스마트 컨트랙트를 위한 SSDLC

스마트 컨트랙트를 위한 SSDLC 지침

Institute : 고려대학교 블록체인 연구센터
Date : 2025/06/10
Version : 2.0

<제목 차례>

<표 차례>

<그림 차례>

I. 목적

스마트 컨트랙트를 통한 디지털 금융 서비스와 같은 민감한 데이터와 자산의 처리가 중요하다. 이에 따라 스마트 컨트랙트의 신뢰성뿐만 아니라 보안성의 중요성도 대두되고 있다. 현재 디지털 금융 서비스와 같은 분야에서 스마트 컨트랙트는 핵심적인 역할을 수행하고, 민감한 데이터와 자산을 처리한다. 이러한 환경에서 스마트 컨트랙트의 신뢰성과 보안성은 그 어느 때보다 중요하며, 이는 기업의 명성, 고객의 신뢰 및 금융적 안정성에 직접적인 영향을 미친다. 따라서, 보안이 내재화된 스마트 컨트랙트 개발을 위한 Secure Software Development Life Cycle (SSDLC)의 적용은 선택이 아닌 필수적인 요소로 자리 잡고 있다.

본 문서에서는 스마트 컨트랙트의 신뢰성을 확보하고 보안을 내재화 시킬 수 있는 스마트 컨트랙트 개발 방법론을 제시한다. 신뢰성은 소프트웨어가 사용자의 요구 사항과 기대를 충족시키며, 예측 가능하고 일관된 방식으로 작동하는 능력을 의미하고, 보안성은 소프트웨어가 외부 공격과 위협으로부터 데이터와 시스템을 안전하게 보호할 수 있는 능력을 의미 한다.

SSDLC는 전통적인 소프트웨어 개발 생명 주기에 보안 요소를 통합하여, 개발 초기 단계부터 전 주기적으로 보안을 고려하는 프로세스를 의미한다. 이 접근법은 개발, 배포, 유지보수에 이르기까지 소프트웨어의 전체 생명 주기에 걸쳐 보안 내재화를 하고자 한다. 이러한 접근은 보안 취약점을 조기에 식별하고 해결하여, 최종 제품의 보안성을 크게 향상시킬 수 있다.

본 보고서는 스마트 컨트랙트의 개발에서부터 배포, 실행에 이르는 전체 생명 주기 동안 신뢰성과 보안을 강화하기 위한 Secure SDLC의 구체적인 적용 방법을 기술 한다. 이는 각 단계에서 수행해야 할 보안 활동을 포함하여, 보안이 내재화된 스마트 컨트랙트의 개발 과정을 체계적으로 설명하고, 각 단계의 중요성과 실행 방법에 대해 상세히 논의한다. 이를 통해, 개발자, 프로젝트 관리자, 보안 전문가들이 스마트

컨트랙트의 보안 위험을 효과적으로 관리하고, 최종 사용자에게 안전하고 신뢰할 수 있는 서비스를 제공할 수 있도록 지원하는 것을 목표로 한다. SSDLC을 통해 구현되는 개발 방법론은 스마트 컨트랙트의 신뢰성과 보안성을 극대화하여, 궁극적으로 범용적으로 믿고 사용할 수 있는 스마트 컨트랙트 환경에 크게 기여할 것이다.

Ⅱ. 용어 정의 및 약어

1. 위협(Threat) : 의도적이거나 그렇지 않거나 취약점을 악용하여 자산, 정보에 손상, 피해, 파괴 등을 할 수 있는 모든 것

2. 위험(Risk) : 취약점이 악용된 위협으로부터 발생한 정량적인 손상, 피해, 파괴 정도

3. 보안 약점(Weakness) : 소프트웨어 내 포함되는 bug, flaw, mistake, error 등 모든 비정상 코드

4. 취약점(Vulnerability) : 위협으로 악용되어 자산에 무단으로 접근할 수 있는 프로그램의 약점

5. 공격(Attack) : 자산 및 시스템에 접근하거나 손상을 입힐 수 있는 시도

6. SDLC(Software Development Life Cycle) : 소프트웨어 개발 생명 주기, 소프트웨어를 개발하기 위해 명확하게 정의된 개발 방법론

7. SSDLC(Secure-Software Development Life Cycle) : 소프트웨어를 개발하기 위해 명확하게 정의된 방법론에서 단계별 보안 활동이 추가된 개발 방법론

8. 소프트웨어 개발 방법론 : 소프트웨어 공학 원리를 소프트웨어 개발 생명 주기에 적용한 개념으로 시스템 개발을 위한 활동, 절차, 산출물, 기법 등을 체계적으로 정리한 것

Ⅲ. 스마트 컨트랙트 개발을 위한 SSDLC 연구

가. Decentralized App(DApp) 개요 및 특징

DApp은 중앙화된 서버가 아닌 이더리움, EOS와 같은 블록체인 플랫폼에서 운영되는 소프트웨어 개발의 패러다임 전환을 의미합니다. DApp은 P2P 네트워크에서 실행되며 기존의 중앙 집중식 애플리케이션에서 크게 벗어났습니다. DApp의 특징은 다음과 같다.

1. Decentralized - 이더리움 및, EOS 와 같은 플랫폼에서 작동하며 중앙에서 통제할 수 없는 공개 구조로 작동한다.

2. Deterministic - 실행 환경에 관계없이 일관되게 수행한다.

3. Turing complete - 필요한 리소스가 주어지면 모든 작업을 실행 할 수 있다.

4. Isolated - 가상 환경에서 실행되므로 버그가 발생하더라도 블록체인 네트워크의 정상적인 기능을 방해하지 않는다.

DApp은 분산된 특성으로 인해 단일 제어 주체 없이 여러 사용자가 콘텐츠를 생성하고 사용할 수 있다. 해당 구조를 지니기 위해서는 아래와 같은 기본적인 요구사항이 적용된다.

1. Decentralized Nature : 모든 데이터는 분산형 블록체인이나 암호화 기술을 사용해 저장해야한다.

2. Open Source : 소스 코드를 모든 사람이 사용하거나 볼 수 있다. 또한, 변경 사항(change)에 대해서는 사용자 전체 또는 다수에 의해 결정된다.

3. Incentivization : 블록체인 네트워크 내 유효성 검증에 참여한 검증자에게 토큰 및 가치가 있는 디지털 자산의 형태로 보상을 부여한다.

4. Algorithm : 암호화 시스템에서 가치증명을 나타내는 PoV 혹은 PoS 와 같

은 합의 메커니즘이 필요하다.

분산앱을 구축하는 방법은 기존 중앙 집중식 애플리케이션이나 소프트웨어를 구축하기보다 어렵다. 분산앱을 구축하기 위해서는 일반적으로 아래의 다섯 단계로 구성된다.

1. 적절한 비즈니스 사용 사례 식별

2. 아이디어를 이해할 수 있도록 개념증명(POC) 작성

3. 시각 및 기술 디자인 작업 시작 및 플랫폼 파악

4. 테스트 네트워크에서 개발한 것을 테스팅

5. 운영 서버에서 분산앱을 실행

DApp은 상호 작용이나 배포에 실제 ID가 필요하지 않기 때문에 상당한 개인 정보 보호 이점을 제공한다. 네트워크의 어떤 단일 엔티티도 사용자가 블록체인에서 트랜잭션을 제출하거나 배포하거나 데이터를 읽는 것을 차단할 수 없다. 블록체인에 저장된 데이터의 불변성은 정보의 무결성을 보장하여 악의적인 행위자가 정보를 변조하거나 변경하는 것을 방지한다. 중앙 기관에 의존하지 않고도 스마트 계약과 그 실행을 분석할 수 있으나, 블록체인의 코드와 데이터를 수정하는 것이 어렵기 때문에 DApp을 유지가 어려운 점도 존재한다. 다른 문제로는 모든 트랜잭션을 실행하고 저장하는 모든 노드로 인한 성능 오버헤드, 잠재적인 네트워크 트래픽 정체, 중앙 집중식 시스템에 익숙한 사용자가 블록체인 기반 분산형 앱과 상호 작용하는 데 필요한 도구에 적응해야 하는 복잡성 등이 있다.

DApp은 은행 업무, 비즈니스 프로세스 관리 등 다양한 분야에서 실용적인 응용 프로그램을 제공한다. 이를 통해 원활한 국제 거래를 촉진하고, 통화 관리 및 이체를 강화하며, 중개자 없이 대출 등의 프로세스를 신속하게 처리하여 금융 운영의 효율성을 높일 수 있을 것으로 보이며, 추후 물류 분야에서는 구매자와 판매자 간의 스마트 계약을 통해 결제를 자동화하여 전체 프로세스를 간소화 하는 등 사람의 개입 없이 프로세스를 자동화에 있어 큰 이점을 가질 것 이다.

나. SCSVS(Secure Smart Contracts Security Verification Standard)

SCSVS란 개발자, 설계자, 보안 담당자를 위한 스마트 컨트랙트의 표준화된 보안을 제시하기 위해 만들어진 14개 부분으로 구성된 체크리스트이다. 이 목록은 스마트 컨트랙트의 설계에서 구현까지의 단계에서 가이드 라인을 제공하여 알려진 보안 문제를 파악 할 수 있다. SCSVS의 큰 목표는 아래의 4항목과 같다.

1. 스마트 컨트랙트의 고품질 코드 개발
2. 알려진 취약점을 완화
3. 보안 담당자를 위해 체크리스트 제공
4. 적용 범위와 관련하여 스마트 컨트랙트가 얼마나 명확하고 신뢰할 수 있는지를 제공

SCSVS 체크리스트는 일종의 위협 모델링의 일환으로 스마트 컨트랙트의 보안 및 완성도를 측정할 수 있다. 또한, 해당 지표를 활용하여 보안 감사를 진행할 수 있다. 해당 체크리스트를 통해 개발자가 개발할 시 자체적으로 보안 내재화를 시킬 수 있는 이점이 존재하며, 보안과 관련된 개발 항목들을 손쉽게 확인할 수 있다. 체크리스트는 총 14개의 주요영역으로 나뉘며 14개의 주요영역은 아래와 같다.

1. 아키텍처, 설계 및 위협 모델링

- 목표
  - 안전한 스마트 계약 아키텍처를 만들고 구현하기 전에 철저한 위협 모델링을 설계하고 수행
- 요구사항
  - 모든 관련 스마트 계약을 식별하고 적절하게 활용합니다. 설계 단계에서 특정 보안 가정을 고려

2. 액세스 제어

- 목표
  - 정보 및 관련 정보 처리 서비스를 획득 및 사용에 대한 특정 요청을 승인하거나 거부하는 프로세스를 관리

- 요구사항
  - 잘 정의된 역할과 권한으로 사용자와 계약을 연결, 승인된 사용자 및 계약으로만 액세스 제한

3. 블록체인 데이터

- 목표
  - 특히 비밀 데이터를 안전하게 저장하기 위한 메커니즘이 내장되어 있지 않은 퍼블릭 블록체인 스마트 계약에서 중요한 데이터를 읽는 무단 행위자로부터 보호
- 요구사항
  - 스마트 컨트랙트에 저장된 데이터를 식별하고 보호
  - 평문으로 민감한 데이터를 블록체인에 저장하면 안됨
  - 데이터가 잘 못 전송되지 않았는지 확인

4. 통신

- 목표
  - 스마트 컨트랙트와 해당 라이브러리 간의 관계에 대한 주제를 포함
- 요구사항
  - 외부 호출로부터의 악용사례를 고려
  - Openzepplin 과 같은 최신 보안 라이브러리 사용

5. 산술

- 목표
  - 산술 범주는 스마트 컨트랙트의 수학적 연산을 다룸
- 요구사항
  - 모든 수학 연산과 극한 변숫값은 안전하고 예측 가능한 방식으로 처리

6. 악성 입력 처리

- 목표

  - 악의적인 입력 처리는 스마트 컨트랙트에서 매개변수로 얻은 값을 검증
- 요구사항
  - 전달되는 함수 매개변수는 안전하고 예측 가능한 방식으로 처리

7. 가스 사용 및 제한 사항

- 목표
  - 최적화 측면에서뿐만 아니라 가능한 고갈을 방지하기 위한 안전 측면에서도 고려
- 요구사항
  - 가스 사용을 최적화하고 불필요한 손실을 방지
  - 구현은 스마트 컨트랙트의 제한 사항에 따름

8. 비즈니스 로직

- 목표
  - 비즈니스 로직의 보안을 고려
  - 사용된 구성 요소는 검증 없이 안전한 것으로 간주되어서는 안됨
  - 비즈니스 가정은 스마트 컨트랙트를 구축하는 데 필수적인 최소 신뢰 원칙을 충족
- 요구사항
  - 비즈니스 로직 흐름은 순차적
  - 비즈니스 제한이 지정되고 시행
  - 암호화폐 및 토큰 비즈니스 로직 흐름은 남용 사례와 악의적인 행위자를 고려

9. 서비스 거부

- 목표
  - 스마트 컨트랙트의 가용성과 DoS 위험을 최소화할 수 있는 가능한 방법에 중점
- 요구사항

- 계약 논리는 계약의 가용성에 영향을 미치는 것을 방지

10. 토큰

- 목표
  - 토큰 구현은 확립된 표준을 따라야 함
- 요구사항
  - 구현된 토큰은 보안 표준을 따름

11. 코드 명확성

- 목표
  - 코드를 명확하고 단순하게 유지하면 취약점을 더 쉽게 발견, 수정 가능
- 요구사항
  - 코드가 명확하고 읽기 쉬움
  - 코드에 원하지 않거나 사용하지 않는 부분이 없음
  - 변수 이름은 모범 사례와 일치
  - 사용 중인 기능은 안전

12. 테스트 범위

- 목표
  - 테스트 되는 시기와 대상이 테스트 커버리지가 중점을 두는 방법이 중요
- 요구사항
  - 사양은 공식적으로 테스트
  - 구현은 정적, 동적으로 테스트
  - 구현은 symbolic execution을 사용하여 테스트

13. 알려진 공격

- 목표
  - 알려진 공격으로부터 보호되도록 하는 것

14. 분산 금융

- 목표
  - 안전한 스마트 컨트랙트 생성 아키텍처, 설계 및 위협 모델링
  - 스마트 컨트랙트를 구현하기 전에 가능한 모든 위협을 고려
- 요구사항
  - 탈중앙화 금융이 관리하는 자산은 안전하며 권한이 없는 사람이 인출할 수 없음
  - 탈중앙화 금융 데이터 소스는 조작할 수 없음

다. SSDLC 기법 및 개요

SSDLC(Secure Software Development Life Cycle)는 소프트웨어 개발 프로세스의 각 단계에 보안 관행을 통합하는 프레임워크이다. 소프트웨어 프로젝트 시작부터 보안 조치를 내장하도록 설계되어 보안 고려 사항이 나중에 추가되는 것이 아니라 소프트웨어 개발 전주기에 보안적 요소를 즉각 고려하여 소프트웨어를 만든다. SSDLC는 요구 사항 수집, 설계, 코딩, 테스트, 배포, 유지 관리 등 소프트웨어 개발의 다양한 단계에 걸쳐 모든 단계에서 보안을 내재화 하는 것을 목표로 한다.

요구 사항 수집 및 분석 단계에서는 기능 요구 사항과 함께 보안 요구 사항이 식별된다. 이 단계에는 보안 약점과 취약점을 식별하고 개발 중인 소프트웨어의 보안 영향을 이해하기 위한 위험 평가가 포함된다. 또한 소프트웨어의 표준을 확인하고 보안적 요소를 식별하는 등의 요구 사항을 정의하는 것도 포함된다.

설계 단계에서 소프트웨어를 구체적으로 명세화하는 단계로, 내부 처리 기능을 나타내고, 자료를 정의하며 순서도를 그리는 작업을 포함한다. SSDLC의 경우에는 추가적으로 잠재적인 보안 문제를 예측하고 이러한 위험을 완화하는 방식으로 아키텍처를 설계하기 위해 위협 모델링과 같은 기술을 포함한다. 최소 권한, 심층 방어, 오류 방지 기본값과 같은 보안 설계 원칙을 적용하여 보안이 내재화된 소프트웨어를 구축한다.

구현 단계에서는 시큐어 코딩 가이드를 사용하여 가이드에 따라 방어적으로 코딩하는 과정이 포함된다. 개발자는 SQL 삽입, 크로스 사이트 스크립팅, 버퍼 오버플로와 같은 일반적인 취약점을 방지하기 위해 표준화된 보안 코딩 지침을 따르고, 수동 및 자동 코드 검토 방식은 이 단계에서 중요한 역할을 하며 코드의 보안 결함을 조기에 감지하고 해결할 수 있다.

테스트는 다양한 형태의 보안 테스트를 포함하는 SSDLC의 광범위한 단계입니다. SAST(정적 애플리케이션 보안 테스트) 및 DAST(동적 애플리케이션 보안 테스트)는 취약점을 식별하는 데 활용한다. 소프트웨어에 대한 실제 공격을 시뮬레이션하기 위해 모의 해킹도 수행된다. 이 단계의 목표는 소프트웨어가 명세대로 작동하는지를 확인하는 과정을 넘어 이상 값이 입력으로 들어와도 소프트웨어가 의도한 대로 작동하는지 확인 한다.

SSDLC 배포에는 소프트웨어를 배포하기 전에 최종 보안 검토가 수행되고 보안 업데이트 및 패치를 계획하여 소프트웨어가 수명주기 동안 안전하게 유지되도록 보장한다.

유지 관리는 개발이 완료되어 운영중인 소프트웨어가 제대로 작동 할 수 있게 관리하는 단계로, 소프트웨어에 새로운 취약점이 있는지 모니터링하고 정기적인 업데이트와 패치하는 것을 포함 한다. 또한, 보안 감사 및 규정 준수 검사를 진행하여 소프트웨어가 계속해서 보안 표준을 충족하는지 확인하는 과정을 포함한다.

SDLC에 비해 전 주기적으로 보안을 내재화 시키는 SSDLC에서는 Shift Left, 보안 교육 등을 내포 한다. Shift Left 란, 소프트웨어 개발 프로세스 초기에 보안을 통합하는 관행을 의미합니다. 이는 나중 단계나 소프트웨어 배포 후에 기다리기보다는 개발 초기 단계에서 보안 문제를 해결하는 것입니다. 이 접근 방식은 보안 문제를 조기에 식별 및 수정하여 수정 비용과 복잡성을 줄이는 데 도움이 될 뿐만 아니라 보안이 보안 전문가뿐만 아니라 모든 팀 구성원 간에 공유 책임이 되는 문화를 조성합니다. 또한, 보안 교육과 같은 관행들을 기존의 개발 생명 주기에 포함시킴으로써 보안을 내재화 시킬 수 있습니다. 이러한 생명 주기에 보안을 내재화 시키는 과정을 포함하여 SSDLC 구축하고 이러한 기법을 따르면서 소프트웨어 개발을 진행하게 되면 보안 사고로 인한 피해나, 시간, 비용들을 확연히 줄일 수 있다.

현재 범용적으로 사용하고 있는 SSDLC의 종류로는 MS-SDL, BSI 7 touch point, DevOps에서 전 주기적으로 보안적 속성을 추가한 DevSecOps 등이 존재한다. 그 중 DevSecOps를 사용하여 스마트 컨트랙트를 위한 SSDLC를 구축 하여 MS-SDL 또는 BSI 7 터치포인트와 같은 기존 방법론과 달리 지속적인 보안 통합을 통해 전 주기적으로 보안 내재화를 시킬 수 있다.

IV. DevSecOps

개발과 운영의 합성어인 DevOps는 개발자와 운영팀간의 통합과 협업을 강조하는 소프트웨어 개발 방법론이다. 협업 문화와 공유 문화를 촉진하여 지속적 통합, 전달 및 신속한 배포에 중점을 두고 개발 주기를 단축하고 민첩하게 대응 할 수 있는 것을 목표로 한다. 그러나 전반적인 효율성의 상승과는 달리 통합 보안 측면에서는 적절히 대응하지 못해 보안 사고에 대한 대응을 못 하는 단점이 있다. 이를 방지하기 위해 전 주기적으로 보안을 내재화 시키는 방식인 DevSecOps가 탄생하게 된다.

가. DevOps 개념

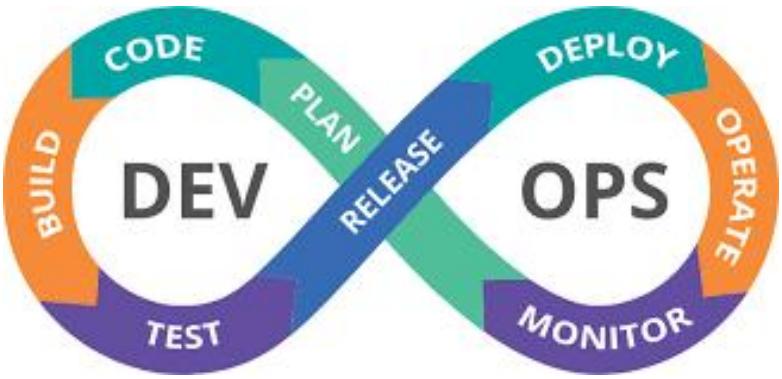


그림 1. DevOps

'개발(Development)'과 '운영(Operations)'의 합성어인 DevOps는 개발팀과 운영팀 간의 향상된 커뮤니케이션과 협업을 강조하는 소프트웨어 프로젝트 관리의 혁신적인 접근 방식입니다. 그림 3에 설명된 것처럼 DevOps는 이러한 팀 간의 격차를 해소하여 소프트웨어 및 서비스의 개발 및 배포를 가속화하는 것을 목표로 한다. DevOps는 크게 5가지 주요 핵심 원칙을 제안하고 있다.

1. 문화(Culture) : 협업을 핵심 문화 측면으로 강조

2. 자동화(Automation) : 테스트 및 지속적 배포의 핵심 요소이다.

3. 간소화(Lean) : 낭비를 최소화하여 고객에게 가치를 빠르게 제공하는 사고방식으로 DevOps 문화를 통해 끊임없이 문제를 해결하고 개선이 필요하다.

4. 측정(Measurement) : 보다 효율적이고 좋은 결과를 위한 피드백과 세부적인 계획을 수립하고, 상호간의 피드백을 하는 과정이 필요하다.

5. 공유(Share) : 결과와 상관없이 실패, 성공의 사례를 다른 사람들이 배울 수 있도록 서로의 경험을 공유할 수 있어야 한다.

DevOps의 단계별 수행 활동은 다음과 같다.

1. 계획 단계
요구사항, 방법론 그리고 절차 등을 미리 생각하여 계획한다.

2. 코드 단계
코드의 개발 및 검토, 사용하는 도구의 버전 관리, 코드 병합을 하며 관련 내용이나 명세를 정리하여 관련된 인원들과 공유한다.

3. 빌드 단계
개발된 코드를 정적분석, 보안 분석을 필히 수행하고 수정한 후 빌드한다. 자동화 도구를 활용하여 코드에 문제가 없는지 주기적으로 검사한다.

4. 테스트 단계
요구사항에 따라 계획한 기능이 제대로 작동되는지 확인 하고, 이상 값에 대한 테스트도 진행하여 개발된 소프트웨어가 제대로 수행하는지 확인한다.

5. 구현단계
완성된 결과물을 개발 서버나 품질서버에 이관한다.

6. 배포단계
서비스를 진행하게 될 운영 서버 및 클라우드에 배포한다.

7. 운영단계

개발된 소프트웨어를 적용한다.

8. 모니터 단계

안정적인 운영을 지속해서 하기 위해 존재하는 것으로 개선할 사항이나 오류 사항을 파악하고 대응하게 소프트웨어가 제대로 작동할 수 있게 유지보수하는 것을 포함한다.

개발팀과 운영팀이 통합된 DevOps는 소프트웨어 개발의 생산성을 크게 향상시킵니다. 이 방법론은 이러한 팀 간의 커뮤니케이션 격차를 해소할 뿐만 아니라 개발, 검증 및 배포를 포괄하는 전체 소프트웨어 주기에 대한 책임을 중앙 집중화한다. 이러한 통합 접근 방식을 통해 전반적인 개발 속도가 빨라지고 제품 출시 시간이 단축되는 이점을 가지며, 팀 구성원의 코드와 관련하여 소유권과 책임감이 높아져 개발 프로세스가 더욱 간소화된다.

그러나 DevOps 하에서 다양한 팀을 통합하려면 효과적인 적응을 위해 충분한 시간이 필요하다. 특히 빈번한 코드 배포가 필요하지 않은 시나리오에서는 이 접근 방식이 비용 증가로 이어질 수 있어 환경에 따라 적합하게 선택을 해야한다. DevOps 구현에 있어 또 다른 중요한 과제는 개발 및 운영 작업을 모두 처리할 수 있는 전문 자동화 도구가 필요하고, 보안보다 속도를 우선시하는 DevOps에는 내재된 위험이 있으므로 신중한 관리가 필요하다. 또한 DevOps 관행에 정통한 숙련된 인재를 찾는 것 또한 하나의 과제이다.

DevOps의 핵심 전략에는 빈번한 소규모 업데이트가 포함되어 있어 각 배포와 관련된 위험을 줄이고 더 빠른 버그 해결이 가능하다. 마이크로서비스 아키텍처를 채택하면 복잡한 시스템을 더 간단하고 독립적인 프로젝트로 분할하여 애플리케이션의 유연성과 혁신 속도를 높일 수 있다. 지속적 통합 및 지속적 전달 방식은 빈번한 배포로 인해 발생하는 운영 문제를 해결하고 빠르고 안전한 소프트웨어 업데이트를 지원하는 데 도움이 된다. 코드형 인프라는 온프레미스 및 클라우드 기반 설정 모두에 적용할 수 있는 신속하고 일관된 인프라 관리를 위한 중요한 접근 방식이다. 모니터링과 로깅은 소프트웨어의 유지보수 측면에서 상당히 중요한 역할을 하며, 실시간 분석과 사전 서비스 모니터링을 통해 성능 문제나 예상치 못한 변화에 신속하게 대응할 수 있다. 또한 개발, 운영, 마케팅, 영업 등 다양한

팀 전반에 걸쳐 DevOps를 통해 촉진된 향상된 커뮤니케이션 및 협업은 조직 목표 및 프로젝트 목표 달성을 위한 응집력 있는 접근 방식을 주도한다. 이러한 방식들을 함께 사용하면 조직이 더 빠르고 안정적으로 고객에게 업데이트를 제공할 수 있다. 주요 DevOps 방식은 아래와 같다.

1. 지속적 통합

지속적 통합은 자동화된 빌드 및 테스트가 수행된 후, 개발자가 코드 변경 사항을 중앙 저장소에 정기적으로 병합하는 소프트웨어 개발 방식, 지속적 통합의 핵심 목표는 버그를 신속하게 찾아 해결하고, 소프트웨어 품질을 개선하고, 새로운 소프트웨어 업데이트를 검증 및 릴리스하는 데 걸리는 시간을 단축

2. 지속적 전달

지속적 전달은 프로덕션에 릴리스하기 위한 코드 변경이 자동으로 빌드, 테스트 및 준비되는 소프트웨어 개발 방식, 빌드 단계 이후의 모든 코드 변경사항을 테스트 환경 및/또는 프로덕션 환경에 배포함으로써 지속적 통합을 확장, 지속적 전달이 적절하게 구현되면, 개발자는 언제나 즉시 배포할 수 있고 표준화된 테스트 프로세스를 통과한 빌드 결과물을 보유

3. 마이크로 서비스

마이크로 서비스 아키텍처는 단일 애플리케이션을 작은 서비스의 집합으로 구축하는 설계 접근 방식이다. 각 서비스는 자체 프로세스에서 실행되고, 주로 HTTP 기반 API와 같은 간편한 메커니즘을 사용하는 잘 정의된 인터페이스를 통해 다른 서비스와 통신, 마이크로 서비스는 비즈니스 기능을 중심으로 구축되며, 각 서비스는 단일 목적으로 한정되어 있다. 다양한 프레임워크 또는 프로그래밍 언어를 사용하여 마이크로 서비스를 작성하고, 이를 독립적으로 단일 서비스 또는 서비스 그룹으로 배포

4. 코드에서의 인프라

조직에 on-premise data center가 있든, 완전히 클라우드에 있든 관계없이 인프라를 빠르고 일관되게 프로비저닝, 구성 및 관리할 수 있다. 엔지니어는 코드 기반 도구를 사용하여 인터페이스, 애플리케이션 코드를 다루는 방법과 유사한 방식으로 인프라를 다룰 수 있다. 인프라가 코드를 통해 정의되므로

인프라와 서버를 표준화된 패턴을 사용하여 배포하고, 최신 패치와 버전으로 업데이트하거나, 반복 가능한 방식으로 복제할 수 있다.

5. 모니터링 및 로깅

조직은 지표와 로그를 모니터링하여 애플리케이션 및 인프라 성능이 제품의 최종 사용자 경험에 어떤 영향을 미치는지 확인, 조직은 애플리케이션과 인프라에서 생성되는 데이터 및 로그를 캡처하고 분류한 다음 이를 분석함으로써 변경 또는 업데이트가 사용자에게 어떤 영향을 주는지 이해하고, 문제의 근본 원인 또는 예상치 못한 변경에 대한 통찰력을 확보, 서비스는 연중무휴 24시간 사용할 수 있어야 하고 애플리케이션 및 인프라 업데이트 빈도가 증가함에 따라 적극적인 모니터링이 점점 더 중요해지고 있다. 이러한 데이터에 대한 실시간 분석을 수행하거나 알림을 생성하는 것도 조직이 좀 더 능동적으로 서비스를 모니터링하는 데 도움이 된다.

6. 커뮤니케이션 및 협업

조직에서 커뮤니케이션과 협업이 증가하는 것도 DevOps의 주요 문화적 측면 중 하나이다. DevOps 도구 및 소프트웨어 제공 프로세스 자동화를 사용하면 개발 및 운영의 워크플로와 책임을 물리적으로 합침으로써 협업이 이루어진다. 해당 팀에서는 이 위에 채팅 애플리케이션, 문제 또는 프로젝트 추적 시스템, 커뮤니케이션을 지원하고 정보를 공유하는 강력한 문화적 표준을 확립, 이를 통해 개발자와 운영 그리고 마케팅이나 영업과 같은 다른 팀 간에도 커뮤니케이션이 활발해지면서 조직의 모든 부분에서 목표와 프로젝트에 좀 더 가깝게 다가갈 수 있다.

나. 스마트 컨트랙트를 위한 DevOps

블록체인 솔루션에 대한 DevOps를 제공하는 핵심 개념은 다른 소프트웨어와 다르지 않다. 핵심 개념은 같지만, 블록체인 애플리케이션의 다자간 특성과 원장의 불변성은 블록체인 솔루션에 대한 DevOps 접근 방식을 결정할 때 고려해야 사항들이 아래와 같이 존재한다.

1. 권리 할당

2. 권한을 제거하기 위한 절차

3. 권한 추가 절차

4. 접근 권한 관리

5. pull 요청 권한

6. 승인 프로세스

7. 배포 시기에 대한 요구사항

스마트 컨트랙트를 테스트할 때 직면하는 첫 번째 문제 중 하나는 샘플 데이터의 생성이다. 블록체인에 테스트 데이터를 채우는 것은 다른 유형의 데이터 저장소보다 복잡하다. 블록체인은 시간 순서로 연결된 트랜잭션 블록의 결과이다. 개별 트랜잭션은 주기적으로 블록으로 그룹화되고 블록체인 네트워크의 모든 노드 중에서 블록을 검증하기 위한 일부 작업의 결과로 마지막 노드에 추가된다. 그런 다음 유효한 블록이 네트워크로 브로드캐스팅 된다. 스크립트로 블록체인을 자동으로 채울 수 있지만, 거래의 조정과 순서가 필요하며 관련된 여러 당사자의 맥락에서 이루어져야 한다. 이 결과를 달성하기 위한 새로운 추세는 기존 블록체인의 포크라고 하는 스냅샷을 만드는 것이다. 이것은 종종 원장을 채우기 위해 합성 트랜잭션을 생성하는 시스템을 구축하는 복잡성 없이 같은 결과를 제공할 수 있다.

모든 블록체인 솔루션이 서비스 소비자에게 열쇠를 노출하는 것은 아니지만

모든 시나리오에 키가 나타난다. 종단 간 솔루션의 세부 사항에 따라 인증서, 암호화 및 기타 비밀 및 안전하게 관리해야 하는 API 키도 있을 수 있다. 주요 고려사항 및 관련 조치는 다음과 같다.

1. 비밀 관리

2. 키 관리

3. 인증서 관리

4. 하드웨어 보안 모듈이 필요한지 확인

무엇을 어떻게 테스트할지 합의점을 마련하는 것도 중요하다. 조직마다 우선순위가 다를 수 있으며 스마트 컨트랙트처럼 공유 자산에 대해 무엇을 테스트해야 하는지에 대한 공통된 이해를 확립하는 것이 중요하다.

스마트 컨트랙트는 블록체인 네트워크에서 참여자 간 비즈니스 시나리오에 대한 공유 상태와 논리를 나타내지만, 구성원들이 해당 조직에 특화된 기존 애플리케이션을 통해 스마트 컨트랙트와 상호작용할 수 있다.

DevOps 전략으로 공유 자산에 대한 변경으로 영향을 받을 수 있는 코드의 테스트를 트리거할 위치도 포함해야 한다.

우선순위에 대한 시각이 다른 수십 명의 고객이 있을 수 있다. 그러므로 여러 조직이 기능 범위 지정 및 우선순위 지정, 릴리스 범위 지정, 다양한 환경에 대한 소프트웨어 릴리스 프로세스, 테스트 및 운영을 위한 공유 환경에 대한 배포 타이밍에 대한 고려사항을 정의하고 합의하는 방법에 대한 합의서를 작성하는 것이 중요하다.

일부 사용 사례에서 스마트 컨트랙트는 특정 기간만을 사용하지 않은 프로세스로 사용된다. 테스트에 대한 주요 고려사항은 변경 불가능한 블록체인에 대한 새 버전의 컨트랙트를 릴리스하고, 이러한 요구사항을 테스트에 통합하는 요구사항을 이해하는 것이다. 만약, 계약에 두 가지 버전이 있는 경우 기존 버전의 계약

을 처리하는 방법을 결정해야 한다. 어떤 경우에는 기존의 진행 중인 스마트 컨트랙트가 계속 작동하고 다른 경우에는 최신 버전의 계약으로 마이그레이션해야 한다. 예를 들어, 보험이 새로운 요구사항이 있는 새로운 상품 프로세스를 도입하는 경우 해당 프로세스를 신규 상품 또는 특정 기간이 적용되어있는 상품에만 적용하기를 원할 수 있다. 또 다른 경우에는 규정, 법률, 회사 측 상황들이 처리해야 하는 방법을 변경할 수 있다. 요구사항에 따라 기존 계약 업그레이드, 여러 버전의 계약에서 호환성 등을 포함하는 테스트를 고려해야 한다. 성공적으로 테스팅을 하고 나서 릴리스 파이프라인에서 파일을 복사하고 릴리스 파이프라인에서 사용할 빌드 아티팩트를 게시할 수 있다.

다. DevSecOps 개념

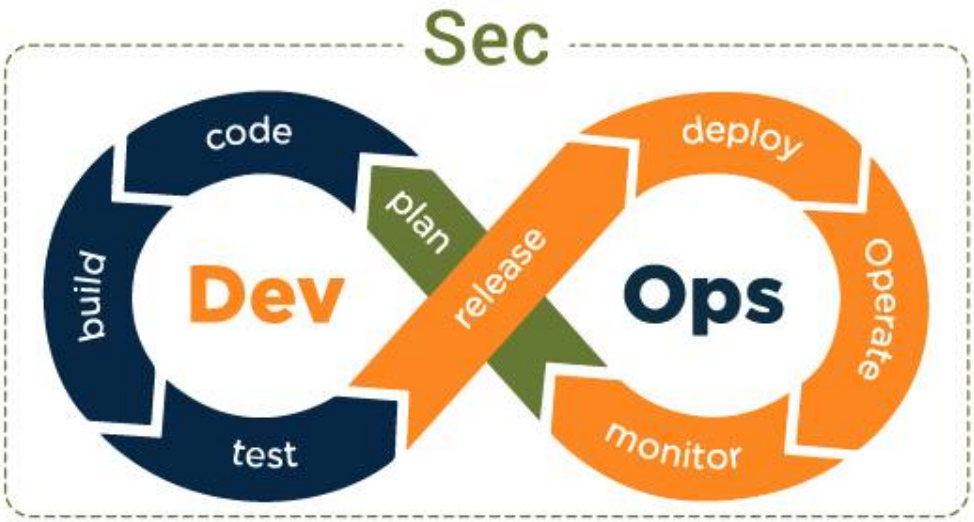


그림 2. DevSecOps

DevOps의 진화인 DevSecOps는 보안의 중요한 요소를 기존의 개발 및 운영 융합에 통합한다. 이는 개발자, 운영자 및 현재 보안 검증자가 소프트웨어 개발 프로젝트 전반에 걸쳐 함께 작업하는 협업 환경을 강조한다. 기존 DevOps에서는 주로 효율성을 높이고 소프트웨어 제공 시간을 단축하기 위해 개발 및 운영을 통합하고 자동화하는 데 중점을 둔다. DevSecOps는 보안을 전 주기에 통합하여 보안 고려 사항이 나중에 고려되는 것이 아니라 개발 수명 주기의 기본 구성 요소가 되도록 보장함으로써 보안을 내재화 시킨다.

특히 현대 소프트웨어 환경에서 DevSecOps의 필요성은 소프트웨어의 복잡성이 증가하고 여러 방면에서의 공격으로 인해 소프트웨어의 전 주기에서 보안을 분리할 수 없다는 인식에 의해 주도되고 있다. 기존 DevOps 모델은 소프트웨어 제공 속도를 높이고 기능을 개선하는 데 효과적이지만 보안적 측면을 간과하여 생성된 소프트웨어의 잠재적인 취약성을 초래한다. DevSecOps는 개발 초기 단계부터 배포 및 유지 관리에 이르기까지 보안을 모든 팀 구성원의 공동 책임으로 도입하여 이러한 문제를 완화하고 있다.

DevSecOps는 기존의 방법론에 여러 가지 보안 내재화 활동을 추가하여 달성

한다. 첫째, 소프트웨어 개발 주기의 모든 단계에서 보안을 고려하는 것의 중요성에 대해 팀 구성원을 지속적으로 교육하는 것에서부터 시작된다. 이러한 교육적 측면을 통해 보안 전문가뿐만 아니라 모든 팀 구성원이 잠재적인 보안 문제를 식별하고 해결할 수 있는 소양을 갖추게 된다.

또한, DevSecOps는 정적 및 동적 분석과 같은 신뢰성 및 보안성을 강화 시키는 과정을 거친다. 정적 분석에는 코드를 실행하지 않고 코드의 취약점을 검사하는 작업이 포함되는 반면, 동적 분석에는 잠재적인 보안 문제를 식별하기 위해 런타임 환경에서 코드를 테스트하는 작업이 포함된다. 이러한 관행을 통해 보안 결함을 조기에 감지하고 해결할 수 있으므로 최종 제품에서 취약점이 발생할 가능성이 줄어든다.

DevSecOps의 또 다른 특징은 개발 프로세스 초기에 보안을 통합하는 관행인 'Shift Left'을 강조한다는 것이다. 순수 보안과 관련된 업무를 보안 관계자만 하는 것이 아니라 개발자와 엔지니어 또한 필수 참여자로 구성시킴으로써 보안 고려사항을 따로 검토하는 것이 아니라 개발자가 개발을 할 때, 엔지니어가 관리를 할 때 즉 모든 상황에서 보안을 고려 시킨다. 이러한 접근 방식을 통해 보안 위험을 조기에 식별하고 해결할 수 있으므로 최종 제품이 기능적일 뿐만 아니라 보안도 보장된다. 이로인해, 보안적 요소를 고려하고 제품을 만든느 시간과 리소스를 효율적으로 줄일수 있게 된다.

또한 DevSecOps는 개발, 보안 및 운영 팀 간의 협업을 강화하여 문제가 발생할 때 이에 대응할 수 있는 집단적 능력을 향상시킨다. 이러한 협업을 통해 취약점을 해결하는 데 필요한 시간이 단축되고 모든 팀 구성원이 보안에 대한 접근 방식을 조정할 수 있다.

DevSecOps는 SDLC의 모든 단계에 보안을 통합함으로써 소프트웨어가 현대 디지털 환경의 증가하는 복잡성과 요구에 맞춰 보안에 대한 포괄적인 접근 방식으로 구축되도록 보장한다. DevSecOps는 보안에 대한 공동 책임 문화를 조성할 뿐만 아니라 설계상 안전한 소프트웨어를 생성하여 끊임없이 진화하는 위협 환경에 대한 향상된 보호 기능을 제공하여 효율적으로 소프트웨어를 개발 할 수 있게 된다.

V. 스마트 컨트랙트를 위한 DevSecOps

스마트 컨트랙트를 위한 SSDLC를 DevSecOps로 통합하는 것은 주로 블록체인 기술에 내재된 고유한 과제와 특성을 해결하기 때문에 점점 더 전략적 결정으로 간주되고 있다. MS-SDL 또는 BSI 7 터치포인트와 같은 기존 방법론과 달리 DevSecOps는 지속적인 보안 통합을 강조하므로 스마트 컨트랙트의 불변적이고 투명한 특성에 특히 적합하다.

스마트 컨트랙트 개발 전주기에서 DevSecOps를 사용하는 주요 이점 중 하나는 계약의 불변성 특성에 대한 적응성이다. 블록체인에 배포되면 스마트 컨트랙트의 오류나 취약점을 수정하기가 매우 어려운 특징이 있다. DevSecOps는 처음부터 개발 프로세스 전반에 걸쳐 지속적으로 보안 고려 사항을 통합하여 이러한 문제를 회피한다. 이러한 사전 예방적 접근 방식은 결함이 있는 계약을 배포할 위험을 줄이는 데 필수적이며, 이는 블록체인 항목의 영구적인 특성을 고려할 때 매우 중요한 과정이다.

또한, DevSecOps에는 전체 개발 주기에 걸쳐 엄격하고 지속적인 보안 프로토콜이 수반되고, 자동화된 보안 테스트, 지속적인 모니터링 및 정기적인 보안 감사를 동반한다. 이러한 관행은 블록체인 기반 애플리케이션에 필요한 지속적인 경계와 잘 조화되어 보안에 대한 지속적인 집중을 보장한다.

DevSecOps가 지원하는 민첩성과 빠른 개발 주기는 블록체인 기술의 역동적인 특성과도 일치 한다. 이 방법론은 수정이 신속하고 안전하게 구현될 수 있는 환경을 조성하며, 이는 빠르게 진화하는 블록체인 부문에서 앞서가는 데 중요한 요소이다. DevSecOps 내의 개발, 보안 및 운영 통합은 프로세스를 간소화하여 보안 내재화와 전체적인 비용 절감에 특화 된다.

협업과 커뮤니케이션으로 개발자, 보안 전문가, 운영 담당자를 포함한 모든 이해관계자가 함께 나아가는 문화를 조성한다. 이러한 협업 방식을 통해 개발 프로세스의 모든 측면에서 기본적인 소프트웨어의 요구사항 뿐만 아니라 보안 고려사항을 포괄적으로 이해하고 통합할 수 있게 된다.

스마트 컨트랙트 개발에서 DevSecOps의 선택은 보안, 규정 준수 및 전 주기

에 보안을 내재화 할 수 있는 방법이다. 이는 스마트 컨트랙트 개발의 고유한 과제를 해결하여 이러한 계약이 안전하고 규정을 준수할 뿐만 아니라 급변하는 블록체인 기술 환경에 적응할 수 있는 방식으로 개발되도록 보장한다.

스마트 컨트랙트 개발을 위한 SSDLC를 DevSecOps의 프로세스와 통합하여 그림과 같이 보안이 내재화된 스마트 컨트랙트를 개발 할 수 있다. 각 단계에 스마트 컨트랙트를 위한 작업들을 추가하여 전 주기에서 보안을 내재화 시킨다. 각 단계에 대한 내용은 아래와 같다.

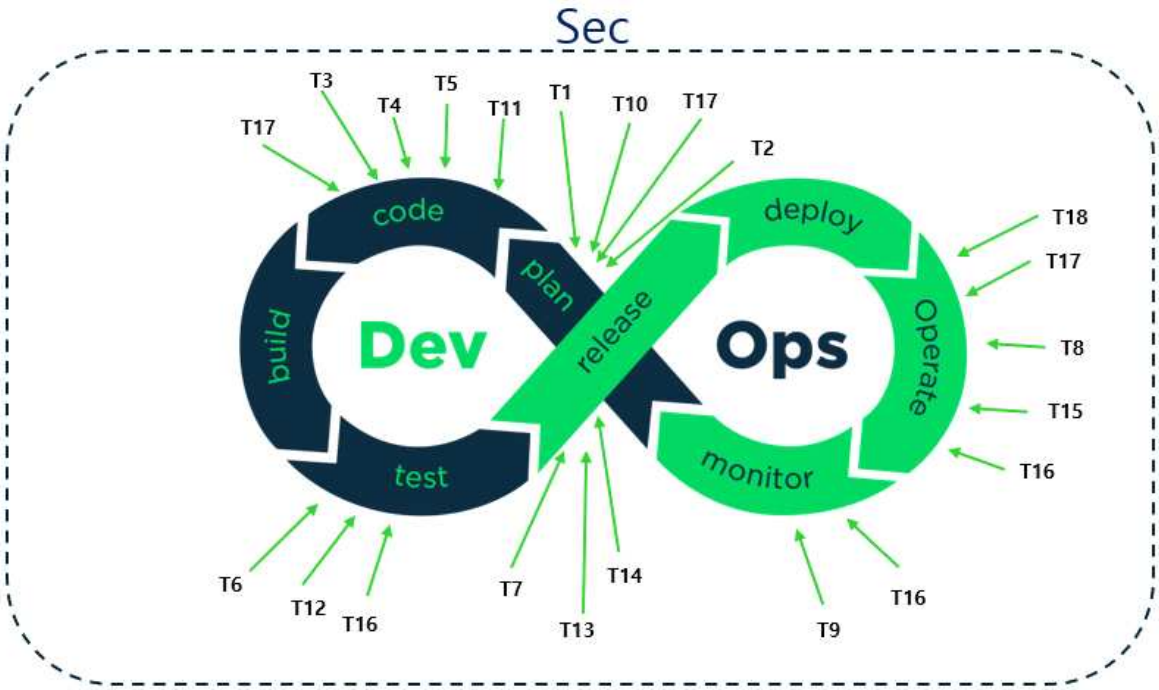


그림 3. DevSecOps for Smart Contract

1. 기획 단계

T1: GDPR 규정 준수 체크리스트 : GDPR 체크리스트를 활용하여 개인 데이터 처리가 규정을 준수하는지 확인한다. 여기에는 데이터 처리 활동을 검토하고 적절한 데이터 보호 조치가 마련되어 있는지 확인하는 것이 포함된다.

T2: 과거 스마트 계약 사건 참조 : 스마트 계약과 관련된 이전 사건을 분석하여 보안 요구 사항을 도출한다. 이 작업에는 그러한 사건이 어떻게, 왜 발생했는

지, 그리고 무엇이 사건을 예방할 수 있었는지 이해하는 것이 포함된다.

T3: 시큐어 코딩 가이드 : 보안 코딩 표준을 기반으로 한 개발 지침을 제공한다. 여기에는 보안 코드 작성에 대한 모범 사례, 일반적인 보안 함정 및 이를 방지하는 방법이 포함된다.

T10: 정형 명세 및 검증 : 스마트 컨트랙트의 기능적 요구사항을 수학적으로 명세화하고 자동 검증 도구를 활용해 오류를 사전에 차단하는 작업을 포함한다. 이는 복잡한 상태 전이를 명확히 하고 불변 조건을 정립하여 논리적 결함을 줄이는 데 기여한다.

T17: Security Champion 운영 : 개발 초기 단계부터 각 팀에 보안 책임자를 지정하여 보안 내재화를 선도하게 한다. 스프린트 단위로 지정된 챔피언은 보안 요구사항 반영, 설계 검토, 팀 간 조율을 담당한다.

2. 코딩 단계

T3: 시큐어 코딩 가이드 : 설정된 지침에 따라 보안 코딩 방식을 구현한다. 여기에는 일반적인 취약점을 방지하기 위해 보안을 염두에 두고 코딩하는 작업이 포함된다.

T4: 사고 패턴 : 이전 스마트 계약 사건의 패턴을 조사하고 현재 프로젝트의 보안 약점을 식별한다.

T5: 정적 분석 및 검증 : 정적 분석 도구를 사용하여 코드의 보안 취약성을 탐지하고 완화한다. 정적 분석에는 취약점을 찾기 위해 코드를 실행하지 않고 코드를 검사하는 작업이 포함된다.

T11: 코드 리뷰 : 개발 중인 코드에 대해 리뷰를 수행하여 논리적 일관성, 보안 취약점, 코딩 표준 준수 여부를 점검한다. 보안 담당자의 승인과 리뷰 이력 관리 또한 포함된다.

T17: Security Champion 운영 : 개발 중간 단계에서도 보안 검토를 지속하고

반복되는 이슈에 대해 개선 가이드를 제시하는 등 실질적인 품질 개선을 유도한다.

3. 테스트 단계

T6: 취약점 탐지를 위한 동적 테스트 : 정적 분석이 놓쳤을 수 있는 취약점을 찾기 위해 동적 테스트를 수행한다. 동적 테스트에는 제어된 환경에서 코드를 실행하여 동작을 테스트하고 보안 문제를 식별하는 작업이 포함된다.

T12: 퍼징 테스트 : 무작위 입력과 경계 조건을 활용하여 스마트 컨트랙트의 예상치 못한 상태 변화를 탐지하고, 상태 불변 조건을 검증하여 보안성을 강화한다.

T16: DevSecOps 자동화 시스템 구성 : 테스트 단계에서 자동화된 CI/CD 환경에 보안 도구와 린터, 퍼징 테스트, 테스트넷 배포 등의 작업을 통합하여 반복적인 검증 체계를 마련한다.

4. 릴리즈 단계

T7: 스마트 컨트랙트 안정성 보장 : 배포 하기 전 모든 주기에서의 테스트, 모니터링의 결과를 분석하고 보안 감사를 통해 보안을 내재화 시키는 전 과정이 포함 된다.

T13: 외부 감사 절차 공식화 : 외부 감사 기관과의 협업을 통해 종합적인 보안 점검을 수행하며, 위험도 분석과 개선 조치를 기반으로 릴리즈 결정을 내린다.

T14: 릴리즈 승인 프로세스 강화 : 바이트코드 해시 검증, 멀티시그 서명, 체크리스트 기반 릴리즈 조건 평가 등을 통해 배포 이전의 보안 통제 절차를 강화한다.

5. 운영단계

T8: 사고 레포지토리 : 운영 스마트 계약에서 버그나 취약점이 발견된 경우

이러한 문제를 데이터베이스화 하고 향후 스마트 컨트랙트 개발에 활용 한다.

T15: 사고 대응 체계 확립화 : 사고 감지부터 보고, 차단, 복구, 후속 조치까지의 일련의 절차를 구조화하고 반복 가능한 실행 체계로 수립한다.

T17: Security Champion 운영 : 사고 대응, 테스트 결과 공유, 보안 문화 확산 등 보안 조직 문화 정착에 기여한다.

T18: 정기 보안 교육 및 버그바운티 프로그램 운영 : 내부 보안 역량 강화를 위한 교육과 외부 화이트해커 참여 기반 취약점 탐지 체계를 정기적으로 운영한다.

6. 모니터링 단계

T9: DApp 모니터링 : 보안 위협 및 악의적 활동에 대해 스마트 계약에 연결된 분산 애플리케이션을 모니터링한다. 이러한 사전 모니터링은 잠재적인 보안 문제를 신속하게 식별하고 완화하는 데 도움이 되며, 전반적인 환경에 대한 확인 및 분석이 가능하다.

T16: DevSecOps 자동화 시스템 구성 : 모니터링 도구와 연계하여 이벤트 발생 시 알람 설정, 로그 수집, 보안 분석 결과 자동 반영 등의 작업을 지속적으로 수행한다.

이러한 작업을 DevSecOps 프레임워크 내의 SSDLC에 통합함으로써 조직은 스마트 계약 개발 수명 주기 전반에 걸쳐 보안이 지속적인 초점이 되도록 보장할 수 있다. 이러한 접근 방식은 위험을 조기에 식별하고 완화하는 데 도움이 될 뿐만 아니라 개발, 운영 및 보안 팀 전반에 걸쳐 보안 인식 및 책임 문화를 조성할 수 있다.

가. GDPR 체크리스트

스마트 컨트랙트를 위한 DevSecOps 구축 계획 단계에서 GDPR 규정 준수 체크리스트를 개발하는 것은 개인정보를 비롯한 데이터를 보호 될 수 있도록 보장하는 필수 단계이다. 이 체크리스트는 표준 GDPR 요구 사항을 해결해야 할 뿐만 아니라 스마트 컨트랙트의 특성으로 인해 발생하는 문제들도 고려해야한다.

데이터 최소화 원칙을 통해 관리하는 것이 중요하다. 스마트 컨트랙트는 블록체인에 과도한 데이터 저장을 피하면서 필요한 개인 데이터만 처리하도록 설계되어야 한다. 데이터가 블록체인에 있으면 이를 변경하거나 제거하는 것이 어려울 수 있으므로 이 단계는 매우 중요하다. 따라서 계획 단계의 핵심 부분은 스마트 컨트랙트가 작동하는 데 필요한 데이터가 무엇인지 정확히 결정하고 데이터 수집을 제한해야 한다.

GDPR의 목적 제한 원칙을 엄격하게 준수해야 한다. 계획 단계에서 개인 데이터가 처리되는 목적을 명확하게 정의 및 확인 해야 한다. 이러한 목적을 문서화하고 데이터가 지정된 목표에만 사용되도록 스마트 컨트랙트를 설계하고 데이터의 오용을 방지해야 한다.

데이터 처리의 합법성을 보장하는 것도 상당히 중요하다. GDPR에 따른 기타 합법적인 근거 등 데이터 처리 활동에 대한 법적 근거를 식별하고 문서화를 진행해야 한다. 이는 스마트 컨트랙트 내 데이터 처리 활동에 대한 법적 기반을 형성하기 때문에 중요하다.

데이터 주체 권리를 다루는 것은 블록체인 환경에서 어렵게 때문에 액세스, 수정, 삭제, 데이터 이동성 등의 권리를 용이하게 하는 방법을 계획해야 하며, 스마트 컨트랙트 내의 데이터가 정확하게 유지되도록 법적 근거를 만들어야 한다.

DevSecOps 프로세스에 계획 단계에 책임과 거버넌스의 구축이 필수적이다. 필요한 경우 데이터 보호 책임자를 임명하고 모든 처리 활동에 대한 자세한 기록을 해야하며, 정기적인 데이터 보호 영향 평가(DPIA) 계획도 필수적이다. 설계 및 기본에 따른 데이터 보호는 GDPR의 핵심 원칙이다. 스마트 컨트랙트 개발 수명주기의 모든 단계에 데이터 보호 고려 사항을 통합하고, 기본 설정이 개인 정보 보호에 적

합한지 확인하는 과정이 필요하다.

스마트 컨트랙트를 위한 DevSecOps 구축 계획 단계에서 이러한 세부 요소를 GDPR 체크리스트에 통합하면 프로젝트 개발 및 운영 수명주기 전반에 걸쳐 규정 준수 및 데이터 보호를 위한 강력한 기반이 마련된다. 점차 중요하게 생각되는 개인정보 및 각종 데이터 보호 규정과 기술 발전에 발맞춰 해당 체크리스트를 정기적으로 검토하고 업데이트를 하여 개인 정보 및 데이터 보호를 지속적으로 적용할 필요가 있다.

GDPR 자가진단 테스트

자가진단이 완료되었습니다.

식별자	내용	예	아니오	점수
INIT-start	controller(컨트롤러)의 요청에 따라 개인정보를 처리하는 processor(프로세서)입니까?	✓		
A-1.1	개인정보 처리는 아래 경우 중 하나 이상에 근거하고 있는가? ① 정보주체의 동의 ② 계약의 이행 또는 계약 전 정보주체 요청에 응하기 위한 처리 ③ 컨트롤러에게 적용되는 법적 의 무 이행 ④ 정보주체 또는 제3자의 중대한 이익 보호 ⑤ 공익 또는 컨트롤러에게 부여된 공적 권한의 행사 ⑥ 컨트롤러 또 는 제3자의 적법한 이익 추구	✓		
A-2.1	정보주체로부터의 동의를 개인정보 처리의 근거로 삼고 있는 가?		✓	심각한 위반
A-3.4	민감정보를 처리하는 경우 GDPR 제9조 2항 각호에서 규정 하고 있는 바와 같이 정보주체의 명시적인 동의 등 명확한 법 적 근거에 기반하고 있는가?	✓		
A-3.5	범죄경력(전과) 및 범죄행위 관련 정보를 처리하는가?		✓	
B-1.1	개인정보를 정보주체로부터 직접 수집시 제공하여야 하는 정 보와 제공시기에 대해 알고 있으며 요청이 있는 경우 그에 맞 게 제공하고 있는가?	✓		
B-1.2	제 3자가 취득한 개인정보를 받았는가?		✓	
B-1.3	제3자가 취득한 개인정보를 제공받을 경우, 정보주체에게 의 무적으로 제공하여야 하는 정보 및 제공시기에 대해 알고 있 으며 그에 맞게 제공하고 있는가?	✓		
B-1.4	정보주체가 제15~22조(예: 접근권, 삭제권, 이동권 등)에 해당 하는 자신의 권리를 행사할 경우 다음 사항을 지켜서 정보를 제공하는가? ① 부당한 지체없이 1개월이내 제공 ② 서면제 공(전자적수단 포함) ③ 명확하고 이해하기 쉬운 언어 사용 ④ 특별한 이유가 없는 한 무상제공	✓		
B-1.5	당초 수집 기간이 종료되었을 경우 즉시 보관 중인 개인정보를 삭제하는가?		✓	심각한 위반
B-1.6	당초 개인정보 수집 목적 이외의 목적으로 개인정보 추가 처 리가 필요한 경우가 있는가?		✓	

그림 4. GDPR Checklist

나. 보안 요구 사항

특히 계획 단계에서 스마트 컨트랙트를 위한 DevSecOps를 구축하려면 보안 요구 사항을 식별하고 지정하는 세심한 접근 방식이 필요하다. 코드에 직접 작성된 계약 조건으로 실행되는 스마트 컨트랙트는 여러 보안적 문제에 직면하게 된다. 또한, 블록체인 기술의 불변성 특성으로 인해 초기 설계부터 보안적 요소를 심도있게 고려하고 확보하는 것이 중요하다.

첫 번째로 위협 분석 및 위험 평가를 수행해야 한다. 여기에는 블록체인 및 스마트 컨트랙트와 관련된 잠재적인 보안 위협을 식별하는 것이 포함된다. 이러한 위협은 재진입, 오버플로/언더플로, 선행 실행 공격과 같은 코드 취약성부터 네트워크 취약성 과 같은 보다 광범위한 시스템 수준까지 다양하다. 초기 설계한 환경과 블록체인, 스마트 컨트랙트의 특성을 이해하는 것이 상당히 중요하다. 이러한 보안 약점 및 취약점들을 분석하기 위해서는 위협 모델링을 진행해서 해당 소프트웨어에서 발생 할 수 있는 여러 가지 보안 위협을 도출하는 과정이 필요하다. 보안 위협을 도출하는 것으로는 STRIDE, TARA, OCTAVE 와 같은 기법들이 존재하며, 이러한 기법을 통해서 위협을 도출 할 수 있다.

위협 모델링에 앞서 목표와 범위를 정의해야하며, 각각의 구성요소와 특징, 아키텍처 분석을 선행해야한다. 선행된 데이터를 기반으로 위협 모델링을 진행하고, 보안 약점 및 발생할 수 있는 여러 가지 공격을 도출한다. 도출된 공격들을 체계화시켜 정리를 하여 치명도 및 발생 가능성과 같은 요소들을 고려하여 위험도를 평가한다. 위험도가 평가되면 평가된 위험도를 기반으로 보안 요구 사항을 도출 한다.

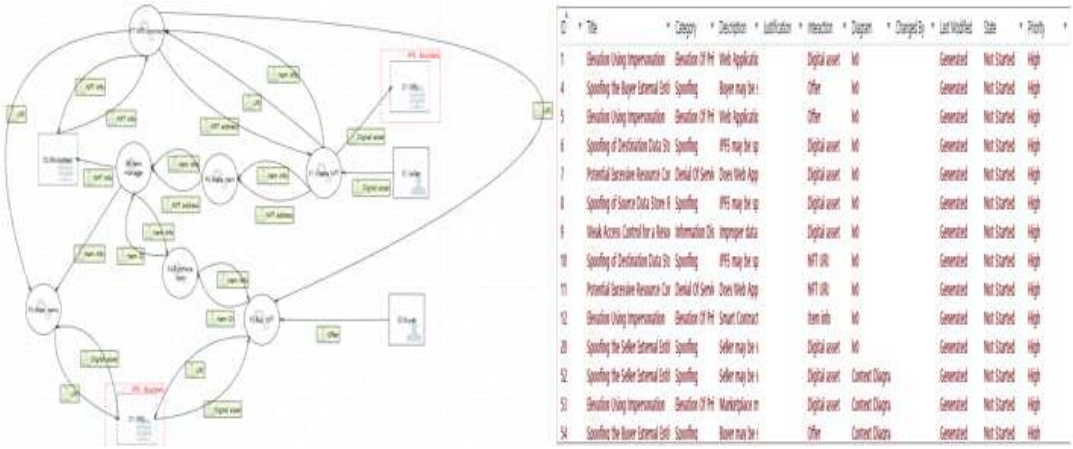


그림 5. Threat Modeling

스마트 컨트랙트 보안 약점은 종종 코딩 오류로 인해 발생하므로 여기에는 코드 품질에 대한 표준 설정을 포함한다. 또한, 암호화 및 키 관리 및 보안 인증 매커니즘, 감사 가능성과 같은 요구사항들을 분류하고 정의해야 한다.

다. 시큐어 코딩 가이드

DevSecOps 프레임워크의 일부로 스마트 컨트랙트를 위한 시큐어 코딩 가이드를 구축하는 것은 필수적인 과정이다. 현재 각 국가 기관 또는, 기업에서는 자체적인 시큐어 코딩 가이드를 가지고 있으며, 이를 준수하면서 보안 사고를 가장 기본적인 프로그래밍 단계에서부터 예방을 하려 하고 있다. 스마트 컨트랙트 또한 고유한 보안 문제에 대한 방어적 코딩 기술에 대한 포괄적인 접근 방식이 필요하다.

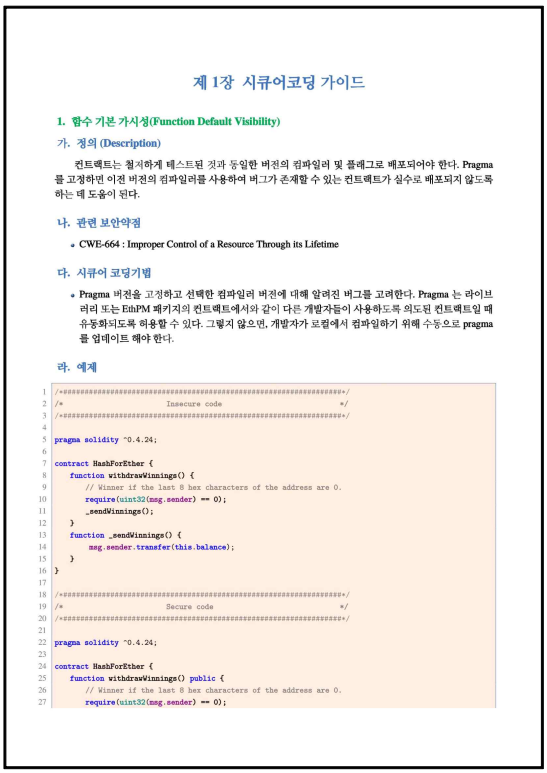
스마트 컨트랙트 위한 시큐어 코딩 가이드의 기초는 Ethereum, Hyperledger 등과 같이 사용되는 특정 블록체인 플랫폼에 대한 철저한 이해하에 작성되어야 한다. 현재 스마트 컨트랙트를 프로그래밍하기 위한 언어로 Solidity와 Go 언어가 있는데, 해당 언어에 대한 시큐어 코딩 가이드가 필수적으로 필요하고, 스마트 컨트랙트의 특성과 보안 약점, 취약점 등을 방지 할 수 있는 내용이 포함되어야 한다.

또한, 가이드에는 방어적 기법의 올바른 사례와 방어적 코딩 기술이 포함되어야 한다. 여기에는 입력 유효성 검사, 오류 처리 및 최소 권한 원칙 준수 등 기본적으로 검사해야하는 모든 항목이 포함된다. 입력 검증은 유효하고 예상되는 데이터만 처리되도록 하기 위해 스마트 컨트랙트에서 매우 중요합니다. 이를 통해 스마트 컨트랙트 공간에서 널리 퍼져 있는 주입 공격, 재진입 공격 등 다양한 공격을 방지할 수 있다.

스마트 컨트랙트의 에러 처리 및 예외 처리 또한 중요하다. 시큐어 코딩 가이드에서는 포괄적이고 오류 처리 메커니즘의 필요성 및 활용 방안을 포함한다. 오류로 인해 프로그램이 중단되거나 다시 시작될 수 있는 기존 소프트웨어와 달리 스마트 컨트랙트의 오류는 되돌릴 수 없는 재정적 손실을 초래할 수 있다. 따라서 시큐어 코딩 가이드에서는 오류를 안전하게 처리하는 방법에 대한 자세한 지침을 제공해야 한다.

일반적인 소프트웨어 보다 스마트 컨트랙트 상에서는 최소 권한의 원칙이 상당히 중요하다. 따라서, 시큐어 코딩 가이드에 접근 권한에 대한 메커니즘을 효과적으로 구현하는 방법과 불 필요한 권한이 없도록 보장하는 방법을 명시 해야 한다. 또한, 데이터 저장, 상태 변경 관리와 같은 항목에 대해서도 명시해야 한다.

스마트 컨트랙트를 위한 시큐어 코딩 가이드는 상당 부분은 블록체인과 관련되어 있기 때문에 블록체인과 관련한 보안 약점에 초점을 맞춰야 한다. 여기에는 재진입 공격, 오버플로 및 언더플로 문제, 선행 실행 및 가스 제한 문제 등이 포함된다. 각 유형의 취약점에 대해 시큐어 코딩 가이드에서는 해당 취약점이 악용될 수 있는 방법에 대한 설명, 예, 이를 방지하는 방법에 대한 자세한 전략을 제공해야 한다. 또한, 가이드는 진화하는 블록체인 기술 환경과 새로운 보안 위협을 반영하기 위해 정기적으로 업데이트되어야 하며, 실제 사고 사례들을 분석하여 반영 해야 한다.



제 1장 시큐어코딩 가이드

1. 함수 기본 가시성(Function Default Visibility)

가. 정의 (Description)

컨트랙트는 철저하게 테스트된 것과 동일한 버전의 컴파일러 및 플래그로 배포되어야 한다. Pragma를 고정하면 이전 버전의 컴파일러를 사용하여 버그가 존재할 수 있는 컨트랙트가 실수로 배포되지 않도록 하는 데 도움이 된다.

나. 관련 보안약점

- CWE-664 : Improper Control of a Resource Through its Lifetime

다. 시큐어 코딩기법

- Pragma 버전을 고정하고 선택한 컴파일러 버전에 대해 알려진 버그를 고려한다. Pragma 는 라이브러리 또는 EthPM 패키지의 컨트랙트에서와 같이 다른 개발자들이 사용하도록 의도된 컨트랙트일 때 유동화되도록 허용할 수 있다. 그렇지 않으면, 개발자가 로컬에서 컴파일하기 위해 수동으로 pragma를 업데이트 해야 한다.

라. 예제

```
/*##################################################################*/
/*                        Insecure code                             */
/*##################################################################*/

pragma solidity ^0.4.24;

contract HashForEther {
    function withdrawWinnings() {
        // Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
    }
    function _sendWinnings() {
        msg.sender.transfer(this.balance);
    }
}

/*##################################################################*/
/*                        Secure code                               */
/*##################################################################*/

pragma solidity ^0.4.24;

contract HashForEther {
    function withdrawWinnings() public {
        // Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
```

그림 6. 시큐어 코딩 가이드

라. 사고 패턴

특히 과거 사건에서 통찰력을 분석하고 추출하는 맥락에서 스마트 컨트랙트를 위한 DevSecOps 프레임워크를 구축하는 것은 중요합니다. 이 과정에서는 잠재적인 위험을 사전에 식별하고 완화하기 위해 블록체인 및 스마트 컨트랙트 도메인의 사고 패턴에 대한 심층적이고 상세한 조사가 포함된다. 블록체인 거래의 불변성 및 공개적 특성은 한 번 발생한 사고가 영구적으로 기록되고 종종 표시되어 분석을 위해 활용 가능한 이점이 존재한다.

스마트 컨트랙트와 관련된 알려진 보안 사고의 포괄적인 기록을 확인 및 기록한다. 여기에는 DAO 공격이나 Qubit 공격과 같은 세간의 이목을 끄는 사건뿐 아니라 규모가 작고 덜 알려진 사건도 포함된다. 이러한 사건들은 공격자들이 어떠한 방식으로 접근을 하고 원하는 정보 및 자산을 탈취하는지를 잘 보여준다. 해당 사건들을 분석해 악용된 취약성 유형, 사건의 영향, 근본 원인과 같은 요소를 기준으로 사건을 분류하여 사건에 대한 자세한 카탈로그를 구축 해야 한다.

데이터가 수집되면 다음 단계는 이러한 사건 간의 패턴과 공통점을 식별하기 위한 분석이 시작된다. 이 분석에서는 즉각적인 기술적 취약점을 넘어 광범위한 패턴을 이해해야 한다. 이 과정은 각 사건에 대한 상세한 기술 분석도 포함 된다. 각 공격이 어떻게 수행되었는지, 어떤 취약점이 악용되었는지, 스마트 컨트랙트가 어떻게 악용을 방지하지 못했는지를 정확하게 이해하고 분석하며 기록하는 것이 중요하다. 이러한 기술적 분석은 계약의 코드, 공격 당시의 블록체인 상태, 사고를 초래한 트랜잭션을 자세히 조사하여 최대한 세부적으로 분류 된다.

사고 패턴의 분석에서 가장 중요한 것은 항상 최신 상태로 업데이트를 꾸준히 해야한다는 점이다. 블록체인과 스마트 컨트랙트 기술의 환경은 빠르게 발전하고 있으며 새로운 유형의 공격 기법과 실제 공격들이 꾸준히 새롭게 등장하고 있다. 그러므로 사고 카탈로그를 정기적으로 업데이트하고, 패턴을 재분석하고, 모범 사례를 업데이트하는 것이 중요하다.

마. 정적 분석

모든 소프트웨어는 정적 분석을 거쳐 보안을 내재화 시켜야 한다. 스마트 컨트랙트 또한 정적 분석 및 검증을 하여 보안을 내재화 시켜야 한다. 특히 오류를 되돌릴 수 없고 오류로 인하여 심각한 결과를 초래할 수 있는 블록체인 기술의 불변적인 특성을 가지고 있어 사전에 검증하는 과정이 상당히 중요하다.

스마트 컨트랙트에서의 정적 분석은 보안 약점, 결함, 에러 또는 개발 표준 준수 문제를 찾아내는 것을 포함하고, 이를 찾기 위해 실행하지 않고 코드를 분석한다. 이 과정은 런타임 중에 명확하지 않을 수 있는 문제를 감지하는 데 도움이 된다. 스마트 컨트랙트로 인한 복잡성과 고유한 보안 문제를 고려할 때 정적 분석은 철저하고 블록체인의 세부 사항과 Ethereum용 Solidity와 같은 스마트 컨트랙트의 프로그래밍 언어에 맞게 조정되어야 한다.

정적 분석은 자동화된 도구 및 스캐너를 활용하여 코딩 스타일 오류, 코딩 표준의 편차, 재진입 공격, 정수 오버플로 등과 같은 복잡한 보안 취약성을 감지하는 것부터 시작한다. 또한, 사고 패턴과 사고 래퍼지토리를 활용하여 최신 보안 약점을 탐지하고 도구를 정기적으로 업데이트하는 과정이 필요하다. 수동 검토 프로세스도 포함되며, 숙련된 개발자의 코드 검토는 계약 논리가 의도한 기능을 정확하게 반영하고 보안, 효율성 및 유지 관리 가능성에 대한 모범 사례를 준수하는지 확인하는 데 중점을 두어야 한다.

정적 분석 후에는 스마트 컨트랙트가 지정된 요구 사항을 충족하고 모든 시나리오에서 예상대로 작동하는지 확인하는 철저한 검증 프로세스를 거쳐야 한다. 검증에는 엣지 케이스를 포함한 포괄적인 테스트 케이스 및 시나리오를 사용하는 것이 포함된다. 가능한 경우, 공식 검증을 통해 스마트 컨트랙트의 기반 알고리즘의 정확성을 수학적으로 입증하거나 반증하는 것이 포함된다.

정적 분석 및 검증을 DevSecOps 파이프라인에 원활하게 통합하는 것이 중요하며, 이러한 프로세스는 개발 및 배포 워크플로의 표준 부분이 되어야 한다. 스마트 컨트랙트 코드의 모든 업데이트는 새로운 라운드의 정적 분석 및 검증을 촉발하여 지속적인 보안과 규정 준수를 보장해야 한다.

바. 동적 분석

스마트 컨트랙트의 개발에 있어 동적 분석 및 테스트는 보안과 기능을 검증하는 데 있어 필수적인 요소이다. 이는 정적 분석만으로는 발견하기 어려운 실시간 환경에서의 취약점과 문제점을 식별할 수 있다. 정적 분석이 코드를 실행하지 않고 이루어지는 반면, 동적 분석 및 테스트는 실제 또는 시뮬레이션된 환경에서 스마트 컨트랙트의 실행을 포함한다. 이는 블록체인 기술의 복잡하고 상호작용적인 특성을 감안할 때 특히 중요하다.

동적 테스트는 다양한 테스트 기법을 적용할 수 있다. 단위 테스트는 각 스마트 컨트랙트의 기능과 모듈을 개별적으로 평가하여 예상대로 작동하는지 확인한다. 하지만 스마트 컨트랙트와 블록체인의 상호 연결된 특성을 고려하면, 단위 테스트만으로는 충분하지 않다. 따라서 통합 테스트가 필요하며, 이는 스마트 컨트랙트의 다양한 부분과 다른 계약, 블록체인 자체와의 상호작용을 검토한다. 이러한 통합 테스트는 분산된 특성으로 인해 발생할 수 있는 종속성과 상호작용의 문제를 식별하는 데 도움이 된다. 엔드투엔드 테스트는 실제 환경에서 스마트 컨트랙트의 전체 작동을 시뮬레이션하여 계약의 기능, 성능 및 보안을 포괄적으로 평가한다. 이러한 테스트는 높은 거래량, 다양한 유형의 지갑 및 노드와의 상호작용, 실제 이벤트 및 입력에 대한 반응을 포함한 다양한 조건을 시험한다. 성능 테스트는 다양한 부하 및 스트레스 조건에서 스마트 컨트랙트의 성능을 평가하는 데 중요하다. 이는 특히 트랜잭션 처리 속도가 느려질 수 있는 블록체인 환경에서 중요하며, 이러한 테스트는 성능 문제로 인한 비용 증가와 사용자 경험 저하를 방지하는 데 기여한다. 보안 테스트는 재진입, 경쟁 조건, 트랜잭션 순서 의존성과 같은 일반적인 공격에 대한 취약성을 검사하며, 스마트 컨트랙트의 보안을 강화하는 중요한 요소이다. 블록체인 및 스마트 컨트랙트 보안을 위해 설계된 도구와 프레임워크를 활용하면 자동화된 방식으로 이러한 보안 테스트를 수행할 수 있다.

동적 분석은 스마트 컨트랙트의 성능 및 보안의 다양한 측면에 대한 데이터를 수집하고, 문제를 식별하고, 사용 패턴을 이해하며, 계약의 성능 및 보안을 최적화하는 데 필수적인 역할을 한다. DevSecOps에 동적 분석 및 테스트를 통합하는 것은 보안 내재화를 위한 필수적인 과정이다. 이 과정은 포괄적인 테스트 사례 및 시나리오의 설계, 그리고 이를 지속적인 통합 및 배포를 포함한다. 이러한 통합은 스마트 컨트랙트가 보안을 염두에 두고 개발되고, 수명 주기 전반에 걸쳐 지속적으로 테스트되고 모니터링되는 것을 보장한다.

사. 안정성 확보

DevSecOps에서 릴리스 단계의 안정성 확보 계획, 포괄적인 테스트, 그리고 지속적인 모니터링을 포함한다. 릴리스 단계는 소프트웨어의 전체 개발 수명주기 중 가장 중요한 부분 중 하나로, 여기서 코드는 실제 사용 환경으로 이관 하는 과정을 거친다. 이 단계에서의 안정성 확보는 소프트웨어의 신뢰성을 보장하기 위해 필수적이다.

철저한 테스트는 릴리스 단계의 안정성 확보에 있어 핵심적인 요소이다. 이는 단위 테스트, 통합 테스트, 시스템 테스트, 그리고 사용자 수용 테스트를 포함하여, 각 단계에서 발견되지 않은 버그나 결함을 찾아내고 해결하는 데 중점을 둔다. 이러한 테스트들은 코드의 모든 변경 사항이 기존 시스템과 원활하게 통합되고, 예상대로 작동하는지를 보장한다.

안정성을 확보하는 데 있어, 보안도 중요한 요소 중 하나이다. 보안 감사 및 취약점 평가는 소프트웨어를 공격자로부터 보호하는 데 중요하다. 이는 코드를 철저히 검토하여 잠재적인 보안 취약점을 식별하고, 적절한 보안 조치를 취하는 것을 포함한다. 또한, 규제 준수를 위해 법적 및 산업 표준에 따라 소프트웨어가 제대로 구성되고 운영되는지 확인하는 것도 중요하다.

릴리스 단계에서의 안정성을 확보하기 위해 지속적인 모니터링도 필수적입니다. 이는 배포된 소프트웨어의 성능을 지속적으로 추적하고, 문제가 발생하면 즉시 대응할 수 있도록 해야 한다. 모니터링 도구는 트래픽 패턴, 시스템의 자원 사용, 오류 로그 등을 추적하며, 이러한 데이터는 시스템의 전반적인 상태를 파악하고, 필요한 경우 조정을 통해 성능을 최적화하는 데 사용된다. 또한, 체계적인 배포 프로세스를 통해 전략을 명확하게 정의하고, 필요한 경우 즉각적인 롤백이 가능하도록 해야 한다.

아. 사고 레포지토리 확보

DevSecOps 프레임워크 내에서 스마트 컨트랙트를 위한 사고 레퍼지토리의 구축은 보안을 내재화 시키는 과정에서 참고할 수 있는 데이터들을 수집하기 위한 필수적 과정이다. 이 레퍼지토리는 발생하는 모든 버그와 보안 취약점을 포함하는 포괄적인 데이터베이스 역할을 하며, 과거 사건을 기록하고 향후 문제를 예방하고 빠르게 해결하기 위한 지식 기반으로 활용할 수 있다.

사고 레퍼지토리의 구축과 유지에는 몇 가지 주요 단계와 고려사항이 포함된다. 일단, 스마트 컨트랙트의 맥락에서 사고의 요소를 정의해야 한다. 이는 사소한 버그부터 자금 손실이나 데이터 유출과 같은 주요 보안 취약점에 이르기까지 다양하게 정의 될 수 있다.

그 다음 사건 보고 및 문서화를 위한 체계적인 프로세스를 구축해야 한다. 이 프로세스는 발견 시간, 사건 설명, 영향, 재현을 위한 조치, 즉각적인 해결 방법 등 필수적인 세부 정보를 포함하는 표준화된 양식을 사용하고, 데이터베이스를 구축해야 한다.

수집된 데이터는 분류된 상태로 데이터베이스에 저장되어야 한다. 이 데이터베이스는 쉽게 접근 할 수 있어야 하며, 민감한 정보는 적절히 보호되어야 한다. 즉, 개발에 참여하는 모든 인원이 데이터를 쉽게 조회하고 분석할 수 있도록 해야 하며, 취약점 및 효과적인 완화 전략을 식별할 수 있도록 지원해야 한다.

데이터 베이스의 내용에서 사후 분석의 내용을 기록하는 것 또한 아주 중요하다. 사고 해결 후 근본 원인, 대응 효과, 사고의 전반적인 이해를 위한 철저한 분석을 수행하고, 이를 데이터베이스화하여 레퍼토리에 추가하여 보안 및 개발 관행에 대한 깊은 통찰력을 제공할 수 있다.

사고 레퍼토리 데이터베이스는 보안 및 개발 프로세스의 필수적인 부분으로, 정기적인 검토를 통해 새로운 패턴이나 취약점을 식별하고, 특정 사고에 대응하거나 업데이트 또는 새로운 스마트 컨트랙트를 계획할 때 보안이 내재화된 스마트 컨트랙트 개발을 위해 필수적으로 사용되어야 하며, 스마트 컨트랙트 설계와 개발 관행, 보안 조치를 개선하는 데 적용되어야 한다.

사고 레퍼지토리의 데이터베이스는 새로운 사건이 발생할 때마다 즉시 업데이트되어야 하며, 변화하는 상황과 통찰력에 적응하기 위해 주기적으로 검토하고 개선해야 한다. 이렇게 함으로써 스마트 컨트랙트의 DevSecOps 환경에서 사고 레퍼토리는 지속적인 학습과 개선 문화에 기여하며, 스마트 컨트랙트의 보안과 안정성을 향상시키고, 보안이 내재화된 스마트 컨트랙트 개발에 있어 중추적인 역할을 한다.

자. DApp 모니터링

블록체인 기술 영역에서 스마트 컨트랙트 및 분산 애플리케이션의 DevSecOps 구축에 모니터링은 필수적 요소이다. DApp 모니터링은 안정성을 보장하고 무결성, 보안, 효율성 및 사고대응에 있어 중요하다. 이는 DApp이 분산형 네트워크에서 작동하며 스마트 컨트랙트와 상호 작용한다는 본질적인 특성 때문이다.

DApp 모니터링은 스마트 컨트랙트의 안정성과 보안성을 보장한다. 이 과정은 스마트 컨트랙트 자체는 물론 사용자 인터페이스 및 이들 간의 상호 작용을 포함한다. 무단 접속, 비정상 거래, 취약점 악용 시도 등의 보안 사고를 실시간으로 탐지하고 대응함으로써 잠재적인 위반이나 손실을 예방할 수 있다.

또한, 모니터링을 통해 성능도 관찰하고 최적화할 수 있다. DApp의 복잡한 동작은 성능에 영향을 미칠 수 있으며, 거래 시간, 가스 비용, 애플리케이션의 응답성 등을 분석함으로써 성능 병목 현상이나 비효율성을 식별하고 해결할 수 있다. 이는 사용자 경험과 운영 비용을 최적화하는 데 기여할 수 있고, 이는 곧 서비스 품질 유지와도 직결된다.

모니터링은 사용자 행동 및 애플리케이션 사용 패턴에 대한 귀중한 통찰력도 제공한다. DApp 내의 트랜잭션 데이터, 스마트 컨트랙트 상호 작용, 사용자 활동 분석을 통해 개발자는 애플리케이션이 어떻게 사용되는지 전반적인 과정에 대해 더 쉽게 이해할 수 있게 되고, 그로인해 새로운 기능이나 개선 사항을 실제 사용자 요구에 맞추는 데 도움이 된다..

스마트 컨트랙트는 재정적인 요소들과 많은 연관이 있기 때문에, 사고 관리 및 대응이 상당히 중요하다. 이를 즉각적이고 효율적으로 관리 하기 위해서는 필수적으로 모니터링을 해야한다. 실시간 모니터링과 경고 시스템은 이상, 장애, 보안 사고를 신속하게 식별하여 적절한 대응을 가능하게 한다. 이는 사고의 영향을 최소화하고 이해관계자의 신뢰를 유지하는 데 아주 중요하다.

차. 정형 명세 및 검증

스마트 컨트랙트는 변경이 어려운 특성상 설계 단계에서의 논리적 완전성과 일관성 확보가 필수적이다. 이에 따라 DevSecOps 설계 단계에서는 정형 명세를 도입하여 계약의 동작을 수학적으로 정의하고, 자동 도구를 통해 사양의 정확성을 사전에 검증하는 과정을 강화해야 한다. 이는 복잡한 상태 전이나 비가역적인 자산 이동이 포함된 스마트 컨트랙트에서 잠재적 결함을 사전에 차단하고, 설계 오류가 코드로 확산되는 것을 방지하는 효과적인 방법이다.

정형 명세는 먼저 시스템의 요구사항을 수학적으로 기술 가능한 모델로 추상화하고, 이를 기반으로 주요 상태(state), 동작(action), 제약조건(invariant)을 명시한다. 대표적으로 사용하는 도구로는 TLA+, Solidity formal verifier, Certora Prover, KEVM 등이 있으며, 이들은 상태 공간 탐색, 경로 검증, 조건 충족 여부 확인 등을 통해 논리적 오류를 사전에 포착할 수 있게 한다. 정형 명세는 다음과 같은 구조로 문서화 및 활용되어야 한다. 첫째, 변수, 초기 상태, 상태 전이 등의 모델링 요소, 둘째, 불변 조건 정의, 셋째, 모델 체킹과 같은 기법을 통한 테스트 시나리오 생성, 그리고 마지막으로 검증 결과를 기록 해야 한다. 또한, 스마트 컨트랙트 개발 과정에서 발생할 수 있는 행위와 상호작용 조건을 명확히 정의함으로써, 추론 오류를 방지할 수 있다. 정형 명세를 통해 검증된 사양은 이후 코드 리뷰, 테스트, 감사 단계에서도 일관된 기준으로 활용 가능하다.

정형 명세 기반 설계는 특히 높은 신뢰성과 추적 가능성을 요구하는 시스템에 적합하며, 보안 요구사항을 자동 검증 가능한 구조로 전환함으로써 스마트 컨트랙트 개발의 품질을 근본적으로 제고할 수 있는 핵심 수단이다.

카. 코드 리뷰

스마트 컨트랙트는 코드 그 자체가 계약의 역할을 수행하며, 배포 이후 변경이 불가하거나 매우 제한되기 때문에 개발 단계에서의 코드 리뷰 절차는 DevSecOps 전체 보안 내재화 흐름의 필수 과정이다. 특히 스마트 컨트랙트는 작은 오류도 치명적 결과로 이어질 수 있으므로, 코드 리뷰는 자동화 분석 도구의 한계를 보완하며 필수적인 다층 보안 검토 체계의 일부로 자리 잡아야 한다.

코드 리뷰 과정으로는 첫째, 리뷰 체크리스트를 만들어 초기에 검토를 진행해야한다. 체크리스트의 내용에는 접근 제어 함수에 대한 유효성 검증, 상태 변수의 초기화 및 경계 값 확인, 외부 호출 발생 시 재진입 방지 구조, 오버플로/언더플로 방지등의 점검요소들이 포함된다. 둘째, 보안 담당자의 승인이 필수화 되어야 한다. 이를 통해 보안 검토의 책임 소재를 명확히 하고, 내부 가이드 준수를 강제하게 할 수 있다. 셋째, 리뷰 이력 문서화를 통해 사고 발생시 분석을 진행하여 추후 더 일어날 수 있는 사고를 방지하는 핵심 데이터로 활용 할 수 있다.

또한, 코드 리뷰는 단순한 문법적 오류나 구조적 개선을 넘어 보안 설계와 연계된 논리적 일관성 검토에 초점을 둬야 하며, 이를 위해 리뷰어는 스마트 컨트랙트 설계 문서 및 보안 요구사항 문서를 사전에 숙지하고 있어야 한다. 반복적으로 발견되는 보안 이슈에 대해서는 조직 차원의 코딩 규칙으로 구조화시키고, 해당 내용을 시큐어 코딩 가이드에 반영하는 방식으로 DevSecOps 내의 지속적 개선을 유도할 수 있다.

코드 리뷰 절차는 개발자 간 협업의 품질을 높일 뿐 아니라, 보안 사고 발생률을 획기적으로 낮출 수 있는 가장 실효성 있는 방법 중 하나이며, 특히 실증 단계에서 보안 프로세스의 문서화와 검토 이력 추적이 요구되는 경우 필수적으로 적용되어야 할 핵심 보안 내재화 활동이다.

타. 퍼징 테스트

스마트 컨트랙트는 상태 기반 논리와 외부 입력에 따른 다양한 흐름을 가지므로 예상치 못한 경계 조건에서 치명적인 보안 결함이 발생할 수 있다. 이를 해결하기 위한 핵심적인 테스트 기법이 바로 퍼징 테스트이다. 퍼징 테스트는 일반적으로 사람이 설계한 테스트 케이스로는 포착하기 어려운 오류를 찾는 데 효과적이며, 스마트 컨트랙트와 같이 입력 기반으로 민감하게 동작하는 시스템에 매우 적합하다. DevSecOps 기반 SSDLC의 테스트 단계에서 퍼징 테스트를 도입함으로써 코드가 실제 배포되기 전 무작위 혹은 경계값 기반 입력을 통해 예기치 못한 상태 전이를 사전에 확인할 수 있다.

스마트 컨트랙트 퍼징 테스트에서는 대표적으로 Echidna, Harvey, ContractFuzzer와 같은 도구를 활용할 수 있으며, Solidity나 Vyper 언어 기반으로 작성된 컨트랙트에 대해 자동으로 함수 호출 조합, 부정 입력값, 상태 간섭을 유발할 수 있는 트랜잭션 흐름을 생성하여 테스트를 진행한다. 예를 들어, 특정 함수가 오직 관리자만 호출할 수 있도록 설계되어 있음에도 불구하고, 잘못된 접근 제어나 fallback 함수에 의해 호출되는 경우 퍼징 테스트는 이를 탐지할 수 있다. 이외에도 underflow/overflow, timestamp 의존성, 블록 번호 조건, gas 예외 처리 실패 등 사람이 예측하기 어려운 오류들을 자동으로 탐색한다.

퍼징 테스트는 스마트 컨트랙트의 상태 변화(state transition)를 기준으로 단순 실행 성공 여부가 아닌, 테스트 전후의 상태 불변성을 검증하는 방식으로 진행되어야 한다. 예컨대 “사용자의 잔액은 항상 음수가 될 수 없다“, “컨트랙트 해제 이후에는 어떠한 상태도 변경될 수 없다“와 같은 사양을 불변식(invariant)으로 설정하고, 퍼징 도구가 이를 만족하지 못하는 트랜잭션 조합을 찾아낼 수 있도록 구성해야 한다.

퍼징 테스트는 DevSecOps 파이프라인 내 정적 분석 및 동적 테스트 사이의 중간 단계로, 고위험 함수 또는 자산 이동 관련 함수에 대해 집중적으로 수행되는 것이 바람직하다. 또한 퍼징 테스트 결과는 테스트 커버리지, 실패 케이스, 경고 수준 등으로 구분되어 기록되고, 기존 테스트 시나리오에 대한 보완적 역할을 수행하며 향후 보안 강화 지침에 반영되어야 한다. 실증 환경에서는 퍼징 테스트에서 발견된 실패 트랜잭션을 기반으로 실제 사용자 행동과 유사한 위협 모델을 도출하거나, 잠재적인 비정상 입력 조건을 방어할 수 있는 입력 검증 로직을 추가하는 방식

으로 대응이 가능하다.

결론적으로 퍼징 테스트는 사전에 정의되지 않은 예외적 입력과 실행 흐름을 통해 스마트 컨트랙트의 경계 안정성을 확보하고, 보안 내재화를 실현하는 핵심 테스트 방법이며, DevSecOps의 자동화 기반 보안 검증 전략과도 밀접하게 연계되는 필수 요소이다.

파. 외부 감사 절차 공식화

스마트 컨트랙트는 코드가 블록체인에 배포된 이후 수정이 거의 불가능하다는 특성상, 사전에 이루어지는 보안 점검이 매우 중요하다. 그중에서도 외부 감사는 스마트 컨트랙트의 보안성을 제3자의 객관적인 시각에서 검토하는 핵심 절차로 간주되며, DevSecOps 흐름에서 릴리즈 이전에 반드시 수행되어야 하는 단계이다. 외부 감사 절차를 공식화하는 것은 릴리즈 준비 단계에서의 보안 강화를 제도적으로 구조화하는 작업이며, 이는 실증 환경에서 발생할 수 있는 보안 위협을 선제적으로 통제하는 역할을 한다.

감사 절차는 먼저 코드 동결 시점의 정의로부터 시작된다. 개발이 완료된 스마트 컨트랙트는 기능 확정 이후 코드 변경을 중단하고, 감사기관이 분석할 수 있도록 사전에 지정된 시간에 코드를 freeze 상태로 유지해야 한다. 이후 외부 감사기관과의 계약을 통해 보안 감사 일정을 수립하고, 감사 범위와 방법론, 리포트 제출 형식 등을 명확히 문서화한다. 감사기관은 정적 분석, 동적 분석, 위협 모델링 기반 점검을 포함한 종합적 기법을 사용하여 보안 취약점을 식별하며, 해당 결과를 기반으로 보고서를 작성해 제출한다. 이 보고서에는 발견된 이슈에 대한 설명, 위험도 분류, 제안 조치가 포함되며, 스마트 컨트랙트 배포 여부에 큰 영향을 미친다.

감사 이후에는 내부 검토단이 보고서를 분석하고, 발견된 이슈에 대한 대응 계획을 수립해야 한다. 고위험 항목은 반드시 수정되어야 하며, 중간 및 낮은 위험도는 우선순위에 따라 대응이 결정된다. 모든 대응은 문서화되어 추후 회고 및 사후 검토를 위한 기준이 되며, 필요시 외부 감사기관에 재감사를 요청하거나 수정 후 검토 절차를 반복할 수 있다. 최종적으로 모든 이슈가 해결되었을 때에만 릴리즈가 가능하며, 감사보고서와 대응 내역은 릴리즈 승인 문서와 함께 보관되어야 한다.

이러한 외부 감사 절차의 공식화는 보안 품질을 제3자의 검토로 확보함과 동시에 릴리즈 단계의 신뢰성과 책임성을 높인다. 실증 환경에서는 이를 통해 투자자, 사용자, 파트너로부터의 신뢰를 확보할 수 있으며, 사고 발생 시 대응 책임 소재와 조치 내역을 투명하게 제시할 수 있는 기반이 된다. 따라서 외부 감사는 단순 점검이 아니라 스마트 컨트랙트의 배포 결정과 직결되는 핵심 보안 거버넌스 요소로 작동해야 하며, DevSecOps에서 체계적인 릴리즈 보안 체계를 구성하는 필수 과정이다.

하. 릴리즈 승인 프로세스 강화

스마트 컨트랙트는 릴리즈 이후 변경이 어렵기 때문에 배포 전 마지막 단계인 릴리즈 승인 절차가 보안 측면에서 매우 중요하게 작용한다. 릴리즈 승인 프로세스를 강화하는 것은 단순한 형식적 절차를 넘어서, 개발 전반의 보안 활동이 실제로 효과를 발휘했는지 확인하고, 안전하게 배포 가능한 상태인지 판단하는 핵심 검증 단계이다. DevSecOps 관점에서는 이 과정을 공식화하여 모든 릴리즈에 대해 명확한 기준과 책임 구조를 수립하는 것이 필수적이다.

우선 릴리즈 승인 전에 스마트 컨트랙트의 바이트코드가 정확히 검증되어야 한다. 개발 환경과 배포 환경에서 생성된 바이트코드 해시가 일치하는지 확인함으로써, 감사 이후 코드가 무단으로 변경되지 않았음을 보장할 수 있다. 이를 위해 바이트코드 해시를 기록하고 릴리즈 문서에 포함시켜 이중으로 검증하는 절차가 필요하다. 또한 릴리즈 후보가 사용하는 라이브러리 버전, 디펜던시 구성, 환경설정 등의 변경 이력이 승인 전점검 대상에 포함되어야 한다.

배포 실행은 단일 담당자가 임의로 수행해서는 안 되며, 승인된 배포를 다중 서명 방식으로 처리하도록 구성하는 것이 바람직하다. 이를 위해 멀티시그 기반의 배포 승인 구조를 도입할 수 있으며, 이 과정에는 개발팀, 보안팀, 운영팀 등 각 역할자가 공동으로 참여한다. 각 담당자는 사전에 정의된 체크리스트를 기준으로 릴리즈 승인 여부를 검토하고, 모든 항목이 충족된 경우에만 배포를 허용한다. 체크리스트에는 외부 감사 보고서 반영 여부, 보안 테스트 통과 내역, 퍼징 및 동적 테스트 결과, 사고 대응 프로토콜 준비 여부 등이 포함된다.

이후 스마트 컨트랙트는 메인넷이나 테스트넷 상의 특정 계정에서 멀티시그 서명을 거쳐 배포되며, 배포 로그와 승인 기록은 저장되어야 한다. 릴리즈 승인 과정은 문서화되어 이후 재현 가능하도록 구성되고, 필요시 외부 이해관계자에게도 투명하게 제공될 수 있어야 한다. 실증 프로젝트나 실제 운영 환경에서는 이 승인 절차를 통해 릴리즈 자체의 신뢰도와 법적 책임성을 확보하게 된다.

릴리즈 승인 프로세스는 단순히 기술적 배포의 마무리가 아니라 전체 보안 활동이 실제 배포물로 이어지는 마지막 관문으로, 이 과정이 체계적이고 엄격하게 운영될수록 스마트 컨트랙트 배포의 안전성과 조직의 보안 성숙도는 함께 향상된다. DevSecOps 환경에서는 이와 같은 릴리즈 보안 승인 절차가 표준화된 체계로 통합

되어야 하며, 이를 통해 자동화된 보안 프로세스와 사람 기반의 책임 검토가 균형 있게 작동할 수 있도록 해야 한다.

거. 사고 대응 체계 확립화

스마트 컨트랙트 운영 과정에서 사고 발생 시 신속하고 체계적인 대응이 이루어지기 위해서는 사고 대응 체계를 확립하고 이를 조직 내 절차로 정착시키는 것이 필수적이다. 사고 대응은 단순한 패치 작업을 넘어 조직의 신뢰 회복, 피해 확산 방지, 법적/정책적 책임 이행에 이르기까지 광범위한 영향을 미치기 때문에 DevSecOps 환경에서는 사전에 명확하게 정의된 프로세스를 기반으로 운영되어야 한다.

사고 대응 절차는 감지, 보고, 분석, 차단, 복구, 후속 조치의 일련의 흐름으로 구성된다. 우선 이상 행위 또는 트랜잭션을 감지하는 모니터링 시스템이 실시간으로 작동하고 있어야 하며, 이벤트가 감지되면 즉시 대응 팀에 통보되어야 한다. 보고 단계에서는 사고의 발생 위치, 시간, 영향 범위, 관련 트랜잭션 정보를 신속히 수집하고 기록하는 체계를 갖추고 있어야 하며, 이를 위해 사고 발생 시 자동으로 로그를 백업하고 이를 구조화하는 시스템이 구축되어야 한다.

대응 단계에서는 사전에 정의된 우선순위 기준에 따라 사고의 심각도를 분류하고, 대응 전략을 실행한다. 예를 들어, 치명적인 재진입 공격이나 오작동으로 인한 자산 유실이 발생할 경우 긴급 중단 프로세스를 통해 컨트랙트 기능을 일시적으로 정지시키고, 사용자 인터페이스 또는 프록시 컨트랙트를 통해 제한적인 접근만 허용하는 방식으로 사고를 확산 이전에 차단해야 한다. 이와 같은 비상 대응 절차는 사전에 역할이 명시된 담당자들에게 즉시 할당되어야 하며, 각 단계별 수행 내역은 로그로 남겨야 한다.

복구 과정에서는 이상 상태를 일으킨 원인을 식별하고 해당 취약점을 수정한 후 테스트 및 검증을 거쳐 업데이트를 진행하며, 사용자들에게는 사고 발생 경과 및 대응 계획을 공지함으로써 신뢰를 회복해야 한다. 특히 복구 이후에는 사고 원인 분석과 대응 적절성을 평가하여 보고서를 작성하고, 이 내용을 조직의 사고 레퍼토리에 등록하여 유사 사고 예방 자료로 활용해야 한다.

사고 대응 체계는 정기적으로 점검되고 개선되어야 하며, 실제 사고 발생 가능성을 가정한 시뮬레이션이나 모의 훈련을 통해 절차의 실효성을 검증하는 과정도 필요하다. 이러한 체계는 운영 조직 내부뿐 아니라 외부 이해관계자에게도 명확하게 설명 가능한 형태로 문서화되어 있어야 하며, 사고 대응 절차를 단순한 가이드

라인이 아니라, 실행 가능한 운영 프로세스로 통합하고 반복 가능한 체계로 발전시켜야 한다.

너. DevSecOps 자동화 시스템 구성

DevSecOps 환경에서 스마트 컨트랙트의 보안을 체계적으로 유지하기 위해서는 보안 활동을 수동 절차에 의존하지 않고 자동화 시스템으로 통합하는 것이 필수적이다. 자동화는 보안 점검의 일관성과 반복성을 확보하고, 개발 속도와 보안 품질 간 균형을 유지하는 데 핵심적인 역할을 한다. 스마트 컨트랙트의 경우 릴리즈 이후 수정이 어렵기 때문에 초기 개발 단계부터 보안 검증이 자동으로 이루어져야 하며, 이를 위해 DevSecOps 자동화 시스템이 설계되고 운영되어야 한다.

자동화 시스템은 일반적으로 코드 저장소와 통합된 CI/CD 파이프라인에 보안 검증 단계를 삽입하는 방식으로 구현된다. 예를 들어 GitHub Actions, GitLab CI/CD, Jenkins 등과 같은 도구를 사용하여 pull request 또는 코드 푸시 시 자동으로 보안 테스트가 실행되도록 설정할 수 있다. 자동화된 단계에는 정적 분석 도구, 린터, 취약점 스캐너, 오픈소스 라이브러리 점검 도구 등이 포함되며, 각각은 미리 정의된 규칙에 따라 결과를 산출하고 그에 따라 빌드 승인 여부를 결정한다.

정적 분석 도구는 코드를 실행하지 않고 취약한 코드 패턴을 탐지하며, 린터는 코드 스타일과 일관성을 점검하고, 오픈소스 라이브러리 스캐너는 의존성에 존재하는 CVE 취약점을 탐지한다. 이러한 도구들의 결과는 통합 대시보드에 기록되며, 통과 여부에 따라 코드가 병합되거나 차단된다. 이 과정을 통해 모든 커밋과 배포 전 단계에서 보안 검증이 자동으로 내재화되고, 개발자 실수나 의도치 않은 취약점이 릴리즈로 이어지는 위험을 효과적으로 차단할 수 있다.

또한 자동화 시스템은 테스트넷 배포 및 테스트도 포함하여 운영할 수 있으며, 스마트 컨트랙트가 지정된 테스트 환경에 자동으로 배포되고, 기본 동작 테스트나 퍼징 테스트가 자동으로 실행되도록 설정할 수 있다. 이와 같은 구조는 릴리즈 준비 과정에서 신뢰도를 크게 높이며, 개발자와 보안 담당자가 지속적으로 협업할 수 있는 기반이 된다. 보안 실패가 반복되지 않도록 하기 위해 자동화 시스템은 각 실패 사례에 대한 리포트를 기록하고, 반복 발생하는 문제는 정책적으로 제한하거나 사전 예방 기능을 강화하는 방향으로 확장될 수 있다.

더. Security Champion 운영

스마트 컨트랙트 개발 조직에서 DevSecOps 문화 정착과 실질적인 보안 내재화를 실현하기 위해서는 개발팀 내부에서 보안의식을 주도적으로 확산시키고 실무 적용을 돕는 역할이 필요하다. 이를 위해 Security Champion 제도를 도입하는 것이 효과적이며, 이는 기존 보안 전문가만의 활동에 의존하는 방식에서 탈피하여 개발 조직 전반에 걸쳐 보안을 분산 책임화하고 자발적인 참여 기반으로 보안을 강화하는 방식이다.

Security Champion은 일반적으로 각 개발 스프린트나 프로젝트 단위로 지정되며, 개발자 중 보안에 대한 이해도가 높고 책임감이 있는 인원이 선정된다. 이들은 스프린트 동안 보안 관점에서 코드 변경 사항을 검토하고, 보안 테스트 로그를 확인하며, 의심되는 취약점이나 오탐 결과에 대해 개발자 및 보안 담당자 간의 조율자 역할을 수행한다. 또한 보안 도구의 사용법이나 취약점 대응 사례를 팀에 공유하고, 코드 리뷰 중 시큐어 코딩 가이드가 지켜졌는지를 점검하는 역할도 수행한다.

Security Champion은 특정 보안 전문 인력보다 개발팀 내에서 실질적인 영향을 끼칠 수 있으며, 빠른 피드백과 문화적 동기 부여에 유리하다. 예를 들어 코드 리뷰에서 반복적으로 발생하는 특정 보안 코드 패턴에 대해 가이드를 제작하거나, 취약점이 발견된 코드에 대해 직접 재작성 방향을 제시할 수 있어 실질적인 보안 품질 개선에 기여할 수 있다. 또한 주기적인 스프린트 회의나 회고 과정에서 보안 관련 이슈를 별도로 정리해 팀에 공유함으로써 팀 전체의 보안 감수성을 향상시킨다.

Security Champion은 단순히 역할만 부여하는 것이 아니라 조직적으로 이를 공식화하고 활동 이력을 관리해야 한다. 챔피언 지정 기준, 활동 범위, 권한, 평가 기준 등을 문서화하고, 이들이 수행한 보안 리뷰, 테스트 로그 검토, 개발자 교육 활동 등을 기록하여 정량적/정성적으로 측정할 수 있어야 한다.

러. 정기 보안 교육 및 버그바운티 프로그램 운영

스마트 컨트랙트의 보안을 DevSecOps 환경에서 실질적으로 강화하고 지속 가능한 형태로 유지하기 위해서는 조직 내부의 인적 자산에 대한 역량 강화가 필수적이다. 특히 스마트 컨트랙트는 일반적인 소프트웨어에 비해 배포 이후 수정이 어렵고 금전적 손실이 직접적으로 연결될 수 있기 때문에 개발자, 운영자, 기획자 등 모든 구성원에게 보안 지식을 체계적으로 교육하고 최신 위협 트렌드에 대한 감수성을 유지시키는 것이 중요하다. 이를 위해 정기적인 보안 교육 프로그램과 외부 보안 커뮤니티와 연계된 버그바운티 프로그램을 동시에 운영하는 전략이 요구된다.

정기 보안 교육은 내부 구성원 전체를 대상으로 연 2회 또는 분기별 계획을 수립하여 운영하며, 주요 교육 내용에는 스마트 컨트랙트 공격 사례 분석, 최신 취약점 유형 소개, 안전한 패턴 및 안티패턴, 실전 보안 대응 시뮬레이션 등이 포함된다. 특히 실무 중심 교육을 위해 실제 사고 사례나 테스트넷 기반 실습 자료를 활용하여 교육의 현실성과 몰입도를 높여야 하며, 교육 수료 여부 및 퀴즈, 실습 평가를 통해 이수율과 이해도를 정량적으로 평가할 수 있어야 한다.

이와 함께 버그바운티 프로그램은 외부 화이트해커 커뮤니티 또는 보안 전문가 집단을 대상으로 스마트 컨트랙트의 보안 결함을 사전에 탐지하고 보상하는 체계를 의미한다. 이는 개발 조직이 자체적으로 발견하지 못한 고위험 취약점을 조기 발견하는 데 효과적이며, HackerOne, Immunefi, Bugcrowd와 같은 플랫폼을 활용해 상시 또는 제한된 기간 동안 프로그램을 운영할 수 있다. 버그바운티는 보상 기준과 취약점 심각도 분류 체계를 미리 명시해야 하며, 결과에 대한 투명한 평가 절차와 보상 프로세스를 운영해야 한다.

버그바운티는 단순한 보안 결함 탐지 이상의 의미를 가지며, 외부 커뮤니티와의 신뢰 형성과 협업 생태계를 구축하는 수단이 된다. 스마트 컨트랙트는 누구나 열람 가능한 상태로 배포되기 때문에, 보안도 폐쇄적인 접근보다는 투명하고 개방적인 형태로 검증되었음을 보여주는 것이 사용자 신뢰 확보에도 긍정적 영향을 준다.

VI. 스마트 컨트랙트에서의 DevSecOps 효용

DAO 해킹 사건과 Qubit 해킹 사건을 사례로, SSDLC의 효과성을 검토한다. 이를 통해 보안이 내재화된 소프트웨어의 중요성과 사고를 예방, 완화하는데 효과성이 있음을 보인다. 해당 사례뿐만 아니라 스마트 컨트랙트 작성 시 개인정보와 관련된 경우 T1. GDPR 체크리스트 확인을 하여 개인정보를 보호해야 하며, T4. 사고 패턴 분석 및 T8. 사고 레퍼지토리 확보를 통해 버그나 보안약점을 파악하고, T7. 안정성 확보를 통해 보안을 내재화시켜 사전에 사고를 예방할 수 있도록 한다.

가. DAO 해킹 사례

DAO는 1억 5천만 달러의 자금을 모금했지만, 해킹을 통해 약 6천만 달러가 도난당했다. 공격자는 스마트 컨트랙트에서 재귀 호출 관련 보안약점을 이용하여 공격을 수행했다.

DAO 해킹 사태와 같은 사건들은 다음과 같이 SSDLC를 적용하고 기존의 표 1과 같은 코드를 표 2와 같이 방어코드를 리팩토링 하면, 사전에 재진입과 같은 공격을 방지 할 수 있다.

T2. 토큰 전송 및 인출을 처리하는 계약 기능이 재진입 공격에 취약하지 않도록 보장하기 위한 요구 사항 추가

T3. Checks-Effect-Interactions 패턴을 사용 및 Openzeppelin의 ReentrancyGuard를 사용하여 재진입 공격 방지

T5. 정적 분석 및 확인: 정적 분석 도구를 사용하여 코드의 보안 약점을 식별, 문제점을 사전에 발견 하여 리팩토링 진행

T6. 동적 테스트: Fuzz 테스트 또는 기호, 논리 기반 테스트와 같은 도구를 통해 가능한 모든 경우를 테스트하여 사전에 보안약점을 발견 하여 올바르게 동작하는지 확인

T9. Dapp 모니터링: 반복적인 인출과 같은 비정상적인 거래 패턴을 감지하고

신속하게 대응하여 공격 완화 가능

DAO 해킹 사건과 Qubit 해킹 사건을 사례로, SSDLC의 효과성을 검토한다. 이를 통해 보안이 내재화된 소프트웨어의 중요성과 사고를 예방, 완화하는데 효과성이 있음을 보인다. 해당 사례뿐만 아니라 스마트 컨트랙트 작성 시 개인정보와 관련된 경우 T1. GDPR 체크리스트 확인을 하여 개인정보를 보호해야 하며, T4. 사고 패턴 분석 및 T8. 사고 레퍼지토리 확보를 통해 버그나 보안약점을 파악하고, T7. 안정성 확보를 통해 보안을 내재화시켜 사전에 사고를 예방할 수 있도록 한다

표 1. DAO Bad Case

```
function splitDAO(uint _proposalID, address _newCurator) public {

        uint senderShares = shares[msg.sender];
        require(senderShares > 0);

        uint funds = senderShares * totalFunds / totalShares;

        shares[msg.sender] = 0;
        totalShares -= senderShares;

        withdrawFunds(funds);
        totalFunds -= funds;

    }
```

표 2. Dao Good Case

```
function splitDAO(uint _proposalID, address _newCurator) public {
        uint senderShares = shares[msg.sender];

        // Checks
        require(senderShares > 0);

        uint funds = senderShares * totalFunds / totalShares;

        // Effects
        shares[msg.sender] = 0;
        totalShares -= senderShares;
```

```
        totalFunds -= funds;

        // Interactions
        withdrawFunds(funds);
    }
```

나. Qubit 해킹 사례

Qubit은 Qbridge라는 자체 브릿지를 통해 이더리움을 담보로 BSC를 대출 토큰을 전송을 위해 SafeToken 라이브러리의 safeTransferFrom 함수를 사용하여 주소 0에대한 예외 처리를 하지 않아, 주소 0이 입력되었을 때, Bridge에서 실패하지 않고, 정상적으로 전송되었다는 Event로그가 남고, 자금을 탈취 당했다.

Qubit 해킹 사태와 같은 공격들을 다음과 같은 SSDLC를 적용하고 기존의 표 3과 같이 작성된 코드를 표 4와 같이 방어코드를 리팩토링 하여 예외 처리를 하여 공격을 사전에 예방할 수 있다.

T2. QBridge 프로토콜에 보안 약점을 분석하여 데이터 입력의 유효성 검사하고 address(0) 및 외부 소유 계정(EOA) 문제에 대한 보안 요구사항을 추가함

T3. 데이터 입력 유효성 검사를 진행하고, 보안성이 확보된 OpenZeppelin의 SafeERC20 라이브러리 사용

T5. 정적 분석을 수행하여 deposit() 함수 및 safeTransferFrom() 함수에 대한 보안약점 및 EOA을 사전 식별 하고, 주소, 금액 및 함수 매개변수를 등 입력이 검증되었는지 확인

T7. 동적 테스트를 수행하여 Edge case testing, Functional testing, fuzz test 등을 사용하여 0 주소, 0 값 및 다른 모든 경우를 테스트하여 계약이 예상대로 작동하는지 확인

T9. 이상 또는 의심스러운 활동을 감지 및 인출 트랜잭션을 지속적으로 모니터링

표 3. Qubit Bad Case

```
function deposit(bytes32 resourceID, address depositer, bytes calldata data) external
override onlyBridge {

    uint option;
    uint amount;
    (option, amount) = abi.decode(data, (uint, uint));

    address tokenAddress = resourceIDToTokenContractAddress[resourceID];
    require(contractWhitelist[tokenAddress], “provided tokenAddress is not
whitelisted“);

    if (burnList[tokenAddress]) {
        require(amount >= withdrawalFees[resourceID], “less than withdrawal fee“);
        QBridgeToken(tokenAddress).burnFrom(depositer, amount);
    } else {
        require(amount >= minAmounts[resourceID][option], “less than minimum
amount“);
        tokenAddress.safeTransferFrom(depositer, address(this), amount);
    }

}
```

표 4. Qubit Good Case

```
function deposit(bytes32 resourceID, address depositer, bytes calldata data) external
override onlyBridge {

    // Check if the depositer address is the zero address
    require(depositer != address(0), “depositer address cannot be zero address“);

    uint option;
    uint amount;
    (option, amount) = abi.decode(data, (uint, uint));

    address tokenAddress = resourceIDToTokenContractAddress[resourceID];
    require(contractWhitelist[tokenAddress], “provided tokenAddress is not
whitelisted“);

    if (burnList[tokenAddress]) {
        require(amount >= withdrawalFees[resourceID], “less than withdrawal fee“);
        QBridgeToken(tokenAddress).burnFrom(depositer, amount);
    } else {
```

```
        require(amount >= minAmounts[resourceID][option], "less than minimum
amount");
        tokenAddress.safeTransferFrom(depositer, address(this), amount);
    }
}
```

참고문헌

[1] Smart Contract Security Verification Standard(SCSVS)

[2] Secure Software Lifecycle Knowledge Area Issue 1.0

[3] Blockchain Enabled Smart Contract Based Applications:Deficiencies with the Software Development Life Cycle Models

[4] Software Security(Gary Mcgraw)

[5] Ian Sommerville Software Engineering 10th

[6] SWC Registry

[7] https://devops.com/using-calms-to-assess-organizations-devops/

[8] https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

[9] https://developer.ibm.com/articles/waterfall-model-advantages-disadvantages/

[10] https://aws.amazon.com/ko/devops/what-is-devops/

[11] https://www.ibm.com/docs/en/cloud-private/3.2.0?topic=started-cloud-private-devops

[12] https://azure.microsoft.com/en-us/overview/what-is-devops/#devops-overview

[13] https://www.redhat.com/ko/topics/devops

[14] https://www.synopsys.com/glossary/what-is-devops.html

[15] https://www.atlassian.com/devops

[16] https://searchitoperations.techtarget.com/definition/DevOps

[17] https://cryptotips.eu/en/knowledge-base/what-are-decentralized-applications-dapps/

[18] https://www.geeksforgeeks.org/what-are-decentralized-apps-dapps-in-blockchain/

[19] https://ethereum.org/en/dapps/#what-are-dapps

[20] https://blockchainhub.net/decentralized-applications-dapps/

[21] https://docs.microsoft.com/en-us/archive/msdn-magazine/2019/october/blockchain-devops-for-blockchain-smart-contracts

[22] https://www.scribd.com/document/394591463/DevOps-for-Blockchain-Smart-Contracts

[23] https://www.securing.pl/en/smart-contracts-security-verification-standard/

[24] Myrbakken, Håvard, and Ricardo Colomo-Palacios. “DevSecOps: a multivocal literature review.“ Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4-5, 2017, Proceedings.

Springer International Publishing, 2017.

[25] Rajapakse, Roshan N., et al. "Challenges and solutions when adopting DevSecOps: A systematic review." Information and software technology 141 (2022): 106700.

[26] Dhillon, Vikram, et al. "The DAO hacked." blockchain enabled applications: Understand the blockchain Ecosystem and How to Make it work for you (2017): 67-78.

[27] DeFi protocol Qubit Finance Exploited for $80M, https://www.coindesk.com/markets/2022/01/28/defi-protocol-qubit-finance-exploited-for-80m/