Unveiling the Centralized Security Risks in Decentralized Ecosystems

Kailun Yan, Jilian Zhang, Xiangyu Liu, and Wenrui Diao

***Abstract*—The decentralized ecosystem is claimed to avoid security risks caused by centralization. Decentralized services, such as crypto wallets and decentralized applications (DApps), are purported to offer more reliable security and better protect user privacy. However, our research suggests a different reality: centralized components or scenarios are still prevalent within decentralized ecosystems, introducing security risks typically associated with centralization. This work systematically investigated the centralized security risks in crypto wallets and DApps. We found seven security risks and developed a series of methods to identify these risks. The detection results indicate that centralized security risks are widespread in the decentralized ecosystem. Among the 28 Ethereum-recommended crypto wallets, 96.4% have security risks. Of the 78 Web3 sites (frontends of DApps), 100% contain third-party scripts, and 44.9% expose the user's address to third parties. Furthermore, we developed a high-precision automated tool and inspected 110,506 on-chain smart contracts (backends of DApps), discovering that 83.5% contain at least one security risk. These risks affect 260 well-known tokens with a combined market capitalization exceeding $98 billion.**

***Index Terms*—Decentralized systems, centralized security, crypto wallets, smart contracts, security risks, Web3.**

I. Introduction

BLOCKCHAIN technology is characterized by its decentralized nature, transparency, and immutability [1]. It operates on a distributed ledger system where transactions are recorded across multiple nodes, making it resistant to single points of failure and providing enhanced security. Built upon this technology is a decentralized ecosystem that touts benefits such as security, anonymity, and privacy. Within this ecosystem, key components include crypto wallets and decentralized applications (DApps). Wallets serve as a tool for managing digital assets, while DApps provide a wide range of applications across various sectors like decentralized finance (DeFi) [2], [3], decentralized exchanges (DEXs) [4], NFT (Non-Fungible Token) marketplaces [5], [6], etc.

Since its inception, users have been confident in the potential of decentralized services, believing they can circumvent many risks associated with centralization. Fig. 1 illustrates the main components of a decentralized ecosystem. The red components represent recognized centralized elements, while the yellow ones are considered decentralized. A DApp typically includes a Web3 site as its user interface, also known as the frontend, and uses a smart contract as the backend for processing on-chain transactions. CEXs (centralized exchanges), wallets, and DApps directly interact with users, using RPC (remote procedure call) services to forward transactions to the blockchain network. Our research indicates that decentralized components often rely on or contain centralized elements, undermining the decentralization promise and introducing potential security risks. For example, most decentralized components employ third-party RPC services [7] to forward transactions to the blockchain network. Crypto wallets frequently use third-party SDKs for push notifications and data analytics, which may expose user data to external entities. Similarly, Web3 sites may contain third-party scripts that leak users' wallet addresses. Additionally, many smart contracts implement access controls that restrict certain functions to designated users (referred to as *owners*), which could affect fairness. Some security breaches highlight these concerns. For example, the leading RPC provider Infura [8], which serves as a centralized infrastructure, faced an outage that represented one of the most significant disruptions to Ethereum since The DAO attack [9], [10]. This incident illustrates the inherent security risks of depending on a single, centralized RPC service, such as single points of failure and censorship. In another incident, Slope [11], a well-known crypto wallet, accidentally exposed users' private keys in transfer logs, leading to the theft of tokens valued at over $6 million.

To the best of our knowledge, we are the *first* to systematically explore the centralized security risks in decentralized ecosystems. Prior works focus on blockchain security [12],

Received 29 March 2024; revised 30 September 2025; accepted 9 October 2025. Date of publication 16 October 2025; date of current version 12 March 2026. This work was supported by the Taishan Young Scholar Program of Shandong Province, China under Grant tsqn202211001, and in part by the Xiaomi Young Talents Program. The work of Jilian Zhang was supported in part by the NSFC under Grant 62020106013 and Grant 62472197, and in part by the National Joint Engineering Research Center of Network Security Detection and Protection Technology, Guangdong Key Laboratory of Data Security and Privacy Preserving, Guangdong-Hong Kong Joint Laboratory for Data Security and Privacy Preserving, and Engineering Research Center of Trustworthy AI of Ministry of Education, Jinan University. *(Corresponding author: Wenrui Diao.)*

Kailun Yan is with the Institute for Advanced Study, Tsinghua University, Beijing 100084, China, and also with the School of Cyber Science and Technology, Shandong University, Qingdao 266237, China (e-mail: kailun@mail.tsinghua.edu.cn).

Jilian Zhang is with the College of Cyber Security, Jinan University, Guangzhou 510632, China, and also with the Engineering Research Center of Trustworthy AI, Ministry of Education, Jinan University, Guangzhou 510632, China (e-mail: zhangjilian@jnu.edu.cn).

Xiangyu Liu is with Security Department, Alibaba Group, Hangzhou 311121, China (e-mail: eason.lxy@alibaba-inc.com).

Wenrui Diao is with the School of Cyber Science and Technology, Shandong University, Qingdao 266237, China, also with the Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao 266237, China, and also with the State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao 266237, China (e-mail: diaowenrui@sdu.edu.cn).

Digital Object Identifier 10.1109/TDSC.2025.3622304

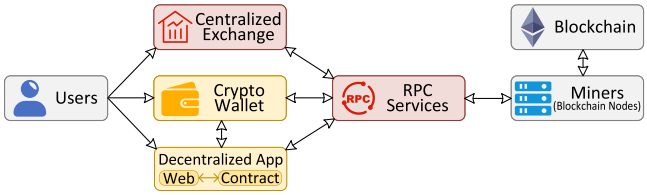


Fig. 1. Main components of the decentralized ecosystem. The red components represent recognized centralized components, while the yellow ones are considered decentralized.

[13], [14], [15], [16], [17], [18] and smart contract vulnerability detection [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. However, from the view of decentralized 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 explored decentralized services and identified seven centralized security risks, with five pertaining to crypto wallets and four (including four sub-issues) associated with DApps. To assess these risks in the wild, we analyzed 28 Ethereum-recommended crypto wallets (Android version) and employed a large language model (LLM) to examine the potential risks of third-party SDKs. Our findings indicate that 27 of these wallets are vulnerable to security risks. In our investigation of the DApps, we discovered that 37.2% (29/78) of popular Web3 sites support centralized login methods, such as Google login, and all feature third-party scripts. Additionally, we found that nearly half (44.9%) of these Web3 sites expose users' wallet addresses to third parties, with the affected DApps having a monthly Unique Active Wallets (UAW) exceeding 11 million. We developed an automated tool, Naga, for detecting risks in smart contracts by identifying state variables. Unlike previous studies that analyzed data dependencies only at the function level, Naga offers a more fine-grained analysis by examining data dependencies in the intermediate representations (IRs) of contract code. Further, we conducted a large-scale evaluation on 110,506 Ethereum on-chain contracts and identified 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 our contributions:

- *New security risks.* We conducted the first systematic study on the centralized security risks in decentralized ecosystems. We identified seven security risks in crypto wallets and DApps.
- *New techniques.* We utilized two methods to examine five security risks in crypto wallets, proposed a semi-automatic approach to inspect Web3 sites, and developed an automated tool to detect smart contract risks.
- *Real-world evaluations.* We evaluated 28 well-known crypto wallets, 78 popular Web3 sites, and 110,506 smart contracts. The results show that the centralized security risks are widespread.

This work extends our conference version presented in the Proceedings of The Web Conference 2023 [29]. This extended version introduces Web3 sites as a new research object and investigates the associated security risks (SR#1, SR#5) associated with them. Specifically, we developed a new semi-automated detection method as detailed in Section IV-B and analyzed 78 popular Web3 sites, revealing their security risks. In Section IV-A, we redesigned the detection of third-party SDKs in crypto wallets, broadening the scope and employing an LLM for SDK classification. Unlike the previous keyword-matching approach, this novel method both automates detection and improves accuracy. Section V has added an evaluation of the LLM-based detection methods and presents an in-depth performance analysis of Naga. Moreover, we have expanded the background in Section II and added a discussion in Section VI.

Roadmap: The remainder of this paper is structured as follows. Section II provides the background of decentralized ecosystems. Section III introduces the seven discovered security risks. Section IV introduces a series of detection methods. Section V presents the measurement results. Section VI discusses designs for decentralized services and suggests mitigation measures. Section VII reviews related work, and Section VIII concludes this paper.

II. Background

Decentralized ecosystems are verifiable, self-governing, and permissionless, allowing participants equal access to services without the need for personal data. Based on blockchain and smart contract technology, Ethereum [30] supports a variety of DApps and stands out as the most famous decentralized platform. This section introduces the four main components of the decentralized ecosystem shown in Fig. 1.

DApps: Decentralized applications (DApps) enhance the ecosystem's prosperity and offer users various decentralized services. A DApp typically includes a frontend, serving as the user interface, and a backend that processes transactions. The user interface often takes the form of a Web3 site and users can connect to the DApp using a crypto wallet. The backend is a smart contract that executes transactions triggered by the frontend. These characteristics ensure the transparency, trustworthiness, and immutability of a DApp. Ethereum Request-for-Comments (ERC) standards provide guidelines for smart contracts. Among these, ERC20 is designed for fungible tokens (FTs), while ERC721 is used for non-fungible tokens (NFTs). ERC1155 supports both FTs and NFTs; however, since it is widely used for NFTs, this paper puts it into the NFT category. Solidity is the most popular programming language for developing smart contracts. Contracts written in high-level languages are compiled into bytecode and then run on the Ethereum Virtual Machine (EVM).

Crypto Wallets: A crypto wallet [31] is the fundamental tool for a user to access the decentralized ecosystem. It offers various functions, such as sending and receiving cryptocurrency, monitoring balances, and connecting to DApps. Most wallets store private keys exclusively on the user's device, ensuring that users have sole control over their digital assets. Crypto wallets come in various formats: mobile applications, browser extensions, desktop software, and physical hardware. Notably, mobile

applications stand out for their accessibility and convenience, making them the preferred choice for many.

RPC Services: The blockchain RPC (remote procedure call) service is essentially a full node that provides external calls [7]. Using RPC services, crypto wallets and DApp sites can interact directly with the blockchain, eliminating the need to run their own full nodes. While RPC services greatly simplify the development of applications, they introduce some centralization risks, as all operations pass through these intermediary RPC services. Well-known third-party RPC services include Infura [32] and Alchemy [33]. As centralized components, RPC services can become single points of failure. Notably, Infura runs on Amazon's cloud servers, compounding these issues by introducing additional cloud-level centralization risks.

Centralized Exchanges: Centralized exchanges (CEXs) bridge traditional fiat currencies with cryptocurrencies, offering robust liquidity for conversion and thereby supporting the growth of decentralized ecosystems. Unlike decentralized exchanges (DEXs), where users retain full control of their private keys, CEXs require users to deposit assets into exchange-controlled wallets. This model improves usability and regulatory compliance but requires users' unconditional trust in the exchange. In addition, CEXs are regulated by governments and typically require personally identifiable information (PII), which raises concerns such as privacy leakage. Conversely, DEXs, as DApps, avoid custody and PII requirements but face challenges in liquidity and user experience. Together, CEXs and DEXs fulfill complementary roles within the ecosystem.

III. Centralized Security Risks

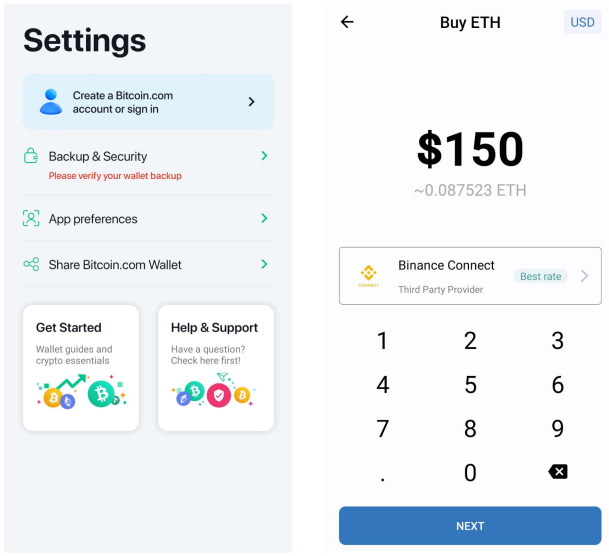
This section introduces the threat model and highlights security risks arising from centralization.

A. Overview

Even within a decentralized architecture, there can be centralized components that a malicious individual could control. Drawing on Section II, this study examines centralized risks in crypto wallets, as well as in the frontend (Web3 sites) and backend (smart contracts) of DApps. We assume decentralized platforms are benign and miners do not collude. In this context, decentralized service providers are treated as potential adversaries, and we consider two scenarios:

- *First-party Centralization.* The adversary incorporates centralized services or backdoors into the decentralized service he developed. Typically, the adversary intentionally blurs the lines of decentralization and misleads users by promoting the service as operating on a decentralized platform.
- *Third-party Centralization.* The adversary, as a third party, supplies centralized components to decentralized services, thereby contaminating the decentralized ecosystem. This often involves offering services or software development kits (SDKs) to persuade developers to embed those components in their decentralized services.

Crypto Wallets: Due to the overwhelming dominance of Android users, this study focuses on Android-based wallets. Many



(a) Settings in Bitcoin.com Wallet (b) CEX in Trust Wallet

Fig. 2. Crypto wallets with centralized services.

wallets, despite claiming decentralization and anonymity, often fall short in practice. Some request personally identifiable information (PII) during registration, while others rely on centralized third-party components (e.g., RPC providers, SDKs). Such dependencies undermine decentralization and expose wallets to security risks.

DApps: Since Web3 sites and contracts are not strictly one-to-one, this paper examines popular Web3 sites and representative contracts, focusing on ERC20, ERC721, and ERC1155, the three most widely used standards. DApps have long been regarded as decentralized, anonymous, and more secure applications. However, in practice, Web3 sites accessed through browsers or crypto wallets may expose user privacy to embedded third-party scripts. Meanwhile, smart contracts often implement access control, restricting certain functions to specific addresses. These issues undermine the ideals of decentralization and create potential security risks.

B. Security Risks

We conducted a systematic investigation of the decentralized ecosystem and identified a series of security risks associated with centralization. This section describes these security risks, with SR#1 to SR#5 identified in crypto wallets. Additionally, SR#1 and SR#5 relate to Web3 sites, whereas SR#6 and SR#7 are associated with smart contracts.

SR#1: Anonymity Loss (AL): Some crypto wallets require registration and ask users to provide personally identifiable information (PII), such as email addresses and phone numbers. As shown in Fig. 2(b), a popular crypto wallet with over 5 million downloads, Bitcoin.comWallet, offers optional registration. Some Web3 applications also provide multiple login methods. Wallet-based login typically exposes only the user's public key to the application, helping preserve anonymity. In contrast, social logins (e.g., Google, Facebook) often bind users'

social identities to their wallet addresses. As a result, wallets and sites can associate PII with the user's public key, reducing privacy [34], [35]. Attackers may also exploit leaked PII to launch phishing attacks. For example, the Trezor wallet exposed the email addresses of 106,856 users, leading to widespread phishing campaigns [36].

SR#2: Private Key Leakage (PL): Private keys, crucial for users' crypto assets, should be securely stored locally (on the device). However, some wallets encourage users to back-up their private keys on first- or third-party servers. Fig. 2(a) is the settings page of Bitcoin.comWallet, which contains the option for private key back up. Another wallet, ZenGo, requires users to back up private keys to Google Drive before using it. Such practices might lead to private key leakage because servers could either misappropriate or unintentionally leak these keys. Additionally, during network transmission, private keys could be vulnerable to man-in-the-middle (MITM) attacks. For instance, the Slope wallet leaked users' private keys during transmission, resulting in a theft approximating $6 million in token value [11].

SR#3: Built-in Centralized Services (BS): Certain wallets have built-in centralized services. Fig. 2(b) shows the cryptocurrency purchases service in TrustWallet, provided by Binance, a renowned CEX. These centralized services are under government regulation and could share users' PII. Most users do not recognize the difference between centralized and decentralized services in wallets, so these platforms must inform users about the risks associated with centralized services. Also, users may suffer financial losses due to centralized service outages. For example, as of 2020, 75 exchanges had ceased operations [37]. CelsiusNetwork is a centralized finance platform with 1.7 million users that paused its services due to "*extreme market conditions*" [38].

SR#4: RPC Services (RS): RPC services or providers suffer from issues inherent in centralization, such as denial-of-service (DoS) [7]. Also, RPC services can withhold transactions with transaction-ordering dependence (TOD) [39] for huge benefits. Crypto wallets should disclose built-in RPC services 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 [40].

SR#5: Third-Party SDKs or Scripts (TS): Crypto wallets frequently employ third-party SDKs for functionalities like notifications and fraud protection. In doing so, wallets might share users' PII, device IDs, and crash logs with these third-party entities. In the Slope incident [11], white-hat hackers discovered that Slope was transmitting users' sensitive information in plaintext to Sentry, a bug-tracking SDK. Similarly, Web3 sites often embed third-party analytics scripts or host their own scripts on external platforms such as CDNs (content delivery networks). These scripts are loaded from external domains and executed in the user's browser, which may expose the user's IP address and even leak wallet addresses (public keys) to third parties.

SR#6: Overpowered Owner (OO): Smart contracts can verify the user's address to allow or deny the user to invoke a specific function, i.e., access control. Access control enables

```
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.

```
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.

creators (owners) to exert undue influence over contracts, aka *overpowered owner (OO)*. Listing 1 provides an example. First, the `constructor()` sets the creator (`msg.sender`) as the owner (`_owner`) in line 2. The modifier `onlyOwner()` has a `require` statement in line 7, ensuring that the caller matches the owner. Consequently, only the owner (creator) can access the `mint()` function due to its protection by `onlyOwner()`. OpenZeppelin [41] is a reputable library that provides two types of access controls, namely *Ownable* and *AccessControl*. Some contracts opt for a basic `mapping` to manage a set of privileged addresses, termed *AdminControl*. The variable of *Ownable* is named `owner`. The variables of *AccessControl* and *AdminControl* are called `roles` and `admins`, respectively. For clarity, this paper uses *the owner* or *owners* to represent roles within these three access controls.

Generally, contracts adopt access control to protect accounts' interests and maintain contracts. However, an 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 a failure is the KickICO incident [42]. 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. This paper define the term *liquidity* as whether anyone can *transfer* or *allowance* tokens without limitations. If the owner can control the related functions in a contract, the contract has *limited liquidity*. Listing 2 shows two kinds of limited liquidity, one is to set a `bool` variable to pause the function (line 2), and the other to blockthe address to prevent it from calling the function (line 3).

SR#6.b: Vulnerable Scarcity (VS): Tokens are scarce because usually the total supply of a contract is limited. *Vulnerable scarcity* means that the owner can arbitrarily increase the supply

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

Listing 3. Example of the ERC20 Metadata.

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

Listing 4. Example of an `Event`.

of tokens. 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. For example, the Yearn.finance team proposed to issue an additional $225 million worth of YFI to incentivize developers, which caused fierce protests from current holders [43].

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 contract deployment. Wallets and browsers display contract metadata to users, so mutable metadata may mislead users into sending unexpected transactions.

SR#6.d: Mutable Parameters (MP): The function `transfer()` of ERC20 often has customized parameters, such as *transferTax*. 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). However, if a function does not emit an event after updating a state variable, no one will be notified unless they actively checks the contract in the blockchain.

IV. Detection Approaches

This section outlines our approaches to detecting the seven centralized security risks identified previously. First, we propose four research questions and design an automated detection method for identifying wallet risks in Section IV-A. Next, we introduce a semi-automated method for detecting two Web3 site risks in Section IV-B. Finally, we present an automated tool for identifying contract risks in Section IV-C.

A. Detection on Crypto Wallets

The Ethereum website lists 44 wallets [44]. Of these, 30 support the Android platform, accounting for the most significant proportion. Therefore, we focus on Android wallets as our research object. We analyzed the security risks of wallets from two perspectives. The first is function checking from the user's point of view (D#1), and the second involves leveraging a large language model to classify SDKs (D#2).

D#1: Function Check: We manually checked the main functions of crypto wallets, including generating and importing private keys, trading, modifying RPCs, and more. Our attention was directed towards the following four research questions:

- RQ1 Does the wallet require users to register or provide personal identifiable information before use? (SR#1)
- RQ2 Does the wallet recommend users to back up their private keys to the cloud? (SR#2)
- RQ3 Does the wallet have built-in centralized services and inform users of their centralization? (SR#3)
- RQ4 Can users modify the RPC of the wallet? (SR#4)

D#2: LLM-based Detection: Large language models (LLMs) have recently been applied to code security analysis [45], [46], showing advantages in natural language understanding and contextual reasoning. Building on this, we leverage LLMs as expert assistants to automatically analyze SDK-related websites and determine their functionalities. As illustrated in Fig. 3,

the detection process begins with decompiling a wallet's APK using Apktool to access its decompiled resources. Then, it extracts third-party packages (SDKs) by performing a breadth-first search in the *smali* folders and parsing the *AndroidManifest.xml* file. Because package names typically adhere to the Java naming convention, such as *domain.company.project*. For instance, *com.sensetime.senseid* suggests a connection to http://senseid.sensetime.com. It then extracts the first two or three segments of these package names, indicative of the domain's first and second levels, and reverses them to form URLs. Next, the webpages from these URLs are crawled, and the SDKs are classified using OpenAI's GPT-4. The domain name and website content of the SDK are provided to the GPT-4 API, which is then prompted to determine whether the site offers services from predefined categories.

Prompt

Is this website ({url}) used to introduce or promote the following business services:{sdk_categories}
Your answer should start with Yes or No followed by an array, such as [1,1,0,0,0].
Website content: {text}

We structured the GPT-4 prompt as follows: To constrain its open-ended responses, we first categorized the services commonly found on third-party SDK websites. These services were integrated into the prompt as an array. We then instructed GPT-4 to evaluate whether the website introduces or promotes the services enumerated in the array. Moreover, we specified that GPT-4's answers should conform to a certain structure to ease automated parsing. GPT-4 first verifies the service's presence (replying with *Yes* or *No*), followed by providing an array. Within this array, '1' indicates the service's presence at the corresponding index, while '0' denotes its absence. The details of SDK categories are provided in Section V.

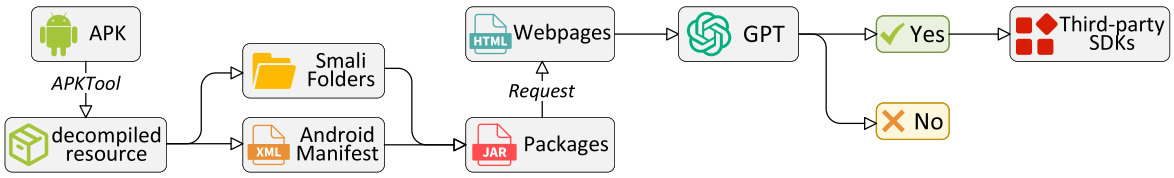


Fig. 3. Process of identifying risky third-party SDKs.

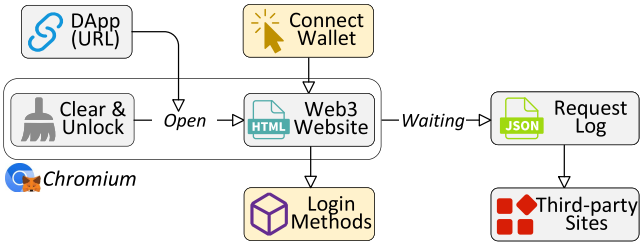


Fig. 4. Semi-automatic detection process of SR#1 and SR#5 in DApp. The yellow ones are manual operations, and the others are automated operations.

TABLE I
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	`block`	`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 II	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.

B. Detection on Web3 Sites

We propose a semi-automatic detection method to detect SR#1 and SR#5 that exist in the front end of DApps, namely Web3 sites. For a given Web3 site, we access the site and capture the network requests made by its scripts to detect SR#5. During the detection process, we manually check the login methods supported by the site to identify SR#1, and connect the wallet to the site to further identify SR#5.

D#3: Semi-Automatic Detection: We develop the tool based on playwright to implement the above method; playwright is an end-to-end web applications testing tool that can perform automated operations in the browser. Also, we use the most popular crypto wallet, Metamask, for testing. Notably, we do not use the browser's privacy mode because the privacy mode disables some third-party scripts. Unlike approaches that directly test Web3 site APIs [47], our method interacts with the site through the browser, thereby better reflecting real user behavior. Fig. 4 illustrates our detection process. Specifically, the tool pre-installs the Metamask into the open-source browser Chromium and then tests a set of Web3 sites. Before testing each site, the tool clears the browser's cache and unlocks . Then, the tool opens the Web3 siteusing the given URL and captures all network requests made by the site during the test. Since the login button of each site is different and cannot be triggered automatically, we manually trigger the login button and record the login methods supported by the site (SR#1). If the site supports wallet login, we manually connect Metamask to the site, so that the site can access our wallet address. After connecting the wallet, we wait 15 seconds to ensure that the script execution on the web page is completed. Finally, the tool exports all captured requests as a log file for SR#5 analysis.

For Anonymity Loss (SR#1) We classify the login methods of Web3 sites into three categories: *only support social login*, *only support wallet login*, and *support both wallet and social login*. Social login includes single sign-on (SSO), such as Google and Facebook, as well as logging in with an email address or mobile phone number. Social login exposes the user's PII to the Web3 site, allowing it to associate wallet addresses with personal data, thereby compromising anonymity. Therefore, DApps that support social login have SR#1.

For Third-Party Scripts (SR#5) We analyze the request log to determine if a DApp has risky third-party scripts. If a script on the DApp sends requests to third-party domains, we regard it as risky because it may leak device information, IP addresses, and other user data.. Furthermore, we check whether our wallet address appears in any third-party requests; if so, the script also leaks the user's wallet address.

C. Detection on Smart Contracts

This section proposes four detection methods based on state variable identification and implements an automated tool to detect security risks in smart contracts.

1) State Variables Identification: We detect security risks in the Solidity source code of contracts. Solidity is by far the most widely used smart contract language in the Ethereum ecosystem, and most existing auditing practices and research efforts focus on this level [48], [49], [50]. Solidity uses state variables to store a contract *state* on the blockchain. A contract with an overpowered owner typically defines a state variable (called *owner*) that stores the owner's address 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 govern the contract by modifying other state variables. Therefore, to detect SR#6 and its four sub-issues, we need to find out the owner variable and the owner-controlled variables. Table I lists these state variables and detection methods. Below we introduce four methods (D#4∼7) to identify state variables in Table I.

TABLE II
OPTIONAL METADATA FOR TOKENS

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

TABLE III
OPENZEPPELIN CONTRACTS

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

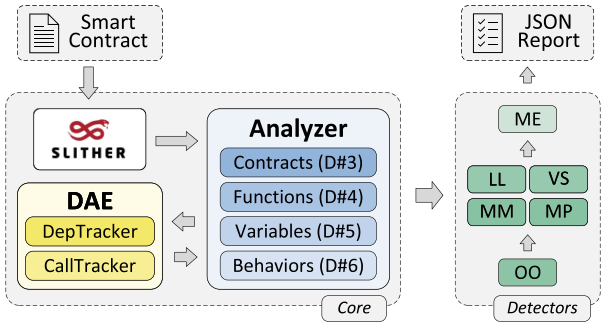


Fig. 5. Overview of naga architecture.

D#4: Inheritance: Solidity supports multiple inheritance, and many contracts inherit the OpenZeppelin library to provide functionality. Table III lists the representative contracts, which define critical features such as ownership, access control, and token standards. By annotating their state variables, our approach can directly identify the corresponding variables in Table I when contracts inherit from them.

D#5: Getters & Modifiers: If a contract does not inherit any given contracts, D#4 will no longer be applicable. In such cases, we leverage *getters* and common modifiers (e.g., `onlyOwner()`, `onlyRole()`) to extract state variables. Listing 3 shows the getters of the ERC20, and we can identify variables through these getters. By matching these functions against those in Table III, we can still identify relevant variables through their return statements.

D#6: Variable Name & Type: Solidity can automatically create getters for state variables declared `public`, so that a contract may omit getters. Therefore, we match variable names and types to identify state variables that are not covered by D#4 and D#5.

D#7: Behavior Patterns: The above methods cannot handle custom variables, e.g., a contract may have an owner variable with an uncommon name. 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 it must appear in a `require` statement to protect other state variables.

Following the four methods, we detect state variables in Table I and evaluate the security risks (SR#6, SR#6.a ∼ d) of a contract based on whether the owner 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 an *owner-related ME* ; otherwise, it is a *user-related ME*.

Because D#4 and D#5 rely on library matching, they achieve high accuracy. However, D#6 may generate false positives due to potential overlaps in variable names and types. Therefore, we limit the scope of D#6 to the variables `totalSupply` (SR#6.b) and `metadata` (SR#6.c), which are widely recognized. We rely on D#7 to address the variability associated with SR#6, SR#6.a, and SR#6.d. Note that SR#6.b and SR#6.c do not utilize D#7, as their variables are inherently read-only, making behavioral pattern definition unnecessary.

2) Design of Naga*:* Based on the D#4∼7 detection methods, we designed a static analysis tool called Naga[1] to detect security risks in smart contracts. The core idea of Naga is to trace how critical state variables are governed by access control, which enables robust vulnerability detection across diverse contracts. Fig. 5 illustrates the architecture of Naga, which consists of two modules, i.e., *Core* and *Detectors*. Naga is built on top of SLITHER [51], a static analysis framework. First, SLITHER compiles the available Solidity source code to generate an intermediate representation (IR). If only the bytecode is provided, IR generation is not feasible. Then, the Analyzer loads the SLITHER object and analyzes it at multiple levels. Finally, *Detectors* module identifies security risks using predefined detectors and outputs a JSON report. When analyzing functions, Naga begins from all `public/ external` functions, as they are the only ones directly accessible to external users. These entry functions may further invoke `internal` or `private` functions, thereby forming call chains. Therefore, the *Core* module incorporates a *Data-Dependency Analysis Engine (DAE)* for fine-grained tracing, which tracks the dependencies of key variables or statements, and supports the analysis of internal or external calls. D#5 and D#7 benefit from *DAE*.

Data-dependency Analysis Engine (DAE): The *DAE* uncovers the data-dependencies of a given variable or statement by tracing intermediate representations (IRs) produced by SLITHER. Owing to the advantages of the static single assignment (SSA) form of the IR, the *DAE* is notably accurate. Here, we provide an example.

In Listing 6, the function `transferOwnership()` contains a compound `require` in line 8, which is equivalent to lines 9 and 10. Without the *DAE*, we only learn that line 8 depends on two local variables (`cOwner` and `newOwner`) and one constant address (`0x0`). Listing 5 showcases the SSA form of IRs from Listing 6. Line 11 in Listing 5 represents the IR of the tainted line 8 in Listing 6. The rvalue `TMP_4` in line 11 depends on `TMP_1` and `TMP_3` in line 10. *DAE* supports the decomposition of the && operator in conditional statements. Upon detecting the && operator in line 10, the *DAE* initiates two *sub-DAEs* to evaluate `TMP_1` (♦) and `TMP_3` (▲). In line 6,

[1]Naga is available at github.com/d0scoo1/Naga

```
address public _owner; // the owner variable
function _msgSender() internal returns (address) {
  return msg.sender;
}

function transferOwnership(address newOwner) external {
  address cOwner = _owner;
  require(cOwner==_msgSender() && newOwner!=address(0));
  // require(cOwner == _msgSender());
  // require(newOwner != address(0));
  _owner = newOwner;
}
```

Listing 5. Example of Transferring Ownership.

```
_msgSender() IRs:
  RETURN msg.sender ▲

transferOwnership() IRs:
  cOwner_1(address) := _owner_1(address) ▲
  TMP_0(address) = INTERNAL_CALL, _msgSender()() ▲
  TMP_1(bool) = cOwner_1 == TMP_0 ▲
  TMP_2 = CONVERT 0 to address ♦
  TMP_3(bool) = newOwner_1 != TMP_2 ♦
  TMP_4(bool) = TMP_1 && TMP_3 ◀
  TMP_5(None) = SOLIDITY_CALL require(bool)(TMP_4)
  _owner_2(address) := newOwner_1(address)
```

Listing 6. Static Single Assignment Form of IRs of Listing 5.

the *sub-DAE* delves into the internal call `_msgSender()`, retrieving the return value `msg.sender`. By tracing the green marks, it becomes evident that `TMP_3` relies on `newOwner_1` (equivalent to `newOwner`) and the address `0x0`. Naga learns 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#7, Naga reports that this function is controlled by the owner due to the first condition.

Detectors: Based on the four detection methods, we design six detectors to find risks. Also, Naga supports defining additional detectors to extend its capabilities. The details of these detectors are described below.

SR#6: Overpowered Owner Initially, we identify the three *access* inheritances from Table III in D#4 and the two modifiers, `onlyOwner()` and `onlyRole()`, in D#5. Subsequently, we detect the remaining variables in D#7. The criteria for D#7 include: (1) The `owner` must be compared against `msg.sender` in a `require` statement; (2) The `owner` must be protected either by himself or by other `owners`. If such `owner` variables are discovered, we conclude that the contract possesses an *overpowered owner*.

SR#6.a: Limited Liquidity The variable `paused` is associated with the *Pausable* inheritance and is often complemented by the modifier `whenNotPaused()`. These are covered by D#4 and D#5. Within D#7, we assess the state variables contained in `require` statements of user-writable functions. If a state variable is either of type `bool` or `mapping(address ⇒ bool)`, and is protected by owners, we deduce that it is an LL variable, indicating the contract exhibits *limited liquidity*.

SR#6.b: Vulnerable Scarcity We focus on identifying two specific inheritances, *ERC20* and *ERC1155*, in D#4, given that *ERC721* lacks the `totalSupply` attribute. The `totalSupply` attribute is accompanied by a getter, `totalSupply()`, which we address in D#5. In D#6, we scrutinize the name keyword `totalSupply` and its type (`uint`). A contract is labeled as exhibiting *vulnerable scarcity* if the following conditions hold true: (1) the variable `totalSupply` is protected by owners, (2) owners can increase the value of the variable, and (3) `totalSupply` is not bound by an immutable `uint` variable or a constant.

SR#6.c: Mutable Metadata In D#4 and D#5, we identify token inheritances and their associated getters. Moving on to D#6, we examine the name keywords and types of the remaining variables. Finally, we check whether any owner-only functions can modify metadata.

SR#6.d: Mutable Parameters Given the diversity of parameters in contracts, direct matching proves challenging. For D#7, each mutable parameter should meet the following criteria: (1) be of type `uint`; (2) appear in user-writable functions; and (3) have a write function exclusively callable by owners.

SR#7: Missing Events Upon recognizing state variables, we perform missing-event detection. When a function modifies a state variable without triggering any events, we consider the identity of the state variable and flag that function as a missing-event case.

V. Measurement and Findings

A. Experiment Setup

Crypto Wallet Dataset: Wallets typically offer the same features across different platforms. Given the widespread usage of the Android platform for wallets, this paper focuses on Android wallets as a representative example. We downloaded 30 Android crypto wallet apps recommended by Ethereum from Google Play. However, two of these wallets, Ledger and Keystone, require additional hardware and were thus excluded from our experiment. The remaining 28 wallets collectively account for over 131 million downloads. This includes one wallet with over 100 million downloads and two wallets with more than 10 million downloads each.

DApp Dataset: Web3 sites and contracts do not correspond one-to-one, and the scale of smart contracts is much larger than Web3 sites. Therefore, we collected the Web3 site dataset and the smart contract dataset respectively. In total, our dataset includes 78 well-known Web3 sites and 110,506 contracts.

Web3 site Dataset We selected 78 Web3 sites from Dappradar [52], a well-known decentralized application distribution platform. We sorted the DApps by Unique Active Wallets (UAW) in the past 30 days, and then selected the top 20 DApps from the five categories of DeFi, DEX, Gambling, Marketplace, and Social Media. We excluded duplicates and unavailable Web3 sites and obtained a dataset of 78 Web3 sites. The monthly UAW of 78 DApps exceeds 19 million, and the monthly transaction volume exceeds $137 billion.

TABLE IV
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

Smart Contract Dataset Table IV lists two contract datasets,[2] totaling 110,506 contracts. These include 43,152 ERC20, 29,522 ERC721, 4,161 ERC1155, and 33,671 non-token contracts. The first dataset, termed *Etherscan token (ET)*, is sourced from Etherscan's token tracker [53]. Etherscan has identified several ERC20, ERC721, and ERC1155 tokens. We crawled the source codes of these tokens to form the *ET* dataset. This dataset is of high value, with the ERC20 tokens boasting 30 million holders and a cumulative market cap exceeding $310 billion. The second dataset, *Ethereum mainnet (EM)*, originates from the *SCS* project [54], which aggregates the source code of contracts. Due to compilation failures of half the contracts in *SCS*, we independently crawled all 143,900 contracts from Etherscan. We eliminated duplicates between the two datasets and excluded contracts written in Solidity versions earlier than 0.5.0, as these versions do not support the `require` keyword. Token identification in the *EM* dataset was performed by matching function signatures, a common method employed by crypto wallets and platforms, including Etherscan.

Setup: We performed D#1 on a Lenovo Lemeng K12 phone with Android 10, and ran D#2, D#3, and Naga on a server with two Intel(R) Xeon(R) Gold 6226R CPUs (2.90 GHz) and 256 GB memory, running Ubuntu 20.04 and Python 3.8.10. The timeout for contract compilation and Naga analysis was set to 60 seconds. We detected all risks (SR#6, SR#6.a ∼ d, and SR#7) in token contracts, and only SR#6 and SR#7in non-token contracts, since the variables defined in SR#6.a∼ d. are not always present in non-token contracts.

B. Security Risks in Crypto Wallets

Table V lists the security risks (SR#1 to SR#5) of crypto wallets. The results show that 96.4% (27/28) of wallets have at least one security risk.

To SR#1: Anonymity Loss: Table VI lists five wallets that require mandatory registration. Among the five wallets, four require email addresses, two require phone numbers, and two require face verification. In particular, ZenGo requires users to link their Google accounts and back up private keys to Google Drive, while Aktionariat requests additional personal details such as name and address. Three wallets offer optional registration (◐ in Table V) but still require email addresses. Almost all wallets with SR#1 offer supplementary services such as online backup (SR#2) and centralized exchange (SR#3), so they tend to make users sign up.

To SR#2: Private Key Leakage: Typically, wallets advise users to write down the mnemonic words associated with their private keys. Nonetheless, seven wallets prioritize convenience by suggesting users opt for online backups, thus exposing them to the risk of private key leakage. The Bitcoin.comWallet and the Coin98Wallet, boasting 5 M and 1 M downloads respectively, prompt users to store their private keys on their servers. Additionally, five wallets, including ZenGo, advocate for users to back up their private keys to Google Drive.

To SR#3: Built-in Centralized Services: Two wallets come with built-in exchanges, while 19 others offer cryptocurrency purchase functions. However, none of these wallets inform users that such services are centralized. Hiding the centralization of these services may be a strategy to attract users. Only nine wallets disclose the service providers (◐ in Table V). Notably, none of the wallets request Android permissions, with the sole exception being Numio. This crypto wallet requires users to grant location permissions, which it utilizes to verify whether users are eligible to purchase cryptocurrencies. It's worth noting

[2]Both datasets are available at github.com/d0scoo1/naga_contracts

TABLE V
SECURITY RISKS OF CRYPTO WALLETS

Crypto Wallet	DLs	SR#1	SR#2	SR#3	SR#4	SR#5
Brave Wallet	100M+	○	○	◐	○	1
Coinbase Wallet	10M+	○	○	○	●	12
MetaMask	10M+	○	○	◐	○	4
Bitcoin.com Wallet	5M+	◐	Cloud	●	●	7
Exodus	1M+	○	○	◐	●	3
Opera Wallet	1M+	○	○	○	●	3
Status	1M+	○	○	◐	○	3
TokenPocket	1M+	○	○	○	○	0
Coin98 Wallet	500K+	◐	Cloud	●	●	4
imToken	500K+	○	○	○	○	6
MEW Wallet	500K+	○	○	●	●	5
AlphaWallet	100K+	○	○	◐	○	5
Argent	100K+	●	Google	◐	●	11
Coin Wallet	100K+	○	○	◐	●	2
Guarda	100K+	○	○	●	●	1
Pillar	100K+	○	○	●	●	7
ZenGo	100K+	●	Google	●	●	12
Zerion Wallet	100K+	○	○	◐	●	5
1inch Wallet	50K+	○	Google	○	●	3
Loopring Wallet	50K+	●	○	○	●	2
AirGap Wallet	10K+	○	○	○	●	1
Bridge Wallet	10K+	◐	○	●	○	5
FoxWallet	10K+	○	○	◐	○	5
Gnosis Safe	10K+	○	○	○	●	4
Numio	10K+	●	Google	●	●	4
Rainbow	10K+	○	Google	●	●	6
Unstoppable	10K+	○	○	○	●	1
Aktionariat	1K+	●	○	●	●	2

● Risk present; ◐ Potential risk; ○ No risk.

TABLE VI
CRYPTO WALLETS REQUIRING ACCOUNT REGISTRATION (SR#1)

	Personally Identifiable Information		
Crypto Wallet	**Email**	**Phone Number**	**Face**
Argent	✓	✓	
ZenGo	✓		✓
Loopring Wallet	✓		
Numio			✓
Aktionariat	✓	✓	

TABLE VII
CATEGORIES OF THIRD-PARTY SDKS (SR#5)

	Categories	SDKs	Wallets
C#1	Notification, SMS, Email	18	26
C#2	Data Analysis, Customer Analysis	22	27
C#3	ID Verification, Fraud Protection	16	17
C#4	Bug Reporting, Error Tracking	9	25
C#5	In app chat, Chatbot	11	26

An SDK can fit multiple categories.

TABLE VIII
TOP TEN THIRD-PARTY SDKS

SDK (SR#5)	C#1	C#2	C#3	C#4	C#5	Wallets
com.google.firebase	✓	✓		✓	✓	25
com.squareup		✓				14
io.sentry				✓		8
io.branch	✓	✓	✓			6
com.intercom	✓	✓			✓	5
com.appsflyer		✓	✓			5
com.nimbusds			✓			5
io.intercom	✓	✓			✓	5
com.wix	✓	✓				4
com.google.cloud		✓				3

TABLE IX
TOP 10 THIRD-PARTY SITES WITH EMBEDDED SCRIPTS IN DAPPS (SR#5)

Third-party Sites	DApps	Third-party Sites	DApps
googletagmanager.com	52	google.com	25
googleapis.com	51	cloudflareinsights.com	20
gstatic.com	48	sentry.io	20
google-analytics.com	45	doubleclick.net	19
walletconnect.com	33	walletconnect.org	18

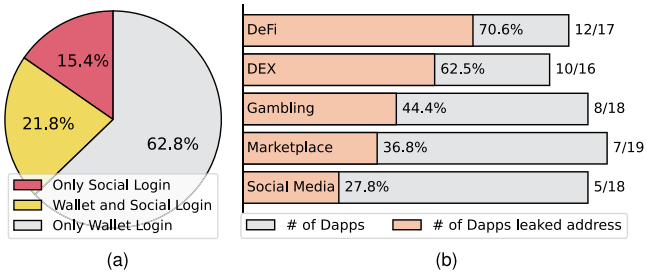


Fig. 6. (a) Supported login methods in 78 DApps (SR#1). (b) The number of DApps in each category and user address leakage (SR#5).

that certain services like DEX Swap are decentralized and have been excluded from our analysis.

To SR#4: RPC Services: Within our dataset, 20 crypto wallets neither disclose their RPC providers nor allow modifications to RPCs. Of the remaining wallets, eight allows users to add new RPCs, and six of them also disclose their RPC providers. Among these six, five have centralized RPC providers, which include entities like *infura.io*, *nodereal.io*, and *ankr.com*. Exceptionally, the wallet imToken operates its own RPC service through *token.im*.

To SR#5: Third-Party SDKs: We investigated websites of common third-party SDKs and categorized them into five groups, as shown in Table VII. SDKs in Category C#1 may access users' contact information, while those in Categories C#2∼C#5 may access other PII, threatening user anonymity. Using GPT-4, we then identified 39 third-party SDKs that fall into these categories, and manual verification confirmed the results. Table VIII showcases the 10 most frequently used SDKs in APKs. Notably, *com.google.firebase* appears in 25 APKs. Table VII summarizes the distribution of SDKs and their adoption across APKs. In total, 27 of the 28 crypto wallets use third-party SDKs for data analysis, posing risks of user information disclosure. Additionally, Category C#3 includes a facial recognition SDK, *com.facetec*, which is used by Numio and ZenGo. Another notable SDK is *io.sentry* (Sentry), a bug-reporting tool adopted by eight prominent crypto wallets, together accounting for over 11 million downloads. Sentry offers customization options, and the aforementioned Slope incident was a result of misconfiguration.

C. Security Risks in DApps

This section analyzes the detection results of DApps, which reveal extensive security risks. Among Web3 sites, 37.2% (29/78) have anonymity loss (SR#1), and 100% have risky third-party scripts (SR#5). Table X presents SR#6 to SR#7 related to smart contracts, showing that 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.

To SR#1: Anonymity Loss: Fig. 6(a) shows the distribution of login methods among 78 DApps. Specifically, 12 (15.4%) DApps support only social login, while 17 (21.8%) support both wallet-based and social login. In practice, many users still choose social login due to their familiarity with Web2 habits, while only experienced users prefer wallet-based login. This tendency undermines anonymity and leaves a considerable proportion of users exposed to privacy risks. Further investigation revealed that Google Sign-In is the most widely supported social login method, with 24 out of 29 DApps supporting it. Also, the DApp *qorpo.world* mandates users to link their wallet addresses with their email, significantly compromising user anonymity.

To SR#5: Third-Party Scripts: In our study, we found that across 78 DApps, third-party scripts sent requests to a total of 268 external sites. Additionally, 35 (44.9%) DApps leaked users' wallet addresses to 59 third-party sites. The monthly UAW and transaction volume of the 35 DApps are more than 11 million and $125 billion, respectively. Fig. 6(b) depicts the distribution of DApps across five categories. Notably, a single DApp could fall into more than one category. We can see 70.6% of DeFi and 62.5% of DEX DAppsexpos user wallet addresses to third-party sites, directly contradicting their claims of decentralization, privacy, and security. Most DApps only leaked wallet addresses to one or two sites, mainly to RPC providers. However, *app.rhino.fi*, a popular DeFi platform with over 320 K monthly UAW and more than $398 million in monthly transaction volume, exposed user addresses to as many as 23 third-party sites, including analytics services like *heapanalytics.com*. Table IX reveals the top 10 third-party sites embedded in DApps scripts,

TABLE X
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 XI
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	80.7%	38.3%	6.9%	0.6%	43.0%	68.3%
ERC721	97.8%	8.4%	N/A	70.1%	N/A	87.6%
ERC1155	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	88.0%	25.5%	N/A	29.9%	N/A	76.3%

ERC20 is FT, ERC721 and ERC1155 are NFT.

TABLE XII
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**%)

with Google-related sites dominating the list. This indicates that a significant number of DApps still rely on centralized services.

To SR#6: Overpowered Owner: As shown in Table X, 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 are at 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. 70.4% (47,635/67,645) of tokens with overpowered owners have at least one risk of SR#6.a $\sim$ d. Table XI 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 XI shows that 25.5% of tokens and 38.3% of ERC20 tokens have this risk. As mentioned before, owners use `block` and `paused` to limit liquidity for security purposes. 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 `block`, and 96.8% of NFTs have `paused`.

To SR#6.b: Vulnerable Scarcity: We only detected vulnerable scarcity only in ERC20 and ERC1155 tokens, as ERC721 tokens lack `totalSupply`. Since investors often value scarcity, most contracts set limited total supplies. Nevertheless, 3,044 contracts still exhibit vulnerable scarcity risks, as shown in Table X, indicating an increased likelihood of token depreciation from additional issuance. For example, RESERVE[3] is an ERC20 token with a market value exceeding $28M. Although it sets a *maxSupply* to limit the *totalSupply*, the contract also includes a *changeMaxSupply()* function that allows the owner to modify this limit. Another ERC1155 contract, VERSE WORKS,[4] does not limit the *totalSupply*, granting the owner the autonomy to mint new NFTs at their discretion. Our tool NAGA effectively identified and reported these issues.

To SR#6.c: Mutable Metadata: Table XI shows that 67.5% of NFTs have mutable metadata, while only 0.6% of ERC20 tokens. Since most NFTs store users' digital assets on centralized servers, owners need to retain the ability to change `baseURI`, and NFTs with mysteryboxes 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 ERC20 tokens 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 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 XII 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.

D. Effectiveness and Efficiency Analysis

In this section, we analyze the effectiveness of LLM in SDKs detection and evaluate the performance of Naga.

Effectiveness of LLM: In our experiments, the LLM-based detection method exhibited superior performance compared to the traditional keyword matching method [29]. When we set

[3]RESERVE: 0x196f4727526ea7fb1e17b2071b3d8eaa38486988
[4]VERSE WORKS: 0xec43e92046c1527586dfaf02031622c30af9a1d6

TABLE XIII
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

the minimum number of keywords for a match to three, the keyword matching method identified 29 correct SDKs and six incorrect ones. In contrast, GPT-4 identified 39 correct SDKs with zero errors. Moreover, even though we prioritized English web pages during our web scraping process by setting the request headers accordingly, some websites still return content in other languages. GPT-4 can automatically handle various languages. For instance, GPT-4 successfully processed the Japanese content of the website *zendesk.com*, determining that it offers third-party services, while the keyword method failed in this scenario.

FP & FN Analysis: We randomly selected 100 token contracts to test the effectiveness of Naga. Table XIII shows the number of correct (TP), incorrect (FP), and missed (FN) state variables detected by Naga compared to the results obtained manually. Overall, Naga demonstrates high accuracy and few false positives. It detected 96.5% (715/741) of variables about overpowered owner (SR#6, 6.a $\sim$ 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 owners of SR#6. D#4 and D#5 missed some variables because their names are uncommon, and some contracts violate the specification, such as using `if` statement instead of `require` statement, which led to D#6 missing them. Naga directly identifies events by IRs of contract code. Hence, FP and FN are not present in SR#7.

Cost and Robustness: NAGA is a highly efficient tool, with an average cost of about 2.11 seconds per contract, of which SLITHER parsing takes 2.08 seconds, and NAGA analyzing takes less than 0.024 seconds. We examined 110,506 contracts in the experiment. Of these, 1,834 failed to compile due to source file errors, ten contracts cannot find entry points, and only one encountered an analysis error. Notably, no contract resulted in a timeout during this process. Overall, NAGA demonstrates strong robustness.

Benefit of DAE: Among all contracts, there were 6,533,826 internal calls and 1,446,446 external calls. On average, this amounts to 72 calls per contract. Notably, 99.18% (109,595 out of 110,506) of the contracts had at least one internal or external call. Using DAE, NAGA was able to resolve every call it encountered. Consider NIFTYWALLS[5], an ERC721 NFT with a floor price of 0.1 ETH (approximately $158). It relies on an external contract for metadata management and uses an internal call to retrieve the *baseURI* from the external contract. Our DAE effectively traces both external and internal calls, pinpointing the metadata *baseURI* with precision.

[5]NIFTYWALLS: 0x1718af1613d86e21488a9d985a586ae9c5dbd62e

VI. DISCUSSION

This section discusses the decentralized design and provides suggestions to mitigate these risks.

A. Decentralized Design

Decentralized ecosystems aim to utilize the distributed nodes of blockchain to reduce single points of failure and prevent the concentration of power. In decentralized services, the key lies in which entities can access users' data and privacy. An ideal decentralized model should ensure that user data is stored locally, and servers can only access user data in a de-identified form, while third parties cannot obtain non-anonymous user data. In practice, however, this is unrealistic, For example, almost all Web3 sites and crypto wallets need to query RPC services to obtain data (such as address balances). However, we can note that sending requests to the RPC service directly on the user's device will expose the user's public key, IP address, and device information, thereby compromising anonymity. A safer approach is to have the server of the Web3 site or wallet query the RPC service to avoid leaking the user's privacy. Similarly, DApps and wallets should anonymize user identities when utilizing third-party services like data analytics, instead of directly using the user's public key address for identification. However, our detection results show that many DApps and wallets still rely on third-party SDKs and scripts provided by traditional Internet giants such as Google, and do not anonymize users, which is inconsistent with the principle of decentralization contradictory and raise concerns about the safety of our ecosystem.

B. Mitigation

In the decentralized ecosystem, users need to better understand decentralized services and take part in governance. Developers are crucial in maintaining this ecosystem and should promote decentralized solutions to enhance safety and privacy. For seven security risks, this section will propose mitigation measures from both the user and developer perspectives.

For Users: Users should be cautious and avoid sharing PII with wallets or DApps (SR#1). To protect their digital identities, users should consider using privacy-enhancing technologies like VPNs and Tor. Running their own blockchain node, rather than relying on RPC services provided by wallets, can also increase their control and security (SR#4). Furthermore, users need to be aware of potential risks and utilize tools like Naga to identify contracts' *overpowered owners* (SR#6).

For Developers: Developers should give priority to decentralized login methods, like Web3 authentication, as they provide better security and privacy than traditional login methods (SR#1). They must follow development standards and clearly inform users about the permissions involved (SR#3, SR#5, SR#7). Developers should also consider allowing wallets to connect to multiple RPC services, which can mitigate the risks associated with the failure or compromise of a single service (SR#4). Additionally, implementing access control with multi-signatures can decentralize ownership and mitigate the risks arising from the leakage of a single private key (SR#6).

VII. Related Work

Previous works were limited in security risks of decentralized ecosystems, but our work focuses more on security risks caused by centralization.

Horus [34] 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. Houy et al. [55] provide a structured review of vulnerabilities in crypto wallets. They provide a high-level overview of crypto wallet security issues, while our work is from a more specific, centralized perspective, so our work is complementary. Winter et al. [35] measured 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 or phish users. Li et al. [7] 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.

Prior studies in distributed systems and network security have examined security issues from multiple perspectives. HIDIM [56] addresses class imbalance in intrusion detection through hierarchical dependency modeling. Sun et al. [57] studied cross-domain service function chain orchestration with a full-mesh aggregation approach. The resilient formation tracking method [58] enables swarm systems to withstand deception attacks via anomaly estimation and secure control. RFL − APIA [59] mitigates poisoning attacks in IIoT federated learning with fuzzy-rule-based detection and adaptive aggregation, while the smart contract-based incentive mechanism [60] ensures fair reward distribution in AIoT federated learning.

Using the large language model (LLM) to process human-readable documents such as protocols, code, etc. is becoming increasingly popular, and this approach has shown better results than traditional methods [61], [62], [63], [64]. Xia et al. [64] perform the first extensive study on directly applying LLMs for automated program repair. TITANFUZZ [65] used an LLM to automatically generate test cases for Deep Learning software libraries. Meng et al. [66] utilized the LLMs to guide protocol fuzzing; the results show that the LLMs can help the fuzzer find more vulnerabilities. Our work is the first to utilize the LLM to classify SDKs in Android applications.

Static analysis tools mainly use formal verification and symbolic execution to detect contracts, such as ZEUS [48], VERISMART [49], OSIRIS [67], Securify [50]. Dynamic analysis tools rely on fuzzers and SMT solvers, such as ContractFuzzer [68], ReGuard [69], Sereum [9], Oyente [39], ConFuzzius [24]. However, these existing tools all focus on contract vulnerabilities, and none of them directly support detecting *overpowered owner*. TokenScope [70] 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.

VIII. Conclusion

The primary motivation for users to adopt decentralized services is to circumvent the security risks associated with centralization. This work takes the first step in exploring the centralized security risks within decentralized ecosystems. We focus on two predominant decentralized services: crypto wallets and DApps, and highlight seven previously overlooked security risks. We propose a series of effective methods to detect these security risks, including a semi-automated tool and two automated tools. Our findings underscore the widespread nature of centralized security risks; specifically, 96.4% of crypto wallets, 100% Web3 sites and 83.5% contracts exhibit security risks. We propose potential mitigation strategies for users and developers, but we believe that building a truly decentralized ecosystem will require a collaborative effort across the entire community. For future work, we will expand our detection of centralized risks to a broader range of decentralized components, such as consensus mechanisms, node distributions, and cross-chain bridges.

References

[1] Z. Liu et al., "Make Web3.0 connected," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 5, pp. 2965–2981, Sep./Oct. 2022.

[2] L. Zhou et al., "SoK: Decentralized finance (DeFi) attacks," in *Proc. 44th IEEE Symp. Secur. Privacy*, 2023, pp. 2444–2461.

[3] Z. Li et al., "Demystifying DeFi MEV activities in flashbots bundle," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 165–179.

[4] P. Daian et al., "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 910–927.

[5] D. Das, P. Bose, N. Ruaro, C. Kruegel, and G. Vigna, "Understanding security issues in the NFT ecosystem," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 667–681.

[6] B. White, A. Mahanti, and K. Passi, "Characterizing the OpenSea NFT marketplace," in *Proc. Companion Web Conf.*2022, 2022, pp. 488–496.

[7] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain DApps at RPC service," in *Proc. 28th Annu. Netw. Distrib. System Secur. Symp.*, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_3C-1_23108_paper.pdf

[8] Y. Khatri, "Ethereum infrastructure provider infura is down, crypto exchanges begin to disable ETH withdrawals," Nov. 11, 2020, Accessed: Mar. 28, 2024. [Online]. Available: https://www.theblock.co/post/84232/ethereum-infrastructure-provider-infura-is-down

[9] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in *Proc. 26th Annu. Netw. Distrib. System Secur. Symp.*, 2019. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_09-3_Rodler_paper.pdf

[10] C. Staff, "What was the DAO?," Mar. 17, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://www.gemini.com/cryptopedia/the-dao-hack-makerdao

[11] E. Tan, "Solana's $6M exploit likely tied to slope wallet, developers say," Aug. 4, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://www.coindesk.com/business/2022/08/03/solanas-latest-6m-exploit-likely-tied-to-slope-wallet-devs-say/

[12] X. Li, A. Yepuri, and N. Nikiforakis, "Double and nothing: Understanding and detecting cryptocurrency giveaway scams," in *Proc. 30th Annu. Netw. Distrib. System Secur. Symp.*, 2023. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_f584_paper.pdf

[13] I. Abraham and G. Stern, "New Dolev-Reischuk lower bounds meet blockchain eclipse attacks," in *Proc. 26th Int. Conf. Principles Distrib. Syst.*, 2023, pp. 16:1–16:18.

[14] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 375–392.

[15] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *Proc. 1st IEEE Eur. Symp. Secur. Privacy*, 2016, pp. 305–320.

[16] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 104–121.

[17] S. Long, S. Basu, and E. G. Sirer, "Measuring miner decentralization in proof-of-work blockchains," 2022, *arXiv:2203.16058*.

[18] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against Bitcoin peer-to-peer network," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 894–909.

[19] S. Smolka et al., "Fuzz on the beach: Fuzzing solana smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 1197–1211.

[20] K. Babel, M. Javaheripi, Y. Ji, M. Kelkar, F. Koushanfar, and A. Juels, "Lanturn: Measuring economic security of smart contracts through adaptive learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 1212–1226.

[21] K. Babel, P. Daian, M. Kelkar, and A. Juels, "Clockwork finance: Automated analysis of economic security in smart contracts," in *Proc. 44th IEEE Symp. Secur. Privacy*, 2023, pp. 2499–2516.

[22] C. Sendner et al., "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning," in *Proc. 30th Annu. Netw. Distrib. System Secur. Symp.*, 2023. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2023/02/ndss2023_s263_paper.pdf

[23] B. He et al., "TxPhishScope: Towards detecting and understanding transaction-based phishing on ethereum," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 120–134.

[24] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, "ConFuzzius: A data dependency-aware hybrid fuzzer for smart contracts," in *Proc. 6th IEEE Eur. Symp. Secur. Privacy*, 2021, pp. 103–119.

[25] S. So, S. Hong, and H. Oh, "SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 1361–1378.

[26] L. Su et al., "Evil under the sun: Understanding and discovering attacks on ethereum decentralized applications," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 1307–1324.

[27] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection," in*Proc. 29th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2020, pp. 415–427.

[28] J. Krupp and C. Rossow, "teEther: Gnawing at ethereum to automatically exploit smart contracts," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1317–1333.

[29] K. Yan, J. Zhang, X. Liu, W. Diao, and S. Guo, "Bad apples: Understanding the centralized security risks in decentralized ecosystems," in *Proc. ACM Web Conf.*, 2023, pp. 2274–2283.

[30] V. Buterin, "A next-generation smart contract and decentralized application platform," 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://ethereum.org/en/whitepaper/

[31] "Metamask: A crypto wallet & gateway to blockchain apps," Accessed: Mar. 28, 2024. [Online]. Available: https://metamask.io/

[32] "Infura," Accessed: Mar. 28, 2024. [Online]. Available: https://www.infura.io/

[33] "Alchemy: The Web3 development platform," Accessed: Mar. 28, 2024. [Online]. Available: https://www.alchemy.com/

[34] M. S. Uddin, M. Mannan, and A. M. Youssef, "Horus: A security assessment framework for android crypto wallets," in *Proc. 17th EAI Int. Conf. Secur. Privacy Commun. Netw.*, 2021, pp. 120–139.

[35] P. Winter, A. H. Lorimer, P. Snyder, and B. Livshits, "What's in your wallet privacy and security issues in web 3.0," 2021, *arXiv: 2109.06836*.

[36] L. Abrams, "Fake trezor data breach emails used to steal cryptocurrency wallets," Apr. 3, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://www.bleepingcomputer.com/news/security/fake-trezor-data-breach-emails-used-to-steal-cryptocurrency-wallets/

[37] M. Young, "75 crypto exchanges have closed down so far in 2020," Oct. 7, 2020, Accessed: Mar. 28, 2024. [Online]. Available: https://cointelegraph.com/news/75-crypto-exchanges-have-closed-down-so-far-in-2020

[38] S. Wan, "Celsius network continues to make moves, prompting calls to resume withdrawals," Jul. 7, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://cryptoslate.com/celsius-network-continues-to-make-moves-prompting-calls-to-resume-withdrawals/

[39] L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269.

[40] J. Benson, "Ethereum's infura cuts off users to separatist areas in Ukraine, accidentally blocks Venezuela," Mar. 4, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://decrypt.co/94315/ethereum-infura-cuts-off-users-separatist-areas-ukraine-accidentally-blocks-venezuela

[41] OpenZeppelin, "Openzeppelin contracts," Oct. 5, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts

[42] P. Paganini, "KICKICO security breach–hackers stole over $7.7 million worth of KICK Tokens," Jul. 30, 2018, Accessed: Mar. 28, 2024. [Online]. Available: https://securityaffairs.co/wordpress/74910/hacking/kickico-hack.html

[43] B. Aleks-blockchaincap et al., "YIP-57: Funding yearn's future," Jan. 21, 2021, Accessed: Mar. 28, 2024. [Online]. Available: https://gov.yearn.finance/t/yip-57-funding-yearns-future/9319

[44] "Find a wallet," Oct. 5, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://ethereum.org/en/wallets/find-wallet/

[45] A. Lekssays, H. Mouhcine, K. Tran, T. Yu, and I. Khalil, "LLMxCPG: Context-aware vulnerability detection through code property graph-guided large language models," in *Proc. 34th USENIX Conf. Secur. Symp.*, 2025, pp. 489–507.

[46] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proc. 31st Annu. Netw. Distrib. System Secur. Symp.*, 2024. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

[47] K. Yan, X. Zhang, and W. Diao, "Stealing trust: Unraveling blind message attacks in Web3 authentication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2024, pp. 555–569.

[48] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. 25th Annu. Netw. Distrib. System Secur. Symp.*, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_09-1_Kalra_paper.pdf

[49] S. So, M. Lee, J. Park, H. Lee, and H. Oh, "VERISMART: A highly precise safety verifier for ethereum smart contracts," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1678–1694.

[50] P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. T. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 67–82.

[51] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain*, 2019, pp. 8–15.

[52] "DappRadar," Accessed: Mar. 28, 2024. [Online]. Available: https://dappradar.com/rankings

[53] Etherscan, "Etherscan," Accessed: Mar. 28, 2024. [Online]. Available: http://etherscan.io/

[54] M. Ortner and S. Eskandari, "Smart contract sanctuary," Jul. 1, 2022, Accessed: Mar. 28, 2024. [Online]. Available: https://github.com/tintinweb/smart-contract-sanctuary

[55] S. Houy, P. Schmid, and A. Bartel, "Security aspects of cryptocurrency wallets - a systematic literature review," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 4:1–4:31, 2024.

[56] W. Zhou et al., "HIDIM: A novel framework of network intrusion detection for hierarchical dependency and class imbalance," *Comput. Secur.*, vol. 148, 2025, Art. no. 104155.

[57] G. Sun, Y. Li, D. Liao, and V. Chang, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 3, pp. 1175–1191, 2018.

[58] Y. Liu, W. Li, X. Dong, and Z. Ren, "Resilient formation tracking for networked swarm systems under malicious data deception attacks," *Int. J. Robust Nonlinear Control*, vol. 35, no. 6, pp. 2043–2052, 2025.

[59] C. Li, A. He, G. Liu, Y. Wen, A. T. Chronopoulos, and A. Giannakos, "RFL-APIA: A comprehensive framework for mitigating poisoning attacks and promoting model aggregation in IIoT federated learning," *IEEE Trans. Ind. Informat.*, vol. 20, no. 11, pp. 12935–12944, Nov. 2024.

[60] G. Xu et al., "A model value transfer incentive mechanism for federated learning with smart contracts in AIoT," *IEEE Internet Things J.*, vol. 12, no. 3, pp. 2530–2544, Feb. 2025.

[61] T. B. Brown et al., "Language models are few-shot learners," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.

[62] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program.*, 2022, pp. 1–10.

[63] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "CodaMosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *Proc. 45th IEEE/ACM Int. Conf. Softw. Eng.*, 2023, pp. 919–931.

[64] C. S. Xia, Y. Wei, and L. Zhang, "Automated program repair in the era of large pre-trained language models," in *Proc. 45th IEEE/ACM Int. Conf. Softw. Eng.*, 2023, pp. 1482–1494.

[65] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proc. 32nd ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2023, pp. 423–435.
[66] J. Ackerman and G. Cybenko, "Large language models for fuzzing parsers (registered report)," in *Proc. 2nd Int. Fuzzing Workshop*, 2023, pp. 31–38.
[67] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 664–676.
[68] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, 2018, pp. 259–269.
[69] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," in *Proc. 40th Int. Conf. Softw. Engineering: Companion*, 2018, pp. 65–68.
[70] T. Chen et al., "TokenScope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1503–1520.

Kailun Yan received the PhD degree in cyber science and technology from Shandong University, Qingdao, China, in 2025. He is currently a postdoctoral fellow with the Institute for Advanced Study, Tsinghua University, Beijing, China. His research interests include blockchain security, Web3 security, and smart contract analysis.

Jilian Zhang received the PhD degree from Singapore Management University, Singapore, in 2014. He is currently an associate professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include data management, information security, and machine learning.

Xiangyu Liu received the PhD degree from the Chinese University of Hong Kong in 2016. He is currently a staff algorithm engineer with the Security Department of Alibaba Group. His research interests include cyber security, artificial intelligence, and deep learning.

Wenrui Diao received the PhD degree in information engineering from the Chinese University of Hong Kong, in 2017, under the supervision of Prof. Kehuan Zhang. He is currently a professor with the School of Cyber Science and Technology, Shandong University, Qingdao, China. His research interests include mobile security and IoT security. He received the 2019 ACM SIGSAC China Rising Star Award.