스패로우
정적 분석 기술

화이트 페이퍼

주식회사 스패로우

SPARROW

CONTENTS

1 핵심 기술 개요

당사의 솔루션에는 복잡하고 다양한 기술이 적용됩니다. 그 중에서 핵심 기술은 크게 분석 기술과 통합 기술이라는 두 가지로 구분할 수 있습니다. 분석 기술은 솔루션의 코어라고 볼 수 있는 분석 엔진의 분석 수행 방법에 관련된 기술입니다. 여기에 포함된 기술의 종류는 정적 분석, 동적 분석, 구성요소 분석으로 나누어집니다. 통합 기술은 이러한 분석의 결과를 활용하는 방법과 관련된 기술입니다. 여기에는 개별 분석 기술의 결과를 종합적으로 판단해서 결과를 표시하는 결과 통합 기술과 사용자의 소프트웨어 통합 및 배포(CI/CD) 파이프라인에 당사의 솔루션을 연결하여 활용하는 DevSecOps 기술이 포함됩니다.

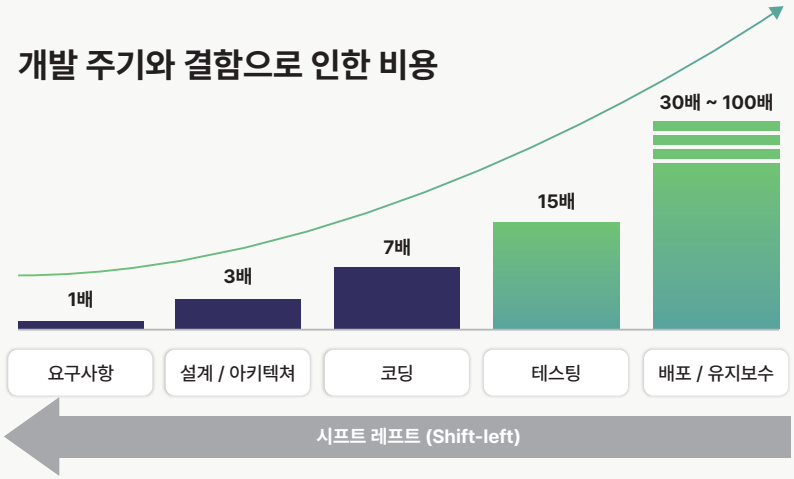
정적 분석 기술은 당사의 전주기적 통합 애플리케이션 보안 테스팅 솔루션에서 제공하는 분석 기술인 정적 분석, 동적 분석, 구성 요소 분석 중 하나입니다. 세 가지 기술은 분석에 사용하는 목적물(분석 대상)의 차이를 기준으로 분류되며, 각 기술을 통해 분석을 수행하는 절차 혹은 방법(분석 방법), 분석 후 표시되는 내용(분석 결과)에도 고유한 특징이 있습니다. 이 문서에서는 정적 분석 기술의 분석 대상에 따른 분석 방법 및 분석 결과에 대한 설명을 간략히 서술하도록 합니다.

2 정적 분석 기술

정적 분석(SAST, Static Application Security Testing)은 애플리케이션을 실행하기 전에 소스 코드, 바이트 코드, 또는 바이너리 코드를 분석하여 보안 취약점을 식별하는 기술입니다. 정적 분석에 사용되는 분석 대상을 넓은 의미로 소스 코드라고 부릅니다.(이후 소스 코드로 통일) 정적 분석은 소프트웨어 개발 라이프사이클(SDLC) 초기에 보안 문제를 식별하고 해결할 수 있는 주요한 방법입니다.

소프트웨어 개발 운영에서 보안에 대한 인식이 높아지면서 소프트웨어의 통합 및 배포(CI/CD) 파이프라인에 정적 분석 기술 도구가 통합되는 사례가 늘어나고 있습니다. 개발 초기 단계에서 보안 결함을 발견하는 경우 전반적인 소프트웨어 품질을 향상시킬 수 있을 뿐만 아니라 소프트웨어를 수정하는 비용을 크게 절감할 수 있기 때문입니다.

당사의 정적 분석 기술도 애플리케이션을 직접 실행하지 않고 소스 코드만으로 분석할 수 있기 때문에 애플리케이션이 완성되지 않은 개발 초기 단계부터 지속적으로 적용할 수 있다는 장점이 있습니다. 소프트웨어에서 오류나 취약점으로 인해 발생하는 비용은 해당 문제를 발견하고 해결하는 시점이 늦어질수록 기하급수적으로 증가합니다. (아래 그래프 참고) 따라서 정적 분석은 시프트 레프트(Shift-left) 접근 방식에 따라 조기에 결함을 찾아내려는 현대 소프트웨어 개발 흐름에 매우 유용한 기술입니다.



모양 분석(Syntactic Analysis)은 소스 코드의 문법과 구조를 분석하는 방법입니다. 즉, 코드가 문법적으로 올바른지를 확인하고 코드의 구조를 분석합니다. 따라서 모양 분석은 코드의 로직을 고려하지 않고 생김새와 관련된 검출 규칙을 검사할 수 있습니다. 예를 들어, 취약하다고 알려진 오래된 버전의 API를 사용하거나 코드에서 예외 처리 블록을 비워둔 경우 모양만으로 문제를 파악할 수 있습니다.

반면, 의미 분석(Semantic Analysis)은 코드의 문맥과 의미를 해석하여 논리적으로 맞는지를 분석합니다. 의미 분석은 실행 과정에 대한 이해를 바탕으로 하기 때문에 모양 분석에 비해 기술적으로 난이도가 높고 분석하기 위해 더 많은 시간과 메모리 자원이 필요합니다. 의미 분석에서는 복잡한 검출 규칙을 검사할 수 있습니다. 예를 들어, 널 값 역참조, 버퍼 오버런, 잘못된 메모리 해제 등의 치명적인 오류가 여기에 해당합니다.

검출 규칙은 국내외를 아우르는 다양한 소프트웨어 보안 및 품질 컴플라이언스에 따라 설계되었습니다. 사용자가 특정 코딩 가이드 혹은 스탠다드에 따른 규칙을 확인하려고 하는 경우, 매핑된 검출 규칙을 활성화하여 해당하는 검사만 수행하고 결과를 확인할 수 있습니다.

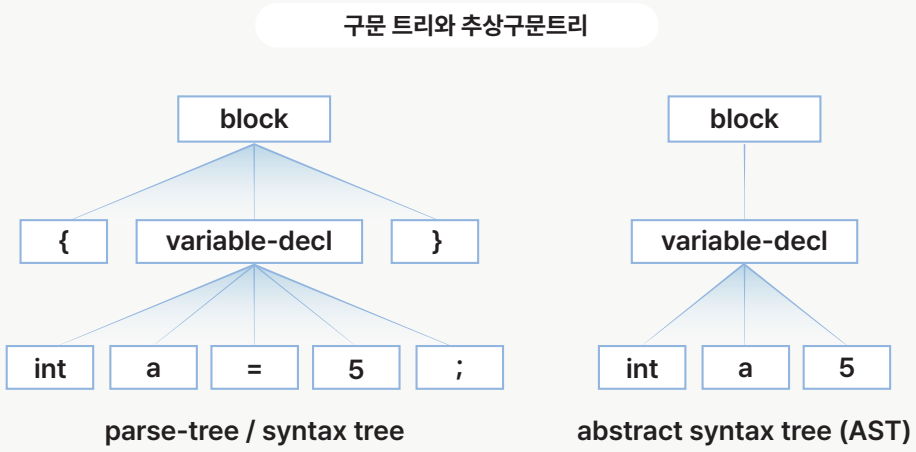
이렇게 모양 분석 및 의미 분석에 따라 분석이 수행되면 사용자는 분석 결과를 확인하고 적절한 수정 방법을 찾아야 합니다. 액티브 서제스천(Active Suggestion)은 검출된 문제를 해결할 수 있도록 방법을 제시하는 기술입니다. 일반적인 예시 혹은 케이스로 나눈 예제와 달리 실제 코드에서 구체적인 수정 내용을 알려준다는 점에서 당사의 기술이 돋보이는 기능이기도 합니다.

이러한 분석 방법 및 해결 방법은 폭넓은 프로그래밍 언어를 분석할 수 있다는 점에서 유의미한 결과를 가져오게 됩니다. 당사에서 개발한 공통 모듈은 최신 개발 트렌드를 반영한 새로운 언어를 빠르게 지원할 수 있는 특징을 가지고 있습니다. 이제 각각의 기술에 대해 자세한 내용을 살펴보겠습니다.

2.1 모양 분석

모양 분석(Syntactic Analysis)은 정적 분석 기술에서 필수적인 요소 기술이며 모양 분석의 주요 목적은 소스 코드가 프로그래밍 언어의 문법 규칙을 준수하는지를 확인하는 것입니다. 따라서 소스 코드의 구조를 확인하기 위해 먼저 원시 소스 코드를 구조화된 형식으로 변환해야 합니다. 모양 분석을 통해 변환된 코드는 이후 의미적 검증, 최적화 또는 더 높은 수준의 분석 작업을 수행할 수 있는 기초가 됩니다.

이 작업이 바로 소스 코드의 문법 구조를 나타내는 추상구문트리(Abstract Syntax Tree)를 생성하는 과정입니다. 추상구문트리란 구문 트리를 단순화시킨 형태로서 문법적 세부 사항을 생략하고 코드의 본질적인 구조를 파악하기 위해 초점을 맞춥니다.



위 그래프의 왼쪽에서 일반적인 구문 트리는 코드 블록 안에 포함된 중괄호 및 = 연산자를 모두 트리에 표시하게 됩니다. 따라서 현재 표시된 트리에는 표시되지 않았지만 트리의 구조가 더 복잡하고 레벨이 깊은 형태로 정리됩니다. 반면 오른쪽에 있는 추상구문트리에서 괄호와 같은 그룹화 기호는 생략될 수 있으며, 특정 노드는 코드의 계산 구조를 보다 직접적으로 표현하기 위해 축약될 수 있습니다.

추상구문트리는 이렇게 소스 코드의 구조를 단순하게 표현하기 때문에 의미가 더 간결해지고 읽기 쉬운 특징이 있습니다. 즉, 코드 자체가 아닌 추상구문트리에 포함된 정보를 사용하더라도 함수 호출, 변수 선언, 제어 흐름, 반복문, 조건문 등의 코드 구조를 명확하게 파악할 수 있습니다. 그렇기 때문에 추상구문트리를 사용함으로써 대규모의 코드를 분석하는 경우에도 빠르게 소스 코드의 구조를 확인하는 작업이 가능합니다.

모양 분석에서는 추상구문트리를 생성한 다음, 생성된 트리를 탐색하면서 트리의 노드를 하나씩 방문하게 됩니다. 이 때 방문한 트리의 구조와 검출 규칙의 분석 시작 조건이 일치하는 경우 검출 규칙에서 정해준 특정한 코드 패턴이나 구조가 존재하는지를 검사합니다. 이런 방법으로 프로그래밍 언어의 문법 규칙이 준수되었는지를 확인하고 나아가 보안 취약점이 존재할 가능성이 있는 부분을 식별하게 됩니다. 반복적인 변수 재할당, 안전하지 않은 함수 호출, 불완전한 조건문 등은 이러한 모양 분석 기술을 통해 검출될수 있는 결과입니다.

2.2 의미 분석

당사의 정적 분석은 의미 분석을 통해 모양 분석으로는 쉽게 찾을 수 없는 취약점과 치명적인 오류도 검출할 수 있는 성능을 갖추고 있습니다. 의미 분석(Semantic Analysis)이란 정적 분석의 중요한 요소 기술로써 소스 코드 프로그램의 의미적 일관성과 정확성을 검증하는 과정입니다. 모양 분석이 코드의 문법적 구조를 확인하는 단계라면, 의미 분석은 이 구조가 실제 의미적으로 올바른지를 확인하는 데 중점을 둡니다. 즉, 의미 분석을 통해 프로그램이 의도한 대로 동작할 것인지, 그리고 코드가 프로그래밍 언어의 의미적 규칙을 준수하는지를 검증하게 됩니다.

모양 분석에서 미리 생성한 추상구문트리를 사용하여 실제 소스 코드가 실행되는 순서에 따라 추상구문트리를 순회하며 실제 프로그램이 실행되는 의미를 계산합니다. 이를 위해 코드를 통해 다양한 정보를 추출하게 됩니다. 예를 들어, 함수의 경우 전체 프로그램의 함수 간 호출 그래프를 만들고 이 정보를 바탕으로 각각의 함수에 검출 규칙을 적용합니다. 이 때 함수로 인해 발생하는 메모리 및 이벤트에 관한 정보도 함께 계산됩니다.

의미 분석은 기본적으로 변수와 함수를 정의하고 사용하는 방법, 데이터 타입의 일관성, 제어 흐름의 논리적 타당성 등을 검사합니다. 따라서 이 과정을 통해 구문적으로는 문제가 없지만 의미적으로 오류가 있는 코드를 탐지할 수 있습니다. 예를 들어, 정의되지 않은 변수를 참조하거나 잘못된 타입의 데이터를 사용하는 경우 의미 분석을 통해 오류를 찾아냅니다.

이러한 의미 분석을 구현하기 위해 당사에서 개발한 주요 세부 기술은 다음과 같습니다.

- **스코프 분석(Scope Analysis)**

스코프 분석은 변수, 함수, 클래스 등의 이름이 유효한 범위를 결정하고, 해당 식별자를 범위 내에서 올바르게 참조했는지를 확인하는 기술입니다. 이 기술을 구현하기 위해서는 특정 블록이나 함수 내에서만 유효한 변수의 스코프를 정확히 파악할 수 있어야 합니다. 이를 통해 변수의 중복 선언, 정의되지 않은 변수의 사용, 변수 범위를 벗어난 참조 등과 같은 오류를 사전에 발견할 수 있습니다.

- **제어 흐름 분석(Control Flow Analysis)**

제어 흐름 분석은 프로그램이 실행되는 로직을 분석하여 실행 상 발생할 수 있는 논리적 오류를 탐지하는 기법입니다. 이 기술은 먼저 코드를 분석하여 제어 흐름 그래프(CFG, Control Flow Graph)를 그려냅니다. 이를 통해 프로그램의 실행 흐름을 시각화하고, 각 경로에서 발생할 수 있는 변수 값과 상태를 추적하여 논리적 오류를 발견합니다. 이 기술을 통해 조건문이나 반복문에서 발생할 수 있는 논리적 오류, 무한 루프, 도달 불가능한 코드(Dead code) 등을 효과적으로 탐지할 수 있습니다.

- **값 분석(Value Analysis)**

값 분석은 프로그램을 실행할 때 변수나 표현식에 입력될 수 있는 기능 값을 예상하고 분석하는 기술입니다. 프로그램의 변수에는 다양한 값이 입력될 수 있고 이 값에 따라 프로그램이 동작하게 됩니다. 값 분석은 특정한 값이 입력되었을 때 프로그램이 어떻게 동작할지를 예측하고, 잘못된 값이 사용되었을 때 발생할 수 있는 오류를 확인할 수 있습니다. 예를 들어, 배열의 인덱스가 배열의 크기를 초과하는지를 확인하거나, 정수 오버플로우가 발생할 수 있는 가능성을 탐지할 수 있습니다.

• 값의 범위 분석(Constraint Range Analysis)

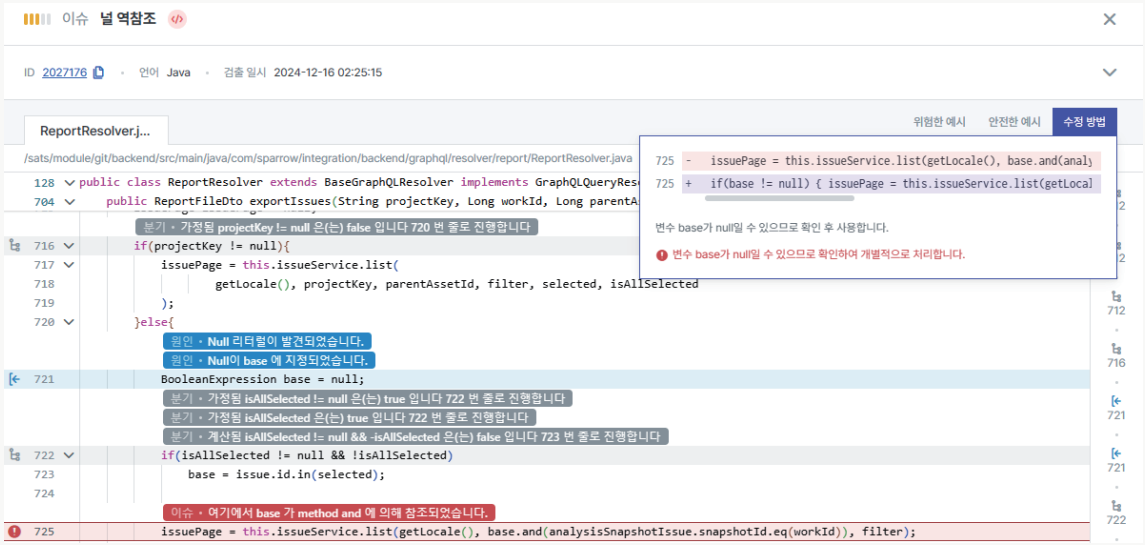
앞서 설명한 값 분석의 한 종류인 값의 범위 분석 기술을 연구하여 변수나 표현식이 가질 수 있는 값의 범위를 추정하는 기술을 구현했습니다. 값의 범위 분석 기술은 변수의 값이 예상 범위를 벗어나는지, 조건문에서 특정 조건이 항상 참이거나 거짓인지를 판단할 수 있습니다. 이 기술을 통해 반복문에서 무한 루프 발생 가능성을 확인하거나, 분기문에서 불필요한 조건을 식별함으로써 코드의 안정성과 신뢰성을 높일 수 있습니다. 특히 여러 식별자가 다른 유효 범위를 가지는 경우 해당 식별자를 적절히 병합하고 값을 선정하도록 설계함으로써 정탐율을 향상시켰습니다.

위와 같은 세부 분석 기술은 프로그램의 의미적 일관성과 정확성을 보장하기 위해 필수적인 기술입니다. 당사의 정적 분석은 스코프 분석, 제어 흐름 분석, 값 분석 및 값의 범위 분석 외에도 타입 검사, 데이터 흐름 분석 등 다양한 기법을 통해 프로그램의 잠재적 오류를 탐지하고, 소프트웨어의 품질과 안정성을 높이는 데 기여하고 있습니다.

2.3 액티브 서제스천 (Active Suggestion)

정적 분석 기술을 통해 보안 취약점이 검출되면 사용자는 취약점이 검출된 코드를 수정해야 합니다. 일반적인 정적 분석 도구에서는 취약점의 발생 원인이나 유형에 따라 케이스를 구분하여 대략적인 해결 방법을 제시하게 됩니다. 따라서 사용자가 구체적으로 코드를 어떻게 수정할 것인지는 제공되지 않을 수 있습니다. 반면, 당사의 액티브 서제스천 기술은 실제 코드를 수정하는 방법을 알려주는 기술입니다. 코드의 구조나 의미를 파악하고 대처할 수 없는 사용자의 경우에도 액티브 서제스천 기술에서 제공하는 수정 방법을 사용해서 취약점을 해결할 수 있습니다.

액티브 서제스천 기술을 이용한 수정 예시



액티브 서제스천은 코드를 수정하는 방법을 알아내기 위해 먼저 수정 예시 코드를 추론합니다. 이 때 정적 분석 기술의 모양 분석에서 생성한 추상구문트리(Abstract Syntax Tree)를 사용합니다. 앞서 설명한 것처럼 모양 분석 혹은 의미 분석 기술이 추상구문트리를 통해 취약점을 검출합니다. 따라서 취약점이 검출되지 않기 위해 수정해야 할 코드의 예시도 동일한 방법으로 추론할 수 있습니다. 다만 수정 예시 코드는 구체적인 수정 방법이 아니라 특정 취약점을 해결하기 위해 사용할 수 있는 분석한 코드에 대한 예시에 가깝습니다.

추상구문트리에서 수정 예시 코드를 성공적으로 추론하게 되면, 취약점이 검출된 소스 코드 원본에서 수정해야 할 코드의 영역을 계산합니다. 원본에서 삭제해야 하는 코드줄을 확인하고 원본에 추가해야 하는 코드를 생성한 다음, 미리 확인한 위치에 해당 코드를 추가합니다. 만약, 수정 코드를 생성할 때 코드만으로 표현할 수 없는 추론 결과가 있다면 사용자가 해당 내용을 유의하도록 미리 알리는 메시지를 생성합니다. 이런 메시지를 통해 사용자가 제시된 수정 코드를 직접 붙여넣더라도 추가적으로 확인할 수 있도록 지침을 줄 수 있습니다.

2.4 폭넓은 프로그래밍 언어 분석 지원

당사의 정적 분석은 프로그래밍 언어학 이론을 기반으로 하여 코드를 분석하는 분석 엔진을 개발해 왔습니다. 그 결과, 모든 언어에 공통적으로 존재하는 문법 구조나 규칙을 언어 체계로 표현할 수 있는 특정 엔진을 구현해두었습니다. 이렇게 설계된 엔진은 공통 모듈이라는 이름으로 불리며 정적 분석에서 탐지하는 검출 규칙 중 다수가 해당 모듈 위에서 동작하게 됩니다.

만약 공통 모듈 없이 프로그래밍 언어마다 다른 엔진을 제공해야 한다면 특정 언어를 분석하기 위해 문법과 구조를 분석하고 새로 프로그램을 개발해야 합니다. 따라서 특정 언어에 대한 요구가 증가하는 시점과 언어 분석을 지원하는 시점 사이에 상당한 시간이 필요합니다.

그러나 공통 모듈은 보편 문법(Universal Grammar)을 구현한 프로그램이기 때문에 Kotlin, Go, Rust와 같이 비교적 최근에 개발되어 사용 점유율이 빠르게 증가하고 있는 언어도 신속하게 분석 대상에 포함할 수 있었습니다. 이로 인해 당사는 Java 및 C/C++과 같은 전통적인 언어부터 최신 트렌드 언어까지 다양한 범주의 언어를 포함하여 총 25개 이상의 언어와 프레임워크를 분석할 수 있도록 지원하고 있습니다.

당사에서 분석을 지원하는 지원하는 프로그래밍 언어 및 프레임워크는 다음과 같습니다.

- 프로그래밍 언어: ABAP, Apex, ASP, ASP.Net MVC, C, C++, C#, Go, HTML, Java, JavaScript, JSP, Kotlin, Objective-C, PHP, Python, Rust, SQL, Swift, TypeScript, VB.NET, VBS, Visualforce, XSL, 각종 설정 파일
- 프레임워크: iBatis, MiPlatform, MyBatis, Nexacro, Node.js, Proframe, React, Spring, Struts, Vue.js, XPlatform, 전자정부 프레임워크

또한 C/C++ 언어의 경우, 아래와 같은 컴파일러 및 IDE 환경을 프리셋으로 지정함으로써 코드를 빌드한 후 분석하는 방법으로 일반적인 빌드 없는 분석에 비해 정교한 분석 기능을 제공합니다.

- 컴파일러: GNU C/C++, IBM AIX C/C++, HP-UX C/C++, Solaris C/C++
- IDE 프리셋: Visual Studio 6, 2005, 2008, 2010, 2012, 2013, 2015, 2017, 2019, 2022

2.5 다양한 컴플라이언스 지원

정적 분석 기술에서 탐지하는 검출 규칙의 기준은 국내의 경우 관련 법령과 정부 가이드라인에 따르고 있습니다. 또한 국제적인 컴플라이언스 및 코딩 표준이 요구하는 규칙을 폭넓게 지원하고 있습니다. 이를 통해 검출 규칙은 포괄적인 검출 범위를 보유한 동시에 정밀한 기준을 제공하게 되었습니다. 각 검출 규칙은 제공하는 범위나 기준에 따라 컴플라이언스에 매핑되어 있습니다.

따라서 사용자가 원하는 컴플라이언스에 해당하는 검출 규칙을 사용하여 정적 분석을 사용할 수 있습니다. 이런 방법으로 시간 대비 효율적으로 보안 취약점을 탐지하고 소프트웨어 품질을 개선할 수 있습니다.

• 국내 컴플라이언스: 행정안전부 소프트웨어 보안약점 진단가이드 (2021), JavaScript 시큐어코딩 가이드 (2022), Python 시큐어코딩 가이드 (2022), 방위사업청 코딩규칙, 무기체계 소프트웨어 보안약점 점검 목록

• 해외 컴플라이언스: .Net framework design guideline, .Net framework design guideline 2023, BSSC C/C++ 2000, CERT-C 2016, CERT-C++ 2016, CERT-Java, CWE 658(4.14), CWE-659(4.14), CWE-660(4.14), HIC++ 2.2, HIS 1.0.3, ISO26262, JPL, JPL Power of Ten, MISRA-C 2004, MISRA-C 2012, Amendment 1/2/3, MSDN-C# 2015, MISRA-C++ 2008, MSDN-C# 2023, Oracle Coding Standards, PA-DSS, PCI DSS

3 맺음말

지금까지 정적 분석 기술의 의미, 주요 기능의 동작, 특징에 대해 설명했습니다. 정적 분석과 같은 핵심 기술로 분석할 수 있는 범위는 지속적인 연구 개발을 통해 확장되고 있으며 정확도 또한 향상되고 있습니다. 그리고 이렇게 수행된 분석의 결과를 직관적으로 이해시킬 수 있도록 설명, 예시, 해결 방법 등을 포함한 다양한 컨텐츠를 포함하고 있습니다. 당사의 정적 분석을 사용하는 분석 솔루션을 직접 경험해보시고 당사에서 제공하는 기술 지원 및 컨설팅 서비스를 통해 소프트웨어 개발 및 운영에 활용해보시기 바랍니다.

추가적인 문의 사항은 스패로우 고객센터 (https://cs.sparrow.im/ko/tickets)를 방문하세요.