KAI PROTOCOL
SECURITY ASSESSMENT REPORT

MAY. 4 - MAY. 18, 2021
MAY. 26 - JUN. 4, 2021
JUN. 13 - JUN. 14, 2021

시작하기 전에

- 본 문서는 블록체인 보안 전문업체 SOOHO에서 진행한 취약점 검사를 바탕으로 작성한 문서로, 보안 취약점의 발견에 초점을 두고 있습니다. 추가적으로 코드 퀄리티 및 코드 라이센스 위반 사항 등에 대해서도 논의합니다.
- 본 문서는 코드의 유용성, 코드의 안정성, 비즈니스 모델의 적합성, 비즈니스의 법적인 규제, 계약의 적합성, 버그 없는 상태에 대해 보장하거나 서술하지 않습니다. 감사 문서는 기술적 논의 목적으로만 사용됩니다.
- SOOHO는 회사 정보가 대외비 이상의 성격을 가짐을 인지하고 사전 승인 없이 이를 공개하지 않습니다.
- SOOHO는 업무 수행 과정에서 취득한 일체의 회사 정보를 누설하거나 별도의 매체를 통해 소장하지 않습니다.
- SOOHO는 스마트 컨트랙 분석에 최선을 다하였음을 밝히는 바입니다.

SOOHO 소개

SOOHO는 Audit Everything, Automatically란 슬로건으로 지속적인 보안을 위해 필요한 기술을 연구하고 서비스합니다. 자체 취약점 분석기들과 오픈소스 분석기들을 기반으로 모든 개발 생애 주기에 걸쳐 취약점들을 검사합니다. SOOHO는 자동화 도구를 연구, 개발하는 보안 분야 박사 연구원들과 탐지 결과와 컨트랙트 코드를 깊게 분석하는 화이트 해커들로 구성되어 있습니다. 보안 분야 전문성을 바탕으로 파트너 사의 컨트랙트를 알려진 취약점과 Zero-day 취약점의 위협으로부터 안전하게 만들어줍니다.

개 요

2021년 5월 4일에서 5월 18일, 2021년 5월 26일에서 6월 4일, 2021년 6월 13일부터 14일 동안 SOOHO는 KAI Protocol의 스테이블 코인 프로젝트에 대한 취약점 분석을 진행하였습니다. 감사 기간 동안 아래의 작업을 수행했습니다.

- SOOHO의 자체 취약점 검사기를 통한 취약점 탐지 및 결과 분석
- 컨트랙트 보안 취약점 의심 지점에 대한 익스플로잇(Exploit) 코드 작성
- 컨트랙트 코드 모범 사례와 시큐어 코딩 가이드를 바탕으로 코드의 수정 권고 사항 작성

총 2명의 보안 전문가가 컨트랙트의 취약점을 분석하였습니다. 참여한 보안 전문가는 Defcon, Nuit du Hack, 화이트햇, SamsungCTF 등 국내외의 해킹 대회에서 수상을 하고 보안분야 박사 학위의 학문적 배경을 가지는 등 우수한 해킹 실력과 경험을 가지고 있습니다.

SOOHO를 통해 알려진 취약 코드 시그니쳐를 해당 컨트랙트에서 스캐닝하였습니다. 추가적으로 SOOHO의 VeriSmart를 이용해 Arithmetic 연산에 대해 형식 검증하였습니다.

발견된 취약점은 총 10개로 순서대로 Critical 3개, High 3개, Low 3개, Note 1개 입니다. 이슈에는 가스 최적화나 모범 사례 준수 등은 제외되었습니다. 발생할 수 있는 여러 가능성에 대해 많은 고려를 한 개발진들의 기술력을 확인할 수 있었습니다. 꾸준한 코드 감사를 통해 서비스의 안정을 도모하고 잠재적인 취약점에 대한 분석을 하는 것을 추천 드립니다.

분석 대상

2021년 5월 4일에서 5월 18일, 2021년 5월 26일에서 6월 4일, 2021년 6월 13일부터 14일 동안 아래의 프로젝트를 분석하였습니다.

Project	kdollar-contract-20210428
File Hash	f6fd2135
# of Files	35
# of Lines	2,622

Project	kdollar-contract-20210525
File Hash	ba3f5f03
# of Files	46
# of Lines	3,404

Project	kaiprotocol-core-20210614
File Hash	e1e7d92f
# of Files	27
# of Lines	2,069

주요 감사 포인트 및 프로세스

KAI Protocol은 알고리즘 기반의 스테이블 코인 프로젝트입니다. KAI Protocol은 Basis Cash 프로젝트와 해당 프로젝트에서 파생된 Basis Dollar. 그리고 Basis Dollar로부터 파생된 프로젝트로 Share와 Bond를 통해 안정화된 가격의 스테이블 코인을 유지하는 시스템입니다. 이에 따라 자산의 소유권이 탈취 가능한지, 화폐 발행 과정에서 문제는 없는지, 운영 과정에서 발생할 수 있는 해킹 시나리오에 대해 취약한지를 위주로 검증하였습니다.

예를 들어, 관리자가 아닌 임의의 유저가 거래소에 영향을 끼칠 수 있을지, 거래는 문제없이 중개되는지, 트랜잭션의 성공/실패에 대해 모두 잘 처리되는지 등의 시나리오가 이에 해당됩니다. 관리자에 의한 내부 해킹(e.g., 러그풀)은 발생하지 않음을 전제하였습니다. 또한, 가격의 변동성 등의 프로토콜의 설계는 본 감사의 범위가 제외됨을 밝힙니다.

1

취약점 자동 분석
소프트웨어 검사

2

보안 전문가의
코드 직접 분석

다음을 고려합니다.

- 보다 큰 리스크를 내포하는 코드를 중점적으로 분석합니다.
- 접근 권한의 관리가 잘 되고 있는지 검사합니다.
- 파트너 사가 의도한대로 코드가 작성되었는지 분석합니다.

3

Exploit 분석과
Poc 작성

Exploit 분석과 PoC 작성은 다음을 목표로 합니다.

- 코드 실행을 통한 동적 분석
- 발견된 취약점의 악용을 통해 실제 금전적인 이득을 얻을 수 있는지 살펴봅니다. (e.g., 무한 인출)
- 발견된 취약점을 악용해 서비스에 악영향을 끼칠 수 있는지 살펴봅니다. (e.g., Mint)

취약점의 심각성 척도

발견된 취약점은 심각성 척도를 기준으로 나열해서 설명합니다.

Critical High Medium Low Note

심각성 척도는 우측 OWASP의 Impact & Likelihood 기반 리스크 평가 모델을 기반으로 정해졌습니다. 해당 모델과 별개로 심각도가 부여된 이슈는 해당 결과에서 그 이유를 서술합니다.

Impact	Likelihood: Low	Likelihood: Medium	Likelihood: High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Note	Low	Medium

Severity

1차 분석 결과

분석 결과는 취약점 분석 과정에서 중점적으로 살펴본 이슈들과 결과에 대한 내용을 포함하고 있습니다

(RESOLVED) ACCESS CONTROL High

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `Bond.sol`
File Location : `kdollar-contract-20210428/assets/`
`└── Bond.sol`

```
27      function burn(uint256 amount) public {
28          super.burn(amount);
29      }
```

이슈 설명 `burn` 함수에는 접근 권한이 없기 때문에 누구나 토큰 소각이 가능합니다. `onlyOperator`를 통해 `operator`만이 실행하도록 수정하는 것을 권장합니다.

(RESOLVED) ACCESS CONTROL High

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `Cash.sol`
File Location : `kdollar-contract-20210428/assets/`
`└── Cash.sol`

```
34      function burn(uint256 amount) public {
35          super.burn(amount);
36      }
```

이슈 설명 `burn` 함수에는 접근 권한이 없기 때문에 누구나 토큰 소각이 가능합니다. `onlyOperator`를 통해 `operator`만이 실행하도록 수정하는 것을 권장합니다.

(RESOLVED) NEXT OWNER CAN BE NULL High

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `AirdropOperator.sol`
File Location : `kdollar-contract-20210428/`
`└── AirdropOperator.sol`

```
58      function changeNextOwner(address _nextOwner) public onlyOwner {
59          nextOwner = _nextOwner;
60      }
```

이슈 설명 `nextOwner`가 연산이 될 때 주소가 Null 인지 검사를 하는 로직이 빠져 있어서 `changeOwner` 함수가 무용지물이 될 수 있습니다. Null 검사를 추가하는 것을 권장합니다.

2차 분석 결과

분석 결과는 취약점 분석 과정에서 중점적으로 살펴본 이슈들과 결과에 대한 내용을 포함하고 있습니다

(RESOLVED) TREASURYIMPL CAN BE REINITIALIZED Critical

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryImpl.sol`
File Location : `kdollar-contract-20210525/`
`└── TreasuryImpl.sol`

```
163     function initialize (
164         address _dollar,
165         address _bond,
166         address _share,
167         uint256 _startTime
168     ) external onlyAdmin {
```

이슈 설명 함수 `initialize`는 단 한번만 호출되게 제어되어야 합니다. 이에 따라 호출된 적이 있는지를 저장하여 단 한번만 호출되도록 설계해야 합니다.

(NON ISSUE) DOLLAR PRICE SHOULD BE UPDATED Critical

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryImpl.sol`
File Location : `kdollar-contract-20210525/`
`└── TreasuryImpl.sol`

```
289     function buyBonds(uint256 _dollarAmount, uint256 targetPrice)
290         require(_dollarAmount > 0, "Treasury: cannot purchase bonc
```

```
309         IKayAsset(dollar).burnFrom(msg.sender, _dollarAmount);
310         IKayAsset(bond).mint(msg.sender, _bondAmount);
311
312         roundSupplyContractionLeft = roundSupplyContractionLeft.sub(_dollarAmount);
```

```
317     function redeemBonds(uint256 _bondAmount, uint256 targetPrice)
318         require(_bondAmount > 0, "Treasury: cannot redeem bonds wi
```

```
362     function allocateSeigniorage() external
363         _updateDollarPrice();
```

개발팀에서 해당 부분에 대해 확인하여 가격 변동에 영향이 없기 때문에 호출을 하지 않음을 확인하였습니다. 이에 따라 논이슈 처리하였습니다.

이슈 설명 함수 `buyBonds`에서 dollar를 소각하고 bond를 mint 한 후에 유통량 변동에 따라 `_updateDollarPrice` 함수를 호출해야 합니다. 마찬가지로 bonds의 양과 dollar의 양에 영향을 주는 `redeemBonds`와 `allocateSeigniorage`의 함수 호출부 마지막에서도 `_updateDollarPrice` 함수를 호출해야 합니다.

2차 분석 결과

분석 결과는 취약점 분석 과정에서 중점적으로 살펴본 이슈들과 결과에 대한 내용을 포함하고 있습니다

PENDINGIMPLEMENTATION CAN BE NULL Low

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryUni.sol`
File Location : `kdollar-contract-20210525`
└── `TreasuryUni.sol`

```
function _acceptImplementation() public {
    // Check caller is pendingImplementation
    require(msg.sender == pendingImplementation, "accept pending
```

이슈 설명 `pendingImplementation`에 대한 조건식이 없기 때문에 null이 될 수 있고 이에 따라 `implementation` 또한 null이 될 수 있는 여지가 있습니다. Compound 프로젝트와 같이 `pendingImplementation != address(0)` 조건식을 추가해야합니다

(RESOLVED) BOARD ROOM CAN BE NULL Low

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `VoteProxy.sol`
File Location : `kdollar-contract-20210525`
└── `VoteProxy.sol`

```
23      function setBoardroom(address newBoardroom)
24          address oldBoardroom = boardroom;
25          boardroom = newBoardroom;
```

해당 파일이 삭제되어 이슈 해결됨 처리하였습니다.

이슈 설명 `boardroom`이 null이 될 수 있기 때문에 이에 대한 조건식을 추가해야 합니다

TREASURYIMPL CAN BE CHANGED Note

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryUni.sol`
File Location : `kdollar-contract-20210525`
└── `TreasuryUni.sol`

```
function _acceptImplementation() public {
    // Check caller is pendingImplementation
    require(msg.sender == pendingImplementation, "accept pending

    // Save current values for inclusion in log
    address oldImplementation = implementation;
    address oldPendingImplementation = pendingImplementation;

    implementation = pendingImplementation;
```

이슈 설명 `treasury`에 대한 로직이 정의되는 컨트랙트 주소를 관리자가 변경될 수 있습니다. 서비스 이용자들의 동의와 별개로 변경될 수 있는 부분은 잠재적인 보안 이슈를 내포합니다.

3차 분석 결과

분석 결과는 취약점 분석 과정에서 중점적으로 살펴본 이슈들과 결과에 대한 내용을 포함하고 있습니다

BOARDROOM CAN BE REINITIALIZED Critical

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `Boardroom.sol`
File Location : `kaiprotocol-core-20210614`
└── `Boardroom.sol`

```
107     function initialize(
108         IERC20 _kai,
109         IERC20 _skai,
110         ITreasury _treasury
111     ) public onlyOperator {
112         kai = _kai;
113         skai = _skai;
114         treasury = _treasury;
115 
116         emit Initialized(msg.sender, block.number);
117     }
```

이슈 설명 함수 `initialize`는 단 한번만 호출되게 제어되어야 합니다. 이에 따라 호출된 적이 있는지를 저장하여 단 한번만 호출되도록 설계해야 합니다.

ADDRESS CAN BE NULL Low

분석 결과에 대한 추가적인 자료 및 코멘트

File Name : `TreasuryImpl.sol`
File Location : `kaiprotocol-core-20210614`
└── `TreasuryImpl.sol`

```
203     function setBoardroom(address _boardroom) external onlyAdmin {
204         boardroom = _boardroom;
205     }
206 
207     function setDollarOracle(address _dollarOracle) external onlyAdmin {
208         dollarOracle = _dollarOracle;
209     }
210 
211     function setRound(uint256 _round) external onlyAdmin {
212         round = _round;
213     }
```

이슈 설명 `boardroom, dollarOracle, round`가 null이 될 수 있기 때문에 이에 대한 조건식을 추가하는 것을 권장합니다.

분석 결과

분석 결과는 취약점 분석 과정에서 중점적으로 살펴본 이슈들과 결과에 대한 내용을 포함하고 있습니다

분석하였습니다 - MATHEMATICAL OPERATIONS ✓

분석 결과에 대한 추가적인 자료 및 코멘트

설명 KAI Protocol의 연산 상에서 문제가 발생하는지를 확인하였습니다.

분석하였습니다 - REENTRANCY ✓

분석 결과에 대한 추가적인 자료 및 코멘트

설명 KAI Protocol에서 발생가능한 재진입 공격에 대해 분석하였습니다.

최종 결과 요약 및 결론

KAI Protocol의 코드는 이해하기 쉽게 명명되고 용도와 쓰임에 따라 잘 설계되어 있습니다. 특히, 발생가능한 상황에 대해 적절한 처리를 신경 쓴 부분과 테스트 코드를 작성한 노력이 돋보였습니다. 대부분의 경우 모범 사례를 따르고 있습니다. 코드 검사 결과, **발견된 취약점은 총 10개로 순서대로 Critical 3개, High 3개, Low 3개, Note 1개 입니다.** 분석한 코드에 대한 꾸준한 코드 감사를 통해 서비스의 안정을 도모하고 잠재적인 취약점에 대한 분석을 하는 것을 추천드립니다.

Project	kaiprotocol-core-20210614
File Hash	e1e7d92f
# of Files	27
# of Lines	2,069

```
kaiprotocol-core-20210614
├── BBFund.sol
├── BBFundImpl.sol
├── BBFundStorage.sol
├── Boardroom.sol
├── OracleKlayswap.sol
├── Timelock.sol
├── TreasuryImpl.sol
├── TreasuryStorage.sol
├── TreasuryUni.sol
├── assets
│   ├── KAI.sol
│   ├── KAIBond.sol
│   ├── KAIShare.sol
│   └── kERC20.sol
├── interfaces
│   ├── IBoardroom.sol
│   ├── IKAIAsset.sol
│   ├── IKlayExchange.sol
│   ├── IKlayswapFactory.sol
│   ├── IKlayswapStore.sol
│   ├── IOracle.sol
│   └── ITreasury.sol
├── lib
│   ├── Babylonian.sol
│   ├── FixedPoint.sol
│   └── UQ112x112.sol
├── owner
│   ├── Operator.sol
│   └── Ownable.sol
└── utils
    ├── ContractGuard.sol
    └── Epoch.sol
```