

Sicherheitsanalyse von Smart Contracts auf Ethereum

Bachelorarbeit im Studiengang Informatik (B.Sc.)

Frankfurt University of Applied Sciences | 2025

Yousef Ghanem

Agenda

- Motivation, Zielsetzung und Forschungsfrage
- Methodik und Vorgehensweise
- Blockchain, Ethereum und Smart Contract
- Entwicklung und Analyse (CryptoBank)
- Verbesserungen und erneute Analyse (CryptoBankSecure)
- Ergebnisse und Diskussion
- Zusammenfassung und Ausblick

Motivation, Zielsetzung und Forschungsfrage

Motivation

- Hohe Relevanz in der Praxis
- Interesse an Sicherheit & Risiken
- Theorie & Praxis verbinden

Zielsetzung

- Sicherheit von Smart Contracts auf Ethereum analysieren
- Typische Schwachstellen in der Praxis identifizieren

Forschungsfrage

„Welche sicherheitsrelevanten Risiken treten bei der Entwicklung von Smart Contracts auf Ethereum auf und wie können diese wirksam erkannt und behoben werden?“

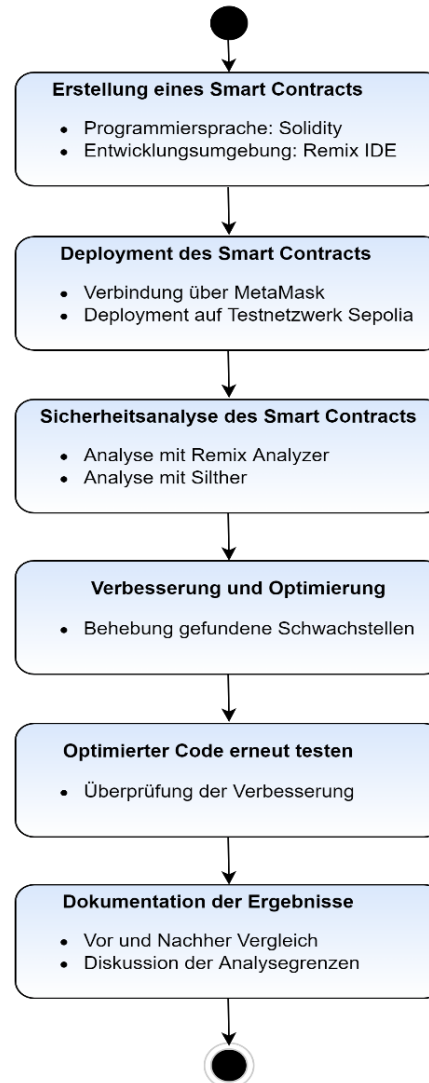
Methodik und Vorgehensweise

Theorie

- Blockchain, Ethereum und Smart Contracts

Praxis

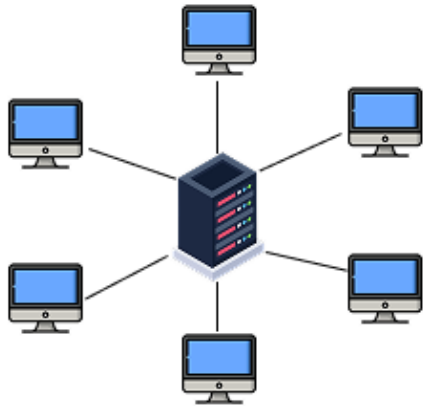
- Entwicklung mit Solidity & Remix
- Deployment auf Sepolia-Testnetz mit MetaMask
- Sicherheitsanalyse mit Remix und Slither
- Codeverbesserung & erneuter Test



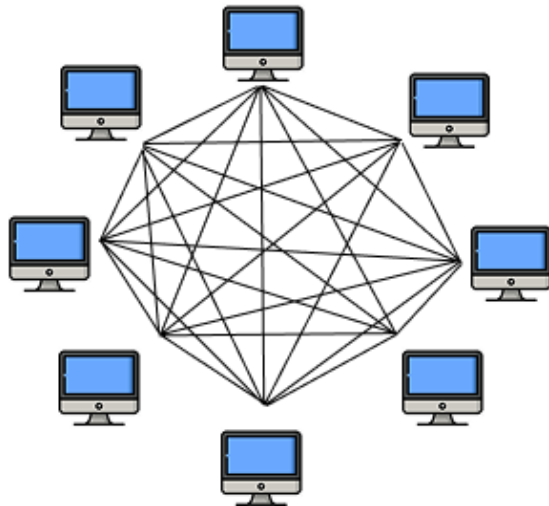
Blockchain, Ethereum und Smart Contract

Blockchain

- Dezentrale Datenstruktur aus verketteten Blöcken
- Manipulation kaum möglich



Zentralisiert



Dezentralisiert

Ethereum

- Plattform für dezentrale Anwendung
- Führt Programme in der Ethereum Virtual Machine aus

Smart Contracts

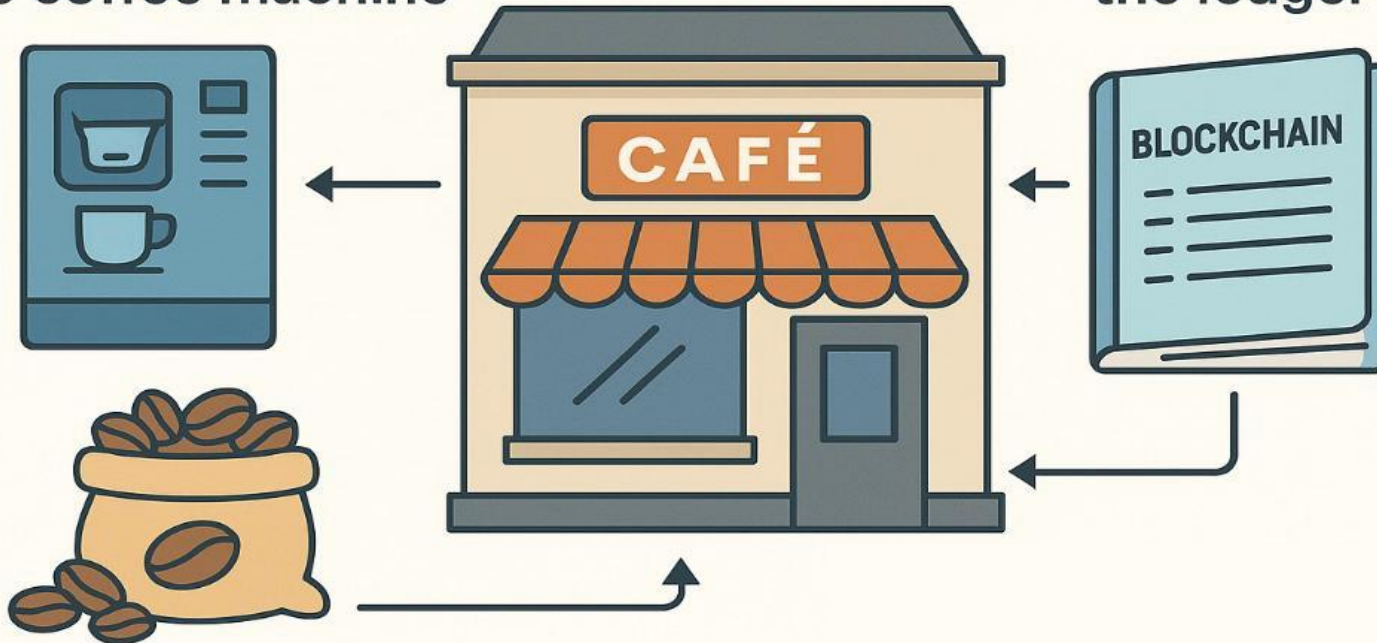
- Automatisierte Verträge auf der Blockchain
- Unveränderlich & deterministisch

Blockchain, Ethereum und Smart Contract

Ethereum = the café

Smart Contract
the coffee machine

Blockchain =
the ledger



Ether =
the coffee beans

Entwicklung und Analyse (CryptoBank)

Entwicklung

- Remix IDE
 - Entwicklungsumgebung
 - Kompilieren & Deployen
- Solidity
 - Programmiersprache
 - Ähnlich wie JS & C++

Deployen

- MetaMask
 - Wallet & Schnittstelle zum Ethereum Netzwerk
 - Verbinde Remix mit Sepolia
- Testnetz Sepolia
 - Offizielles Ethereum Testnetz
- Etherscan (Sepolia)
 - Blockchain-Explorer für Sepolia
 - Zeigt alle Transaktionen und Contracts

Analyse

- Remix Analyzer
 - Analyse-Plugin
- Slither
 - Externes Analysetool (Terminal)
 - Detaillierte Codeprüfung auf Sicherheitslücken

Entwicklung und Analyse (CryptoBank)

```
contract CryptoBank {  
  
    // hier werden die Attribute definiert  
    address public besitzer;  
    mapping(address => uint) public kontostand;  
  
    // Events für Transparenz und Analyse  
    event eingezahlt(address indexed benutzer, uint betrag);  
    event ausgezahlt(address indexed empfaenger, uint betrag);  
    event etherEmpfangen(address absender, uint betrag);  
    event FallbackGenutzt(address absender, uint value, bytes data);
```

```
    // funktion für die Einzahlung  
    function einzahlen() public payable {  
        require(msg.value > 0, "Ether wurde nicht gesendet.");  
        kontostand[msg.sender] += msg.value;  
        emit eingezahlt(msg.sender, msg.value);  
    }
```

```
    constructor() {  
        besitzer = msg.sender;  
    }
```

```
    // funktion für die Auszahlung: Diese Funktion wird nur von Besitzer ausgeführt  
    function auszahlen(address payable _empfaenger, uint _betrag) public {  
        require(msg.sender == besitzer, "Sie sind nicht der Besitzer!");  
        require(kontostand[_empfaenger] >= _betrag, "Kontostand ist nicht genug!");  
        kontostand[_empfaenger] -= _betrag;  
        _empfaenger.transfer(_betrag);  
        emit ausgezahlt(_empfaenger, _betrag);  
    }
```

```
    receive() external payable {  
        emit etherEmpfangen(msg.sender, msg.value);  
    }  
  
    // vordefinierte Funktion: wenn Funktion nicht existiert  
    fallback() external payable {  
        emit FallbackGenutzt(msg.sender, msg.value, msg.data);  
    }
```

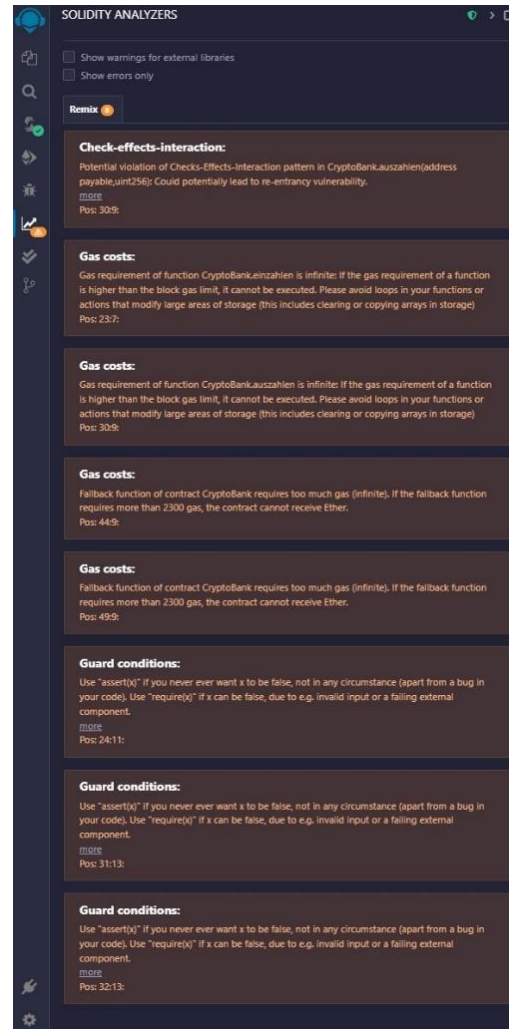

Entwicklung und Analyse (CryptoBank)

Remix Analyzer

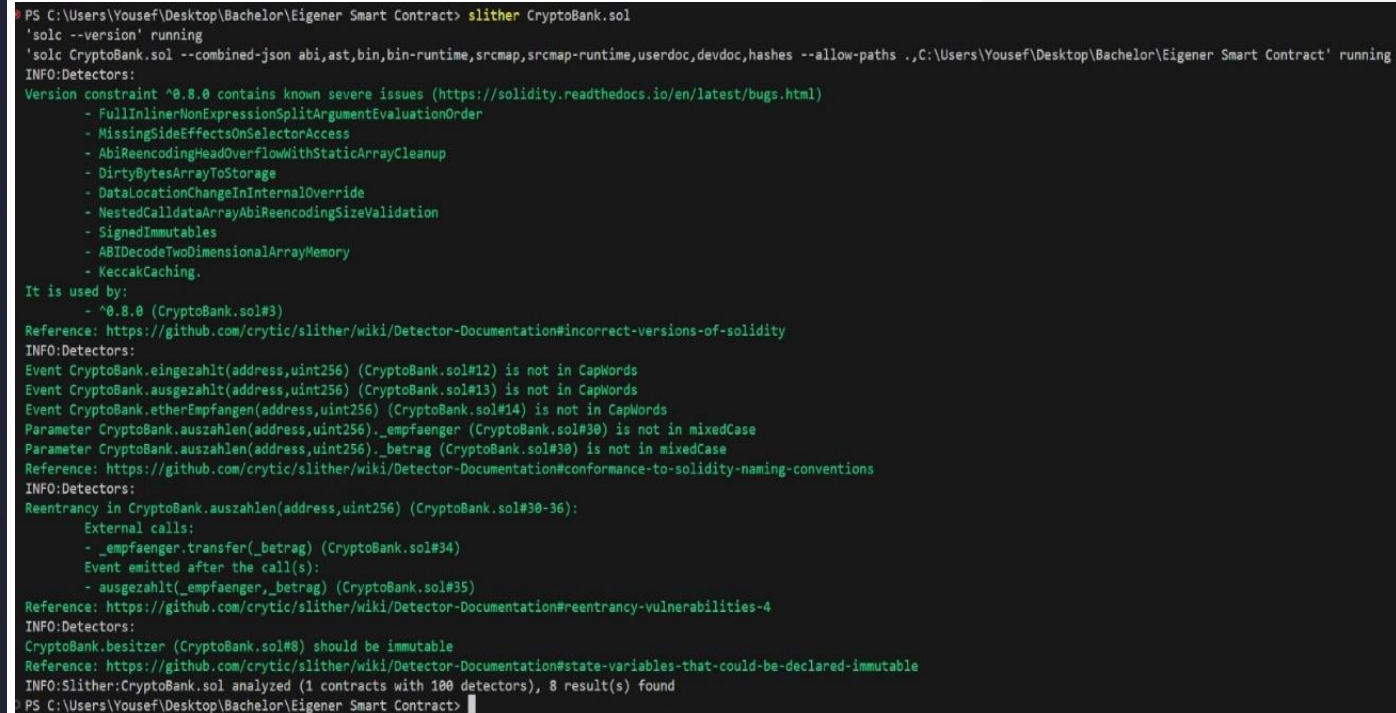
- Check-Effects-Interaction (Reentrancy-Risiko)
- Gasverbrauch
- Guard Conditions

Slither Analyzer

- Reentrancy-Warnung in `auszahlen()`
- Solidity-Version `^0.8.0`
- Stilistische Hinweis (Events & Parameter)
- Empfehlung: `besitzer` als `immutable`



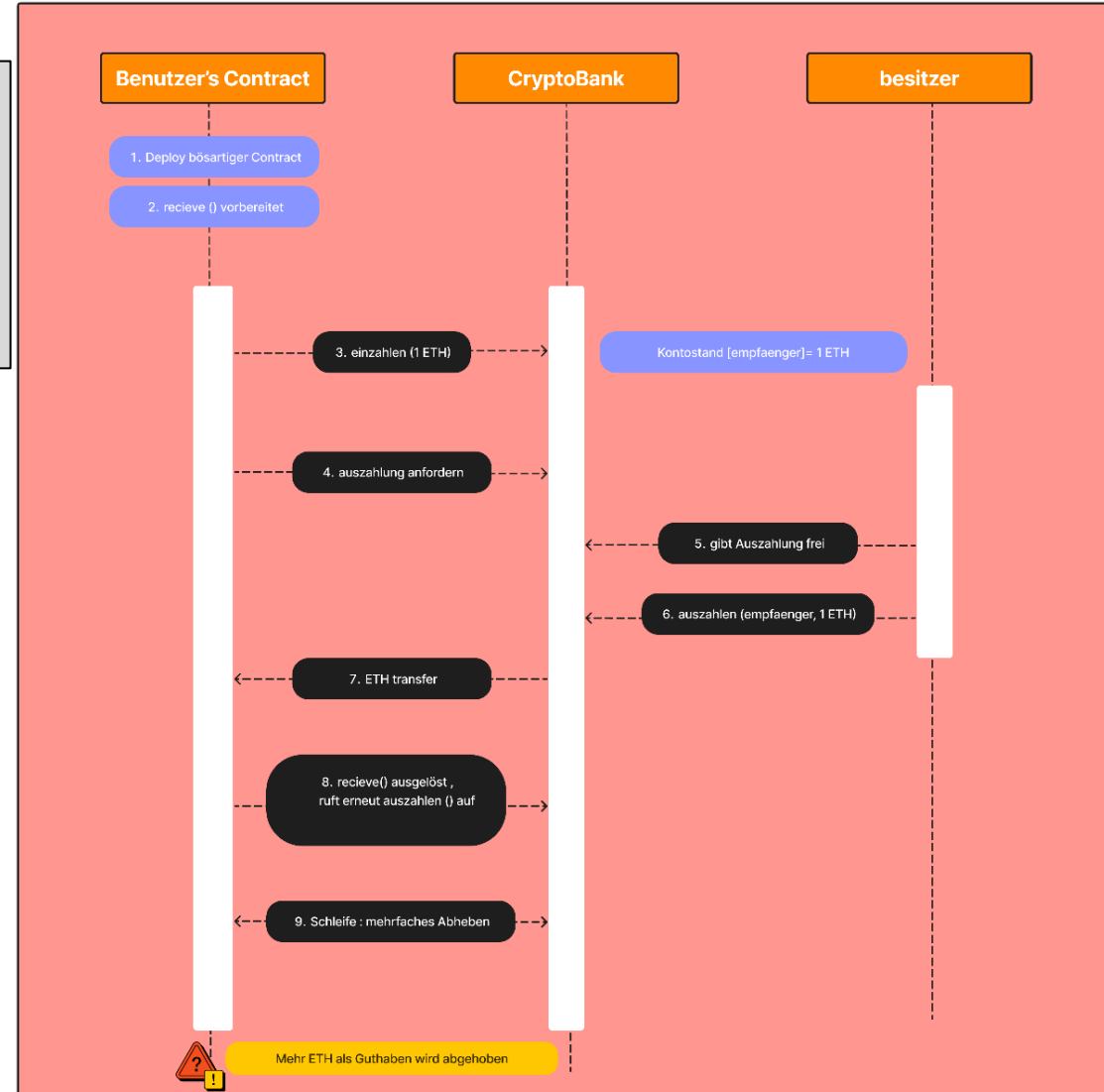
Remix Analyzer



Slither

Entwicklung und Analyse (CryptoBank)

```
// funktion für die Auszahlung: Diese Funktion wird nur von Besitzer ausgeführt
function auszahlen(address payable _empfaenger, uint _betrag) public {
    require(msg.sender == besitzer, "Sie sind nicht der Besitzer!");
    require(kontostand[_empfaenger] >= _betrag, "Kontostand ist nicht genug!");
    kontostand[_empfaenger] -= _betrag;
    _empfaenger.transfer(_betrag);
    emit ausgezahlt(_empfaenger, _betrag);
}
```



Verbesserungen und erneute Analyse (CryptoBankSecure)

```
// NEU: Importieren des Sicherheitsmodules für Reentrancy Schutz
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
// NEU: Erben von ReentrancyGuard
contract CryptoBankSecure is ReentrancyGuard {
```

```
    address public immutable besitzer;
    mapping(address => uint) public kontostand;
```

```
// geändert: Events in CamelCase für bessere Konventionen
event Eingezahlt(address indexed Benutzer, uint Betrag);
event Ausgezahlt(address indexed Empfaenger, uint Betrag);
event EtherEmpfangen(address Absender, uint Betrag);
// Fallback Event wurde entfernt: Gas sparen
```

```
// Geändert: fallback() enthält nur revert und vermeidet Gaswarnung
fallback() external payable {
    revert("Unbekannte Funktion aufgerufen.");
}
```

```
// Geändert mit Reentrancy Schutz
function auszahlen(address payable empfaenger, uint betrag) public nonReentrant {
    require(msg.sender == besitzer, "Sie sind nicht der Besitzer!");
    require(kontostand[empfaenger] >= betrag, "Kontostand ist nicht genug!");

    kontostand[empfaenger] -= betrag;

    // Geändert mit Sicherer Transfer durch call + require
    (bool success, ) = empfaenger.call{value: betrag}("");
    require(success, "Transfer fehlgeschlagen.");

    emit Ausgezahlt(empfaenger, betrag);
}
```

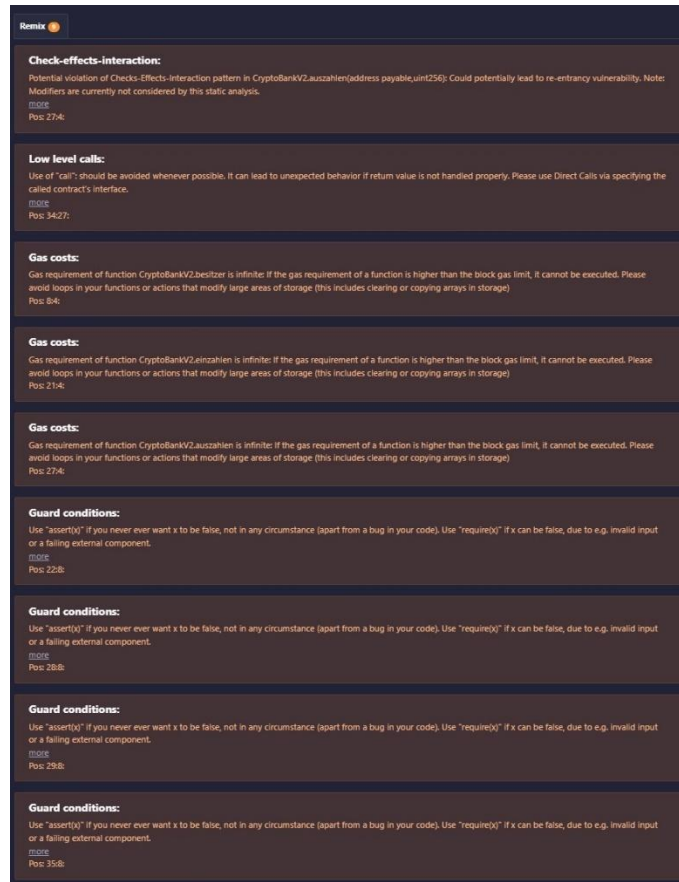
Verbesserungen und erneute Analyse (CryptoBankSecure)

Remix Analyzer

- Check-Effects-Interaction
- Low-Level-Call
- Guard Conditions
- Gas Coasts

Slither Analyzer

- Low-Level-Call
- Solidity-Version ^0.8.0



```
PS C:\Users\Yousef\Desktop\Bachelor\Eigener Smart Contract> slither CryptoBankSecure.sol --solc-args "--allow-paths . node_modules" --solc-remaps "@openzeppelin"
>>
'solc --version' running
'solc @openzeppelin=node_modules/@openzeppelin CryptoBankSecure.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,userdoc,devdoc,hashes -
running
INFO:Detectors:
ReentrancyGuard._reentrancyGuardEntered() (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#74-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Version constraint ^0.8.0 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
- FullInlinerNonExpressionSplitArgumentEvaluationOrder
- MissingSideEffectsOnSelectorAccess
- AbiReencodingHeadOverflowWithStaticArrayCleanup
- DirtyByteArrayToStorage
- DataLocationChangeInInternalOverride
- NestedCalldataArrayAbiReencodingSizeValidation
- SignedImmutables
- ABIDecodeTwoDimensionalArrayMemory
- KeccakCaching.
It is used by:
- ^0.8.0 (CryptoBankSecure.sol#2)
- ^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in CryptoBankV3.auszahlen(address,uint256) (CryptoBankSecure.sol#25-35):
- (success,None) = empfaenger.call(value: betrag)() (CryptoBankSecure.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:CryptoBankSecure.sol analyzed (2 contracts with 100 detectors), 3 result(s) found
```

Slither

Remix Analyzer

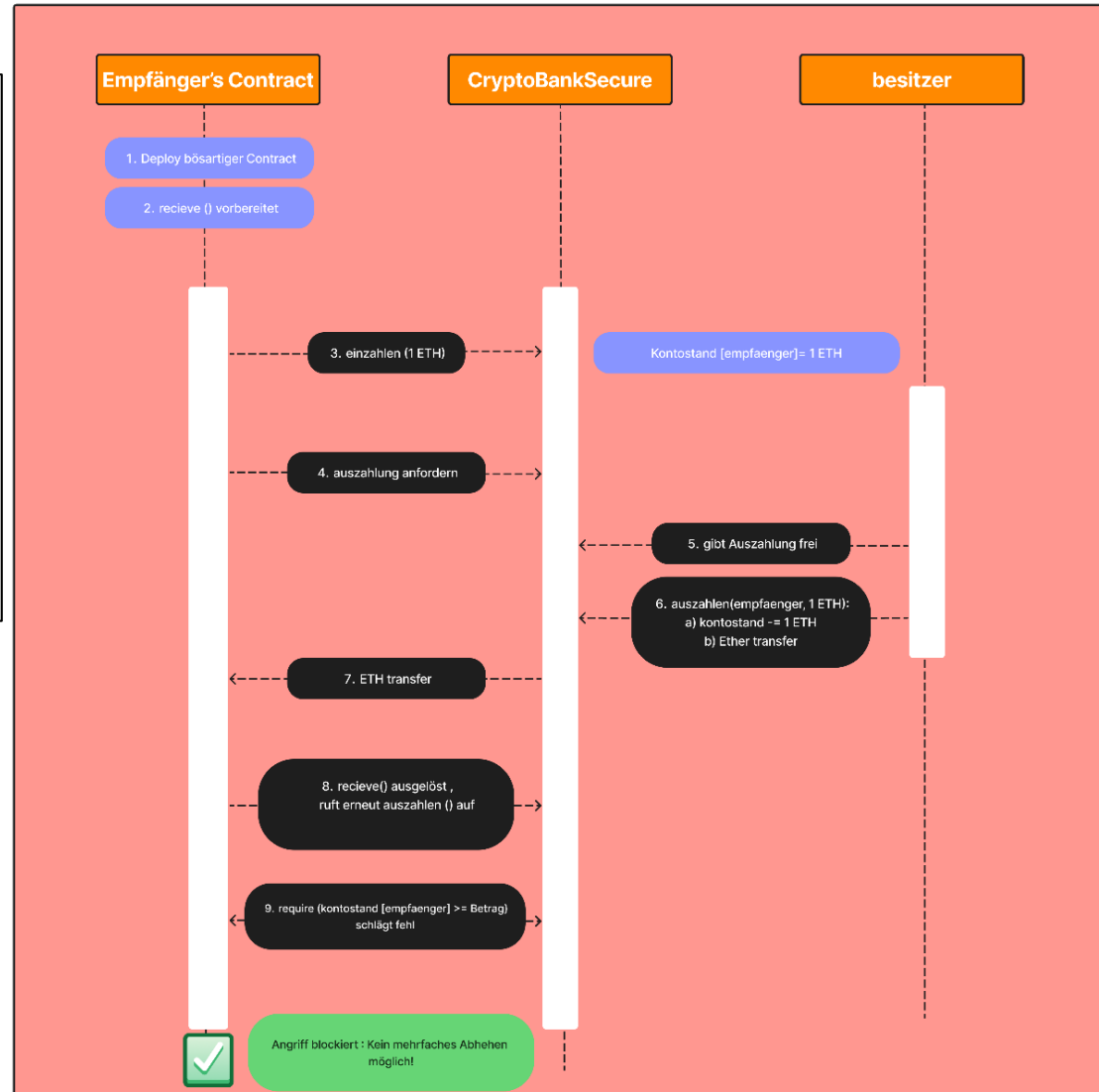
Verbesserungen und erneute Analyse (CryptoBankSecure)

```
// Geändert mit Reentrancy Schutz
function auszahlen(address payable empfaenger, uint betrag) public nonReentrant {
    require(msg.sender == besitzer, "Sie sind nicht der Besitzer!");
    require(kontostand[empfaenger] >= betrag, "Kontostand ist nicht genug!");

    kontostand[empfaenger] -= betrag;

    // Geändert mit Sicherer Transfer durch call + require
    (bool success, ) = empfaenger.call{value: betrag}("");
    require(success, "Transfer fehlgeschlagen.");

    emit Ausgezahlt(empfaenger, betrag);
}
```



Ergebnisse und Diskussion

Analysepunkt	CryptoBank (ursprünglich)	CryptoBankSecure (optimiert)	Tool-Meldung	Behandlungsmaßnahme	Zweck / Verbesserung
Ether-Transfer	<code>transfer()</code> ohne Rückgabekontrolle	<code>call {value: ..}()+ require (success)</code>	Remix + Slither	Low-Level-Call mit Prüfung	Absicherung gegen fehlgeschlagene Auszahlungen
Reentrancy-Schutz	Kein Schutz vorhanden	<code>nonReentrant</code> verwendet (OpenZeppelin)	Remix (Warnung)	Modifizierer eingeführt	Verhinderung verschachtelter Wiederaufrufe
Fallback-Logik	Event + Speicherzugriff	nur <code>revert("..-")</code>	Remix	Logik entfernt	Minimierung von Gasverbrauch & Angriffspotenzial
Event-Namen	<code>eingezahlt, ausgezahlt, etherEmpfangen</code>	<code>Eingezahlt, Ausgezahlt, EtherEmpfangen</code>	Slither	Namensformat korrigiert	Stilistische Konformität mit Solidity Standards
Zugriffsprüfung	<code>require(msg-sender == besitzer)</code>	identisch	Remix	unverändert korrekt	Schutz vor unbefugtem Zugriff
besitzer-Variable	<code>public</code> , veränderbar	<code>public immutable</code>	Slither	auf <code>immutable</code> geändert	Effizienz & Absicht der Unveränderbarkeit klar dargestellt
require / assert	korrekt, aber Remix schlägt <code>assert()</code> vor	bei <code>require()</code> belassen	Remix	keine Änderung nötig	gemäß Dokumentation korrekt eingesetzt
Gasabschätzung	"infinite gas" bei mehreren Funktionen	weiterhin vorhanden	Remix	bekanntes Verhalten	keine tatsächliche Einschränkung
Compiler-Version	<code>^0.8.0</code> mit allgemeinen Warnungen	beibehalten	Slither	Version bewusst beibehalten	Stabilität und Kompatibilität mit Bibliotheken

Zusammenfassung und Ausblick

Zusammenfassung

- Ziel: Sicherheitsanalyse von Ethereum Smart Contracts
- Theorie +Praxis kombiniert (CryptoBank & CryptoBankSecure)
- Risiken mit Remix & Slither erkannt
- Optimierung: Reentrancy-Schutz, sicherer Ether Transfer, vereinfachte fallback()
- Forschungsfrage positiv beantwortet: Risiken erkennbar und gezielt behebbar

Ausblick

- Blockchain gewinnt an Bedeutung über Technik hinaus
- Smart Contracts: starkes Potenzial aber auch Hürden
- Herausforderungen: Skalierung, Recht, Bedienbarkeit
- Zukunft: Kombination mit KI, Oracles, automatischem Vertragsrecht

Quellen

Folie 5: Jain, Shashank Mohan. 2023. A Brief Introduction to Web3: Decentralized Web Fundamentals for App Development. Berkeley, CA : Apress, 2023.

Antonopoulos, Andreas M. und Wood, Gavin . 2019. Ethereum – Grundlagen und Programmierung: Smart Contracts und DApps entwickeln. Heidelberg : dpunkt.verlag, 2019.

Folie 7: (1) Remix. Remix IDE Documentation. [Online] <https://remix-ide.readthedocs.io/> . (2) Solidity. Solidity Documentation. [Online] <https://docs.soliditylang.org/en/v0.8.29/> . (3)

Remix Project. Solidity Analyzers. [Online] https://remixide.readthedocs.io/en/latest/static_analysis.html . (4) Trail of Bits. Slither – Solidity Static Analysis Framework. [Online] <https://github.com/crytic/slither>.

Folie 10: (1) Wood, Gavin, Ethereum: A Secure Decentralised Generalised Transaction Ledger – Yellow Paper. [Online] <https://ethereum.github.io/yellowpaper/paper.pdf>. (2) Ethereum Foundation. Solidity – Security Considerations – Re-Entrancy. [Online] <https://docs.soliditylang.org/en/latest/security-considerations.html#re-entrancy> . (3) ConsenSys Diligence. Smart Contract Best Practices – Known Attacks: Reentrancy. [Online] https://consensys.github.io/smart-contract-best-practices/known_attacks/#reentrancy .

Folie 13: (1) Solidity. Security Considerations. [Online] <https://docs.soliditylang.org/en/v0.8.30/securityconsiderations.html#re-entrancy> . (2) Solidity. Control Structures. [Online] <https://docs.soliditylang.org/en/v0.8.30/controlstructures.html#external-function-calls> . (3) Solidity. Control Structures. [Online]

<https://docs.soliditylang.org/en/v0.8.30/controlstructures.html#error-handling-assert-require-revert-and-exceptions>. (4) Solidity. Contracts. [Online]

<https://docs.soliditylang.org/en/v0.8.30/contracts.html#fallbackfunction> . (5) Trail of Bits. Slither Usage Guide. [Online] <https://github.com/crytic/slither/wiki/Usage#usingimports-from-a-package-manager>. (7) Trail of Bits. Slither Detector Documentation. [Online] <https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls>.

**Vielen Dank fürs Interesse
an meiner Bachelorarbeit!**