정 보 보 호 학 회 지
제 31 권 제2호, 2021. 4

# 소프트웨어 타입 및 메모리 안전 강화 연구

전 유 석*

요 약

C/C++ 언어는 빠른 속도를 가지는 저 수준(low-level) 언어로 운영체제, 응용 프로그램, 인공지능 라이브러리 및 IoT 장비 운영체제 등 광범위한 분야에서 사용되고 있다. 하지만 C/C++ 프로그램은 타입 및 메모리 안전을 보장하는 프로그램을 구현하는 것은 프로그래머의 몫으로 남겨두기 때문에 프로그래머의 실수 등으로 인한 타입 및 메모리 안전 위반 사례는 주요 프로그램의 취약점 중 60% 이상을 차지할 만큼 심각하다. 이 논문에서는, 저자가 수행하였던 타입 및 메모리 안전 위반 문제를 해결하기 위한 두 가지 접근 방식을 소개한다. 첫 번째로, 특정 객체의 할당 패턴을 처리하면서 경량 메타데이터 구조와 컴파일러 최적화를 제공하는 타입 안전 위반 취약점 탐지 도구인 HexType은 취약점 탐지 범위 및 성능 오버헤드를 기존 연구에 비해 크게 개선하였다. 다음으로, 메모리 안전 위반 취약점을 효율적으로 탐지하기 위한 퍼징 기반의 FuZZan을 소개한다. FuZZan은 퍼징 테스트 처리량을 개선하여 주어진 동일 시간 안에 더 많은 취약점을 찾는 데 도움을 준다. 또한, 본 논문은 저자가 수행할 추가적인 소프트웨어 타입 및 메모리 안전 강화 연구 등을 포함한 향후 연구 방향을 소개한다.

## Ⅰ. 서 론

C/C++는 빠른 속도로 하드웨어를 제어할 수 있는 시스템 프로그래밍 언어로 대부분 운영체제 (Windows, Linux, MacOS), 응용 프로그램 (Chrome, Firefox, Wireshark), 인공지능 라이브러리(Tensor Flow, Pytorch) 및 IoT 장비들의 운영체제(TinyOs, freeRTOS) 등 광범위한 분야에서 사용되고 있다. C/C++ 프로그램은 타입 안전(Type Safety) 및 메모리 안전(Memory Safety)을 보장하지 않기 때문에, 구현 시 이에 유의하여 프로그램을 작성하여야 한다. 하지만, 프로그래머의 부주의 등으로 인해 타입 및 메모리 안전 위반 사례는 전체 안드로이드, MacOS, Ubuntu, 크롬, 파이어폭스 등 프로그램의 취약점 중 60%[1, 2] 이상을 차지할 만큼 빈번하게 발생한다. 따라서, 저자는 광범위하게 사용되는 C/C++ 프로그램의 취약점 중 대부분을 차지하는 타입 안전 위반(Type safety violation) 및 메모리 안전 위반(Memory safety violation) 취약점 분석 및 대응기술 연구를 수행하였다.

첫 번째, 타입 안전 위반(Type safety violation) 취약점은 주로 잘못된 타입 캐스팅으로 발생할 수 있다. 타입 캐스팅은 C++에서 모듈화를 가능하게 하는 핵심 기능 중 하나이며 성능을 위해 대부분의 타입 캐스팅은 정적으로만 검사하고 있다. 하지만, 정적 검사는 객체의 실제 실행 시간의 타입 정보를 무시하고 포인터 타입 등의 타입 정보에 의존하여 제한적인 검사를 수행하는 문제를 가지고 있다. 따라서, 실행 시간에 포인터가 잘못된 캐스팅으로 인해서 유효하지 않은 타입을 가리키게 된다면 허용되지 않은 메모리 영역에 접근하여 큰 문제를 일으킬 수 있다. 공격자는 이러한 유형의 문제를 악용하여 Adobe Flash, PHP, Google Chrome 또는 Firefox와 같은 소프트웨어를 공격한 사례가 계속 보고되고 있다. 따라서 이러한 문제를 해결하기 위해서 위험성이 존재하는 모든 캐스팅 부분에 명시적으로 검사 코드를 삽입하고 실행 시간에 포인터가 가리키는 객체의 정확한 타입 정보를 바탕으로 타입 안전 위반 공격을 탐지 할 수 있는 HexType[3]을 제안하였다. 실험 결과 이전의 최신 취약점 분석 도구와 비교하여 Firefox 테스트에서 최소 1.1배 - 최대 6.1배 더 높은 취약점 탐지능력을 보여주었다. 또한, 여러 가지 최적화 노력으로 SPEC CPU2006 벤치마크의 실험 결과 오버헤드가 최소 2배 - 최대 33.4배 감소함을 확인하였다. 또한, 11

* 울산과학기술원 컴퓨터공학과 (조교수, ysjeon@unist.ac.kr)

개의 새로운 타입 위반 취약점 버그를 탐지하였고 탐지된 버그는 모두 해당 관리자에게 보고되어 현재 패치되었다.

두 번째, 퍼징(Fuzzing)은 가장 널리 쓰이는 취약점 탐지 방법의 하나며, 메모리 안전 위반 취약점 탐지 등에도 널리 사용되고 있다. 더 많은 메모리 안전 위반 취약점을 탐지하기 위해서 퍼저는 Address Sanitizer(ASan)[4]와 같은 메모리 안전 위반 취약점 탐지 센이타이져(Sanitizer)와 주로 함께 동작한다. 하지만, ASan을 비롯한 센이타이져들은 유닛 테스트 (Unit tests) 및 장시간 러닝 테스트(long-running integration tests) 등의 테스트에 적합하게 디자인되어 있고 퍼저와의 연동을 목적으로 개발되지 않았기 때문에 퍼징처럼 각 테스트 별 매우 짧은 테스트 시간을 가지는 환경에서 불필요한 오버헤드를 발생시키고 퍼징의 실행 속도를 낮추어 주어진 시간 동안 검사능력을 저하하는 문제를 가지고 있다. 이 오버헤드의 근본적인 원인은 취약점 분석 도구들이 가지는 짧은 퍼징 시간에 적합하지 않은 무거운 메타데이터 구조(Metadata structure) 때문이다. 따라서 이 문제를 해결하기 위해 퍼징 환경에 맞는 가벼운 새로운 메타데이터 구조들을 설계하였고 퍼징을 수행하면서 얻어지는 결과에 따라서 기존 취약점 분석 도구의 무거운 메타데이터 구조 대신 제안된 새로운 메타데이터 구조를 선택적으로 사용하게 함으로써 퍼징 오버헤드를 줄일 수 있는 FuZZan[5]을 제안하였다.

FuZZan은 기존의 취약점 도구의 메타데이터 구조를 교체하였지만, 실험을 통해서 FuZZan의 새로운 메타데이터 구조는 기존 취약점 분석 도구와 같은 취약점 탐지능력을 갖추며 퍼징 속도를 48% 향상하여 주어진 테스트 시간 동안 더 많은 취약점을 찾아낼 수 있음을 실험을 통해 확인하였다.

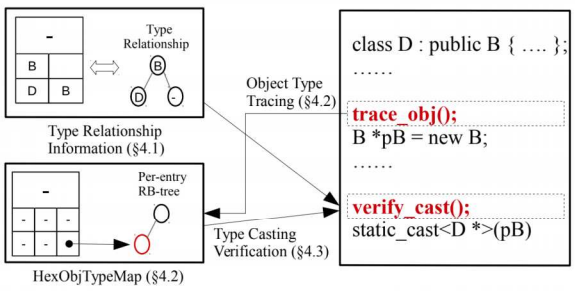
## Ⅱ. 타입 안전 강화 연구

타입 안전 강화를 목표로 수행한 HexType 프로젝트는 C++ 소프트웨어를 공격하기 위해 이용되는 주요 취약점인 타입 안전 위반(Type safety violation) 취약점을 효율적으로 탐지하기 위해서 수행되었다. C++에서 타입 안전 위반 공격은 모듈화를 가능하게 하는 핵심 기능 중 하나인 타입 캐스팅을 통해 주로 발생할 수 있다. 하지만, 타입 캐스팅은 성능을 위해 대부분의 타입 캐스팅은 정적으로만 검사하고 있다. 정적 검사는 객체의 실제 실행 시간의 타입 정보를 얻을 수 없어 포인터 타입 등 제한된 타입 정보에 의존하여 검사를 수행하는 문제 때문에 타입 안전 위반 취약점을 제대로 탐지 못하는 단점을 가지고 있다. 잘못된 캐스팅으로 인한 타입 안전 위반 취약점은 허용되지 않은 메모리 영역에 접근하여 중요한 데이터 접근 및 변조를 할 수 있다. 공격자는 이러한 문제를 악용하여 다음과 [표 1]과 같은 타입 안전 위반 취약점이 보고 되고 있다. 이러한 취약점을 제거하기 위해서 다양한 기존 연구[7, 8]들이 수행되었지만, 제한된 탐지 능력 그리고 탐지 속도 등의 문제점 때문에 타입 안전 위반 취약점을 효율적으로 탐지하는데 제약이 있어서 아직도 많은 타입 안전 위반 취약점이 남아 지속해서 보고되고 있다.

이러한 문제를 해결하기 위해서 제안된 HexType은 [그림 1]에서 처럼 위험성이 존재하는 모든 캐스팅 부분에 LLVM을 활용하여 명시적으로 검사 코드를 삽입한다. 캐스팅이 타입 안전 위반 취약점을 초래하는지 검증하기 위해서 캐스팅에 쓰이는 원천(Source) 타입 및 목적지(Destination) 타입 정보가 필요하다. 하지만, 컴파일 타임에 알 수 있는 목적지 타입과 달리 원천 포인터가 가리키는 객체의 정확한 타입 정보를 파악하는 것은 어려운 문제이다. 원천 포인터가 가리키는 객체의 정확한 타입을 파악하기 위해서, HexType은 객체가 할당되고 해제되는 부분에 추가적인 코드를 삽입하여 객체가 할당될 때 할당된 주소 및 타입 정보를 메타데이터에 저장하고 객체가 해제될 때 해당 정보를 메타데이터에서 삭제하는 방식의 디자인을 선택하였다. 또한, 객체의 할당에서 해제까지의 시간이 상대적으로 짧고 대량

〔표 1〕 타입 안전 위반 취약점 목록

| 프로그램 | CVE # |
|---|---|
| Google Chrome V8 | CVE-2020-6418 |
| WebKitGTK | CVE-2020-3897 |
| Autodesk FBX SDK | CVE-2020-7081 |
| Adobe Flash Player | CVE-2020-3757 |
| PhantomPDF | CVE-2020-15638 |
| Firefox | CVE-2019-17026 |
| Libxslt | CVE-2019-5815 |
| Ghostscript | CVE-2018-19134 |
| Adobe Flash Player | CVE-2018-4944 |
| Foxit PDF Reader | CVE-2018-9942 |




〔그림 1〕 HexType 프로젝트 개요

| Fast-path Slot | | | Slow-path Slot (RB-tree Ref) |
|---|---|---|---|
| Allocated Object Ref | Hashvalue for Object Name | Type Relationship Information Ref | |
| 0x417000 | 2341234 | 0x51723D | |
| 0x41563C | 1312321 | 0x51724D | _ |
| 0x41723D | 7231234 | 0x51724D | _ |
| _ | _ | _ | _ |
| 0x41563E | 4232123 | 0x51623D | |

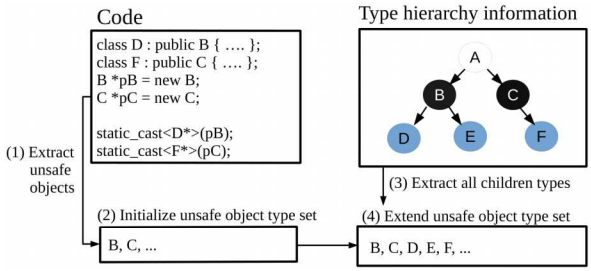
Per-entry RB-tree

〔그림 2〕 HexType 메타데이터 자료구조

으로 할당되는 스택 객체 등을 고려할 때, 해쉬 맵(hash map)에 기반한 새로운 메타데이터 구조[그림 2]를 제안하였다. 또한, 모든 객체의 할당을 모두 추적하는 것은 성능상의 큰 오버헤드를 초래하기 때문에 [그림 3]에서처럼 정적 분석을 통해서 캐스팅에 실제 사용되는 타입 정보를 뽑아 이 타입과 관련된 객체가 할당되고 해제될 때만 정보를 메타데이터 구조에 저장하는 최적화 방식을 적용하였다. 캐스팅과 관련이 없는 타입은 추후 타입 캐스팅에 사용되지 않을 것이기 때문에 HexType은 이를 추적할 필요가 없다.

기존의 취약점 분석 도구가 가지고 있는 제약적인 탐지 범위 문제를 해결하기 위해서, 기존 취약점 도구가 처리하지 못했던 CMA(Customized Memory Allocator)에서 관리되는 객체들을 추적하기 위해서 추가로 Placement_new 및 Reinterpret_cast 등을 활용한 CMA 할당 패턴을 찾아내고 추적하여 기존 연구에 대비하여 취약점 탐지능력을 크게 향상했다.

실험 결과 이전의 최신 취약점 분석 도구와 비교하여 Firefox 테스트에서 CMA를 통한 객체의 할당 정보를 추가로 추적하고 관리함으로써 최대 6.1배 더 높은 취약점 탐지능력을 보여주었다. 또한, 캐스팅 검증에 적합한 메타데이터 구조 및 객체 추적 최적화 방법 등을 통




〔그림 3〕 객체 추적 최적화 방법 개요

하여 SPEC CPU2006 벤치마크의 실험 결과 오버헤드가 최대 33.4배 더 빠르게 테스트할 수 있는 것을 확인하였다. 또한, HexType은 Qt 및 Apache Xerces-C++에서 11개의 새로운 타입 위반 취약점 버그를 탐지하였고 탐지된 버그는 모두 해당 관리자에게 보고되어 현재 패치되었다. 타입 안전 위반 취약점 점검 및 제거를 통한 보안성 향상에 이바지하기 위해서, HexType은 오픈소스[9]로 공개하였다.

## Ⅲ. 메모리 안전 강화 연구

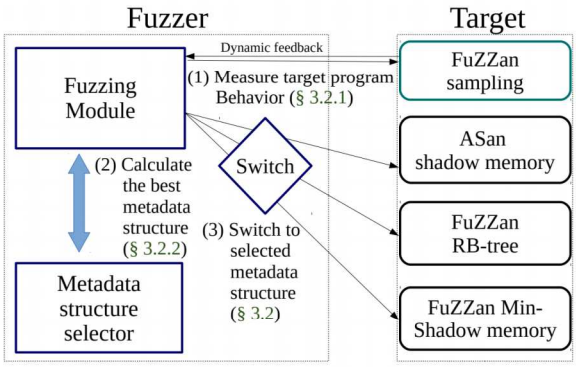
메모리 안전 강화를 목표로 수행한 FuZZan 프로젝트는 소프트웨어에서 발생하는 주요 취약점인 메모리 안전 위반 취약점을 효과적으로 탐지하기 위해서 수행된 프로젝트이다. 퍼징(Fuzzing)은 널리 사용되고 있는 효과적인 취약점 탐지 기법의 하나로 많은 메모리 안전 위반 취약점이 퍼징을 통해서 검출되고 있다. 하지만 현재 퍼저는 테스트 중에 발생한 취약점이 세그먼트 폴트(Segment fault)등으로 인한 크래시(crash)가 발생하지 않는 취약점은 찾을 수 없는 문제점을 가지고 있다. 따라서, 발생한 취약점을 탐지하기 위해서 퍼저는 Address Sanitizer(ASan)와 같은 메모리 안전 위반 취약점 탐지 센이타이저(Sanitizer)와 함께 동작한다. ASan는 잘못된 메모리 영역 탐지가 발생하고 이 접근이 크래시를 발생하지 않더라고 퍼저가 인식할 수 있도록 도울 수 있다. 하지만 ASan과 같은 일반적인 취약점 분석 도구들은 유닛 테스트(Unit tests) 및 장시간 러닝 테스트(long-running integration tests)등의 테스트에 최적화되어 있다. 따라서, 퍼징처럼 각 테스트 별 매우 짧은 테스트 시간을 가지는 환경에서 취약점 분석 도구가 매번 실행 될 때마다 발생하는 상당한 수준의 초기화 및 종료 함수들은 불필요한 오버헤드를 발생시키고 퍼징의 실행 속도를 낮추어 주어진 시간 동안 검사능력

을 감소시키는 문제를 가지고 있다. 이 오버헤드의 근본적인 원인은 취약점 분석 도구들이 가지는 짧은 퍼징 시간에 적합하지 않은 무거운 메타데이터 구조(Metadata structure) 때문이다. 이러한 메타데이터 구조는 매번 테스트할 때마다 초기화 및 정리 작업을 계속 반복해야 하므로, 기존의 장시간 테스트 환경에서는 한 번의 메타데이터 구조 관련 작업이 큰 문제는 아니었으나, 퍼징처럼 매번의 짧은 테스트를 위한 무거운 메타데이터 구조 초기화 작업은 높은 오버헤드를 가지는 문제를 초래한다. 따라서 이 문제를 해결하기 위해, [그림 4]처럼 퍼징 환경에 맞는 가벼운 새로운 메타데이터 구조인 RB-tree 기반 메타데이터 구조 및 쉐도우 메모리(shadow memory) 기반 Min-shadow 메타데이터 구조를 설계하였다.

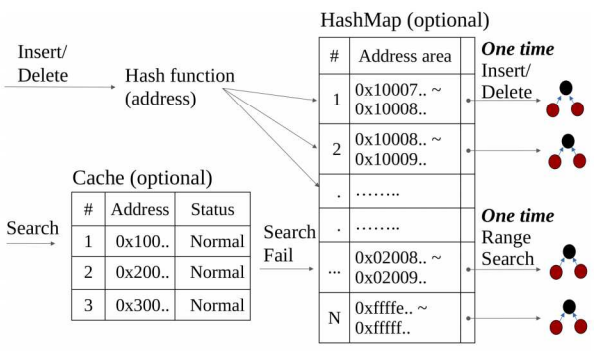
RB-tree [그림 5] 메타데이터 구조는 기존의 쉐도우 메모리처럼 넓은 공간을 예약하고 사용하는 대신에 짧은 퍼징 시간을 고려하여 초기화 및 종료가 상대적으로 간편한 RB-tree에 기반한 메타데이터 구조를 설계하였다. 비교적 짧은 퍼징 시간 동안에 할당되는 객체의 양은 적을 것이기 때문에 RB-tree 메타데이터 구조에 저장되는 객체의 양 또한 적을 것이라는 가정을 바탕으로 설계하였다. Min-shadow 메타데이터 구조의 경우, 기존 쉐도우 메모리를 기반으로 하지만, 실제 메모리의 크기에 대응하여 쉐도우 메모리 메타데이터 구조 크기가 증가하는 특성을 고려하여, 실제 메모리를 강제적으로 압축하여 대응되는 쉐도우 메모리의 크기를 줄이는 Min-shadow 메타데이터 구조를 고안하였다. [그림 6]처럼, 64bit 플랫폼에서 실제 메모리에 대응되는 16TB 영역을 쉐도우 메모리로 지정하고 사용하던 방식에서 강제적으로 메모리 영역을 줄여, 상대적인 메타데이터 구조 크기를 줄여서 관련 초기화 및 종료 작업의 오버헤드를 줄 일 수 있다. 예를 들어, [그림 6]의 경우, 4GB 메모리 영역으로 압축 시에 512MB의 쉐도우 메모리만 필요하다. 프로그램별로 다른 힙 메모리 크기 등을 요구할 수 있으므로 다양한 힙 메모리 크기를 가지는 Min-shadow 모드를 만들어서 이에 대응할 수 있도록 디자인하였다.

[그림 4]에서처럼 ASan의 기본 쉐도우 메타데이터 구조 및 다양한 Min-shadow 메타데이터 구조, RB-tree 메타데이터 구조를 적용한 각각의 다른 버전의 빌드가 필요하다. 퍼징 테스트 시 주기적으로 샘플링(sampling) 모드를 실행하여 해당 테스트 프로그램이 현재 퍼저가 테스트하고 있는 인풋을 대상으로 어느 정도의 객체 및 메모리를 할당하는지를 파악하여 퍼징 테스트 도중에 적합한 메타데이터 구조로 전환되는 방식을 통해 추가적인 최적화를 수행하였다.

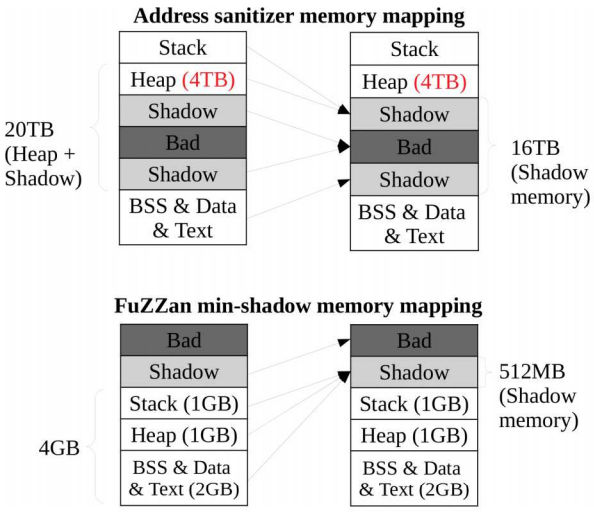
FuZZan은 기존의 취약점 도구의 메타데이터 구조를




〔그림 4〕 FuZZan 프로젝트 개요




〔그림 5〕 RB-tree 메타데이터 구조




〔그림 6〕 Min-shadow 메타데이터 구조

교체하였지만, 기존에 알려진 다양한 취약점 점검을 통해서 FuZZan의 새로운 메타데이터 구조는 기존 취약점 분석 도구와 동일한 취약점 탐지 능력을 갖추고 있는 것을 확인하였다. 또한 효율적인 메타데이터 구조 덕분에, 퍼징 속도를 48% 향상하여(Google fuzzer test suite 벤치마크[6]를 활용한 실험) 주어진 테스트 시간 동안 더 많은 취약점을 찾아낼 수 있음을 실험을 통해 확인하였다. 메모리 안전 위반 취약점 점검 및 제거를 통한 보안성 향상에 이바지하기 위해서, FuZZan은 오픈소스[10]로 공개하였다.

## Ⅳ. 향후 연구 계획

앞으로 저자가 향후 수행하고자 하는 연구 방향은 다음과 같다.

### 4.1. C/C++ 메모리 및 타입 안전 강화 연구

현재 타입 및 메모리 안전 위반 취약점 탐지 도구들은, 테스트 용도가 아닌 실제 제품에 적용되어 제로데이 공격을 예방할 수 있는 수준에 이르기에는 높은 오버헤드, 제한된 탐지능력 그리고 잘못된 긍정 오류(false positive) 혹은 부정 오류(false negative) 등의 문제점이 남아 있다. 따라서, 이러한 문제를 해결할 수 있는 연구를 계속해서 수행하고자 한다. 현재, 이러한 문제를 해결하기 위한 노력으로, 타입 정보를 기록하는 데 필요한 추가적인 저장 공간을 할당하지 않으며 할당된 객체의 타입 정보를 추적하기 위한 노력을 최소화할 수 있는 V-Type 연구를 수행할 계획에 있다. 다형성(polymorphic) 객체는 내부적으로 할당된 객체의 타입 정보를 가상 테이블을 통해 얻을 수 있지만, 비 다형성(non-polymorphic) 객체는 타입 안전 위반 취약점 탐지를 위한 타입 정보를 내부적으로 가지고 있지 않다. 따라서 V-Type은 비 다형성 객체를 컴파일러를 수정하여 타입 정보를 가지는 다형성 유형의 타입으로 강제로 변경한다. 이를 통해, V-Type은 개체 할당 및 해제를 추적하지 않고 분리된 저장 공간을 할당 및 관리하는 부담을 제거하여 탐지 성능을 크게 향상하는 연구를 수행하고 있다.

### 4.2. RUST 취약점 분석

RUST는 시스템 프로그래밍 언어로, C/C++와 동등한 수준의 속도를 달성하면서 주요 프로그램의 취약점의 60% 이상을 차지하는 메모리 안전 위반 취약점으로부터 안전한 높은 보안성을 제공하는 언어로 마이크로소프트, 파이어폭스 등 많은 기업의 프로그램이 RUST을 이용해 구현되고 있다. 또한, 2016년부터 5년 동안 스택 오버플로우 설문조사에서 매년 가장 좋아하는 언어로 선정되었다. 이러한 RUST가 국내 소프트웨어 개발에 적용이 된다면 메모리 안전 위반 취약점 등을 제거한 안전 소프트웨어 개발을 개발하여 금융, 포탈, 정보 서비스, 빅데이터 분석 등의 주요 IT 산업에 사용되는 소프트웨어에 적용하여 보안성을 크게 향상할 수 있을 것으로 기대된다. 하지만, 최근 unsafe RUST 및 C/C++과 같은 이기종 언어와의 통신 등에서 지속해서 RUST의 보안 취약점이 보고되고 있다. 따라서 이러한 RUST의 보안 위협 제거를 위한 대응기술 개발을 수행하고자 한다.

### 4.3. 그 외 관심 분야

저자는 기존에 수행하였던 프로세스에 최소 권한 정책 강화를 통해 공격의 피해를 최소로 하고자 했던 POLPER[11]의 후속 프로젝트로 더 발전된 정적 분석 기법을 바탕으로 자동으로 개발자와 사용자의 추가적인 노력 없이 해당 프로그램에 필요한 권한을 자동으로 정확하게 추출하고자 하는 연구를 수행하고자 한다. 이 기술은 최근 많이 사용되고 있는 컨테이너에 적용되어 도커(Docker)와 같이 커널을 공유하는 경우 해당 컨테이너에 최소 권한만을 부여하여 공격 피해를 최소화하는 데 활용될 수 있을 것으로 기대된다.

자율 주행 기술은 자율 주행 자동차의 완전한 실용화를 목표로 많은 발전을 이루고 있다. 하지만, 자율 주행은 이를 가능하게 하는 정확한 감지, 계획, 제어 및 수많은 불확실성을 처리하기 위한 수많은 알고리즘을 포함하여 매우 복잡하다. 모든 구성 요소가 정확하게 동작하지 않으면 인명을 위협하는 큰 사고로 이어질 수 있다. 따라서, 저자는 퍼징 등의 테스트 기법을 활용하여 자율 자동 기술의 보안성을 향상할 수 있는 연구를 수행하고자 한다.

## Ⅴ. 결 론

C/C++ 언어는 빠른 속도 및 높은 자유도를 바탕으로 광범위하게 우리 생활에 사용되고 있지만, 프로그램은 타입 및 메모리 안전을 보장하는 프로그램을 구현하는 것은 프로그래머의 몫으로 남겨두기 때문에 주요 프로그램의 대부분 취약점이 타입 및 메모리 안전 위반과 관련 있을 만큼 심각하다. 이 논문에서는, 저자가 기존에 타입 안전 강화를 위해서 수행하였던 HexType 및 메모리 안전 위반 문제를 해결하기 위해 수행하였던 FuZZan을 소개하였다. HexType의 경우 기존의 타입 안전 위반 취약점 탐지 도구가 가지는 문제점을 해결하여 성능을 개선하였으며, FuZZan의 경우 메모리 안전 위반 취약점을 찾는데 주로 쓰이는 퍼징 및 ASan 결합 구조에서 생기는 비효율적인 디자인상의 이슈를 해결하여 주어진 시간에 안에 버그를 더 많이 찾을 수 있도록 성능을 개선하였다. 마지막으로, 향후 저자가 수행할 C/C++ 메모리 및 타입 안전강화 연구, RUST 취약점 분석 및 그 외 관심 연구 분야를 소개하였다.

## 참 고 문 헌


[1] Alex Gaynor, Fish in a Barrel, “QUANTIFYING MEMORY UNSAFETY AND REACTIONS TO IT,” ENIGMA, https://www.usenix.org/conference/enigma2021/presentation/gaynor, 2021.

[2] Microsoft Securtiy Team, “Trends, challenge, and shifts in software vulnerability mitigation,” https://github.com/microsoft/MSRC-Security-Research/tree/master/presentations/2019_02_BlueHatIL, 2019.

[3] Yuseok Jeon, Priyam Biswas, Scott Carr, Byoungyoung Lee, and Mathias Payer, “HexType: Efficient Detection of Type Confusion Errors for C++,” In Proceedings of the ACM Conference on Computer and Communications Security(CCS), 2017.

[4] Serebryany, Konstantin and Bruening, Derek and Potapenko, Alexander and Vyukov, Dmitriy, “AddressSanitizer: A fast address sanity checker,” In Proceedings of the USENIX Annual Technical Conference(ATC), 2012

[5] Yuseok Jeon, WookHyun Han, Nathan Burow, and Mathias Payer, “FuZZan: Efficient Sanitizer Metadata Design for Fuzzing,” In Proceedings of the USENIX Annual Technical Conference (ATC), 2020

[6] Google, “*Fuzzer test suite,*” https://github.com/google/fuzzer-test-suite

[7] Istvan Haller, Yuseok Jeon, Hui Peng, Mathias Payer, Cristiano Giuffrida, Herbert Bos, and Erik van der Kouwe, “TypeSan: Practical Type Confusion Detection,” In Proceedings of the ACM Conference on Computer and Communications Security(CCS), 2016.

[8] Lee, Byoungyoung and Song, Chengyu and Kim, Taesoo and Lee, Wenke, “Type casting verification: Stopping an emerging attack vector,” In Proceedings of the USENIX Security, 2015

[9] Yuseok Jeon, Priyam Biswas, Scott Carr, Byoung young Lee, and Mathias Payer, “HexType,” https://github.com/HexHive/HexType

[10] Yuseok Jeon, WookHyun Han, Nathan Burow, and Mathias Payer, “FuZZan,” https://github.com/HexHive/FuZZan.

[11] Jeon, Yuseok and Rhee, Junghwan and Kim, Chung Hwan and Li, Zhichun and Payer, Mathias and Lee, Byoungyoung and Wu, Zhenyu, “Process-aware restriction of over-privileged setuid calls in legacy applications,” In Proceedings of the ACM Conference on Data and Application Security and Privacy (CodaSpy), 2019.


## 〈저 자 소 개〉


**전 유 석 (Yuseok Jeon)**

정회원

2007년 8월 : 인하대학교 컴퓨터공학 졸업

2010년 2월 : 포항공과대학교 정보통신학 석사

2020년 12월 : 퍼듀대학교 컴퓨터공학 박사

2021년 2월~현재 : 울산과학기술원 김퓨터공학과 조교수

<관심분야> 소프트웨어 보안 및 시스템 보안