Use After Free 취약점 탐지 기술에 관한 연구

김진욱[1], 박상우[2], 송병진[3], 김지은[4], 김관빈[5], 임준섭[6], 고광만[7]
[1,2,6,7]상지대학교 컴퓨터공학과,[3,4,5]한국폴리텍대학교
2023015001@sj.sangji.ac.kr, bjsong@kopo.ac.kr, 202151007@sj.sangji.ac.kr,
kkman@sangji.ac.kr, 202048008@sj.sangji.ac.kr

A Study on Use After Free Vulnerability Detection Technique

Jin-Wook Kim[1], Sang-Woo Park[2], Byung-Jin Song[3], Ji-eun Kim[4], Gwan-bin Kim[5], Jun-Seop Lim[6], Kwang-Man Ko[7]
[1,2,6,7]Dept. of Computer Engineering Sang-Ji University, [3,4,5]Medical Engineering, KOREA POLYTECHNICS)

요 약

Use After Free(UAF) 취약점은 의료 소프트웨어, 의료 시스템, 임베디드 시스템 등 다양한 분야에서 널리 사용되고 있는 보안 취약점이다. C/C++는 메모리를 직접 할당하고 해제하는 작업을 프로그래머가 수동으로 수행하기 때문에 UAF의 발생 가능성이 높다. UAF는 과거로부터 현재까지 해결되지 않은 버그로써 존재하고 있으며, [1]2023년CWE 제일 위험한 취약점 Top25중에 4위를 차지하고 있다. 본 논문에서는 C/C++가 널리 사용되는 만큼 UAF를 빠르고 정확하게 탐지하기 위한 연구 내용과 성능 결과를 소개한다.

1. 서론

C/C++는 컴퓨터 프로그래밍 언어 중에서 가장 오래되고 널리 사용되는 언어 중 하나다. C언어를 기반으로 객체 지향 프로그래밍 개념을 도입하여 만들어진 C++는 시스템 프로그래밍, 게임 개발, 임베디드 시스템 등 다양한 분야에서 활용된다. C/C++는 하드웨어에 대한 직접적인 제어가 가능하여 매우 빠른 실행 속도를 제공하고 풍부한 표준 라이브러리와 다양한 서드파티 라이브러리를 활용할 수 있는 장점이 있다. 고성능 컴퓨딩의 다양한 분야나 저수준 코딩에서 C/C++는 매우 중요한 역할을 담당하고 있다.

(그림 1) 2023 CWE TOP 25

하지만 C/C++의 특징중 하나인 메모리 관리는메모리를 직접 관리해야 하므로 프로그래머의 실수로 인해 메모리 누수, 버퍼 오버플로우 등의 문제가 발생할 수 있다. 이러한 특징은 강력한 제어 능력을 제공하지만 수동적 관리에 의해 Use After Free 취약점이 발생하게 된다. Use After Free는 빈 공간에 필요에 따라 동적으로 메모리를 할당하기도 하고 해제 하기도 하는 Heap영역에서 일어난다. Heap영역에서 할당된 공간을 해제하고 메모리를 다시 할당할 시 공간을 재사용하면서 이러한 취약점이 발생한다. 이러한 취약점을 이용해 공격자는 웹 브라우저, 운영체제 커널, 어플리케이션 소프트웨어 등에 공격을 할 수있다.

2. UAF를 활용한 피해

C/C++에서 메모리 누수에 의해 발생한 Use After Free를 이용해 공격자는 임의 코드 실행, 정보 유출, 권한 상승, 서비스 거부(Dos), 데이터 조작 등 다양한 피해를 입힐 수 있다.

2.1 임의 코드 실행

원본 코드에 임의의 코드를 실행해 해제된 메모리 공간을 조작하여 프로그램의 제어 흐름을 변경할 수

있다. 이렇게 시스템을 완전히 장악할 수 있는 길이 열리게 된다. 이렇게 열린 길을 이용해 여러 가지 메모리에 접근할 수 있고 공격자에게 유리하게 코드를 변경할 수 있다.

2.2 정보 유출

공격자는 해제된 메모리 공간을 통해 민감한 정보를 획득할 수 있다. 예를 들어, 메모리 재사용 과정에서 민감한 데이터가 새롭게 할당된 객체에 남아있을 수 있다. 이러한 데이터에 공격자가 접근하게 되면서 정보가 유출된다.

2.3 권한 상승

메모리 누수를 통해 공격자가 임의 코드를 실행해 시스템의 권한을 상승시킬 수 있다. 이를 통해 낮은 권한의 사용자로 시작하여 루트 또는 관리자 권한을 획득할 수 있다.

2.4 서비스 거부(Dos)

프로그램의 제어 흐름을 망가뜨려 서비스 거부 상태를 유발할 수 있다. 이는 프로그램이 비정상적으로 종료되거나 응답하지 않게 하는게 가능하다.

2.5 데이터 조작

공격자는 중요한 데이터를 조작하여 시스템의 무결성을 해칠 수 있다. 예를 들어, 데이터 베이스의 값을 변경하거나 파일 시스템의 데이터를 변조할 수 있다.

3. UAF 보안 취약점 탐지

3.1 보안 취약점

Use After Free를 해결하기 위해서는 동적으로 할당된 메모리 공간을 해제한 후 그 메모리를 참조하고 있던 포인터를 참조 추적 등으로 사용하여 메모리에 접근하도록 해서는 안된다. 또한, 메모리 해제 후 포인터에 NULL값을 저장하거나 다른 적절한 값을 저장하면 의도하지 않은 코드의 실행을 막을 수 있다. 이러한 수정을 하기 위해 디버깅을 통해 코드를 검사하고 메모리가 재사용 되는 부분을 수정하거나 NULL값을 저장할 수 있다.

이렇게 프로그래머가 직접 관리해서 UAF 취약점의 발생을 방지할 수 있지만, 이러한 방법은 프로그래머의 실수가 일어나면 쉽게 다시 일어날 수 있기 때문에 C/C++ 같은 메모리 관리가 필요한 언어를 대신해서 Rust나 Go와 같은 메모리 안전 언어를 사용할 수 있다. 또한, [2]Address Sanitizer(ASan)와 같은 동적 분석 도구를 사용해 런타임 중에 메모리 오류를 탑지할 수 있고 컴파일 타임에 취약점을 탐지하는 분석 도구를 사용할 수 있다.

3.2 UAF 취약점 탐지

이 장에서는 예제 프로그램을 통해 어떠한 부분에서 취약점이 발생하고 그 취약점을 어떻게 해결하는지 탐지한다.

```
int main(int argc, const char *argv[]) {
  char *temp;
  temp = (char *)malloc(BUFFER_SIZE);
  ......
  free(temp);
  //해제한 자원을 사용하고 있어 의도하지 않은 결과가 발생하게 된다.
  stmcpy(temp, argv[1], BUFFER_SIZE-1);
}
```

(그림 2) 예제 프로그램

(그림 2)의 예제 프로그램을 보게되면temp라는 공간에 malloc함수로 BUFFER_SIZE 크기의 동적 메모리를 할게하게 된다. 그후에 free함수를 이용해 temp를 해제한다. 하지만 해제한 후에 stmcpy함수로 temp를 다시 사용하게 되면서 해제한 메모리의 재사용이 이루어지면서 취약점이 발생하게 된다. 이러한 문제를 해결하기 위해서는 해제한 자원을 재사용하지 않기 위해 stmcpy함수를 호출한 후에 free함수를 사용하거나 free함수뒤에 temp에 NULL값을 저장함으로써 메모리 공간을 해제한 후에 다시 사용할 수 없게하면 UAF취약점이 악용되는 상황을 방지할 수 있다. 이렇게 수동으로 메모리 관리를 할 때 프로그래머의 작은 실수로 코드의 취약점이 발생하는데 이러한 취약점이 발생하지 않도록 코드를 검토할 필요가 있다.

4. 결론

본 논문에서는 Use-After-Free버그를 공격자가 어떻게 사용하는지 버그의 예방또는 감지하는 방법들을 분석하였다. Use-After-Free의 [1] CWE순위가 높은 걸로 보아 그 만큼 중요하고 빈번하게 일어나는 버그라는걸 알 수있고, 이를 해결하기 위해서는 상당한 노력이 필요하다고 볼 수 있다. UAF를 해결

하기 위해 이미 많은 탐구가 이루어지고 있지만 아직은 정확성이나 속도 면에서 부족함이 느껴지고 이 버그를 해결하기 위한 획기적인 방법이 필요하다. 본 논문에서 UAF의 탐구를 통해 버그 방지를 위한 더욱 섬세한 연구 필요성과 다양한 공격 시나리오 분석을 통해 추가 연구를 진행할 예정이다.

참고문헌

[1] CWE Top25: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html)

[2] AddressSanitizer : https://github.com/google/sanitizers/wiki/AddressSanitizer

[3] 행정안전부 한국인터넷진흥원의 소프트웨어 보안약점 진단 가이드