Penny Wise and Pound Foolish: Quantifying the Risk of Unlimited Approval of ERC20 Tokens on Ethereum

Dabao Wang
Monash University
Melbourne, Australia
dabao.wang@monash.edu

Hang Feng
Zhejiang University
Hangzhou, China
feng.h@zju.edu.cn

Siwei Wu
Zhejiang University
Hangzhou, China
wusw1020@zju.edu.cn

Yajin Zhou
Zhejiang University
Hangzhou, China
yajin_zhou@zju.edu.cn

Lei Wu*
Zhejiang University
Hangzhou, China
lei_wu@zju.edu.cn

Xingliang Yuan
Monash University
Melbourne, Australia
xingliang.yuan@monash.edu

ABSTRACT

The prosperity of decentralized finance motivates many investors to profit via trading their crypto assets on decentralized applications (DApps for short) of the Ethereum ecosystem. Apart from Ether (the native cryptocurrency of Ethereum), many ERC20 (a widely used token standard on Ethereum) tokens obtain vast market value in the ecosystem. Specifically, the approval mechanism is used to delegate the privilege of spending users' tokens to DApps. By doing so, the DApps can transfer these tokens to arbitrary receivers on behalf of the users. To increase the usability, *unlimited approval* is commonly adopted by DApps to reduce the required interaction between them and their users. However, as shown in existing security incidents, this mechanism can be abused to steal users' tokens.

In this paper, we present the first systematic study to quantify the risk of unlimited approval of ERC20 tokens on Ethereum. Specifically, by evaluating existing transactions up to 31st July 2021, we find that unlimited approval is prevalent (60%, 15.2M/25.4M) in the ecosystem, and 22% of users have a high risk of their approved tokens for stealing. After that, we investigate the security issues that are involved in interacting with the UIs of 22 representative DApps and 9 famous wallets to prepare the approval transactions. The result reveals the worrisome fact that all DApps request unlimited approval from the front-end users and only 10% (3/31) of UIs provide explanatory information for the approval mechanism. Meanwhile, only 16% (5/31) of UIs allow users to modify their approval amounts. Finally, we take a further step to characterize the user behavior into five modes and formalize the good practice, i.e., *on-demand approval and timely spending*, towards securely spending approved tokens. However, the evaluation result suggests that only 0.2% of users follow the good practice to mitigate the risk. Our study sheds light on the risk of unlimited approval and provides

suggestions to secure the approval mechanism of the ERC20 tokens on Ethereum.

CCS CONCEPTS

• **Security and privacy** → **Software and application security**.

KEYWORDS

Decentralized Finance, Ethereum, ERC20, Unlimited Approval

1 INTRODUCTION

As a blockchain-based financial system, Decentralized Finance (DeFi) has been blooming in recent years. The prosperous development of DeFi brings a rapid growth of decentralized applications (DApps). Many investors are attracted to profit via trading their crypto assets on DApps of the Ethereum ecosystem. Apart from Ether (the native token on Ethereum), the ERC20 [1] token standard [18] allows users to use their tokens in other arbitrary contracts. Due to the convenience, ERC20 tokens rapidly become popular crypto assets that help boost the prosperity of DApps. Specifically, the ERC20 token standard introduces an approval mechanism enabling users to delegate the privilege to DApps to spend users' tokens. In particular, users first grant the permission of their tokens to DApps with a transaction (i.e., *approval transaction*). Then on behalf of users, DApps can launch another transaction to transfer users' approved tokens to provide the requested service.

As such, the front-end users need to construct at least two transactions to spend their ERC20 tokens, including the approval transaction and another transaction to transfer tokens. Specifically, the front-end users will directly interact with DApps' and wallets' UIs to construct these transactions. To minimize the cost of repeatedly sending approval transactions, *unlimited approval* is used to grant the privilege of using an unlimited number of users' tokens to DApps.

---

*Corresponding Author: Lei Wu (lei_wu@zju.edu.cn).

---

[1]ERC20 is the most widely used token standard on Ethereum.

Despite that unlimited approval can help users save money (gas fee) from repeatedly sending approval transactions, it can also be *abused* to steal users' tokens. For example, malicious DApps [29, 33] may elaborately trick the users into granting the token approvals, and surreptitiously transfer those approved tokens (e.g., through backdoor functions). Moreover, many recent incidents [30, 36, 41] reveal that hackers can steal users' approved tokens (worth over $180M$) by exploiting vulnerabilities laid in the legitimate DApps. On the other hand, this situation is exacerbated by the strategy adopted by those popular DeFi DApps with a considerable TVL [2] (e.g., Uniswap [45]), and crypto wallets with a large number of users (e.g., Trust Wallet [46]). Specifically, the DApps tend to use unlimited approval as the default settings on their UIs to interact with the users, while the wallets usually (if not always) keep these settings for the interactions. Furthermore, both of them do not provide the functionality for users to adjust the approval amounts.

As a result, there is an urgent need to understand the risk of unlimited approval and mitigate attacks against the approval mechanism. Although there exist several studies focusing on the improper implementation [14] and use (or even abuse) [15, 22, 38, 51] of the ERC20 token, to the best of our knowledge, there is still a lack of study to systematically and comprehensively reveal and measure the risk of unlimited approval in the ecosystem, including: 1) *to what extent unlimited approval is abused and how risky of users' approved tokens?* 2) *what security issues are involved in interacting with DApps and wallets to prepare approval transactions?* 3) *What is the current status of user behaviors and how do users achieve good practice to spend approved tokens towards mitigating the risks?*

In this paper, we present the first systematic study of unlimited approval of ERC20 tokens. By analyzing all transactions (before 31st July 2021), we identify $25.4M$ approval transactions. Then, we analyze the distribution and growth trend of unlimited approval transactions, as well as the sender, spender, and token participating in the approval transactions. Furthermore, we define *RiskAmount* and *RiskLevel* to measure the potential risk of approved tokens (**Section 5**). To gain an in-depth understanding of the abuse of unlimited approval, we investigate 22 DApps and 16 wallets from two aspects: *Interpretability* and *Flexibility*. *Interpretability* indicates the status of explanatory information that the UIs provide for users to understand the risk of unlimited approval, while *Flexibility* indicates whether the UIs provide the modification feature for users to adjust the approval amount. By doing so, our investigation can disclose whether DApps and wallets guide users to securely construct approval transactions (**Section 6**). Finally, to further understand the user behavior of spending approved tokens, we first detect users' temporal sequence of approving and spending actions, as well as characterize user behaviors into five modes. Then, we analyze the distribution of the user behavior that follows the good practice of using unlimited approval (**Section 7**).

To sum up, this study reveals a number of worrisome facts about unlimited approval:

- **Unlimited approval is prevalent (60%, 15.2M/25.4M) on Ethereum and 22% of users have a high risk of their approved tokens for stealing.** We identify $25.4M$ approval transactions and 60% of them are unlimited approval. Meanwhile, 60%

of unique users participate in the unlimited approval transactions. The evaluation result of the *RiskAmount* and *RiskLevel* on the top three tokens (i.e., USDC, USDT, and DAI) suggests that 22% of users are threatened by token stealing with a high risk.

- **Most DApps and wallets do not provide comprehensive understandings (around 90%) and flexibility (84%) for users to mitigate the risk of unlimited approval.** We discover that all selected DApps (22) request unlimited approval from users. However, only 10% DApps and wallets (i.e., 9% (2/22) and 11% (1/9), respectively) provide explanatory information on the approval mechanism. Moreover, only 16% (5/31) of UIs enable users to adjust the approval amounts. Surprisingly, we also find two DApps (i.e., Curve Finance [19] and Yearn Finance [20]) mislead users to construct unlimited approval transactions.

- **Only 0.2% of user behaviors follow the good practice to spend approved tokens.** After the in-depth investigation of users' transactions regarding the approval mechanism, we characterize the user behavior into five modes and formalize the good practice of spending approved tokens. Through this, we further measure the number of user behaviors following the good practice. The result suggests that 76% of user behaviors follow modes 1 and 2, and 99% of their user behaviors are using unlimited approval. However, only 0.2% (2,475/1,496,886) of detected user behaviors securely spend approved tokens, which suggests that most users are not aware of the risk.

2 BACKGROUND

2.1 Ethereum Primitives

Accounts.   Ethereum has two types of accounts: External Owned Account (EOA) and Contract Account (CA). EOAs are controlled by private keys owned by users. In contrast, CAs are controlled by smart contracts, which are known as snippets of JavaScript-like code. To create a CA, users need to send a signed transaction to deploy their smart contracts on the Ethereum blockchain.

Transactions.   Ethereum blockchain is a state machine that can be altered by validated transactions. An EOA can initiate a transaction based on different purposes such as transferring Ether, invoking a function of a smart contract or deploying a smart contract. However, during the execution of a transaction, a transaction can trigger more transactions by invoking functions of other smart contracts. Therefore, to distinguish them, we call transactions initiated by EOAs as external transactions and transactions triggered within an external transaction as internal transactions. In Section 5, the word 'transaction' indicates the external transaction.

Smart Contract.   A smart contract on Ethereum is an account controlled by an immutable program (with many executable functions) written in a high-level language (i.e., Solidity). To execute the smart contract, one can send transactions by invoking the function of the smart contract. A function of an Ethereum smart contract can be identified by the function signature. The function signature contains the first four bytes of the hash value (SHA3) of the function name with the parenthesized list of parameter types. Initiating a transaction by invoking a function of a smart contract requires a user to provide a function signature with corresponding parameters in the call data section. Then, the transaction will be executed based on the logic of the smart contract.

---

[2]I.e., Total Locked Value.

2.2 Decentralized Applications.

The decentralized application (DApp) is an application running in the Ethereum system with accessible and transparent smart contract(s). In addition, the DApp normally builds a user interface (e.g., website) for front-end users to use its services. In the following, we introduce the high-level ideas of two popular types of DApps to help understand this work.

Decentralized Exchanges (DEXes). Unlike centralized exchanges (CEXes), decentralized exchanges (DEXes) are exchanges without any centralized authority. Instead, users are allowed to exchange their cryptocurrencies with full control of their capital. Moreover, there are two main categories of DEXes: automated market maker and order book.

Lending Platforms. Lending platforms in DeFi offer loans of cryptocurrencies for users without any intermediary. Borrowers can directly take loans on the lending platforms by paying interest periodically. Moreover, most lending platforms have their liquidation mechanism (with different collateral ratios) to protect their loaned assets from serious price slippage of borrowers' collateral.

2.3 Wallets

In the cryptocurrency ecosystem, the users maintain a pair (or multiple pairs) of public and private keys to claim cryptocurrency ownership. Specifically, the users use public keys to receive cryptocurrencies and private keys to transfer their cryptocurrencies. In this context, the wallets function like a bank account to manage users' cryptocurrencies by storing their key pairs safely. The purpose of a secure wallet is to prevent users' private keys from leaking. There are two main types of wallets: software and hardware wallets. The *software* wallets are designed as software available in three main formats (e.g., Mobile, Desktop, and Web-based Extension). Users' information (e.g., private keys) is regularly stored online or on local devices. The *hardware* wallets leverage some hardware devices (e.g., hard drive) to store users' information offline.

2.4 Approval Mechanism of ERC20 Tokens

To understand the approval mechanism of ERC20 tokens, we elaborate on the details of two variables and two functions implemented in the ERC20 token contract regarding the approval mechanism:

- `mapping(address=>uint)` **balanceOf**: `balanceOf` is a mapping list recording the amount of tokens owned by users.
- `mapping(address=>mapping(address=>uint))` **allowance**: `allowance` is a mapping list recording the amount of tokens approved by users to spenders.
- **approve**(`address _spender, uint256 _amount`): `approve` is the function executed by accounts (i.e., CA or EOA) to grant the permission of the tokens to indicated spenders. The approval transaction will update the variable `allowance`.
- **transferFrom**(`address _sender, address _receiver, uint256 _amount`): `transferFrom` is the function executed by approval recipients to transfer tokens. The transaction including the invocation of `transferFrom` will update both `allowance` and `balanceOf`.

Apart from the native coin (Ether) on Ethereum, various ERC20 tokens circulate in the Ethereum ecosystem and have obtained a great market value (e.g., USDC [13] and USDT [3]). The ERC20

token standard, introduced in 2015, is one of the most popular token standards. Different from Ether, there are two ways to spend ERC20 tokens. First, the user can directly transfer their ERC20 tokens to other accounts by invoking the `transfer` function. Second, according to the approval mechanism, the user can delegate their ERC20 tokens [3] to other accounts by invoking the `approve` function [4]. Once the user's approval transaction is processed, the accounts that received approval can spend the user's approved tokens by invoking `transferFrom` function [5]. Note that spending ERC20 tokens on DApps normally requires users to send approval and execution transactions sequentially by interacting with DApps and wallets (see Section 6.1). However, if the user's approved tokens are used out by the DApp, the user has to construct an approval transaction again before requesting services from the DApp. Therefore, users often construct unlimited approval transactions to save money from repeatedly sending approval transactions with limited approval amounts. To explore the good practice of spending approved tokens, the user behaviors (e.g., how do users specify the approval amounts and construct the corresponding transactions) and their distribution need to be characterized and analyzed (see Section 7).

3 ATTACK MODELS AND MOTIVATING EXAMPLES

In this section, we will elaborate on the high-level ideas of some infamous attacks to understand the risk of the approved ERC20 tokens. Specifically, our investigation suggests that users' approved tokens might be stolen in two ways: 1) approving tokens to malicious or hijacked DApps, and 2) approving tokens to vulnerable DApps. Accordingly, the related security incidents can be classified into two attack models (as shown in Figure 1). For each model, we will first describe the attack flow, then demonstrate it with a concrete example, and finally summarize the representative attacks in recent years. Moreover, we list all incidents and summarize the loss in Table 1.
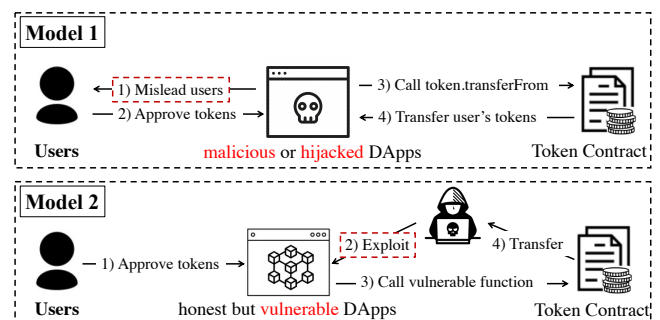

Figure 1: The attack models of stealing users' approved tokens.

---

[3]Named as **approved tokens** in this paper.
[4]Named as **approval transactions** in this paper.
[5]Named as **execution transactions** in this paper.

3.1 Attack Model 1

Attack model 1 is mainly related to the front-end of DApps. The victims are first misled to approve their tokens to the malicious or hijacked DApp. Then, with users' token approvals, the malicious or hijacked DApp can further invoke `transferFrom` of ERC20 token contracts to steal users' approved tokens.

A Concrete Example: In September 2020, a DApp called Uni-Cat [33] stole over $140, 000 worthy of UNI [44] tokens from users. Initially, UniCat pretended as a yield farming DApp promising to distribute great profits based on users' deposited UNI tokens. With attractive advertisements, UniCat induced many users to approve UNI tokens to their accounts. After receiving token approvals from many users, UniCat triggered their backdoor function `setGovernance` to steal all users' tokens. Most importantly, even if the user did not deposit all his/her UNI tokens in UniCat, with unlimited approval, UniCat can still steal all of the user's UNI tokens by internally triggering `transferFrom` with the backdoor function.

Other Representative Attacks: On 28th August 2020, Nik Kunkel [29] revealed a double approval exploit launched by the project called Degen Money. The approach used by Degen Money is more straightforward than what was done in the Unicat incident. Concretely, the UI provided by Degen Money intentionally tricked users to send two approval transactions to a functioning contract and a different address, where the latter stole users' tokens. On 2nd December 2021, the front-end of Badger was manipulated by a malicious party [11]. As a result, users who interacted with Badger's UI automatically sent approval transactions to delegate their tokens to the injected addresses, which led to a loss of 120M USD.

3.2 Attack Model 2

In attack model 2, users lost their approved tokens by sending token approvals to an honest but vulnerable DApp. Specifically, the attacker first detected the vulnerability of the DApp's smart contracts (i.e., lack of access control). Then, the attacker internally triggered the `transferFrom` function using DApp's identity to transfer users' approved tokens.

A Concrete Example: On 27th Feb 2021, a DApp called Furucombo experienced a serious hack [21]. The root cause is that Furucombo forgot to initialize the implementation contract address while immigrating the flash loan [37, 47] service from AAVE [6]. Therefore, a hacker had the chance to take over the place and injected his malicious contract to gain access control of users' approved assets. As a result, the hacker stole more than 15M$ worthy tokens that had been approved to Furucombo by users.

Other Representative Attacks: On 22nd Feb 2020, Primitive Finance [36], which is an option market protocol, launched a set of white-hat attacks to preserve over 10M$ worthy tokens approved by users. The high-level idea of this incident is that the malicious attackers can craft fake and worthless tokens to bypass the assertion in function `flashMintShortOptionsThenSwap` and transfer the fund by leveraging the flash swap service of Uniswap. On 19th June 2020, Bancor ran a white-hat attack on their contract to transfer users' approved tokens to other safe contracts. The root cause is that Bancor mistakenly set the function `safeTransferFrom` as public, so that any user can invoke this function to transfer tokens

---

Table 1: The loss summary of 8 real-world incidents.

Attack Model	Date	Platform	Loss
	2020.08.28	*Degen Money*	-
Model 1	2020.10.05	*Unicat*	0.14M$
	2021.12.02	*Badger*	120M$
	2020.02.22	*Primitive Finance*	10M$*
	2020.06.19	*Bancor*	0.14M$*
Model 2	2020.06.20	*DeFi Saver*	3.5M$*
	2021.02.27	*Furucombo*	15M$
	2022.01.19	*Multichain*	44.5M$

The value with '*' represents the fund which is eventually saved.

---

approved to Bancor. As a result, $455, 349 worth of tokens were saved. Somehow, the massive transfer action was detected by some arbitrage bots to take an advantage of $135, 299. On 20th June 2020, a vulnerability [41] was disclosed to the DeFi Saver team. Instantly, the DeFi Saver team performed a white-hat attack to transfer all vulnerable funds to a safer address where they can only be accessed by the asset owners to withdraw their funds. However, once again, the flaw discovered in the function `SwapTokenToToken` and `takeOrder` allowed hackers to circumvent the requirements and perform their self-designed logic (e.g., transferring approved tokens in the Exchange protocol). On 19th January 2022, the platform Multichain [34] was exploited by a group of hackers and lost around 44M$ because that their smart contract allows users to invoke `permit` function (see more details in Section 8.1) for ERC20 tokens, which do not adopt `permit` function for the approval mechanism.

4 RESEARCH DESIGN

4.1 Terminology

To favor the readability, we summarize a list of terminologies used in the rest of the paper.

Approval Sender ($\mathcal{U}$): The approval sender is an Ethereum account (i.e., EOA or CA) creating approval transactions to delegate permission of spending their ERC20 tokens. In this study, we mainly consider the case that an approval sender acts as an EOA to construct approval transactions. The word 'user' and 'sender' both indicate one EOA in the rest of this paper.

Approval Spender ($\mathcal{S}$): The approval spender is an Ethereum account (i.e., EOA or CA) receiving the permission of spending approved tokens. In this paper, we mainly target DApps (normally act as a CA) which spend users' approved tokens based on the logic of their smart contracts.

ERC20 Token ($\mathcal{T}$): ERC20 tokens are deployed smart contracts which are responsible to perform users' requests (i.e., approving and transferring), and recording the correlated information (e.g., $\mathcal{T}$.balanceOf, $\mathcal{T}$.allowance).

Wallet ($\mathcal{W}$): The wallet is normally used by front-end users to manage their accounts and digital assets. It also plays an important role to help users connect to DApps and construct transactions and send it to the blockchain network for confirmation.

Approval Transaction ($\mathcal{A}$): A transaction that is initiated by the ERC20 standard function `approve`. Constructing an approval

---

[6]AAVE is the first DApp officially announcing the flash loan service.

transaction requires the specification of $\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$, and the approval amount. Based on the approval amount, we can further categorize approval transactions into three types: *1)* Unlimited approval ($\mathcal{UA}$) indicates that the approval amount of $\mathcal{A}$ reaches the maximum value (i.e., `uint256 - 1`) or the total supply of token $\mathcal{T}$; *2)* Zero approval ($\mathcal{ZA}$) is the approval transaction with zero approval amount. $\mathcal{ZA}$ indicates that $\mathcal{U}$ tries to revoke their permission of their approved tokens; *3)* Other Approval ($\mathcal{OA}$) represents the rest of approval transactions.

Execution Transaction ($\mathcal{E}$): A transaction contains an internal transaction initiated by the ERC20 standard function `transferFrom`. Through invoking the function `transferFrom`, the spender spends users' approved token based on the logic of its smart contract code. The validated execution transaction will further decrease the value of $\mathcal{T}$.`balanceOf[U]` and $\mathcal{T}$.`allowance[U][S]` based on the spending amount.

4.2   Research Questions

Our objective of this work is to first reveal the usage of unlimited approval in the ecosystem, and then provide a comprehensive empirical study towards a better understanding of the risk of abusing unlimited approval. To this end, this study is driven by answering the following three research questions.

RQ 1: What is the usage of unlimited approval in the ecosystem and to what extent are the potential risks taken by users? Although the usage and risk of unlimited approval have been loosely mentioned in the public (e.g., the social media), the scale of unlimited approval remains unknown and there is a lack of quantification analysis for the risk of approved tokens. We conduct a transaction-based analysis in Section 5 to answer this question.

RQ 2: what security issues are involved in interacting with DApps and wallets to prepare approval transactions? Front-end users construct approval transactions through directly interacting UIs provided by both DApps and wallets. It is critical to understand how the UIs of DApps and wallets guide users to construct their approval transactions. We conduct an interaction-oriented investigation in Section 6 to answer this question.

RQ 3: What is the current status of user behaviors and how do users achieve good practice to spend approved tokens towards mitigating the risks? Analyzing the user behavior regarding the approval mechanism may help determine the good practice for users to mitigate the risks. By doing so, we provide guidelines for users to securely use approval transactions. We conduct a user behavior analysis in Section 7 to answer this question.

5   TRANSACTION-BASED ANALYSIS

In this section, we conduct the transaction-based analysis to answer RQ 1. Specifically, based on the previous definition of the approval transaction, we first apply a fully-automatic approach to detect approval transactions on Ethereum (Section 5.1). After that, we conduct a comprehensive measurement for identified approval transactions to gain a more detailed view of the approval's usage in the ecosystem (Section 5.2). Finally, we further reveal the risk of users' approved tokens in terms of *RiskAmount* and *RiskLevel* (Section 5.3). To engage the community, we will release the collected dataset and the source code to perform the analysis on https://github.com/PanicWoo/RAID2022_Approval.
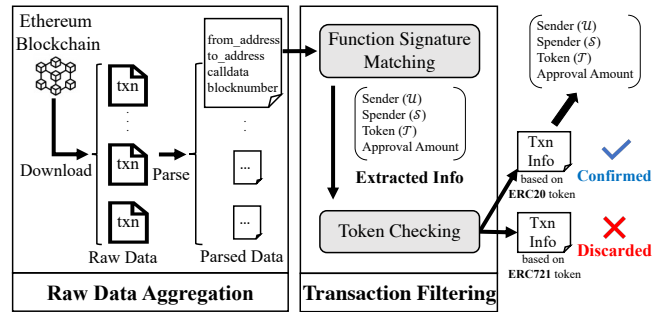

Figure 2: The process of detecting approval transactions.

5.1   Approval Transaction Detection

Figure 2 depicts the process of detecting approval transactions in two stages, as follows:

Raw Data Aggregation.   In this stage, we first take advantage of Geth [1], which is a popular Ethereum client, to synchronize the Ethereum ledger up to block 12936339 (the latest block on 31st July 2021). The synchronized database contains the validated transactions and is built based on a cluster with 12 machines. Each machine consists of an Intel CPU (i7-8700 3.2GHz) and 32GB of memory. Then, we take one more step to parse downloaded transactions and extract desired information (i.e., from_address, to_address, calldata, and block number) for the next stage of filtering.

Transaction Filtering.   In this stage, we accurately identify approval transactions through sequentially processing the parsed data with the following two components:

- **Function Signature Matching.** As aforementioned in Section 2.1, the call data extracted from each raw transaction will include the invoked function signature and associated values mentioned in the invoked function. In this component, we take the advantage of function signatures (Section 2.4) provided by the ERC20 token standard to filter approval transactions. Once the function signature of the transaction is matched with the hash value of the `approve` function, we further identify the value of *sender*, *spender*, *token*, and *approval amount* and pass the value set to the next component. In this study, we only focus on the external transaction initiated by invoking the `approve` function.

- **Token Checking.** The `approve` function of ERC20 and ERC721 [2] token standards have the same format and hash value. Therefore, we leverage the extracted token address to conduct a token checking process to reduce the false-positive rate of our identification process. As mentioned in the standard, ERC721 token contracts must include the variable `uri`. Therefore, in this component, we label tokens having the variable `uri` in their contracts and discard approval transactions interacting with ERC721 token contracts.

5.2   Overall Statistics of Approval Transactions

In total, we discover 25.4M approval transactions up to block 12936339. Based on our definition of unlimited approval, we identify two-thirds of approval transactions (15.2M, 60%) are unlimited approval, while 0.45M (1.8%) and 9.75M (38.4%) are $\mathcal{ZA}$ and $\mathcal{OA}$

Table 2: The overall statistic of identified approval transactions. In total, $4,935,215$ **unique** $\mathcal{U}$, $192,172$ **unique** $\mathcal{S}$ **and** $94,748$ **unique** $\mathcal{T}$ **are identified among all collected approval transactions.**

Approval Type	# of Transaction	Unique Sender ($\mathcal{U}$)	Unique Spender ($\mathcal{S}$)	Unique Token ($\mathcal{T}$)
$\mathcal{UA}$	15.2M (59.8%)	2975K(60.3%)	106.8K(55.6%)	72.4K(76.5%)
$\mathcal{ZA}$	0.45M (1.8%)	116.1K(2.4%)	18.7K(9.7%)	12.5K(13.2%)
$\mathcal{OA}$	9.75M (38.4%)	2777K (56.3%)	97.5K(50.7%)	45.5K(48.0%)
Total Count	**25.4M**	**4,935,215**	**192,172**	**94,748**

In the column of **Sender ($\mathcal{U}$)**, **Spender ($\mathcal{S}$)**, and **Token ($\mathcal{T}$)**, the sum of the percentages is not equal to 100%. For instance, each sender can participate in both unlimited and zero approvals. Similarly, the $\mathcal{S}$ and $\mathcal{T}$ can be involved in different types of approval transactions.

respectively (the first column of Table 2). Note that the evaluation result is conservative due to the strict definition of $\mathcal{UA}$ [7]. Moreover, in the rest columns of Table 2, we discover that a great percentage of $\mathcal{U}$ (60.3%), $\mathcal{S}$ (55.6%), and $\mathcal{T}$ (76.5%) have participated in $\mathcal{UA}$. The statistical result indicates that $\mathcal{UA}$ has been widely used in the ecosystem up to block 12936339.
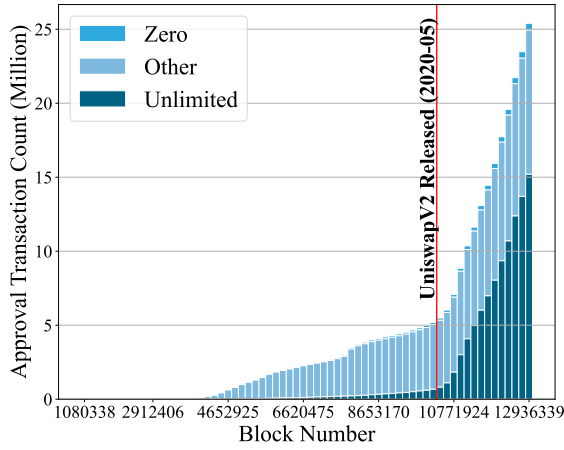

Figure 3: The growth trend and distribution of approval transactions.

We also present the growth trend of approval transactions and different participants ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$) in Figures 3 and 4. In Figure 3, it is worth noting that the number of approval transactions has experienced a rapid increase since May 2020. In the meanwhile, as shown in Figure 4, the number of $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{T}$ has also increased quickly. The reason for the increase is the introduction of *Uniswap V2* [45], which is one of the largest DEXes on Ethereum. Through investigating all unlimited approval transactions in terms of $\mathcal{S}$, we find that there is 35% (8.9M/25.4M) of approval transactions specifying *UniswapV2 Router* as the spender. Moreover, 94% (8,361,001/8,925,110) of those approval transactions belong to $\mathcal{UA}$.

---

[7]As a matter of fact, we also discover 23% of approval transactions with an extremely large approval amount (greater than 2^{248}), which could be regarded as $\mathcal{UA}$ to some extent, as the corresponding users are still at a risk. However, we decide to follow the strict criterion to simplify the measurement.

5.3 Risk Analysis for Approved Tokens

Unlimited approval does not directly cause users to lose their approved tokens. However, abusing unlimited approval will expose these tokens in a dangerous place. In the following, we will investigate the top three most frequently used tokens (i.e., USDC, USDT and DAI) to evaluate the potential risk of users' approved tokens.

Risk Amount Analysis. We define the *RiskAmount* to quantify the potential risk of users' approved tokens. For a given tuple ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$), the *RiskAmount* indicates the amount of $\mathcal{U}$'s owned tokens that a $\mathcal{S}$ can transfer. Specifically, *RiskAmount* equals the smaller amount of a $\mathcal{U}$'s owned tokens and the tokens approved to a $\mathcal{S}$ by the $\mathcal{U}$, as shown in Equation 1:

$$RiskAmount = Min(\mathcal{T}.allowance[\mathcal{U}][\mathcal{S}], \mathcal{T}.balanceOf[\mathcal{U}]) \quad (1)$$

Accordingly, Figure 5 presents the growth trend of the accumulative *RiskAmount* of all users in different tokens. As for the time range, we obtain the data in the past whole year (2020-08-01 to 2021-7-31) to plot the trend graph of *RiskAmount*. From Figure 5, we discover that *RiskAmount* of each top token is generally increasing in the past whole year (especially USDT), which suggests that more and more approved tokens are in danger as time goes by.

Risk Level Analysis. We define the term *RiskLevel* to facilitate classifying users' approved tokens based on the amount of $\mathcal{U}$'s owned tokens ($\mathcal{T}$.`balanceOf`) and approved tokens ($\mathcal{T}$.`allowance`). Specifically, there are three risk levels, i.e., *No Risk*, *Low Risk*, and *High Risk*. To better understand *RiskLevel*, we consider a given tuple ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$), and there will be Equation 2:

$$RiskLevel = \begin{cases} No\ Risk, & \mathcal{T}.allowance[\mathcal{U}][\mathcal{S}] = 0 \\ \\ Low\ Risk, & \begin{aligned} &\mathcal{T}.allowance[\mathcal{U}][\mathcal{S}] > 0 \\ &\&\&\ \mathcal{T}.balanceOf[\mathcal{U}] = 0 \end{aligned} \\ \\ High\ Risk, & \begin{aligned} &\mathcal{T}.allowance[\mathcal{U}][\mathcal{S}] > 0 \\ &\&\&\ \mathcal{T}.balanceOf[\mathcal{U}] > 0 \end{aligned} \end{cases} \quad (2)$$

Specifically, there are three cases: 1) if $\mathcal{U}$ have zero `allowance` to $\mathcal{S}$, we consider that there is **no risk** on $\mathcal{U}$'s token $\mathcal{T}$; and 2) if $\mathcal{U}$'s have the `allowance` which is greater than zero and zero `balanceOf`, $\mathcal{U}$'s token $\mathcal{T}$ has **low risk**. Moreover, low-risk indicates that once the users receive tokens (`balanceOf > 0`), the malicious
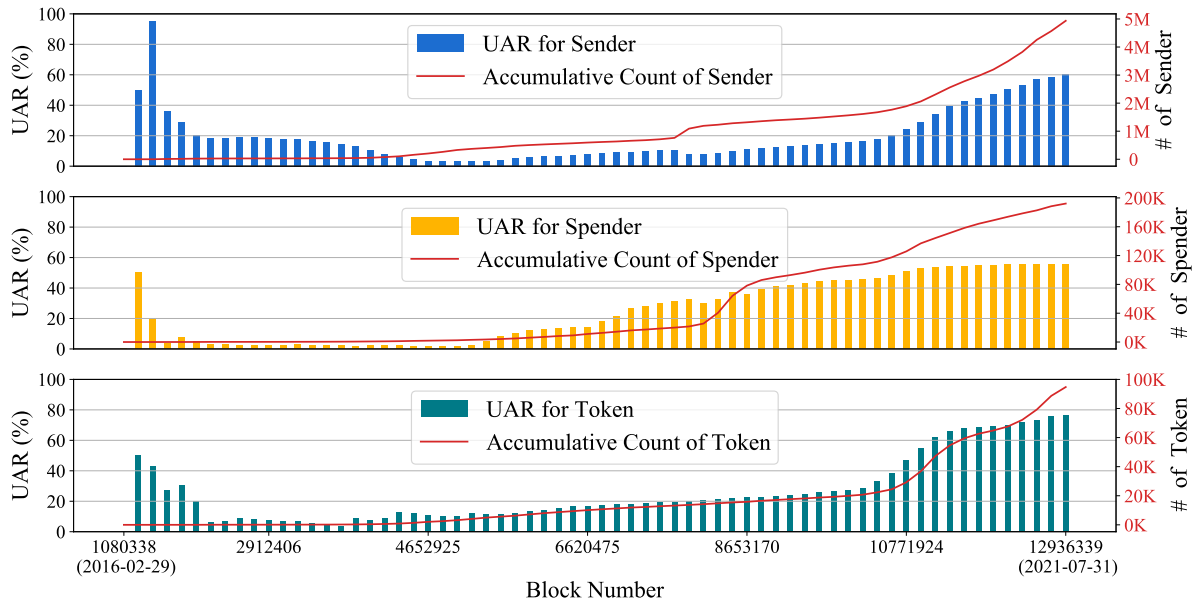
The UAR of $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{T}$ is high in the early stage. The reason causing this bias is due to the small number of approval transactions.

Figure 4: The growth trend of approval transaction participants ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$). From the top to the bottom sub-chart, the line plot presents the cumulative count of unique $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{T}$ respectively. The value of the line plot can be referred to the y-axis on the right side. As for the bar plot, it indicates the percentages of $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{T}$ participating $\mathcal{UA}$ (UAR in short). For example, the last bar of the top sub-chart shows that around 60% of senders construct unlimited approval transactions at least once. The value of the bar plot can be referred to the y-axis on the left side.
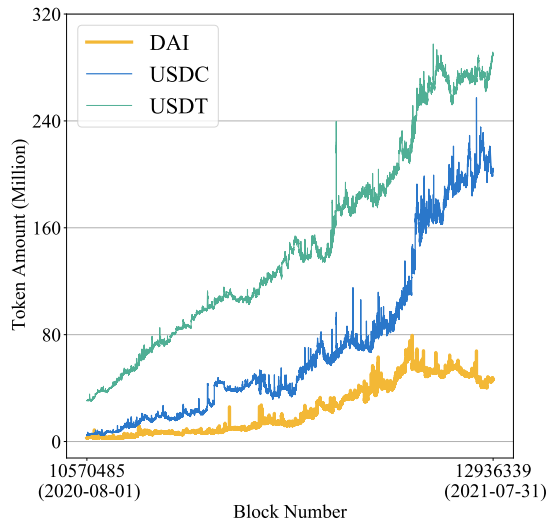

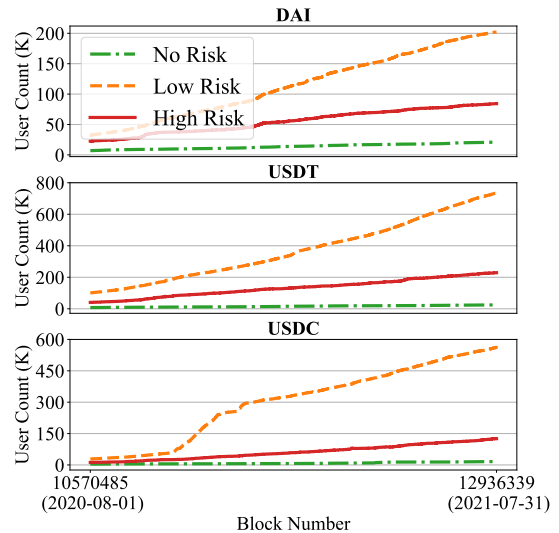**Figure 5: The growth trend of risk amount on DAI, USDC, and USDT.**

Figure 6: The growth trend of users with the different risk levels.

platforms with users' approval can instantly transfer users' received tokens; and 3) if the `balanceOf` and `allowance` of users' tokens are

both greater than zero, users' tokens are at a high-risk level and extremely risky from stealing. In the real world, many security

incidents [30, 33] demonstrate the possibility of stealing high-risk tokens.

By analyzing *RiskLevel* of all users of USDC, USDT, and DAI, we observe that in the past year, the number of users with *low-risk* or *high-risk* tokens has grown steadily compared to users with *no-risk* tokens (Figure 6). Users with low-risk tokens become the majority among all evaluated users. This indicates that most users do not instantly revoke their approval from DApps. According to the fact, users approving tokens to malicious DApps can suffer from token stealing once they receive corresponding tokens. Moreover, we further present the distribution of users with different *RiskLevel* in Table 3. The results suggest that 97% of users are underlying victims to token stealing, as well as 22.4% of users remain their approved tokens with high risk.

Table 3: The distribution of users based on risk levels.

Token	# of Users	Risk Level Distribution		
		No	*Low*	*High*
USDC	734, 307	2.3%	79.2%	18.5%
USDT	1, 058, 617	2.4%	74.6%	23.0%
DAI	319, 913	6.9%	66.0%	27.1%

> **Answer to RQ 1:** *Unlimited approval is widely used (60% of all approval transactions) in the Ethereum ecosystem and a great percentage of unique users (60%, 2.9M/4.9M) have participated in the unlimited approval transactions. Specifically, the number of unlimited approval transactions is extremely concentrated in Uniswap V2, which involves 35% of approval transactions and 94% of them are unlimited approval. For the top three tokens (i.e., USDT, USDC and DAI), most users (97%) are threatened by token stealing and 22% of them are at high-level risk.*

6 INTERACTION-ORIENTED ANALYSIS

Our prior transaction-based analysis suggests the prevalence of unlimited approval on Ethereum. In this section, to fully understand the abuse of unlimited approval, we aim to reveal the security issues involved in interacting with the UIs of DApps and wallets by conducting an interaction-oriented investigation. Specifically, we first demonstrate the process of constructing an approval transaction from the front-end user's perspective (Section 6.1). Then, we investigate the *Interpretability* and *Flexibility* of UIs provided by 22 DApps (Section 6.2) and 14 wallets (Section 6.3), respectively.

6.1 Approval Transaction Construction

As shown in Figure 7, a front-end user constructs an approval transaction in six steps. In **Step 1** and **Step 2**, a front-end user first connects the wallet to a DApp and selects desired services provided by the DApp. In **Step 3**, the DApp constructs the approval transaction (including sender address, spender address, token address, and approval amount) and sends it to the user's wallet for further confirmation. After receiving the approval transaction, in **Step 4** and **Step 5**, the user's wallet presents the transaction information on its
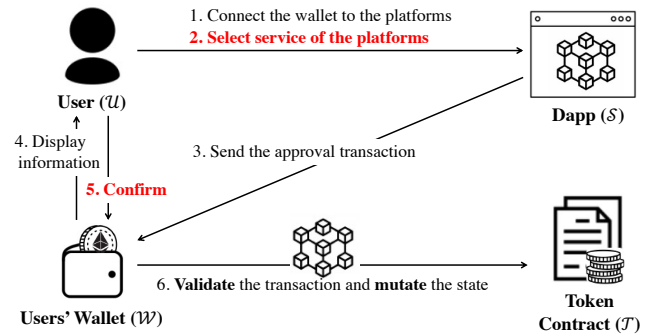

Figure 7: The workflow of constructing an approval transaction by front-end users.

UI and waits for the user to double check the approval transaction. In **Step 6**, once the approval transaction is confirmed by the user, it will be forwarded to the blockchain network for validation. Then, the validated transaction mutates the variable `allowance` in the token contract. It is worth noting that, in **Step 2** and **Step 5** highlighted in Figure 7, the front-end user directly interacts with the UI of the DApp and the wallet to construct approval transactions.

6.2 The Investigation of DApps

We collect 22 DApps according to the number of received approval transactions and total value being locked (i.e., TVL). As of this writing, the TVL of the collected DApps is over 80% and over 50% of approval transactions that are related to the selected DApps.

Investigation Criteria. The term *Interpretability* of DApp's UI indicates the explanatory information of the approval mechanism. In the investigation of DApp's UI, we evaluate the *Interpretability* based on the following three criterion:

- Does the UI explains the purpose of the approval transaction;
- Does the UI presents the existence of the approval transaction;
- Does the UI notifies users that approval and execution transactions are consecutively executed.

As for the *Flexibility*, we check whether the DApp's UI provides the modification option for users to customize the approval amount.

Investigation Result. Table 4 summarizes the investigation result, which reveals that all DApps adopt unlimited approval as the default setting on their UIs. Specifically, 13 DApps do not explain the approval action to users at all, and 7 DApps partially explain the approval mechanism based on our defined criteria. Only *Loopring* [5] and *Yearn.Finance* [20] comprehensively describe the approval mechanism. As for the *Flexibility*, only 2 DApps (Bancor [4] and Keeper DAO [8]) allow users to adjust their approval amounts on their UIs.

Special Cases. We also discover two special cases, i.e., *Curve.fi* and *Yearn.Finance*, which display misleading information for front-end users to grant unlimited approval. Both DApps request an approval with the amount based on users' spending amount on their web UI, but, in fact, they prepare an unlimited approval transaction for users. Figure 8 shows the process of constructing the approval transactions for the two cases. Specifically, *Curve.fi* is a DEX platform for stable coins, and it is supposed to request an approval with

Table 4: The investigation result of DApps' user interfaces.

DApp Name	TVL (USD)	Approval Txn	Approval Setting	Interpretability	Flexibility
Maker	15.305*B*	0.30%	Unlimited	-	NO
Aave	13.084*B*	1.73%	Unlimited	-	NO
Compound	8.905*B*	0.43%	Unlimited	-	NO
*Curve.fi**	8.419*B*	1.24%	Unlimited	-	NO
Uniswap	5.875*B*	40.6%	Unlimited	Criteria 3	NO
Sushi Swap	3.488*B*	2.34%	Unlimited	-	NO
*Yearn.Finance**	3.331*B*	0.33%	Unlimited	Criteria 1,2,3	NO
Balancer	1.557*B*	0.69%	Unlimited	-	NO
Bancor	1.378*B*	1.10%	Unlimited	Criteria 1,2	YES
Alpha Homora	1.042*B*	0.01%	Unlimited	-	NO
Cream Finance	484.192*M*	0.19%	Unlimited	Criteria 1,2	NO
Defi Saver	434.392*M*	0.001%	Unlimited	-	NO
Keeper DAO	408.48*M*	-	Unlimited	-	YES
Harvest Finance	399.808*M*	-	Unlimited	-	NO
Index Coop	280.553*M*	0.01%	Unlimited	-	NO
Set Protocol	198.86*M*	0.04%	Unlimited	-	NO
dYdX	179.429*M*	0.34%	Unlimited	Criteria 1,2	NO
Idle Finance	178.986*M*	0.013%	Unlimited	Criteria 1,2	NO
Loopring	123.151*M*	0.05%	Unlimited	Criteria 1,2,3	NO
Integral	90.762*M*	-	Unlimited	-	NO
1inch	33.349*M*	2.74%	Unlimited	Criteria 3	NO
0x	−	2.29%	Unlimited	Criteria 1	NO
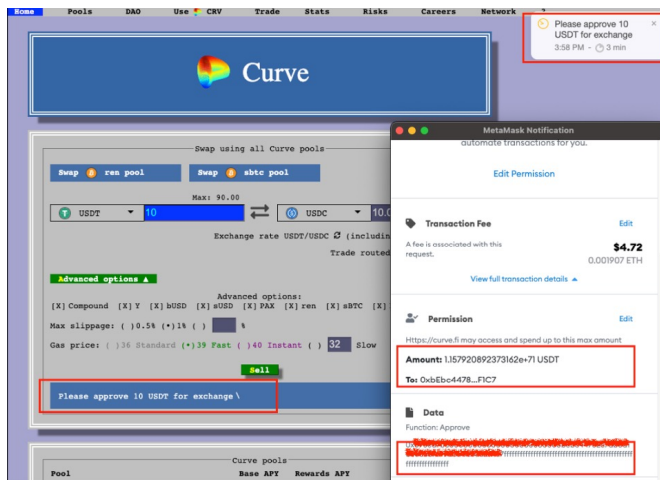
(a) **Curve.fi.**
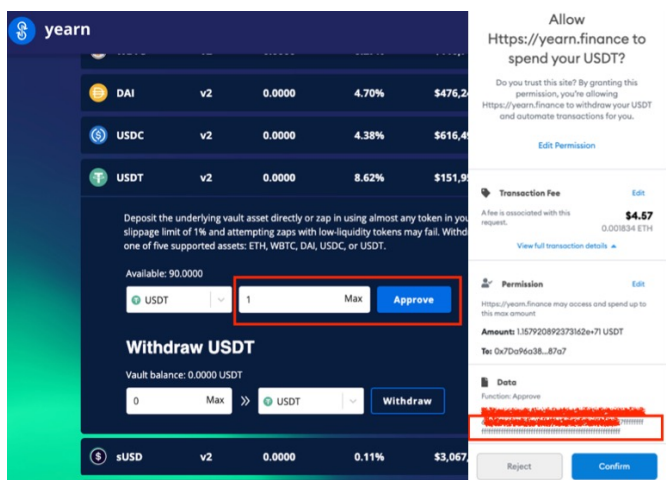
(b) **Yearn.Finance.**

Figure 8: The demonstration of interacting *Curve Finance* and *Yearn Finance*.

the amount according to users' spending amount, i.e., 10 USDT, as shown in the left bottom and right top of Figure 8a. However, the right bottom of Figure 8a suggests that *Curve.fi* directly prepares an unlimited approval transaction without providing any notification

related to the unlimited approval on its web UI. As a result, users might be misled to approve unlimited tokens to *Curve.fi*. Similarly, *Yearn.Finance* provides misleading information on their web UI as well. In Figure 8b, when a user believes that she/he only approves 1

Table 5: The investigation result of wallets' UIs.

Wallet	Google Play Store Downloadings (#)	Apple App Store Score / Reviews (#)	Interpretability				Flexibility
			Spender	Token	Amount	Explanation	
Metamask	10,000,000+	4.2 / 7,800+	✓	✓	✓	✓	YES
Trust Wallet	10,000,000+	4.7 /160,000+	✗	✗	✗	✗	NO
Crypto.com	1,000,000+	4.3 / 68,000+	✗	✓	✓	✓	NO
Coinbase	1,000,000+	4.6 / 89,000+	✗	✗	✗	✗	NO
Argent	100,000+	4.5 / 1,300+	✗	✓	✓	✓	YES
Pillar	100,000+	4.5 / 100+	✗	✗	✗	✗	NO
MEW	500,000+	4.7 / 4,000+	✗	✗	✗	✗	NO
imToken	500,000+	4.4 /400+	✓	✓	✓	✓	YES
Rainbow	N/A	4 / 500+	✗	✗	✗	✓	NO

USDT, *Yearn.Finance* actually provides an unlimited approval transaction by default.

6.3 The Investigation of Wallets

We collect 9 well-known wallets based on the number of downloads ranked by the Google Play Store [23] and the number of reviews in the Apple Store [10].

Investigation Criteria. As aforementioned, the wallet's UI presents the approval transaction prepared by the DApp to the user for further confirmation. To analyze *Interpretability* of wallets' UIs, we mainly investigate the transaction information (i.e., sender ($\mathcal{U}$), spender ($\mathcal{S}$) and approval amount) and the explanation of the approval mechanism presented on the wallet's UI. As for the *Flexibility*, we aim to check whether wallets provide a modification option for users to adjust the approval amount.

Investigation Result. Table 5 gives the result. Specifically, in terms of *Interpretability*, 4 (out of 9) wallets do not display any transaction information and explanation of the approval transaction for users and only *Metamask* presents the comprehensive information. In terms of *Flexibility*, only 33% wallets (3/9, i.e., *Metamask*, *imToken* and *Argent*) embed the modifying feature on their UIs. This indicates that 67% (6/9) wallets will lead users to construct an approval transaction based on DApps' default setting (i.e., unlimited approval). Only front-end users of *Metamask*, *imToken* and *Argent* can customize their approval amounts to avoid constructing unlimited approval. In conclusion, the limited explanation on the approval mechanism and the lack of the flexibility on the wallets' UI can aggravate the abuse of unlimited approval.

> **Answer to RQ 2:** *The investigation result suggests that most DApps and wallets do not provide comprehensive understandings and flexibility for users to mitigate the risk of unlimited approval. Specifically, the result reveals that unlimited approval is adopted by all (22) DApps on their UIs. However, only 10% DApps and wallets (i.e., 9% (2/22) and 11% (1/9), respectively) provide explanatory information of the approval mechanism. Moreover, only 16% (5/31) of UIs enable users to adjust the approval amounts. Surprisingly, we also discover two special DApps, i.e., Curve.fi and Yearn.Finance, which even mislead users to send unlimited approval transactions.*

7 USER BEHAVIOR ANALYSIS

In this section, we seek to explore the good practice of spending approved tokens without risks. Specifically, we first detect and sort the user behavior in a temporal order (Section 7.1). After that, we characterize the user behavior into five modes and formalize the good practice mitigating the risk of approved tokens (Section 7.2). Lastly, we quantify the user behavior based on those modes and the good practice (Section 7.3).
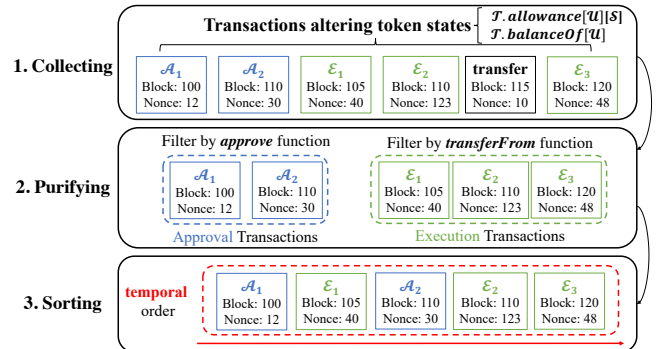

Figure 9: The process of user behavior detection with a given tuple ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$).

7.1 User Behavior Detection

The user behavior indicates a group of ordered approval ($\mathcal{A}$) and execution ($\mathcal{E}$) transactions. Through analyzing this group of ordered transactions, we can understand how a user manages to spend his or her approved tokens. In Figure 9, we present the process of detecting user behaviors for a given tuple ($\mathcal{U}$, $\mathcal{S}$, $\mathcal{T}$). Specifically, there are three steps:

1) We collect all transactions altering states of variables: $\mathcal{T}$.allowance[$\mathcal{U}$][$\mathcal{S}$] and $\mathcal{T}$.balanceOf[$\mathcal{U}$]. However, the extracted transactions also include transactions initiated by $\mathcal{U}$ to directly transfer his or her ERC20 tokens.

2) We purify collected transactions to consist of only the transaction triggered by the function `approve` and `transferFrom`. Since we

Table 6: The classification of user behaviors.

Mode ID	# of $\mathcal{A}, \mathcal{E}$	Sequence Order	Good Practice
1	One, One	$\mathcal{A}_1 \rightarrow \mathcal{E}_1$	$O\mathcal{A}_1 \rightarrow \mathcal{E}_1$, where $amount_{O\mathcal{A}_1} == amount_{\mathcal{E}_1}$
2	One, Many	$\mathcal{A}_1 \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_n$	$O\mathcal{A}_1 \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_n$, where n > 1 && $amount_{O\mathcal{A}_1} == \sum_{i=1}^{n} amount_{\mathcal{E}_i}$
3	One or Many, Zero	$\mathcal{A}_1$ or $\mathcal{A}_1 \rightarrow ... \rightarrow \mathcal{A}_n$	All cases in this mode are considered as bad practices.
4	Many, One	$\mathcal{A}_1 \rightarrow ... \rightarrow \mathcal{A}_n \rightarrow \mathcal{E}_1$	All cases in this mode are considered as bad practices.
5	One or Many, One or Many	Combination of Mode 1, 2, 3, 4	Consisting of only $\mathcal{UA} \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_n \rightarrow \mathcal{ZA}$ or $O\mathcal{A} \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_m$, where n,m >= 1 && $amount_{O\mathcal{A}_1} == \sum_{i=1}^{m} amount_{\mathcal{E}_i}$

only consider the user as an EOA, the `approve` and `transferFrom` are invoked in separate transactions.

3) We merge purified transactions in the temporal order. It is worth noting that we sort transactions based on their block number and transaction nonce [8] so that all extracted transactions regarding the user behavior can be arranged in absolute order.

7.2 User Behavior Characterization

Based on the sequence and the amount of $\mathcal{A}$ and $\mathcal{E}$, we can characterize user behaviors into five modes, including four basic modes and one compound mode, as listed in Table 6.

- **Mode-1 (aka one-to-one mode):** The user only spends approved tokens once after one approval transaction. Achieving good practice in mode 1 requires users to construct $O\mathcal{A}$ for the $\mathcal{A}_1$ with the approval amount that equals the number of tokens spending on the following execution transaction $\mathcal{E}_1$. Through this, the $\mathcal{T}$.`allowance[`$\mathcal{U}$`][`$\mathcal{S}$`]` will remain zero and the *RiskLevel* of the users' approved token becomes no-risk. On the other hand, using unlimited approval in mode 1 leads the approved tokens to become risky.
- **Mode-2 (aka one-to-many mode):** The user spends approved tokens several times with only one approval transaction. To achieve the good practice, the user has to build $O\mathcal{A}$ for the $\mathcal{A}_1$ with the approval amount that equals the total amount of tokens spent on the following execution transactions ($\mathcal{E}_1, ..., \mathcal{E}_n$).
- **Mode-3 (aka only-approval mode):** The user **only** sends the approval transaction(s). We consider the user behavior following mode 3 as bad practices because of the redundant approval transactions. Even though the user might remain their approved tokens without risk by constructing a zero approval transaction at the end like ($\mathcal{A}_1 \rightarrow ... \rightarrow \mathcal{A}_n \rightarrow \mathcal{ZA}$), the user wastes money (gas fee) for only sending approval transactions.
- **Mode-4 (aka many-to-one mode):** The user sends many approval transactions before spending approved tokens. The user behavior following mode 4 can be interpreted as the combination of modes 1 and 3, which is a bad practice due to the redundant approval transactions.

- **Mode-5 (aka compound mode):** The user behavior of mode 5 is the combination of at least two basic mode behaviors. To achieve good practice in mode 5, the user has to construct approval and execution transactions by only using two fixed patterns: *1)* $\mathcal{UA} \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_n \rightarrow \mathcal{ZA}$, *2)* $O\mathcal{A} \rightarrow \mathcal{E}_1 \rightarrow ... \rightarrow \mathcal{E}_m$ to ensure that the approved token has no risk.

Achieve the good practice. Table 6 also explains the way to achieve the good practice regarding the *RiskLevel* of users' approved tokens after the last transaction of the user behavior. Accordingly, users can mitigate the risk of approved tokens by following the good practice mentioned in modes 1, 2, and 5. Specifically, the good practice requires *on-demand approval and timely spending*, which means users need to approve the right amount of tokens and trade them as soon as possible based on the approval mechanism, and do not delegate any unnecessary tokens to the DApp thereafter.

7.3 User Behavior Distribution

To conduct a comprehensive analysis of the user behavior, we run the user behavior detection for ten representative pairs of $\mathcal{S}$ and $\mathcal{T}$ which cover the most users [9]. As shown in Table 7, based on these ten pairs, we totally collect **1,496,886** user behaviors, which covers 1,314,995 (30% of all collected users in our data set) unique users. We also measure the distribution of the user behavior based on five defined modes in Figure 10 and present the corresponding statistics in Table 7. The result shows that the characterized user behaviors are extremely concentrated in modes 1, 2 and 3, i.e., 46%, 30% and 20% of unique users that use approval transactions, respectively. Specifically, 60% (6/10) and 30% (3/10) of pairs are mostly used to spend approved tokens by following modes 1 and 2. Meanwhile, in pair 3, 68% of user behaviors follow mode 3, which only sends approval transactions. However, only 2.3% of unique users act based on mode 5 which can possibly become the good practice.

Moreover, we take a further step to measure the distribution of the user behaviors following the good practice. We observe that only 0.01% (83/1,138,840) of user behaviors follow good practice in modes 1 and 2. The reason is that over 99% of user behaviors belonging to modes 1 and 2 start with unlimited approval. The fact reveals that most users in mode 1 and mode 2 have remained their

---

[8]The transaction nonce implies the order of transactions in the same block.

[9]Of course, $\mathcal{S}$ or $\mathcal{T}$ is of a great amount of TVL or market value as well.

Table 7: The user behavior distribution of 10 pairs of $\mathcal{S}$ and $\mathcal{T}$. In total, we have detected 1,496,886 user behaviors based on the ten selected pairs. There are 1,314,995 identical users among detected user behaviors. This is because a user might spend approved tokens with different pairs.

| Pair ID	Spender ($\mathcal{S}$)	Token ($\mathcal{T}$)	Identical User (# / %)	Behavior Mode Distribution (%)				
				Mode1	Mode2	Mode3	Mode4	Mode5
1	UniswapV2	USDT	338,855 / 6.8%	37.1%	**41.1%**	18.8%	1.2%	1.8%
2	Compound	USDC	249,407 / 5.0%	**83.1%**	7.5%	3.0%	3.5%	2.8%
3	UniswapV2	USDC	197,474 / 4.0%	35.7%	**46.6%**	14.2%	1.1%	2.4%
4	Wyern	WETH	141,242 / 2.8%	15.0%	13.8%	**68.0%**	0.7%	2.5%
5	UniswapV2	SHIB	117,381 / 2.4%	**38.7%**	28.5%	30.6%	1.2%	1.1%
6	UniswapV2	DAI	115,538 / 2.3%	39.3%	**42.9%**	14.1%	1.3%	2.4%
7	Shiba	SHIB	102,162 / 2.1%	**52.6%**	31.5%	6.6%	4.5%	4.8%
8	UniswapV2	WETH	81,142 / 1.6%	**40.8%**	35.5%	20.1%	1.5%	2.1%
9	Wyern	USDT	78,437 / 1.6%	**43.9%**	23.2%	30.7%	1.3%	0.9%
10	Polygon	Matic	75,248 / 1.5%	**67.3%**	26.1%	3.9%	1.3%	1.3%
User Behavior Count			1,496,886	687,817(46%)	451,023(30%)	297,528(20%)	26,734(1.7%)	33,784(2.3%)
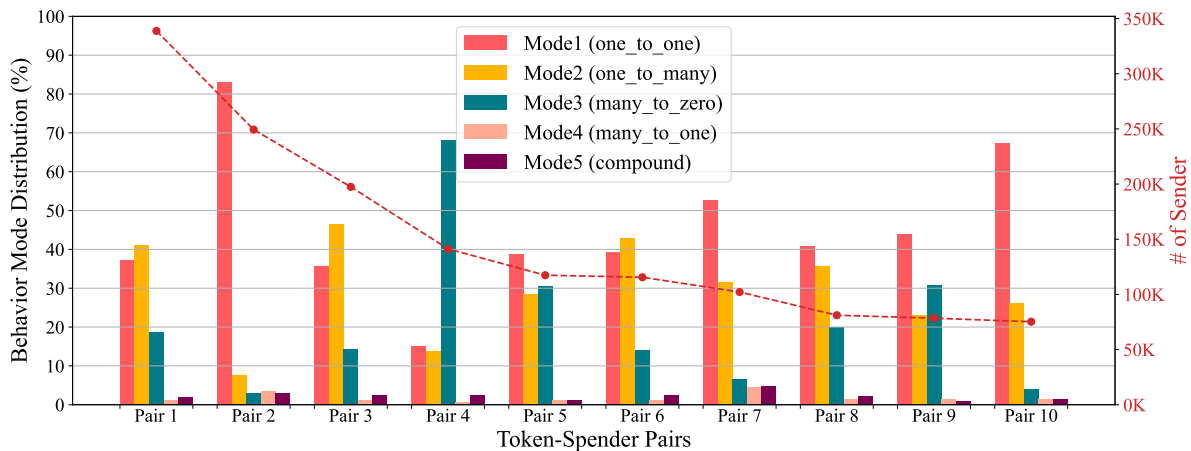
Figure 10: The distribution of user behaviors. Note that the points on the dot line represent the number of unique users.

approved tokens with risk of stealing. As for the user behavior in mode 5, 7% (2,392 / 33,784) of user behaviors comply with the good practice of using approval transactions. In total, only **0.2% (2,475 / 1,496,886)** of user behaviors follow the good practice to spend approved tokens. The revealed worrisome result may be caused by DApp's unclear interpretation and low flexibility regarding the ERC20's approval mechanism.

Answer to RQ 3: *The analysis result suggests that 76% of user behaviors comply with modes 1 and 2, and 99% of their user behaviors are using unlimited approval. Theoretically, modes 1, 2, and 5 may lead to the good practice. To this end, users shall spend out approved tokens by either granting limited approval on demand or revoking the approved tokens after the last execution transaction. However, the result reveals a worrisome fact that only 0.2% of user behaviors follow the good practice to spend the ERC20 tokens based on the approval mechanism.*

---

8 EXISTING SOLUTIONS AND SUGGESTIONS

In this section, we first discuss two existing solutions attempting to address the trade-off between the convenience and security of unlimited approval. After that, we provide suggestions for the stakeholders (i.e., users, DApps, and wallets) regarding the approval mechanism to mitigate the risks of unlimited approval.

8.1 Existing Solutions

ERC777. Compared to the ERC20 token standard, ERC777 [26] has the distinct ability to send tokens in a single transaction with the introduction of the *operator*, which is an (arbitrary) address authorized by users to transfer users' tokens with provided logic (i.e., the hook function). Moreover, users have the flexibility to authorize/revoke multiple *operator*s. Obviously, the advantage of ERC777 is to simplify the process of transferring tokens with only one transaction. However, it is also a two-folded solution. First, transferring ERC77 tokens requires a high transaction fee due to the hook functions. Second, ERC777 also introduces a trust issue that

users have to select a trustworthy *operator*. Third, spending ERC777 on a new platform requires users to authorize a new *operator* again.

EIP2612.   EIP2612 [31] introduces a function `permit` for ERC20 tokens to validate and process users' approval via an off-chain signed message. This proposal can help users save gas costs from sending an approval transaction, as users only need to pay the gas fee for the execution transactions. Moreover, since there is no cost in the approving process, users can customize their approval amounts for their execution transactions. Therefore, users will have no risk on their tokens because the variable $\mathcal{T}$.allowance[$\mathcal{U}$] will always remain at zero. However, to the best of our knowledge, only a few tokens (e.g., DAI [6] and UNI [7]) adopt the `permit` function in their contracts, which may cause problems for platforms that try to support different tokens in a unified way. For example, the platform *Multichain* [34] was exploited and lost around 44M USD. The root cause is that it allows malicious users to invoke the `permit` function for ERC20 tokens which do not adopt EIP2612.

8.2   Suggestions

In our study, we reveal the fact that unlimited approval is abused in the ecosystem. Our findings may shed some light on stakeholders involved in the approval mechanism.

ERC20 Token Users.   Our study reveals that front-end users send at least two transactions (approval and execution transactions) to perform actions in a DApp. Front-end users should be aware of the approval setting designed by the interacting DApps and wallets before approving their tokens. Moreover, users should have security consciousness regarding interacted DApps. For example, for a given DApp, whether the smart contracts are verified by some reputable platforms (e.g., Etherscan [42]), or they are officially audited by trustworthy security companies? To protect approved tokens, we suggest that users only approve the customized amount of tokens needed for further executions to minimize the risk. Moreover, users should actively monitor their approved tokens so that they can timely revoke approved tokens if necessary. Users can easily revoke their approved tokens via some platforms (e.g., *approved.zone* [12] and *revoke.cash* [9]).

DeFi DApps and Crypto Wallets.   Unlimited approval is often designed as default settings on their web UI to improve the user experience. However, it may mislead novice users without (comprehensive) explanatory information. As such, DApps and wallets should precisely explain the risk of unlimited approval so that users can find the suitable approval transaction according to their further execution. More importantly, DApps and wallets also need to enable the feature that allows users to customize their approval amounts, which may further motivate users to construct secure approval transactions. Apart from that, it might be a good idea for them to develop revoking functionality on their UIs so that users can timely withdraw their approvals. Lastly, DApps should consider adopting emergency operations in their smart contract (e.g., pausing the operation) to protect users' funds.

9   RELATED WORK

Analysis for ERC20 Token.   In the academic field, several studies have been published to analyze ERC20 tokens [14, 15]. For

example, Chen et al. [14] proposed a tool, called *TokenScope*, to investigate inconsistent token behaviors among 7, 421 ERC20 tokens. Chen et al. [15] conduct a graph analysis to characterize money transfer, contract creation, and invocation of ERC20 tokens in the Ethereum ecosystem. Besides, some works have addressed the improper use of ERC20 tokens [22, 38, 51]. For example, Rahimian et al. [38] proposed two protocols to resolve the multiple withdrawal problem raised by the ERC20 token standard. Gao et al. [22] revealed the prevalence of counterfeit cryptocurrencies on Ethereum. Xia et al. [51] systematically analyzed the behavior, the working mechanism, and the financial impacts of scam tokens and identified over 10K scam tokens and scam liquidity pool on Uniswap V2.

DeFi Security.   Apart from the security issues of the ERC20 token, in recent years, DeFi security has been widely researched and has drawn great attention. Numerous studies have been conducted to address the security issues like price manipulation [49], frontrunning [16, 17, 52], governance [25], and flash loan attacks [37]. Furthermore, Sam et al. [48] systematically studied the security issues introduced in theory or occurred in practice.

Smart Contract Vulnerability.   The vulnerability of the smart contract can cause severe loss for DApps and users. Many studies dedicated to detecting or protecting vulnerable smart contracts with static analysis techniques (e.g., symbolic execution and formal verification) [24, 28, 32, 43] or dynamic analysis techniques (e.g., fuzzing) [27, 39, 40, 50]. Moreover, Perez et al. [35] surveyed 23,327 vulnerable contracts identified by six academic projects and discovered that only 1.98% of them have been exploited.

10   CONCLUSION

In this paper, we present the first in-depth study of quantifying the risk of unlimited approval of ERC20 tokens on Ethereum. Our study proposes a fully-automatic approach to detect the approval transactions, and reveals the prevalence (60%) of unlimited approval in the ecosystem. We also conduct an investigation to reveal the security issues involved in interacting with 31 UIs (22 DApps and 9 wallets) to send approval transactions. The result shows that only a few UIs provide explanatory understandings (10%) and flexibility (16%) for users to mitigate the risk of unlimited approval. Furthermore, we perform the user behavior analysis to characterize five modes of user behaviors and formalize the good practice to use approved tokens. The result reveals that users (0.2% of user behaviors) barely follow the good practice towards mitigating the risks of unlimited approval. Finally, we discuss two existing solutions attempting to address the trade-off between convenience and security of unlimited approval, and provide possible suggestions.

REFERENCES

[1] 2014. Geth. https://geth.ethereum.org/. [Online; accessed August-2021].
[2] 2015. ERC20 Token Standard. https://eips.ethereum.org/EIPS/eip-721. [Online; accessed August-2021].
[3] 2015. Tether (USDT). https://tether.to/. [Online; accessed August-2021].
[4] 2017. Bancor. https://bancor.network/. [Online; accessed August-2021].

[5] 2019. Loopring. https://loopring.org/#/. [Online; accessed August-2021].
[6] 2020. The Application of EIP2612 in DAI. https://docs.makerdao.com/smart-contract-modules/dai-module/dai-detailed-documentation#3-key-mechanisms-and-concepts. [Online; accessed August-2021].
[7] 2020. The Application of EIP2612 in Uniswap. https://github.com/Uniswap/uniswap-v2-core/blob/master/contracts/UniswapV2ERC20.sol. [Online; accessed August-2021].
[8] 2020. Keeper DAO. https://www.keeperdao.com/. [Online; accessed August-2021].
[9] 2020. revoke.cash. https://revoke.cash/. [Online; accessed August-2021].
[10] Apple. 2008. Apple App Store. https://www.apple.com/app-store/. [Online; accessed August-2021].
[11] Badger. 2021. Badger Incident. https://twitter.com/peckshield/status/1466356911842856967. [Online; accessed Jan-2022].
[12] Anton Bukov and Sergej Kunz. 2019. approved.zone. https://approved.zone/. [Online; accessed August-2021].
[13] Centre. 2018. USD Coin. https://www.centre.io/usdc. [Online; accessed August-2021].
[14] 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*. 1503–1520.
[15] Weili Chen, Tuo Zhang, Zhiguang Chen, Zibin Zheng, and Yutong Lu. 2020. Traveling the token world: A graph analysis of ethereum erc20 token ecosystem. In *Proceedings of The Web Conference 2020*. 1411–1421.
[16] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 910–927.
[17] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. Sok: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*. Springer, 170–189.
[18] Vitalik Buterin Fabian Vogelsteller. 2015. ERC20 Token Standard. https://eips.ethereum.org/EIPS/eip-20. [Online; accessed August-2021].
[19] Curve Finance. [n.d.].
[20] Yearn Finance. [n.d.].
[21] Furucombo. 2021. Furucombo Incident. https://medium.com/furucombo/furucombo-post-mortem-march-2021-ad19afd415e. [Online; accessed August-2021].
[22] Bingyu Gao, Haoyu Wang, Pengcheng Xia, Siwei Wu, Yajin Zhou, Xiapu Luo, and Gareth Tyson. 2020. Tracking Counterfeit Cryptocurrency End-to-end. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 3 (2020), 1–28.
[23] Google. 2008. Google Play Store. https://play.google.com/store. [Online; accessed August-2021].
[24] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–27.
[25] Lewis Gudgeon, Daniel Perez, Dominik Harz, Arthur Gervais, and Benjamin Livshits. 2020. The Decentralized Financial Crisis: Attacking DeFi. *arXiv preprint arXiv:2002.08099* (2020).
[26] Thomas Shababi Jacques Dafflon, Jordi Baylina. 2017. ERC777 Standard. https://eips.ethereum.org/EIPS/eip-777. [Online; accessed August-2021].
[27] Bo Jiang, Ye Liu, and WK Chan. 2018. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 259–269.
[28] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts.. In *Ndss*. 1–12.
[29] Nik Kunkel. 2020. https://twitter.com/nomos_paradox/status/1299215849018937345. [Online; accessed August-2021].
[30] Yudi Levi. 2020. Bancor Incident. https://blog.bancor.network/bancors-response-to-today-s-smart-contract-vulnerability-dc888c589fe4. [Online; accessed August-2021].

[31] Martin Lundfall. 2020. ERC777 Standard. https://eips.ethereum.org/EIPS/eip-2612. [Online; accessed August-2021].
[32] 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*. 254–269.
[33] Alex Manuskin. 2020. Unicat Incident. https://www.zengo.com/unicats-go-phishing/. [Online; accessed August-2021].
[34] Multichain. 2022. Multichain Incident. https://twitter.com/MultichainOrg/status/1483733455296860160. [Online; accessed Jan-2022].
[35] Daniel Perez and Ben Livshits. 2021. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.
[36] Primitive. 2021. DeFi Saver Incident. https://primitivefinance.medium.com/postmortem-on-the-primitive-finance-whitehack-of-february-21st-2021-17446c0f3122. [Online; accessed August-2021].
[37] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2020. Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. *arXiv preprint arXiv:2003.03810* (2020).
[38] Reza Rahimian, Shayan Eskandari, and Jeremy Clark. 2019. Resolving the multiple withdrawal attack on erc20 tokens. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 320–329.
[39] Michael Rodler, Wenting Li, Ghassan O Karame, and Lucas Davi. 2018. Sereum: Protecting existing smart contracts against re-entrancy attacks. *arXiv preprint arXiv:1812.05934* (2018).
[40] Clara Schneidewind, Ilya Grishchenko, Markus Scherer, and Matteo Maffei. 2020. eThor: Practical and provably sound static analysis of Ethereum smart contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 621–640.
[41] DeFi Saver Team. 2020. DeFi Saver Incident. https://medium.com/defi-saver/disclosing-a-recently-discovered-exchange-vulnerability-fcd0b61edffe. [Online; accessed August-2021].
[42] The 'Etherscanners' Team. [n.d.]. Etherscan. https://etherscan.io/. [Online; accessed August-2021].
[43] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 67–82.
[44] Uniswap. 2020. UNI Token. https://uniswap.org/blog/uni/. [Online; accessed August-2021].
[45] Uniswap. 2020. Uniswap V2. https://uniswap.org/. [Online; accessed August-2021].
[46] Trust Wallet. 2017. Trust Wallet. https://trustwallet.com/. [Online; accessed August-2021].
[47] Dabao Wang, Siwei Wu, Ziling Lin, Lei Wu, Xingliang Yuan, Yajin Zhou, Haoyu Wang, and Kui Ren. 2020. Towards understanding flash loan and its applications in defi ecosystem. *arXiv preprint arXiv:2010.12252* (2020).
[48] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. 2021. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778* (2021).
[49] Siwei Wu, Dabao Wang, Jianting He, Yajin Zhou, Lei Wu, Xingliang Yuan, Qinming He, and Kui Ren. 2021. DeFiRanger: Detecting Price Manipulation Attacks on DeFi Applications. *arXiv preprint arXiv:2104.15068* (2021).
[50] Valentin Wüstholz and Maria Christakis. 2020. Harvey: A greybox fuzzer for smart contracts. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1398–1409.
[51] Pengcheng Xia, Haoyu Wang, Bingyu Gao, Weihang Su, Zhou Yu, Xiapu Luo, Chao Zhang, Xusheng Xiao, and Guoai Xu. 2021. Trade or Trick? Detecting and Characterizing Scam Tokens on Uniswap Decentralized Exchange. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–26.
[52] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2020. High-Frequency Trading on Decentralized On-Chain Exchanges. arXiv:2009.14021 [cs.CR]

A   THE ADOPTION OF EIP2612

As aforementioned, with `permit` function, users can delegate DApps to spend their tokens with a customized amount of tokens in one transaction. The proposal can help users save gas fees from sending additional approval transaction and mitigate the risk of unlimited approval. However, as shown in Figure 11, DAI and UNI tokens force users to approve unlimited tokens in the `permit` function according to their newest smart contract.

B   THE DEMONSTRATION OF MITIGATING THE RISK OF UNLIMITED APPROVAL

Figure 12 demonstrates a real-world case to present the good practice using unlimited approval. There are two rounds of spending approved tokens by users. The time interval between the two rounds is around 5 days. In the first round, the user first sends an unlimited approval transaction to delegate his or her *USDT* to *Uniswap V2 Router*. Then, the following two execution transactions spend *5567* and *5567.3* USDT on *Uniswap V2*. Finally, the user sends a zero approval transaction to revoke his or her approved tokens. The whole process takes about 11 hours to complete. The 11 hours is the time difference between the unlimited approval transaction $\mathcal{A}_1$ and the zero approval transaction $\mathcal{A}_2$. Similarly, it takes about 2 hours for the user to complete his or her second round of spending approved tokens.

(a) The code of the token DAI.

(b) The code of the token UNI.

Figure 11: The adoption of EIP2612.

~ 11 hrs

~ 2 hrs

$\mathcal{A}_1$
Block: 11254351
Nonce: 187
Amount: **Unlimited**

$\mathcal{E}_1$
Block: 11254362
Nonce: 188
Amount: **5567.0**

$\mathcal{E}_2$
Block: 11254370
Nonce: 188
Amount: **5567.3**

$\mathcal{A}_2$
Block: 11257349
Nonce: 204
Amount: **Zero**

$\mathcal{A}_3$
Block: 11290771
Nonce: 218
Amount: **Unlimited**

$\mathcal{E}_3$
Block: 11290778
Nonce: 219
Amount: **11124**

$\mathcal{A}_4$
Block: 11291309
Nonce: 233
Amount: **Zero**

$$\begin{pmatrix} \mathcal{U} \\ \mathcal{S} \\ \mathcal{T} \end{pmatrix} = \begin{pmatrix} \text{0x0af9231ba6d097e2600d2763e449d2bec8a46c47} \\ \text{0x7a250d5630b4cf539739df2c5dacb4c659f2488d} \\ \text{0xdac17f958d2ee523a2206206994597c13d831ec7} \end{pmatrix}$$

$\mathcal{T}.allowance[\mathcal{U}][\mathcal{S}] = 0$

No Risk

$\mathcal{S}$: Uniswap V2 Router; $\mathcal{T}$: USDT;

Figure 12: An example of mitigating the risk of using unlimited approval.