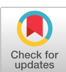

Bad Apples: Understanding the Centralized Security Risks in Decentralized Ecosystems

Kailun Yan
School of Cyber Science and Technology, Shandong University
Qingdao, China
kailun@mail.sdu.edu.cn

Jilian Zhang
College of Cyber Security
Jinan University
Guangzhou, China
zhangjilian@jnu.edu.cn

Xiangyu Liu
Alibaba Group
Hangzhou, China
eason.lxy@alibaba-inc.com

Wenrui Diao*
School of Cyber Science and Technology, Shandong University
Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, SDU
Qingdao, China
diaowenrui@link.cuhk.edu.hk

Shanqing Guo
School of Cyber Science and Technology, Shandong University
Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, SDU
Quan Cheng Laboratory
Qingdao, China

ABSTRACT

The blockchain-powered decentralized applications and systems have been widely deployed in recent years. The decentralization feature promises users anonymity, security, and non-censorship, which is especially welcomed in the areas of decentralized finance and digital assets. From the perspective of most common users, a decentralized ecosystem means every service follows the principle of decentralization. However, we find that the services in a decentralized ecosystem still may contain centralized components or scenarios, like third-party SDKs and privileged operations, which violate the promise of decentralization and may cause a series of centralized security risks. In this work, we systematically study the centralized security risks existing in decentralized ecosystems. Specifically, we identify seven centralized security risks in the deployment of two typical decentralized services – crypto wallets and DApps, such as anonymity loss and overpowered owner. Also, to measure these risks in the wild, we designed an automated detection tool called Naga and carried out large-scale experiments. Based on the measurement of 28 Ethereum crypto wallets (Android version) and 110,506 on-chain smart contracts, the result shows that the centralized security risks are widespread. Up to 96.4% of wallets and 83.5% of contracts exist at least one security risk, including 260 well-known tokens with a total market cap of over $98 billion.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**;

*The corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '23, April 30–May 04, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583393

KEYWORDS

Decentralized ecosystems; Crypto wallets; Smart contracts

ACM Reference Format:
Yan, et al. 2023. Bad Apples: Understanding the Centralized Security Risks in Decentralized Ecosystems. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583393

1 INTRODUCTION

The decentralized platform is the cornerstone of Web3, which promises to build an open, permissionless network [19]. The emergence of decentralized platforms promotes the development of decentralized ecosystems. Decentralized ecosystems consist of a series of services that are widely used due to the features of anonymous login, censorship-free, data security, and zero downtime. These services are decentralized by design and often open-sourced for community review, attracting users and benefiting developers.

From the perspective of most common users, a decentralized ecosystem means every service follows the principle of 100% decentralization. However, in practice, these services may still contain centralized components or scenarios. For example, many decentralized services rely on third-party remote procedure call (RPC) services because they do not run their blockchain nodes. Furthermore, some decentralized applications (DApps) have backdoors to facilitate maintenance. These actions undermine the promise of decentralization and raise centralized security risks, like "*one bad apple spoils the whole barrel*". Also, such risks are practical and vital. In November 2020, Infura [15], the leading RPC provider, was down, which caused the most severe Ethereum incident after The DAO attack [26, 29]. Recently, Slope, a well-known crypto wallet, leaked users' private keys, resulting in the theft of at least $6 million worth of tokens [32].

To the best of our knowledge, we are the first to systematically explore the centralized security risks in decentralized ecosystems. Most prior works focus on the security of blockchain system design [4, 6, 20, 22, 30, 35] and smart contract vulnerabilities detection [12, 16, 27, 31, 33]. However, from the view of ecosystems, the security risks caused by centralization were undervalued.

Our Work. This work systematically evaluates the decentralized ecosystem and reveals the centralized security risks of two crucial decentralized services – crypto wallets and DApps. We delved into decentralized services and discovered seven centralized security risks, of which five are related to crypto wallets and two (with four sub-issues) to DApps. To evaluate these risks in the wild, we proposed two methods to examine 28 Ethereum's officially recommended crypto wallets from the perspective of usage and development. The result shows that 27 wallets have security risks, say 96.4%. Also, we designed an automated tool Naga[1] to detect smart contracts of DApps by identifying state variables. Unlike previous work only analyzing data dependencies at the function level, Naga is a fine-grained tool that analyzes data dependencies in intermediate representations (IRs) of contract code. Further, we conducted a large-scale evaluation on 110,506 Ethereum on-chain contracts and discovered 92,254 contracts (83.5%) with security risks, including 11,419 high-value contracts and 260 famous tokens with a total market cap of over $98 billion.

Contributions. Here we list the main contributions of this paper:

- *New security issues.* We conducted the first systematic study on the centralized security risks in decentralized ecosystems. We identified seven previously unnoticed security risks.
- *New techniques.* We proposed two methods to check crypto wallets and designed an automated tool that can analyze data dependencies at the IR level for smart contract risk detection.
- *Real-world evaluations.* We carried out large-scale evaluations on 28 well-known crypto wallets and 110,506 DApps. The result shows that the centralized security risks are widespread.

Roadmap. The rest of this paper is organized as follows. Section 2 provides the necessary background of decentralized ecosystems. Section 3 discusses the discovered security risks. Section 4 introduces our detection approaches and implementation. Section 5 presents the measurement results. Section 6 proposes some suggestions for risk mitigation. Section 7 reviews related work, and Section 8 concludes this paper.

2 DECENTRALIZED ECOSYSTEMS

Decentralized ecosystems with verifiable, self-governing, permissionless, native payments, etc., allow anyone to access services equally, and no personal data is required. Instead of services controlled and owned by centralized entities, ownership in a decentralized service is distributed amongst its builders and users. Also, each service runs on multiple nodes, so there is no single point of failure, such as denial-of-service (DoS). The booming decentralized ecosystem is driving the transition from Web2 to Web3.

Figure 1 illustrates the main components of a decentralized ecosystem. As a decentralized platform, blockchain is the core infrastructure of decentralized ecosystems, and other services are built on it. Ethereum [7] is currently the largest decentralized platform, powering the cryptocurrency Ether (ETH) and thousands of decentralized applications (DApps). Centralized exchanges (CEXs) facilitate the flow of fiat and cryptocurrencies. As per the protocol, CEXs have an extensive built-in know-your-customer (KYC) policy and operate under regulatory supervision. Crypto or Web3 wallets act as a gateway to the natural and crypto worlds, manage crypto assets, and interact with DApps. DApp is an autonomously

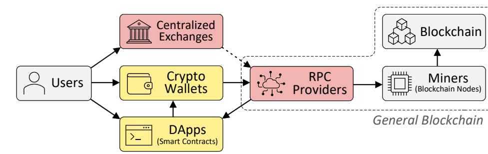

Figure. 1. Basic components of a decentralized ecosystem.

operating application with a backend smart contract and a frontend user interface. Ethereum's smart contracts are written in high-level programming languages such as Solidity [9] and then compiled down to bytecode running on Ethereum Virtual Machine (EVM). Ethereum client offers a set of remote procedure call (RPC) commands, so decentralized services can interact with blockchain by RPC, such as reading data and sending transactions. Most decentralized services rely on third-party RPC services because they do not run blockchain nodes. They send transactions to an RPC service, and the RPC service forwards transactions to the blockchain network. Finally, miners record transactions on the blockchain.

3 CENTRALIZED SECURITY RISKS

This section reveals the neglected security risks caused by centralization in decentralized ecosystems.

3.1 Overview

In a decentralized architecture, there may still exist some centralized components which a single malicious person can control. Here, we describe the threat model and research objects.

We assume decentralized platforms are benign, and miners will not collude with each other. Then, we regard the decentralized service providers as adversaries. We consider two scenarios:

a) *First-party centralization.* The adversary integrates centralized services or backdoors into the decentralized service he developed. In this scenario, the adversary usually intentionally confuses the concept of decentralization and misleads users by advertising that the service runs on a decentralized platform.
b) *Third-party centralization.* The adversary, as a third party, supplies centralized components for decentralized services to contaminate decentralized ecosystems. In this scenario, the adversary provides SDKs or services to induce developers to include those components in the decentralized services.

Based on the description in Section 2, we focus on two widely deployed decentralized services – crypto wallets and DApps.

Crypto Wallets. Crypto wallets claim that they are decentralized and anonymous. But in fact, some crypto wallets require users to provide Email addresses or phone numbers. Meanwhile, crypto wallets may have third-party centralized components, such as RPC services, SDKs, etc. These centralization factors indisputably destroy the decentralized ecosystem and bring security risks.

DApps. DApps emphasize that they run on decentralized platforms and that data is immutable and indisputable. However, the backend of DApps, i.e., smart contracts, may have backdoors that implement privileged operations. ERC20 is an Ethereum request-for-comment (ERC) standard that allows fungible tokens (FTs). ERC721 and ERC1155 are two popular standards for creating unique and indivisible tokens, i.e., non-fungible tokens (NFTs). This work investigated the three most influential DApps.

[1] Naga is available at https://doi.org/10.5281/zenodo.7620441

3.2 Security Risks

After systematic investigations, we identified a series of centralized security risks. Here we discuss them, SR#1 to SR#5 appearing in crypto wallets and SR#6 to SR#7 appearing in DApps.

SR#1: Anonymity Loss (AL). Some crypto wallets require account registration before use, and users have to provide personally identifiable information (PII) in this process, such as Email addresses and phone numbers. Provided PII jeopardizes the user's anonymity [38, 40]. Further, attackers can conduct phishing attacks by leaking PII, e.g., in March 2022, Trezor wallet exposed the Email addresses of 106,856 users, resulting in massive phishing Email attacks [2].

SR#2: Private Key Leakage (PL). Private keys hold users' crypto assets and should be carefully stored locally. Some wallets recommend that users use first- or third-party servers to back up private keys, e.g., Bitcoin.com Wallet, an Android wallet with over 5 million downloads, recommends users back up private keys on their servers, which incurs risks. First, service providers may steal or leak private keys. Second, private keys may be subject to man-in-the-middle (MITM) attacks during network transmission. In August 2022, Slope leaked users' private keys during the network transmission, resulting in the theft of at least $6 million worth of tokens [32].

SR#3: Built-in Centralized Services (BS). Some wallets have built-in first- or third-party centralized services such as exchanges and cryptocurrency purchases. Generally, centralized services are regulated by the government and may share users' PII with third parties. Also, users may suffer financial losses due to the sudden shutdown of services. Most users cannot distinguish whether services in crypto wallets are centralized or not. Therefore, wallets are obliged to inform users of centralized services' risks. For instance, as of 2020, 75 exchanges were closed [41]. In June 2022, Celsius Network, an centralized finance platform with 1.7 million users, suspended its services, citing "*extreme market conditions*" [39].

SR#4: RPC Services (RS). RPC services or providers suffer from issues inherent in centralization, such as denial-of-service (DoS) [17]. Also, RPC providers can withhold transactions with transaction-ordering dependence (TOD) [21] for huge benefits. Crypto wallets should disclose built-in RPC providers and allow users to change RPCs. The centralization of RPC services has caused many security risks. In March 2022, Infura cut off Ukrainian users for policy reasons. Since the default RPC of Metamask (a well-known crypto wallet) is Infura, many Metamask users were also affected [5].

SR#5: Third-Party SDKs (TS). It is common for crypto wallets to use third-party SDKs such as notification and fraud protection. Wallets may share users' PII, device IDs, and crash logs with these third parties. In the Slope incident [32], a white hat hacker found that Slope used plaintext to transmit logs to Sentry, a bug-tracking SDK. Meanwhile, Slope did not clear sensitive information, resulting in Sentry holding sensitive information of Slope's users.

SR#6: Overpowered Owner (OO). The smart contract supports access to the caller's address, so a contract can check the caller's address to see if he can call the function, i.e., access control. Access control allows creators to manipulate contracts, aka *overpowered owner (OO)*. Listing 1 shows an example. First, the `constructor()` sets the creator (`msg.sender`) as the owner (`_owner`) in line 2. Then, the modifier `onlyOwner()` has a `require` statement in line 7 that ensures the caller is the owner. The function `mint()` is only available to the owner (creator) because it is protected by `onlyOwner()`. Openzeppelin [23], the most famous contract library, provides two kinds of access controls, *Ownable* and *AccessControl*. Wild contracts also use a simple `mapping` to maintain a set of privileged addresses, called *AdminControl*. The variable of *Ownable* is called `owner`, as shown in Listing 1. The variables of *AccessControl* and *AdminControl* are called `roles` and `admins`, respectively. For convenience, in the rest of this paper, we use *the owner* or *owners* to denote roles in the three access controls.

```
1  constructor(){
2    _owner = msg.sender; // address public _owner;
3    _maxSupply = 100000; // uint public _maxSupply;
4    _totalSupply = 0;  // uint public _totalSupply;
5  }
6  modifier onlyOwner() {
7    require(msg.sender == _owner);
8    _;
9  }
10 function mint(address to, uint amount) public onlyOwner {
11   //require(msg.sender == _owner); equals to onlyOwner().
12   require(_totalSupply + amount <= _maxSupply);
13   /* ... */
14 }
```

Listing 1. Example of the owner minting tokens.

Generally, contracts adopt access control to protect accounts' interests and maintain the contract environment. However, the risk of overpowered owner could cause the following security risks (SR#6.a to SR#6.d). Furthermore, losing the owner's private key can cause severe damage, as attackers can directly exploit these privileges. An example of such failure is the KickICO incident [25]. Attackers compromised the owner's private key and stole $7.7 million worth of KickICO tokens.

SR#6.a: Limited Liquidity (LL). Liquidity is the ability to buy and sell a cryptocurrency in the market. We define liquidity as whether anyone can *transfer* or *allowance* tokens without limitations. If the owner can freeze a contract, the contract has *limited liquidity*. Listing 2 shows two kinds of limited liquidity, one is to set a `bool` variable to freeze the function (line 2), and the other to blacklist an address to prevent it from calling the function (line 3).

```
1 function transfer(address to, uint amount) public {
2   require(!_paused); // bool
3   require(!_blacklist[msg.sender]); // mapping(address=>bool)
4   /* ... */
5 }
```

Listing 2. Example of a transfer with limited liquidity.

SR#6.b: Vulnerable Scarcity (VS). *Vulnerable scarcity* means that the owner can arbitrarily increase the supply of tokens. Tokens are scarce because usually the total supply of a contract is limited. In Listing 1, only the owner can mint tokens, and line 12 requires that the total supply (`_totalSupply`) is not greater than the maximum supply (`_maxSupply`). The additional issuance of a token harms the interests of holders. In January 2021, Yearn.finance team proposed an extra $225 million worth of YFI to incentivize and retain developers, which caused fierce protests from its holders [3].

SR#6.c: Mutable Metadata (MM). If an owner can change metadata, we call it *mutable metadata*. Token standards offer optional metadata, and Listing 3 lists the metadata of ERC20. In general,

metadata are immutable and only initialized during deployment. Wallets and browsers display contract metadata to users, so mutable metadata may mislead users into sending unexpected transactions.

```
1 function name() public view returns (string) {return _name;}
2 function symbol() public view returns (string){return _symbol;}
3 function decimals() public view returns (uint8) {return 18;}
```

Listing 3. Example of the ERC20 metadata.

SR#6.d: Mutable Parameters (MP). The function `transfer()` of ERC20 often has customized parameters, such as *transferTax* and *maxTxAmount*. Usually, these parameters are immutable and agreed upon by participants. If the owner can update parameters at will, we call it *mutable parameters*.

SR#7: Missing Events (ME). Solidity `Event` encapsulates the logging functionality of EVM. If a function updates state variables without emitting any events, we call it *missing events*. In Listing 4, the caller can know whether the function `transfer()` is executed successfully by listening to the event `Transfer`. Also, users can subscribe to events of functions to be aware of owners' actions (e.g., pause transfer). If a function does not emit an event after updating a state variable, no one will be notified unless he actively checks the contract in the blockchain.

```
1 event Transfer(address from, address to, uint amount);
2 function transfer(address to, uint amount) public{
3   /* ... */
4   emit Transfer(msg.sender, to, amount);
5 }
```

Listing 4. Example of an `Event`.

4 RISK DETECTION APPROACHES

This section describes our approaches to detecting the above seven centralized security risks.

4.1 Detection on Crypto Wallets

Generally, there are four types of crypto wallets: mobile apps, browser extensions, desktop programs, and physical hardware. Ethereum website lists 44 wallets [1], of which 30 support the Android platform, accounting for the most significant proportion, so we focus on Android wallets as the research object. We analyzed the security risks of wallets from two aspects. One is function checking from the users' perspective (D#1), and the other is analyzing SDKs in APKs with a semi-automatic heuristic method (D#2).

D#1: Function Check. We manually check the main functions of crypto wallets, including generating and importing private keys, trading, modifying RPCs, etc. We pay attention to the following four research questions:

RQ1 Does the wallet require users to register or provide additional information before use? (SR#1)
RQ2 Does the wallet recommend users back up their private keys to the cloud? (SR#2)
RQ3 Whether the wallet has built-in centralized services and reminds users that these services are not decentralized. (SR#3)
RQ4 Can users modify RPC providers in the wallet? (SR#4)

D#2: Semi-Automated Detection. We propose a semi-automated heuristic method to identify risky SDKs (SR#5). First, we note that package names usually follow the Java naming convention, i.e., *domain.company.project*, and these commercial third-party SDKs usually have websites to promote business. Specifically, we can infer the website from the package name, e.g., *com.sensetime.senseid* refers to the domain name *http://sensetime.com*. Figure 2 shows our heuristic method to identify SDKs. We use Apktool [37] to decompile the APK of a wallet and then perform a breadth-first search (BFS) on the *smali* directory to obtain all package names. We take the first two parts of the package name and reverse them as the domain name and try to send a request to it. If we receive a response, we crawl the website's content and perform keyword matching. If three keywords are hit, we add this package to the candidate list. Finally, we manually check candidates and get risky SDKs. We pre-selected some risky third-party SDKs from SDKs with high frequency and selected keywords from their websites.

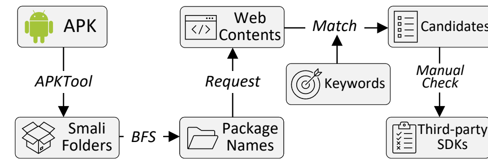

Figure. 2. Process of identifying risky third-party SDKs.

Table 1: Identified state variables.

SR	Variable	Solidity Type	Detection
#6	owner	address, bytes	[D#3, D#4]→D#6
	roles	mapping(bytes⇒RoleData)	
	admins	mapping(address⇒bool)	
		mapping(bytes⇒bool)	
#6.a	blacklist	mapping(address⇒bool)	D#6
	paused	bool	[D#3, D#4]→D#6
#6.b	totalSupply	uint	[D#3, D#4]→D#5
#6.c	Table 2	N/A	[D#3, D#4]→D#5
#6.d	N/A	uint	D#6

[D#3, D#4]→D#6 means that if D#3 and D#4 are executed first, then D#6 is executed for remaining variables.

Table 2: Optional metadata for tokens.

Variable	Solidity Type	ERC20	ERC721	ERC1155
name	string	√	√	
symbol	string	√	√	
decimals	uint	√		
uri	string		√	√

4.2 Detection on Smart Contracts

In this section we propose four detection methods based on state variable identification and implements an automated detection tool.

4.2.1 State Variables Identification. We detect security risks on the Solidity source code of contracts. Solidity uses state variables to store a contract *state* on blockchain. A contract with overpowered owner needs to set a particular state variable (called *owner variable*) that stores owners' addresses and checks the caller's access permissions in critical functions. For example, in Listing 1, if the caller's address is not equal to the owner variable (`_owner`), the function `mint()` will revert. These owner-controlled functions control a contract by modifying other state variables. Therefore, for detecting SR#6 and its four sub-issues, we need to find out the owner variable and the owner-controlled variables. Table 1 lists these state variables and detection methods. Below we introduce four methods (D#3~6) to identify state variables in Table 1.

Table 3: OpenZeppelin contracts.

Directory	Contracts
access	*Ownable, AccessControl, AccessControlEnumerable*
security	*Pausable*
token	*ERC20, ERC721, ERC1155*

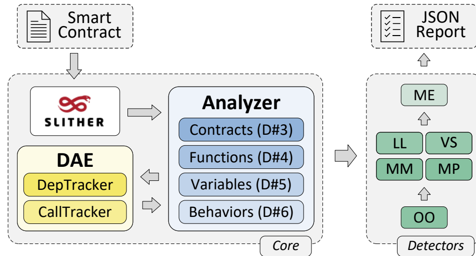

Figure. 3. Overview of Naga architecture.

D#3: Inheritance. Solidity supports multiple inheritance, and many contracts inherit Openzeppelin contracts to provide functionality. Table 3 lists the common contracts in Openzeppelin. We first annotated state variables in these contracts, and if a contract inherits these contracts, we can get variables in Table 1 directly by using variable names.

D#4: Getters & Modifiers. If a contract does not inherit any given contracts, D#3 will be invalid. However, most contracts have *getters*. A getter is a view function that returns the value of a state variable. Listing 3 shows the getters of ERC20, and we can identify variables through these getters. In practice, we match getters of contracts listed in Table 3 and further analyze the `return` statement to identify variables. Additionally, we use a similar approach to match popular modifiers such as `onlyOwner()` and `onlyRole()`.

D#5: Variable Name & Type. Solidity can automatically create getters for state variables declared `public`, so that a contract may omit getters. Therefore, we match names and types to identify state variables that survived the screening of D#3 and D#4.

D#6: Behavior Patterns. The above three matching methods cannot deal with custom variables, e.g., a contract may have an owner variable with an uncommon name. Therefore, we define behavior patterns for most state variables to identify them, e.g., an owner variable can only be written by himself or other owners, and the owner must appear in a `require` to protect other state variables.

Following the above methods, we detect state variables in Table 1 and judge the security risks (SR#6, SR#6.a~d) of a contract according to whether owners can modify these variables. For SR#7, we directly analyze functions and classify them by identified variables. If an owner-controlled function misses an event, we classify it as *owner-related ME* (missing event), otherwise *user-related ME*.

Since D#3 and D#4 are match items in the library, they are accurate. D#5 may have false positives due to the same variable name and type. Thus, we only perform D#5 on `totalSupply` and `metadata` variables (SR#6.b, SR#6.c) because they are well-known. We rely on D#6 to deal with the diversity of SR#6, SR#6.a, and SR#6.d variables. SR#6.b and SR#6.c do not adopt D#6 because their variables are read-only by default that cannot define behavior patterns.

4.2.2 Design of Naga. Based on the detection methods, we designed a static analysis tool called Naga to detect security risks in smart contracts. Figure 3 illustrates the architecture of Naga, which consists of two modules, i.e., *Core* and *Detectors*. Naga is built on top of SLITHER [11], a static analysis framework. First, SLITHER compiles the contract's source code. Then, the Analyzer loads the SLITHER object and analyzes it from multiple levels. Finally, *Detectors* module finds security risks by pre-defined detectors and outputs a JSON report. Only analyzing variables in a single function or statement is not enough because functions can call each other, and there are dependencies between statements. The *Core* module includes a *Data-Dependency Analysis Engine (DAE)* for fine-grained analysis, which can track the dependencies of key variables or statements, and supports the analysis of internal or external calls. D#4 and D#6 benefit from *DAE*.

```
1  address public _owner; //the owner variable
2  function _msgSender() internal returns (address) {
3    return msg.sender;
4  }
5  function transferOwnership(address newOwner) external {
6    address currOwner = _owner;
7    require(currOwner == _msgSender() && newOwner != address(0));
8    // require(currOwner == _msgSender());
9    // require(newOwner != address(0));
10   _owner = newOwner;
11 }
```

Listing 5. Example of transferring ownership.

```
1  _msgSender() IRs:
2    RETURN msg.sender ▲
3
4  transferOwnership() IRs:
5    currOwner_1(address) := _owner_1(address) ▲
6    TMP_0(address) = INTERNAL_CALL, intercall._msgSender()() ▲
7    TMP_1(bool) = currOwner_1 == TMP_0 ▲
8    TMP_2 = CONVERT 0 to address ♦
9    TMP_3(bool) = newOwner_1 != TMP_2 ♦
10   TMP_4(bool) = TMP_1 && TMP_3 ◄
11   TMP_5(None) = SOLIDITY_CALL require(bool)(TMP_4)
12   _owner_2(address) := newOwner_1(address)
```

Listing 6. Static single assignment form of IRs of Listing.5.

Data-dependency Analysis Engine (DAE). *DAE* gets the deep data-dependency of a given variable or statement by tracking the intermediate representation (IR) that SLITHER converted. *DAE* is accurate due to the advantages of the static single assignment (SSA) form of IRs. Appendix A.1 gives the technical details of *DAE*, and here we provide an example.

In Listing 5, the function `transferOwnership()` has a compound `require` in line 7, which is equivalent to line 8 and line 9. Without *DAE*, we only know line 7 depends on two local variables (`currOwner`, `newOwner`) and one constant address (`0x0`). Listing 6 is the SSA form of IRs of Listing 5, and line 11 in Listing 6 is the IR of the tainted line 7 in Listing 5. The rvalue `TMP_4` of line 11 depends on `TMP_1` and `TMP_3` in line 10. *DAE* supports split *&&* operators in conditional statements. Since *DAE* finds the *&&* operator in line 10, it starts two *sub-DAEs* to analyze `TMP_1` (golden ▲) and `TMP_3` (green ♦). In line 6, *sub-DAE* enter the internal call `_msgSender()` and get the return value, i.e., `msg.sender`. Following green marks, we can see `TMP_3` depends on `newOwner_1` (i.e., `newOwner`) and an address `0x0`. Naga learned from *DAE* that the require depends on two conditions. The first condition includes a state variable `_owner` and a global variable `msg.sender`, and the second condition includes a local variable `newOwner` and a constant address `0x0`. According to behavior patterns defined in D#6, Naga reports that this function is controlled by the owner.

Table 4: Contract datasets under detection.

Dataset	ERC20	ERC721	ERC1155	Non-token	Total
ET	351	8,575	2,827	N/A	11,753
EM	42,801	20,947	1,334	33,671	98,753
Overall	43,152	29,522	4,161	33,671	110,506

Detectors. According to Section 4.2.1 and Table 1, we implemented the detectors for SR#6 with four sub-issues (SR#6.a to SR#6.d) and SR#7. We can easily add more detectors to extend the detection capabilities of Naga. Appendix A.2 gives the details of all detectors.

5 MEASUREMENT AND FINDINGS

5.1 Experiment Setup

Crypto Wallet Dataset. We downloaded 30 Android crypto wallet apps recommended by Ethereum from Google Play. Since two wallets (Ledger and Keystone) require additional hardware, the rest 28 wallets were used for the experiment. These 28 wallets have over 131 million downloads, including one wallet with over 100 million downloads and two wallets with over 10 million downloads.

Smart Contract Dataset. Table 4 lists the detected contracts in two datasets[2], with a total of 110,506 contracts, including 43,152 ERC20s, 29,522 ERC721s, 4,161 ERC1155s, and 33,671 non-tokens. The first dataset *Etherscan token (ET)* comes from Etherscan's token tracker [10]. Etherscan identified some ERC20s, ERC721s, and ERC1155s. We crawled the source codes of those tokens and made up the dataset *ET*. It is a high-value dataset where ERC20s have 30 million holders and a total market cap of over $310 billion. The second dataset *Ethereum mainnet (EM)* comes from *SCS* [24], a project that collects the source code of contracts. Since half of the contracts failed to compile in *SCS*, we requested from scratch all 143,900 contracts from Etherscan, which covers a period until July 1, 2022. We removed duplicate contracts between two datasets and did not detect contracts with Solidity versions lower than 0.5.0 because they do not support the `require` keyword. We identify tokens in the *EM* dataset by matching function signatures, a method widely used by crypto wallets and websites, including Etherscan.

Setup. We performed D#1 on a Lenovo Lemeng K12 phone with Android 10. D#2 and Naga were run on a machine with two Intel(R) Xeon(R) Gold 6226R CPUs @ 2.90GHz and 256 GB of memory. The running environment is Ubuntu 20.04 and Python 3.8.10. The timeout of contract compilation and Naga analysis are both 60 seconds. We detected all risks (SR#6, SR#6.a~d, and SR#7) for token contracts and detected SR#6 and SR#7 only for non-token contracts.

5.2 Experiment Results

This section presents our detection results. Table 5 lists the security risks (SR#1 to SR#5) of crypto wallets, and Table 7 lists SR#6 to SR#7 of smart contracts. The results are almost overwhelming. That is, 96.4% (27/28) of wallets and 83.5% (92,254/110,506) of contracts have at least one security risk, including 11,419 high-value contracts and 260 famous tokens with a total market cap of over $98 billion.

[2]Both datasets are available at https://doi.org/10.5281/zenodo.7620479

Table 5: Security risks of crypto wallets.

Crypto Wallet	DLs	SR#1	SR#2	SR#3	SR#4	SR#5
Brave Wallet	100M+	○	○	◐	○	0
Coinbase Wallet	10M+	○	○	○	●	**3**
MetaMask	10M+	○	○	◐	○	**2**
Bitcoin.com Wallet	5M+	◐	Cloud	●	●	**5**
Exodus	1M+	○	○	◐	●	0
Opera Wallet	1M+	○	○	○	●	**1**
Status	1M+	○	○	◐	○	0
TokenPocket	1M+	○	○	○	○	0
Coin98 Wallet	500K+	◐	Cloud	●	●	**4**
imToken	500K+	○	○	○	○	**5**
MEW Wallet	500K+	○	○	●	●	**3**
AlphaWallet	100K+	○	○	◐	○	**1**
Argent	100K+	●	Google	◐	●	**6**
Coin Wallet	100K+	○	○	◐	●	**1**
Guarda	100K+	○	○	●	●	0
Pillar	100K+	○	○	●	●	**4**
ZenGo	100K+	●	Google	●	●	**10**
Zerion Wallet	100K+	○	○	◐	●	**4**
1inch Wallet	50K+	○	Google	○	●	0
Loopring Wallet	50K+	●	○	○	●	**1**
AirGap Wallet	10K+	○	○	○	●	0
Bridge Wallet	10K+	◐	○	●	○	**2**
FoxWallet	10K+	○	○	◐	○	**5**
Gnosis Safe	10K+	○	○	○	●	**2**
Numio	10K+	●	Google	●	●	**3**
Rainbow	10K+	○	Google	●	●	**3**
Unstoppable	10K+	○	○	○	●	0
Aktionariat	1K+	●	○	●	●	**1**

● Security risk exists; ◐ Security risk maybe exist; ○ No security risk.

Table 6: Crypto wallets requiring account registration (SR#1).

	Required Personal Identifiable Information		
Crypto Wallet	**Email**	**Phone Number**	**Face**
Argent	√	√	
ZenGo	√		√
Loopring Wallet	√		
Numio			√
Aktionariat	√	√	

Performance Analysis. Naga cost 2.11 seconds per contract on average. Except for 1,834 contract source file errors, only one contract failed due to IR conversion failure, and no contract was timeout.

To SR#1: Anonymity Loss. Table 6 lists the wallets requiring registration. Among them, four wallets require an Email address, while Argent and Aktionariat also require a phone number. ZenGo and Numio even require face verification, and ZenGo requires users to link Google accounts and back up their private keys to Google Drive before use. Aktionariat requires users to provide personal information such as name and address. In addition, the registration steps of three wallets are optional (marked as ◐ in Table 5), and the user only needs to provide an Email address. Almost all wallets with SR#1 offer extra services like online backup (SR#2), exchange (SR#3), so they tend to make users sign up.

Table 7: Security risks of smart contracts (num. of state variables / num. of contracts).

Contracts	SR#6 (OO)	SR#6.a (LL)	SR#6.b (VS)	SR#6.c (MM)	SR#6.d (MP)	SR#7 (ME)
ERC20	61,170 / 34,812	19,746 / 16,543	2,968 / 2,968	552 / 268	74,381 / 18,552	167,292 / 29,467
ERC721	41,062 / 28,866	2,564 / 2,489	N/A	20,911 / 20,698	N/A	157,280 / 25,862
ERC1155	6,286 / 3,967	561 / 549	76 / 76	2,074 / 2,025	N/A	10,769 / 3,269
Non-token	32,545 / 20,814	N/A	N/A	N/A	N/A	62,114 / 16,198
Total	141,063 / 88,459	22,871 / 19,581	3,044 / 3,044	23,537 / 22,991	74,381 / 18,552	397,455 / 74,796

Table 8: Categories of third-party SDKs (SR#5).

	Categories	# of SDKs	# / DLs of APKs
C#1	Notification, SMS, Email	11	10 / 6,740K+
C#2	Data Analysis, Customer Analysis	12	10 / 11,430K+
C#3	ID Verification, Fraud Protection	13	9 / 11,321K+
C#4	Bug Reporting, Error tracking	5	11 / 26,330K+
C#5	Business Messenger	4	7 / 6,310K+

One SDK may belong to multiple categories, and one app may use multiple SDKs.

Table 9: Percentages of token contracts with security risks.

Contracts	SR#6	SR#6.a	SR#6.b	SR#6.c	SR#6.d	SR#7
ERC20 (FT)	80.7%	38.3%	6.9%	0.6%	43.0%	68.3%
ERC721 (NFT)	97.8%	8.4%	N/A	70.1%	N/A	87.6%
ERC1155 (NFT)	95.3%	13.2%	1.8%	48.7%	N/A	78.6%
NFTs	97.5%	9.0%	N/A	67.5%	N/A	86.5%
Overall (Token)	88.0%	25.5%	N/A	29.9%	N/A	76.3%

To SR#2: Private Key Leakage. Usually, wallets remind users to write down mnemonic words of private keys. However, seven wallets recommend users use online backups for convenience, which leads to SR#2. Bitcoin.com Wallet and Coin98 Wallet, with 5M and 1M downloads, respectively, remind users to back up private keys to their servers, and five wallets (including ZenGo) recommend users to back up private keys to Google Drive.

To SR#3: Built-in Centralized Services. Two wallets have built-in exchanges, and 19 wallets provide cryptocurrency purchase functions. None of the wallets alert users that those services are centralized, and only nine wallets showed providers of services (marked as ◐ in Table 5). No wallets apply for Android permissions, except for Numio. Numio requires users to provide location permission to check whether users can purchase cryptocurrencies. In fact, blurring the centralization of these services will help attract users.

To SR#4: RPC Services. In our dataset, 20 crypto wallets do not disclose their RPC providers and cannot modify RPCs. The remaining eight wallets support users adding new RPCs, and six of them reveal RPC providers. Five of these six disclosed RPC providers are centralized, including *infura.io*, *nodereal.io*, and *ankr.com*. Only imToken wallet uses its own RPC service *token.im*.

To SR#5: Third-Party SDKs. Our heuristic detection (D#2) recognized 72 candidate SDKs. We conducted manual analysis and found 29 risky third-party SDKs. We further investigated these SDKs and found that wallets use them mainly for the five reasons, as listed in Table 8. In C#1, SDK providers hold the wallet user's contact information, which could incur phishing attacks. The SDK providers of C#2 to C#5 may obtain the user's PII, which destroys anonymity. In C#3, we also found a face recognition SDK *com.facetec*, which was confirmed to be used in Numio and ZenGo. Based on the number of APKs, the most influential SDK is *io.sentry* (Sentry), a bug-reporting SDK used by eight well-known crypto wallets with over 11 million total downloads. Sentry supports customization, and the aforementioned Slope incident is caused by misconfiguration. Appendix A.3 lists the detail of these 29 SDKs.

To SR#6: Overpowered Owner. As demonstrated in Table 7, Naga found 141,063 owners in 88,459 contracts, i.e., 80.1% (88,459/110,506) of contracts and 88.0% (67,645/76,835) of tokens have the risk of overpowered owner. The main reason for such high percentages is that, since contracts are immutable once deployed, most creators (owners) want to retain the ability to change contracts. For example, 70.4% (47,635/67,645) of tokens with overpowered owners have at least one risk of SR#6.a~d. Table 9 shows that the percentage of SR#6 in NFTs (97.5%) is significantly more than that in FTs (80.7%) because NFTs usually require owner-controlled functions. For example, to attract potential users, NFT contracts could distribute some NFTs for free, called *airdrop*. Those contracts contain a function `airdrop()` that the owner only controls.

To SR#6.a: Limited Liquidity. Naga found 21,464 contracts with limited liquidity. Table 9 shows that 25.5% of tokens and 38.3% of ERC20s have this risk. As mentioned before, owners use `blacklist` and `paused` to limit liquidity for security. Since `paused` can offer the additional ability to switch transfers, NFTs prefer to use it to delay opening transactions. In our dataset, 79.6% of FTs with limited liquidity have `blacklist`, and 96.8% of NFTs have `paused`.

To SR#6.b: Vulnerable Scarcity. We only detected vulnerable scarcity for ERC20s and ERC1155s because ERC721s do not have `totalSupply`. Since most investors (users) tend to select tokens according to their scarcity, the total supplies of most contracts are limited. However, there are still 3,044 contracts have the risk of vulnerable scarcity, as shown in Table 7, including 2,968 ERC20s and 76 ERC1155s. In other words, these tokens are more likely to depreciate due to additional issuance.

To SR#6.c: Mutable Metadata. Table 9 shows that 67.5% of NFTs have mutable metadata, while only 0.6% of ERC20s. Since most NFTs store users' digital assets on centralized servers, deployers (owners) need to retain the ability to change `baseURI`, and NFTs with blind boxes need owners to reveal `baseURI`. Therefore, there are more NFTs with this risk. However, excluding `baseURI`, there are still 392 contracts with 782 metadata that owners can modify. We further investigated these contracts and found that they intentionally retain the ability to modify metadata for upgrading contracts in the future.

To SR#6.d: Mutable Parameters. About 43.0% of ERC20s support owners in changing transfer parameters. Only 19 contracts (5.4%) have the risk of mutable parameters in the *ET* dataset. However, 18,533 contracts (43.3%) have this risk in the *EM* dataset. The large

Table 10: Num. of functions with missing events (SR#7).

	Total	User	Owner
Token	335,341	13,063 (3.9%)	322,278 (**96.1**%)
Overall	397,455	31,826 (8.0%)	365,629 (**92.0**%)

Table 11: FP & FN analysis – num. of state variables.

	SR#6	SR#6.a	SR#6.b	SR#6.c	SR#6.d	SR#7
TP	136	24	54	274	227	444
FP	5	0	0	0	1	0
FN	2	7	3	14	0	0

discrepancy between the two datasets reflects the constraints of valuable (*ET*) contracts in changing parameters. It indicates that owners of ordinary (*EM*) contracts abuse their power.

To SR#7: Missing Events. The risk of missing events (ME) in contracts is widespread, with 74,796 contracts (67.7%) missing at least one event, indicating that most contracts are not compliant. Table 10 shows the number of functions with ME. The owner-related ME accounts for 92.0% of all contracts and 96.1% of tokens. We further investigated the functions with ME called by owners: owners update parameters (SR#6.d) account for the most significant proportion (35.6%), and other influential ones are SR#6 (18.4%), SR#6.a (18.3%), and SR#6.c (16.2%). The above results show that owners tend not to emit events when manipulating contracts.

FP & FN Analysis. We randomly selected 100 token contracts to test the effectiveness of Naga. Table 11 shows the number of correct (TP), incorrect (FP), and missed (FN) state variables detected by Naga compared to the results obtained manually. Overall, Naga is a tool with high accuracy and low false positives. It found 96.5% (715/741) of variables about overpowered owner (SR#6, 6.a~d) and only had six errors. The errors of Naga are mainly generated by D#6, e.g., whitelisted users can call some special functions. Naga erroneously recognized them as variables of SR#6. D#4 and D#5 miss some variables because their names are uncommon, and some contracts violate the specification, such as using `if` statement instead of `require` statement, which lead to D#6 missing them. Naga directly identifies events by IRs of contract code. Hence, FP and FN are not present in SR#7.

6 MITIGATION

To fundamentally eliminate centralized security risks, we need to make revolutionary changes to the current ecosystems represented by Ethereum. However, these changes will inevitably lead to blockchain forks. This section proposes immediate and practical mitigation from the perspective of users and developers.

For Users. Users can evaluate crypto wallets according to D#1 in section 4.1, and use Naga to evaluate contracts. We recommend that users choose wallets with large downloads because the measurement shows that these wallets have fewer risks. Also, users should not provide any information to wallets (SR#1, 2). Users can use onion routing to hide the actual IP addresses and, if possible, run their own blockchain node instead of the RPC service provided by the wallet (SR#4).

For Developers. Developers should use decentralized services to replace traditional centralized services, such as remote storage with zero-knowledge encryption, and fulfill their obligation to inform users (SR#3, 5, 7). We suggest that developers allow wallets to connect to multiple RPC services simultaneously, which is easy to implement and can avoid malicious or failure of a single RPC service (SR#4). If a contract adopts access control, developers can adopt a multi-signature contract as the owner, which can decentralize the owner's privileges and prevent the risk of the leakage of a single private key (SR#6).

7 RELATED WORK

Previous works on decentralized ecosystems were limited in security and anonymity, but our work focuses more on security risks caused by centralization. Horus [38] is a semi-automated security assessment framework designed to analyze crypto wallet apps. Horus reveals several severe vulnerabilities that can lead to the loss of ownership and anonymization of users. Winter et al. [40] measure the privacy and security properties of DeFi applications. They find that many trackers on DeFi sites can trivially link a user's Ethereum address with PII (e.g., name) or phish users. Li et al. [17] first noticed centralized issues of RPC services and presented the DoERS attack, a Denial of Ethereum RPC service that incurs zero Ether cost to the attacker. TokenScope [8] is an automatic tool that detects inconsistent behaviors resulting from tokens deployed in Ethereum. TokenScope can find inconsistencies in standard events, while our tool Naga is to detect if a function is missing an event.

Static analysis tools mainly use formal verification and symbolic execution to detect contracts, such as ZEUS [14], VERISMART [28], OSIRIS [34], Securify [36]. Dynamic analysis tools rely on fuzzers and SMT solvers, such as ContractFuzzer [13], ReGuard [18], Sereum [26], Oyente [21], CONFUZZIUS [33]. However, these existing tools all focus on contract vulnerabilities, and none of them directly support detecting *overpowered owner*.

8 CONCLUSION

The original intention of users to use decentralized services is to avoid centralized security risks. This work takes the first step in investigating the centralized security risks in decentralized ecosystems and lists seven previously unnoticed security risks. We propose six detection methods and implement an automated tool. The measurement results are not optimistic, with 96.4% of crypto wallets and 83.5% of on-chain contracts having security risks. We hope centralized risks can draw the community's attention.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. This work was partially supported by Taishan Young Scholar Program of Shandong Province, China. Jilian Zhang was supported by NSFC (Grant No. 62020106013 and 61972177).

REFERENCES

[1] Oct. 5, 2022. *Find a wallet.* Retrieved Oct. 5, 2022 from https://ethereum.org/en/wallets/find-wallet/
[2] Lawrence Abrams. Apr. 3, 2022. *Fake Trezor data breach emails used to steal cryptocurrency wallets.* Retrieved Oct. 5, 2022 from

https://www.bleepingcomputer.com/news/security/fake-trezor-data-breach-emails-used-to-steal-cryptocurrency-wallets/

[3] Aleks-blockchaincap, Banteg, Dudesahn, Ekrenzke, Lehnberg, Ryanwatkins, Srsparafi, Tracheopteryx, Vooncer, Yfi-cent, and Milkyklim. Jan. 21, 2021. *YIP-57: Funding Yearn's Future.* Retrieved Oct. 5, 2022 from https://gov.yearn.finance/t/yip-57-funding-yearns-future/9319

[4] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P), San Jose, CA, USA, May 22-26, 2017.*

[5] Jeff Benson. Mar. 4, 2022. *Ethereum's Infura Cuts Off Users to Separatist Areas in Ukraine, Accidentally Blocks Venezuela.* Retrieved Oct. 5, 2022 from https://decrypt.co/94315/ethereum-infura-cuts-off-users-separatist-areas-ukraine-accidentally-blocks-venezuela

[6] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P), San Jose, CA, USA, May 17-21, 2015.*

[7] Vitalik Buterin. Oct. 4, 2022. *A Next-Generation Smart Contract and Decentralized Application Platform.* Retrieved Oct. 5, 2022 from https://ethereum.org/en/whitepaper/

[8] Ting Chen, Yufei Zhang, Zihao Li, Xiapu Luo, Ting Wang, Rong Cao, Xiuzhuo Xiao, and Xiaosong Zhang. 2019. TokenScope: Automatically Detecting Inconsistent Behaviors of Cryptocurrency Tokens in Ethereum. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), London, UK, November 11-15, 2019.*

[9] Ethereum. Oct. 5, 2022. *Solidity Lang.* Retrieved Oct. 5, 2022 from https://github.com/ethereum/solidity

[10] Etherscan. Oct. 5, 2022. *Etherscan.* Retrieved Oct. 5, 2022 from http://etherscan.io/

[11] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB@ICSE), Montreal, QC, Canada, May 27, 2019.*

[12] Asem Ghaleb and Karthik Pattabiraman. 2020. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Virtual Event, USA, July 18-22, 2020.*

[13] Bo Jiang, Ye Liu, and W. K. Chan. 2018. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, France, September 3-7, 2018.*

[14] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 18-21, 2018.*

[15] Yogita Khatri. Nov. 11, 2020. *Ethereum infrastructure provider Infura is down, crypto exchanges begin to disable ETH withdrawals.* Retrieved Oct. 5, 2022 from https://www.theblock.co/post/84232/ethereum-infrastructure-provider-infura-is-down

[16] Johannes Krupp and Christian Rossow. 2018. teEther: Gnawing at Ethereum to Automatically Exploit Smart Contracts. In *Proceedings of the 27th USENIX Security Symposium (USENIX-Sec), Baltimore, MD, USA, August 15-17, 2018.*

[17] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. 2021. As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service. In *Proceedings of the 28th Annual Network and Distributed System Security Symposium (NDSS), virtually, February 21-25, 2021.*

[18] Chao Liu, Han Liu, Zhao Cao, Zhong Chen, Bangdao Chen, and Bill Roscoe. 2018. ReGuard: Finding Reentrancy Bugs in Smart Contracts. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE-Companion), Gothenburg, Sweden, May 27 - June 03, 2018.*

[19] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, Qi Li, and Yih-Chun Hu. 2022. Make Web3.0 Connected. *IEEE Transactions on Dependable and Secure Computing* 19, 5 (2022), 2965–2981.

[20] Sishan Long, Soumya Basu, and Emin Gün Sirer. 2022. Measuring Miner Decentralization in Proof-of-Work Blockchains. *CoRR* abs/2203.16058 (2022). arXiv:2203.16058

[21] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), Vienna, Austria, October 24-28, 2016.*

[22] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrücken, Germany, March 21-24, 2016.*

[23] OpenZeppelin. Oct. 5, 2022. *Openzeppelin Contracts.* Retrieved Oct. 5, 2022 from https://github.com/OpenZeppelin/openzeppelin-contracts

[24] Martin Ortner, Eskandari, and Shayan. Jul. 1, 2022. *Smart Contract Sanctuary.* Retrieved Oct. 5, 2022 from https://github.com/tintinweb/smart-contract-sanctuary

[25] Pierluigi Paganini. Jul. 30, 2018. *KICKICO security breach – hackers stole over $7.7 million worth of KICK tokens.* Retrieved Oct. 5, 2022 from https://securityaffairs.co/wordpress/74910/hacking/kickico-hack.html

[26] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2019. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 24-27, 2019.*

[27] Sunbeom So, Seongjoon Hong, and Hakjoo Oh. 2021. SmarTest: Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts through Language Model-Guided Symbolic Execution. In *Proceedings of the 30th USENIX Security Symposium (USENIX-Sec), August 11-13, 2021.*

[28] Sunbeom So, Myungho Lee, Jisu Park, Heejo Lee, and Hakjoo Oh. 2020. VERISMART: A Highly Precise Safety Verifier for Ethereum Smart Contracts. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, May 18-21, 2020.*

[29] Cryptopedia Staff. Mar. 17, 2022. *What Was The DAO?* Retrieved Oct. 5, 2022 from https://www.gemini.com/cryptopedia/the-dao-hack-makerdao

[30] Gilad Stern and Ittai Abraham. 2022. New Dolev-Reischuk Lower Bounds Meet Blockchain Eclipse Attacks. *IACR Cryptol. ePrint Arch.* (2022), 730.

[31] Liya Su, Xinyue Shen, Xiangyu Du, Xiaojing Liao, XiaoFeng Wang, Luyi Xing, and Baoxu Liu. 2021. Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications. In *Proceedings of the 30th USENIX Security Symposium (USENIX-Sec), August 11-13, 2021.*

[32] Eli Tan. Aug. 4, 2022. *Solana's $6M Exploit Likely Tied to Slope Wallet, Developers Say.* Retrieved Oct. 5, 2022 from https://www.coindesk.com/business/2022/08/03/solanas-latest-6m-exploit-likely-tied-to-slope-wallet-devs-say/

[33] Christof Ferreira Torres, Antonio Ken Iannillo, Arthur Gervais, and Radu State. 2021. ConFuzzius: A Data Dependency-Aware Hybrid Fuzzer for Smart Contracts. In *Proceedings of the 6th IEEE European Symposium on Security and Privacy (EuroS&P), Vienna, Austria, September 6-10, 2021.*

[34] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC), San Juan, PR, USA, December 03-07, 2018.*

[35] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. 2020. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, May 18-21, 2020.* IEEE, 894–909.

[36] Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin T. Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS), Toronto, ON, Canada, October 15-19, 2018.*

[37] Connor Tumbleson. Sept. 20, 2022. *Apktool.* Retrieved Oct. 5, 2022 from https://ibotpeaches.github.io/Apktool/

[38] Md Shahab Uddin, Mohammad Mannan, and Amr M. Youssef. 2021. Horus: A Security Assessment Framework for Android Crypto Wallets. In *Proceedings of the 17th EAI International Conference on Security and Privacy in Communication Networks (SecureComm), Virtual Event, September 6-9, 2021.*

[39] Samuel Wan. Jul. 7, 2022. *Celsius Network continues to make moves, prompting calls to resume withdrawals.* Retrieved Oct. 5, 2022 from https://cryptoslate.com/celsius-network-continues-to-make-moves-prompting-calls-to-resume-withdrawals/

[40] Philipp Winter, Anna Harbluk Lorimer, Peter Snyder, and Benjamin Livshits. 2021. What's in Your Wallet? Privacy and Security Issues in Web 3.0. *CoRR* abs/2109.06836 (2021). arXiv:2109.06836

[41] Martin Young. Oct. 7, 2020. *75 crypto exchanges have closed down so far in 2020.* Retrieved Oct. 5, 2022 from https://cointelegraph.com/news/75-crypto-exchanges-have-closed-down-so-far-in-2020

A APPENDIX

A.1 Data-dependency Analysis Engine (DAE)

Here we give the technical details and algorithms of *DAE*. *DAE* includes a *dependency tracker* (*DepTracker*, Algorithm 1) and a *call tracker* (*CallTracker*, Algorithm 2). *DepTracker* analyzes dominators (statements) of the tainted variable or statement from the bottom up. *CallTracker* traces `Return` statements in internal or external calls and feeds results to *DepTracker*. *DAE* records visited functions to prevent infinite loops caused by recursion.

A.2 Smart Contract Detectors

We elaborate on the implementation details of detectors below.

Algorithm 1: DepTracker

```
Input: tainted_vars: array of tainted values;
       dom_irs: dominators of tainted values;
       walked_funs: array of functions already visited.
Output: dep_vars: dependent variables.
1  dep_vars = tainted_vars
2  while dom_irs do
3      ir = dom_irs.pop()
4      // The lvalue of the expression is not in the dep_vars.
5      if ir.lval not in dep_vars then
6          continue
7      dep_vars.remove(ir.lval) // Remove the old value.
8      dep_vars += ir.rvals // Add rvalues to the dep_vars.
9      if isinstance(ir, (InternalCall, HighLevelCall)) then
10         if ir.function in walked_funs then
11             continue
12         walked_funs.append(ir.function)
13         // Call the InternalCall Tracker
14         dep_vars += CallTracker(ir, walked_funs)
15 end
16 return dep_vars
```

Algorithm 2: CallTracker

```
Input: call_ir: ir that calls the internal function;
       walked_funs: functions already visited.
Output: Dependencies
1  dep_vars = []
2  dom_irs = call_ir.dom_irs // Dominators of call_ir
3  while dom_irs do
4      ir = dom_irs.pop()
5      if isinstance(ir, Return) then // Return statement.
6          dep_vars += DepTracker(ir.rvals, dom_irs,
             walked_funs)
7  end
8  return dep_vars
```

SR#6: Overpowered Owner. First, we match the three *access* inheritances of Table 3 in D#3 and the two modifiers (`onlyOwner()` and `onlyRole()`) in D#4. Then, we recognize the remaining variables in D#6. The rules for D#6 are as follows: (1) The `owner` should compare with `msg.sender` in the `require`; (2) The `owner` must be protected by himself or other `owners`. Finally, if we find `owner` variables, we say this contract has *overpowered owner*.

SR#6.a:Limited Liquidity. The `paused` variable has a *Pausable* inheritance and a popular modifier `whenNotPaused()`. We check the two properties in D#3 and D#4. In D#6, we check the state variables of `require` statements in user-writable functions. If the type of a state variable is `bool` or `mapping(address⇒bool)` and the variable is protected by owners, we believe that the variable is a LL variable, so this contract is *limited liquidity*.

SR#6.b: Vulnerable Scarcity. Since there is no `totalSupply` in *ERC721*, we only detect two inheritances (*ERC20* and *ERC1155*) in D#3. `totalSupply` also has a getter `totalSupply()` for D#4. We check the name keyword `totalSupply` and type (`uint`) in D#5. For `totalSupply`, if (1) the variable is protected by owners, (2) owners can increase the variable, and (3) `totalSupply` is not bound by an immutable `uint` variable or a constant, we believe that `totalSupply` is mutable, so this contract is *vulnerable scarcity*.

SR#6.c: Mutable Metadata. We first detect token inheritances and getters in D#3 and D#4, and then we check the name keywords and types of the remaining variables in D#5. Finally, we check if there are functions owners can modify these metadata.

SR#6.d: Mutable Parameters. Since contracts have various parameters, it is difficult to match them directly. In D#6, for every mutable parameter: (1) `uint` type; (2) presenting in user-writable functions; (3) have a write function that only owners call.

SR#7: Missing Events. After identifying state variables, we perform missing event detection. If a function writes to a state variable without emitting any events, we look up the state variable's identity and report that the function is *Missing Events*.

A.3 Valuable Third-party SDKs

Table 12 lists 29 third-party SDKs in crypto wallets. C#1 is *Notification, SMS, Email* , C#2 is *Data Analysis, Customer Analysis*, C#3 is *Identity Verification, Fraud Protection*, C#4 is *Bug Reporting, Error tracking*, and C#5 is *Business Messenger*.

Table 12: Valuable Third-party SDKs

SDK (SR#5)	C#1	C#2	C#3	C#4	C#5	APKs
io.sentry				√		8
io.invertase	√	√	√	√		6
com.appsflyer		√	√			5
io.intercom	√				√	5
com.intercom	√				√	5
com.onesignal	√					3
com.bugsnag				√		3
com.segment		√				3
com.amplitude		√				3
com.zendesk					√	2
com.mixpanel		√				2
com.geetest	√		√			2
com.facetec			√			2
com.pusher	√					2
cn.jiguang	√	√	√			1
cn.asus.push	√	√	√			1
cn.jpush	√	√	√			1
com.instabug				√		1
com.crashlytics				√		1
com.onfido			√			1
com.adjust		√	√			1
com.microblink			√			1
com.sensorsdata		√				1
com.helpscout					√	1
com.passbase			√			1
com.braze	√	√				1
com.appboy	√	√				1
com.tozny			√			1
org.iban4j			√			1