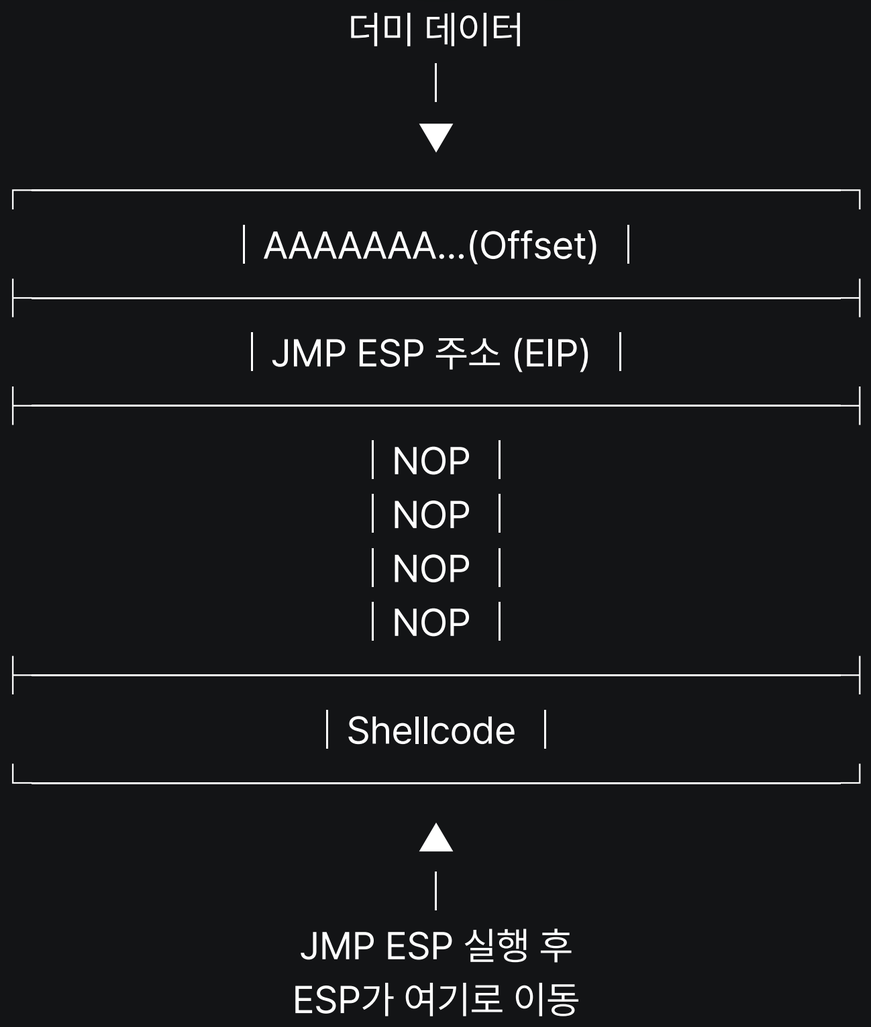
🔴 문제 : NULL Byte

- 쉘코드는 ESP가 가리키는 위치에 저장된다.
- ESP 값에 NULL Byte가 포함될 수 있어 EIP에 직접 사용하기 어렵다.

🟢 해결 : JMP ESP

- NULL Byte가 없는 JMP ESP 주소를 찾는다.
- EIP를 JMP ESP 주소로 덮어쓴다.
- JMP ESP가 실행되면서 ESP가 가리키는 스택 영역으로 이동한다.
- ESP가 가리키는 위치의 Shellcode가 실행된다.

CVE-2009-1329
취약점 분석



NOP SIed(미끄럼틀)

1. NOP(0×90) 명령어를 반복해서 만든 메모리 구간
2. CPU는 NOP을 만나면 다음 명령으로 이동
3. 공격 시 정확한 주소를 몰라도 쉘코드로 유도 가능
4. “미끄러지듯 코드로 도달”하게 만드는 구조다

CVE-2009-1329
코드 실행 원리

```
junk1 = "A" * 20000
junk2="Aa0Aa1Aa2Aa3 ..." # 패턴 코드
payload = junk1 + junk2

f = open("3.m3u", "wb")
f.write(payload)
f.close()
```

❶ 긴 문자열 생성
- 버퍼를 채우기 위한 더미 데이터를 생성한다.

❷ Pattern 문자열 추가
- EIP 위치를 확인하기 위한 문자열을 추가한다.

❸ Payload 생성 및 저장
- 더미 데이터와 Pattern으로 Payload를 생성한다.

❹ 파일 실행
- 파일을 읽는 과정에서 버퍼 오버플로우가 발생한다.

❺ 프로그램 실행
- 원래 반환 주소(EIP)가 Pattern 값으로 변경된다.
- 프로그램이 잘못된 주소를 실행하여 Crash가 발생한다.
- Crash 후 EIP 값을 확인해 Offset을 계산할 수 있다.

CVE-2009-1329
코드 실행 원리

```
import struct

junk = "A" * 26063 # 버퍼 채움 (Offset)
eip = struct.pack("<L", 0×1001b058) # EIP 덮기 (점프 주소)
nop = "\x90" * 20 # NOP 슬라이드
shellcode = "..." # 실행 코드

payload = junk + eip + nop + shellcode # 최종 페이로드

with open("exploit1.m3u", "wb") as f:
f.write(payload) # 파일 저장

print("[+] created") # 완료 출력
```

❶ 버퍼 채우기 (Offset 맞추기)

- 메모리를 꽉 채워서
- 정확히 “덮어야 하는 지점”까지 이동시키는 과정

❷ EIP 덮기 (흐름 탈취)

- 원래 실행 흐름(EIP)을 바꿔버림
- 다음 실행 주소를 공격자가 지정

❸ NOP 슬라이드

- 아무 작업도 안 하는 명령
- 정확한 위치를 못 맞춰도 shellcode로 흘러가게 함

❹ 쉘코드 실행

- 실제로 실행되는 공격 코드
- 계산기 실행 / 쉘 획득 같은 동작 가능

CVE-2009-1329
대응 방안 및 결론

- 1.안전한 입력 처리
- 입력 길이를 검증하여 버퍼 크기를 초과하지 않도록 제한
- 안전한 문자열 처리 함수 사용

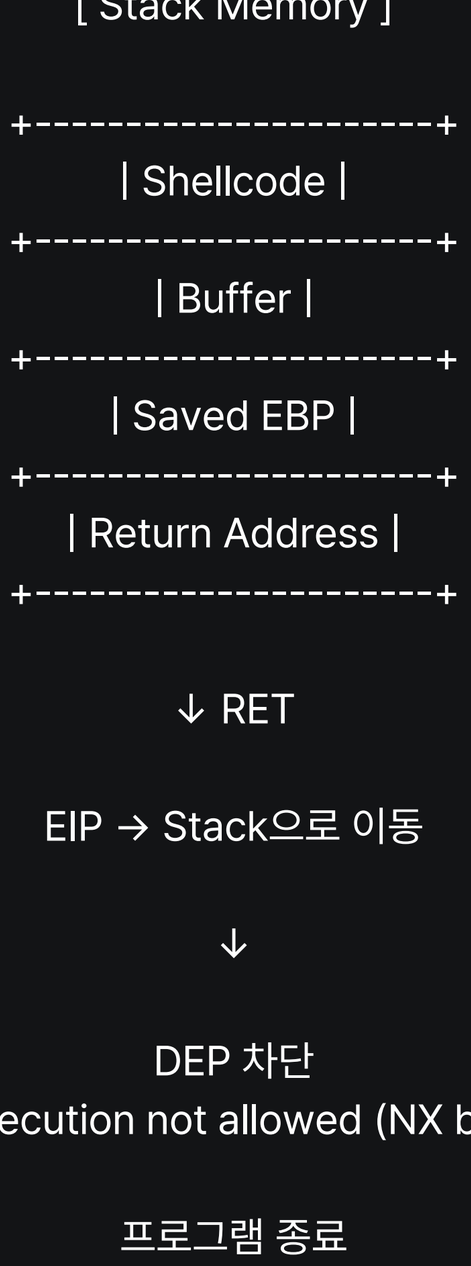
- 2.운영체제 보안 기능 활용
- DEP(Data Execution Prevention) : 데이터 영역의 코드 실행 차단
- ASLR(Address Space Layout Randomization) : 메모리 주소를 무작위화하여 예측 어려움
- Stack Canary : 반환 주소가 덮어쓰기되는 것을 탐지

CVE-2009-1329
대응 방안 및 결론

DEP란?

DEP는 스택이나 힙 같은 데이터 영역에서 코드가 실행되는 것을 차단하여, 버퍼 오버플로우로 삽입된 코드가 실행되지 못하도록 하는 보안 기능입니다.

DEP가 없으면 스택에 있는 Shellcode가 그대로 실행되지만, DEP가 있으면 스택 자체가 실행 금지 영역이 되어 코드 실행이 차단됩니다.



CVE-2009-1329
대응 방안 및 결론

ASLR은 프로그램의 메모리 주소를 실행할 때마다 랜덤하게 변경하여, 공격자가 반환 주소나 실행 위치를 예측하지 못하도록 하는 보안 기술입니다.

예시

원래:

printf 함수 주소 = 0×401000

ASLR 적용:

실행 1: 0×7ffd1234
실행 2: 0×6abc9876
실행 3: 0×55aa1111

CVE-2009-1329
대응 방안 및 결론

Stack Canary는 버퍼 오버플로우로 리턴 주소(RET)가
변조되는 걸 막기 위한 보호 값

왜 필요한가?

버퍼 오버플로우 공격은 RET를 덮어서 실행 흐름을 바꿈
카나리는 RET 앞에서 먼저 깨지기 때문에 공격을 감지 가능

CVE-2009-1329
대응 방안 및 결론

결론

- 버퍼 오버플로우 취약점으로 인해 입력값이 버퍼의 크기를 초과하면 인접한 메모리가 덮어써질 수 있음을 확인하였다.

- 패턴 분석을 통해 EIP가 덮어써지는 위치(오프셋)를 찾고, 원하는 주소로 EIP를 변경하여 프로그램의 실행 흐름을 제어할 수 있음을 확인하였다.

- NOP Sled와 페이로드를 ESP 영역에 배치하여 실행이 원하는 코드로 이어지도록 구성하였다. 최종적으로 ESP에 저장된 페이로드가 실행되어 계산기(calc.exe)가 정상적으로 실행되는 것을 확인하였다.

- 이번 발표를 통해 버퍼 오버플로우의 동작 원리와 공격 과정을 이해할 수 있었으며, 입력값 검증과 메모리 보호 기법이 프로그램 보안에 매우 중요하다는 것을 알게 되었다.

Q&A

질문을 받겠습니다