Demystifying Web3 Centralization: The Case of Off-Chain NFT Hijacking

Felix Stöger[1], Anxin Zhou[2], Huayi Duan[1], and Adrian Perrig[1]

[1] ETH Zürich, Zürich, Switzerland
[2] City University of Hong Kong, Kowloon, HKSAR

Abstract. Despite the ambitious vision of re-decentralizing the Web as we know it, the Web3 movement is facing many hurdles of centralization which seem insurmountable in the near future, and the security implications of centralization remain largely unexplored. Using non-fungible tokens (NFTs) as a case study, we conduct a systematic analysis of the threats posed by centralized entities in the current Web3 ecosystem. Our findings are concerning: *almost every interaction between a user and a centralized entity can be exploited to hijack NFTs or cryptocurrencies from the user, through network attacks practical today.* We show that many big players in the ecosystem are vulnerable to such attacks, placing large financial investments at risk. Our study is a starting point to study the pervasive centralization issues in the shifting Web3 landscape.

1 Introduction

We are witnessing a trend of re-decentralizing web services provided by big corporations, which is portrayed as the transition from Web2 to Web3. In this envisioned Web3 paradigm, decentralized applications (DApps) are hosted on blockchains and other distributed infrastructures, without relying on any single entity for their governance and operation [23]. However, most DApps today deviate from this idealized model and, somewhat inevitably, employ centralized components for cost efficiency, performance, and usability. This creates profitable targets for attackers as observed in many real-world incidents [25,12,11]. Such architecture-level attack surface has not received equal attention from the research community compared with vulnerabilities in the underlying blockchain protocols [28,18], as we further discuss in Section 2. It is important and urgent to fill this gap, given the prevalence of centralized entities in the current Web3 ecosystem and their complex interactions with other parties.

We initiate a systematic study of the security issues *induced by centralization* in Web3, focusing on the sub-ecosystem around non-fungible tokens (NFTs). The reason for choosing NFTs as our subject of study is three-fold. First, they are among the most popular Web3 concepts with a multi-billion dollar market [31]. Second, NFTs establish the fundamental and ubiquitous notion of asset ownership, and therefore they will likely persist even if high market valuations decline. After the initial standard [17], the Ethereum community has proposed a series of improvements to bring NFTs closer to a practical realm from usability [3]

and legal perspectives [20]. Last but not the least, the NFT sub-ecosystem is sufficient to demonstrate common centralization issues, as it involves different centralized entities that interact with users and decentralized infrastructures in various ways. We are particularly interested in the security risks arising from such interactions, as they should not exist in a fully decentralized architecture.

Our work starts with the definition of a functional model that captures the essential entities in today's NFT ecosystem and their dynamics for the creation, tracking, and trading of NFTs. Instantiating this model with concrete architectures that employ different forms of centralization, we systematically examine vulnerable interactions that can be exploited through practical network attacks such as BGP or DNS hijacking. As a result, we find that *almost every interaction of a user with a centralized entity leads to an attack that can hijack NFTs or the associated cryptocurrencies*. Such hijacking is *off-chain* in that it involves no exploitation of the underlying blockchain or smart contracts. We also examine the detectability of these attacks. Some of them can be detected and prevented by prudent inspection of transaction parameters, whereas others require end-to-end data authentication in a decentralized architecture. We have validated most of our proposed attacks on OpenSea and a Ethereum testnet. Furthermore, our analysis of real-world service providers show that 6 out of 10 top NFT marketplaces and many other intermediary services are vulnerable.

Our study of NFTs is just a starting point to investigate the centralization risks in the broad and shifting Web3 ecosystem. The methodology we developed in this work is also applicable to analyzing DApps beyond NFTs.

2 Related Work

Research on NFT security. Marlinspike [26] points out that DApps are not as decentralized as claimed because of their reliance on centralized servers. These servers can return arbitrary NFT-associated data to users, and marketplaces like OpenSea can unilaterally remove NFTs from their listings. This indicates a clear violation of DApps's fundamental principle that their operation should not be influenced by any centralized authority. Das et al. [15] examine today's NFT ecosystem and several security issues therein, including insufficient user authentication and unverified smart contracts, lack of persistent asset data storage, and trader malpractices. Wang et al. [33] measure the risks of disconnection between NFTs and their off-chain assets. Unlike these prior studies that discusses issues arising from (centralized) entities themselves, we inspect architecture-level vulnerabilities rooted in the *extra interactions* induced by centralized entities, and our attacks work *even if these entities themselves remain uncompromised.*

Attacks on DApps. Su et al. [30] analyze common transaction patterns of DApp attacks and develop a tool to automatically identify security incidents. Such attacks exploit design or implementation flaws in smart contracts and thus are orthogonal to the network-based attacks we consider.

As a major class of DApps, decentralized finance (DeFi) aims to remove traditional financial institutions like banks and exchanges. The transparency

and high transaction latency of blockchains makes DeFi services subject to, e.g., front running [14] and sandwiching attacks [34]. Because of their reliance on centralized components like web servers and blockchain gateways, real-world DeFi platforms are also susceptible to the attacks presented in this paper.

Recently, Wang et al. [32] quantify the security risks of unlimited approval of ERC20 tokens that fuel many DApps. Some of our attacks also exploit the fact that trading NFTs requires their owners to delegate control to marketplaces.

Li et al. [24] find that centralized intermediary services used by DApps can be turned into attack vectors for denial of service (DoS). This demonstrates the risk of centralization from another interesting angle.

Blockchain Security. Many attacks on blockchains at the consensus [28,18] or network layer [4,21] have been discovered. In comparison, our work explores a new class of security threats arising from external entities which are not part of a blockchain but widely exist for practical reasons. Programming errors in smart contracts can often lead to vulnerabilities [5]. Different tools have been developed to find such security bugs [27,22]. These tools, however, cannot detect our attacks because we do not exploit flaws in smart contracts themselves.

Network Attacks. The Internet's core building blocks, including the Border Gateway Protocol (BGP) for inter-domain routing and the Domain Name System (DNS) for name resolution, are not secure by design. In a BGP hijacking attack, the attacker can maliciously announce the IP prefix of an autonomous system (AS) and thereby hijack its inbound network traffic; as for DNS, an off-path attacker can inject bogus data into a DNS server's cache and direct clients to malicious servers. These attacks in turn allow the subversion of a wide range of online systems [13], including public key infrastructures (PKIs) which underpin the widely deployed Transport Layer Security (TLS) protocol [29]. Unfortunately, security extensions to BGP and DNS, e.g., RPKI, BGPsec, and DNSSEC, have not received widespread deployment [19,10]. Therefore, network-based attacks are still practical and prevalent in today's Internet.

3 Modeling NFT Functionality

Despite the variety of entities in the current NFT ecosystem, they implement a common set of functions revolving around the creation, tracking, and trading of NFTs. We define a functional model to capture these essential functions and then instantiate it with concrete architectures for detailed security analyses. As depicted in Figure 1, our model contains three types of users that interact with three services—ownership registry, asset storage, and NFT marketplace (NFTM)—via predefined interfaces. This model captures the typical life cycle of NFTs seen today, allowing us to systematically uncover vulnerable interactions involving different centralized entities.

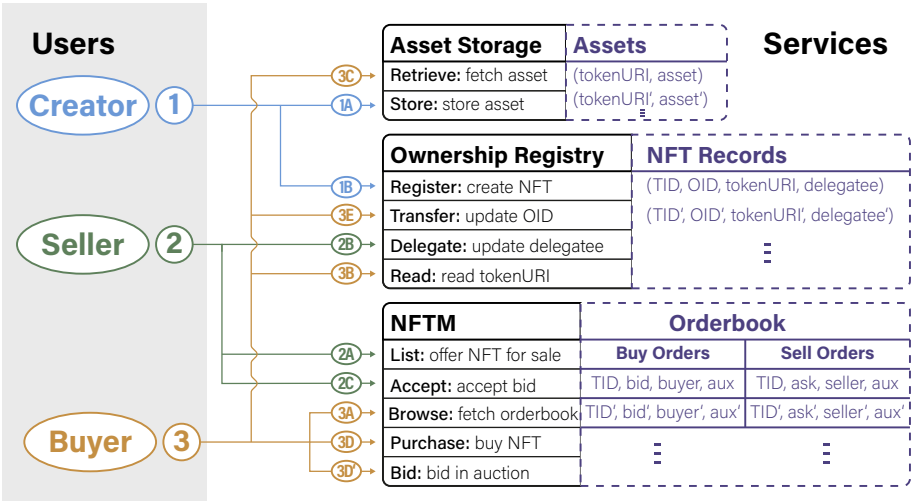


Fig. 1: Our model to capture the essential interactions (arrows) between different users (left) and services (right) commonly found in the current NFT ecosystem.

3.1 Data and Interfaces

Each service exposes a set of interfaces for users to access and modify its data (illustrated in the dashed boxes in Figure 1).

Ownership Registry. In essence, NFTs are ownership records of digital or physical assets, and these records are stored in a (ideally) permanent ownership registry. Each record is defined by 4 fields. The first and foremost is `TID`, which uniquely identifies an NFT; in practice, this is implemented by pairing a globally unique smart contract address and a locally unique token index. The other three fields are: `OID` identifying the token's owner, `tokenURI` pointing to the underlying asset, and `delegatee` identifying an entity who can control the token on behalf of its owner. The registry exposes four interfaces: (1) `Register` to create a new record with all fields except `delegatee` properly initialized to non-empty values, (2) `Transfer` to change the owner of a token by updating its `OID` and clearing the `delegatee` field, (3) `Delegate` to set the delegatee for a token, and (4) `Read` to retrieve a record for a given `TID`. We explain several technicalities below.

The NFT standard EIP-721 [17] specifies only the `Transfer` and `Delegate` functions in our model. Our additional `Register` and `Read` explicitly describe the actions to (1) create NFTs and (2) read NFTs from the ownership registry. This allows us to identify subtle vulnerabilities that otherwise stay concealed. Another remark is that, in practice, `tokenURI` might not reference the asset directly, but instead a separately stored metadata object that contains a further pointer to the actual asset. This extra layer of indirection may increase the attack surface as well, but we refrain from overcomplicating our model with this subordinate interaction whose functions are already subsumed by the major interfaces. We

further assume that once an NFT is created, its `tokenURI` cannot be updated. Finally, we do not consider the possible destruction of an NFT.

Asset Storage. The underlying data associated with NFTs is maintained in an asset storage. We consider a simple yet realistic storage model consisting of (`tokenURI`, `asset`) pairs and two interfaces to store and retrieve assets. Unlike the ownership registry that is hosted on a blockchain by default, practical asset storage systems are almost always off-chain for cost and performance reasons.

NFTM. NFTs must be tradable to create value. This necessitates an NFTM that connect buyers and sellers. The essential data maintained by an NFTM is an orderbook that keeps track of sell and buy orders. Each order contains a token identifier, a bid/ask price, the order's issuer, and some marketplace-specific auxiliary information (e.g., sale duration) not relevant to our security analyses. An NFTM provides 5 interfaces to users: (1) `List` for a seller to offer a token for sale, (2) `Accept` for a seller to accept a buy order or a bid, (3) `Browse` for users to read the catalog of tokens for sale, (4) `Purchase` for a buyer to buy a listed token, and (5) `Bid` for a buyer to bid on a token in auction. The NFTM updates its orderbook according to these actions and process a ownership transfer transaction whenever a buy order matches a sell order.

3.2 NFT Life Cycle

Users can take three roles: creator, seller, or buyer. We describe a typical NFT life cycle through users' interactions with the necessary services. We use the notation (X) `action(in` $\rightarrow$ `out)` to represent the invocation of an interface `action`, which takes `in` as input from and returns `out` to the caller. The `in` or `out` parameters can be empty. We also omit non-critical data in some invocations.

Creation. The creator of an NFT can vary, for example an artist creating the digital asset, or a party entrusted by the asset creator with the task of tokenizing the asset. We do not distinguish these cases. To start with, the creator uploads an asset to the asset storage by calling (1A) `Store(asset` $\rightarrow$ `tokenURI)`. With the returned `tokenURI`, it then creates a token by calling (1B) `Register({OID, tokenURI}` $\rightarrow$ `TID)`, which stores a new record in the ownership registry.

Listing. The owner of a token offers it for sale through (2A) `List({TID, OID, ask}` $\rightarrow$). The invoked marketplace needs permission to transfer the token without the seller's further involvement. This is done via (2B) `Delegate({TID, Mkt}` $\rightarrow$), which sets the `delegatee` of token `TID` as the marketplace identified by `Mkt`.

Trading. A buyer interacts with all three service providers to buy an NFT. It starts by retrieving available sell orders from the marketplace via (3A) `Browse(`$\rightarrow$ `{TID, ask, seller})`. Here we assume only a single sell order is returned. To examine the associated asset, the buyer first gets the token's metadata by calling (3B) `Read(TID`$\rightarrow$ `tokenURI)` from the ownership registry and then fetches the asset by calling (3C) `Retrieve(tokenURI`$\rightarrow$ `asset)` from the storage provider. Marketplaces normally offer two buying options: direct purchase or auction. In the former case, the buyer directly offers the asked price and calls (3D) `Purchase({TID,`

ask, buyer} →). In the latter case, the buyer places a bid via (3D') Bid({TID, bid, buyer} →), which results in a buy order stored in the NFTM's orderbook. Upon a successful sale, the NFTM transfers the token to the new owner by calling (3E) Transfer({TID, buyer} →) without the seller's involvement. This is legitimate because the NFTM has been approved by the seller in advance.

4 System Architecture & Attack Taxonomy

We consider four instantiations of our NFT model that exist (or could exist) in practice. The first one is a fully decentralized architecture where all services are hosted by decentralized infrastructures and all users acces them through their own infrastructure nodes. Each subsequent architecture centralizes one service—that is, the service is either accessed by users through a centralized intermediary (CI), or otherwise hosted centrally and controlled by a single entity. Such centralization creates vulnerable links that can be intercepted to manipulate data communicated between users and services, enabling various attacks to hijack NFTs or cryptocurrencies from different users as summarized in Table 1.

Threat Model. We consider an off-path network adversary who is capable of intercepting communication between a user and a centralized entity through BGP or DNS hijacking. Even if the communication is secured by TLS, the adversary can still acquire a fraudulent certificate to impersonate the victim domain owned by a centralized entity [29]. We assume that decentralized infrastructures themselves, including blockchains and decentralized storage systems, are secure against the adversary, and that their data is always tamper-proof. For example, the adversary cannot attack their underlying consensus mechanisms [18,28] or prevent their users from retrieving data from honest nodes.

4.1 Architecture Type I: Fully Decentralized

In a fully decentralized architecture, the ownership registry and NFTM functions are implemented by smart contracts. Users access these services by sending blockchain transactions that encode function calls to these contracts. The asset storage can be on a blockchain or an off-chain storage system like IPFS or StorJ. We focus on IPFS as it is the de-facto standard decentralized storage system used by many DApps. In IPFS, a file is indexed by a content identifier (CID), a cryptographic token used to retrieve the file and verify its integrity.

Users in this architecture rely on their own blockchain and optionally IPFS nodes to access different services. We do not distinguish between a full blockchain node and a light client [8], because both of them allow a user to verify on-chain data. This idealized (yet still practical!) architecture requires users to use some specialized explorer software to retrieve, organize and render data (e.g., NFTM listings and orderbook) via their local nodes, without depending on any external web services that are prevalent today.

Security. In this architecture, users can locally validate all data they receive from the three services and all actions they perform. In interactions (3A) and

Table 1: A summary of potential NFT hijacking attacks in different architectures. The third column indicates whether and how a user can detect attack attempts. Level 1: the user must carefully audit the transaction parameters to be signed to detect an attack attempt. Level 2: the user must obtain some authenticator (e.g., cryptographic digest or digital signature) for the relevant data (received from a centralized entity) in a secure way (e.g., through a decentralized infrastructure that the user is part of) and verify the data to detect an attack attempt. Level 3: the victim can detect the attacks only retrospectively after it notices the financial loss.

Attacks	Architecture	Detectability	Outcome
$\mathcal{A}$**1**: (1A)	Type II	2/3*	NFT created with attacker-controlled asset
$\mathcal{A}$**2**: (3C)	Type II	2/3*	Wrong NFT bought by buyer
$\mathcal{A}$**3**: (1C)	Type III	3	Royalty paid to attacker
$\mathcal{A}$**4**: (2A)	Type III	1	NFT sold to attacker at a low price
$\mathcal{A}$**5**: (2B)	Type III	1	NFT transferred to attacker
$\mathcal{A}$**6**: (3A)	Type III	2/3**	Increased chance to buy attacker's NFT Higher bids placed by buyer than necessary
$\mathcal{A}$**7**: (3B')	Type III	2	Wrong NFT bought by buyer
$\mathcal{A}$**8**: (3C')	Type III	2	Wrong NFT bought by buyer
$\mathcal{A}$**9**: (3D)	Type III	1	Funds stolen from buyer Wrong NFT bought by buyer
$\mathcal{A}$**10**: (3D')	Type III	1	Buyer's bid amount increased by attacker
$\mathcal{A}$**11**: (2B)	Type III	1	Funds stolen from buyer
$\mathcal{A}$**12**: (3B):	Type IV	3	Wrong NFT bought by buyer

* 2 if IPFS or blockchain, 3 if centralized storage

** 2 if on-chain orderbook, 3 if centralized access or off-chain orderbook

(3B), a user can read integrity-protected data from the blockchain through its local node. Interactions (1B), (3E), (2B), (2A), (2C), (3D), and (3D') are implemented by blockchain transactions cryptographically signed with the user's private key and so they cannot be tampered with. If the asset storage is on-chain, users can verify the integrity of assets in interaction (1A) and (3C); in the case of IPFS, users can also verify retrieved assets using their CIDs. To conclude, this architecture exposes no extra user interaction that can be exploited by our network attacker. Even if an attacker can intercept the communication between a user and other nodes in a decentralized infrastructure, it cannot alter the data undetected thanks to the infrastructure's built-in end-to-end data authentication.

4.2 Architecture Type II: Centralized Asset Storage

To lower the barriers to entry and reduce operational costs (e.g., the high gas fee in Ethereum), NFT participants in practice offer and use the services defined by our model in various centralized forms. We start by analyzing the asset storage.

Centralized Access. Even if many NFT assets nowadays are stored on a decentralized infrastructure by default, most users access them through Blockchain as

a Service (BaaS) providers (e.g., Infura) or IPFS gateways [1]. These CIs provide convenient APIs as a service for users to access a decentralized infrastructure without running their own nodes. As the price of such convenience, however, users must trust these CIs for the authenticity of any received asset data.

For the case of IPFS, it may appear that end-to-end data authentication is still possible with assets' CIDs. However, data integrity verification is rarely implemented outside IPFS nodes. Moreover, IPFS gateways normally do not provide all the parameters[3] needed for data verification to users.

Centralized Hosting. Despite the pursuit of decentralization by NFT participants, it is not uncommon that NFT assets are hosted directly on centralized systems [26]. For example, the Otherside project (`otherside.xyz`), whose NFTs have a sale volume of over 600K ETH, stores asset files on traditional web servers. Unlike decentralized storage, these centralized systems (including cloud storage) provide only simple checksum mechanisms for the detection of data corruption at best. An attacker capable of manipulating a file in transit can also forge its checksum[4]. Hereafter, we assume that end users cannot verify the authenticity of data received from a centralized storage system.

Attacks. We present attacks against interactions (1A) and (3C) (see Figure 1). Both attacks arise from the loss of data verifiability due to centralized access or hosting of asset storage. We highlight the data modified by each attack in red.

$\mathcal{A}$**1:** `Store`(`asset` → `tokenURI`). This attack aims to trick a creator into associating a new NFT with an unexpected asset. Specifically, the attacker can intercept the creator's communication with a CI for decentralized asset storage or directly with a centrally hosted asset storage, and then surreptitiously modify the returned `tokenURI`. As a result, the `tokenURI` included by the creator in a subsequent call to `Register` will reference an attacker-chosen asset.

A creator should normally delete its local copy of the asset file only after the file is uploaded successfully via `Store`(). This gives the creator chances to validate a received `tokenURI`, but such validation is indeed futile. For two of the three architecture variants, centralized access to on-chain assets and centrally hosted asset storage, the attacker can again intercept the creator's validation attempt to retrieve the asset file indicated by the fake `tokenURI`, misleading the creator with a false sense of security. For the case of centralized access to decentralized storage, the aforementioned limitations of IPFS mean that data verification (using CID) by end users is still not practical.

$\mathcal{A}$**2:** `Retrieve`(`tokenURI`→ `asset`). This attack deceives a buyer into purchasing a low-value NFT sold by the attacker, mistakenly believing it to be a high-value one. NFTs by the same creator are generally organized into a collection (e.g., Bored Ape Yacht Club) and have varying values. When a target buyer retrieves an attacker's "bait" asset from a CI or centrally hosted asset storage in interaction (3C), the attacker can intercept the communication and substitute the original

[3] For example, the file chunk size that influences the calculation of CID.

[4] Note that a user in our model can retrieve an asset file's identifier and authenticator (e.g., a CID or digital signature) from a secure decentralized infrastructure.

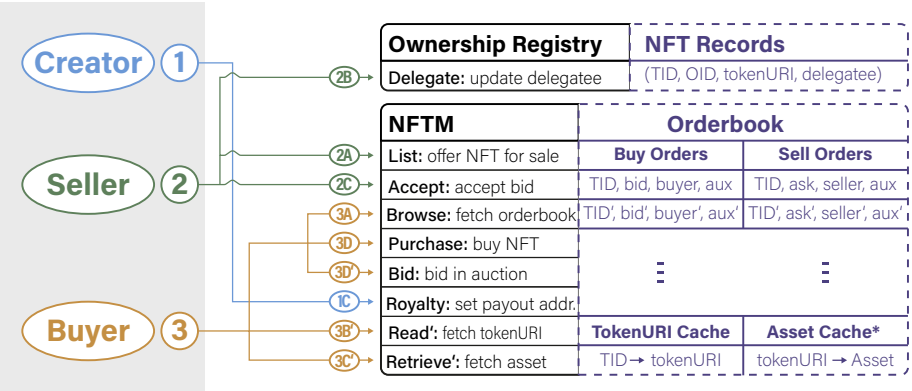


Fig. 2: Centrally hosted NFTM Architecture. The Asset Cache* is optional.

dull asset with a more appealing one from the same collection. The buyer may end up investing the dull asset at a much higher price than necessary. The lack of end-user data verification in the architecture under discussion means that such an attack is hard to detect from a victim user's point of view.

Note that this attack is different from a simple counterfeit NFT where the attacker registers its own NFT with the same `tokenURI` as an expensive, legitimate NFT. Counterfeit NFTs are easily detected because they are not part of the same collection as the genuine NFTs.

4.3 Architecture Type III: Centralized NFTM

At the hub of any NFT ecosystem are versatile marketplaces that bridge all other players. Because of its complex interactions with other entities, this service's centralization can pose most security risks, as explained in this section. Similarly to our previous analysis, we consider two variants of centralized NFTM.

Centralized Access. Developing a full-fledged on-chain NFTM is challenging and inefficient, but it is possible to implement the core functions, including the maintenance of an orderbook and the matching and execution of orders, solely with smart contracts. Few NFMTs are of this type. One example is CryptoPunk, which is dedicated to one single collection of NFTs. Even with such a minimal on-chain NFTM, a non-expert user still needs BaaS CIs to access the underlying blockchain and must trust them for any received trading data.

Centralized Hosting. Most NFTMs today adopt this architecture (e.g., all of the top 10 listed on `DappRadar.com`). They implement the marketplace service as traditional web applications, allowing average users to manage and trade NFTs with ease. To facilitate our analysis, we assume that such an application, which typically consists of a web server and a database among many other components, is hosted entirely by a single ***marketplace server*** (MS).

The MS implements a user-facing storefront to simplify user interactions, stores the orderbook, provides users with buy- and sell order parameters to be

signed, caches the `tokenURI` (replacing interaction (3B) with (3B')), and optionally also caches the asset itself (replacing interaction (3C) with (3C')). Nevertheless, the core functionality of buy- and sell order matching is still implemented in an NFTM smart contract. NFTMs can optionally also replicate the orderbook on-chain for improved auditability by the user.

Our classification of centrally hosted NFTMs is complementary to prior research [15], which classifies them according to which operations are implemented off-chain. We however do not consider fully centralized NFTMs which implement order matching off-chain.

NFTMs also allow NFT creators to earn upon secondary sales of their tokens by deducting a fraction of the sale price as royalties paid to the creator. EIP-2981 [7] describes an on-chain royalty mechanism, though it is not universally supported by NFTs and NFTMs. In its absence, NFTMs rely on proprietary mechanisms. We consider a common approach where the NFTM stores the creator's royalty payout address on the MS [15].

Attacks. The above architectures create the largest attack surface among all forms of centralization. We identify attacks exploiting 9 different interactions. All of them apply to centrally hosted marketplaces and the attack $\mathcal{A}$**6** also applies to centrally accessed marketplaces.

$\mathcal{A}$**3:** `SetRoyalties({addr, amount}` $\rightarrow$`)`. This attack targets royalty mechanisms implemented by NFTMs where the payout address is stored exclusively on the MS. These NFTMs allow logged-in[5] NFT creators to change their payout address without additional authentication. By intercepting and replacing the `addr` parameter sent by the creator to the MS as part of `SetRoyalties` with its own address, an attacker will receive royalty payments from future sales of the NFT. The attack will likely remain unnoticed until a sufficient amount of royalties are siphoned off.

$\mathcal{A}$**4:** `List`({`TID`, `OID`, `ask`} $\rightarrow$). This attack tricks the seller into selling an NFT to the attacker at an attacker-chosen price. To list an NFT for sale, the seller should cryptographically sign a sell order with its blockchain private key. The order's parameters {`TID`, `OID`, `ask`} are provided by the MS as part of (2A) `List`. Intercepting the interaction and reducing the `ask` parameter results in the NFT being listed at a lower price than expected by the seller. This attack can be detected and prevented if the seller carefully audits the sell order's parameters before signing it. However, most crypto wallets fail to display transaction data in a structured and comprehensible way (beyond raw hex data), making transaction auditing difficult for unsophisticated users.

$\mathcal{A}$**5:** `Delegate`({`TID`, `Mkt`} $\rightarrow$). This attack allows an attacker to directly steal a seller's NFTs. Recall that a seller should authorize an NFTM the right to transfer a listed NFT after a successful sale, by making the latter the NFT's `delegatee`. The `Delegate` call occurs right after the call to `List`, and similarly to the listing operation, the MS provides the corresponding parameters {`TID`,

[5] In the case of OpenSea, a user maintains a logged-in status if it cryptographically signed a "login-in" challenge in the last 24h.

`Mkt`} to the seller. An intercepting attacker can change `Mkt` to make itself the delegatee and transfer the NFT to an account under its control. The detection of this attack is also similar to $\mathcal{A}$**4:**, but `Mkt` being a pseudorandom value further complicates manual transaction auditing.

$\mathcal{A}$**6:** `Browse`(→ {`TID`, `ask`, `seller`}). This attack tricks the buyer into perceiving the attacker's NFT offerings as better value than they actually are. It applies to both centrally hosted and CI-accessed decentralized NFTMs. The buyer retrieves a `{TID,ask,seller}` triplet for every NFT it browses from the NFTM's orderbook stored on the MS. An attacker can increase the `ask` value of competing offerings, which makes the buyer perceive the attacker's offerings as better value. This attack can only be detected if a copy of the orderbook is stored on-chain, to which the buyer has untampered access. Centrally accessed NFTMs are also susceptible to this attack because the attacker can intercept and modify the triplet provided by the BaaS CI.

$\mathcal{A}$**7:** `Read'`(`TID`→ `tokenURI`). Similarly to $\mathcal{A}$**2**, this attack misleads buyers into purchasing low-value fraudulent NFTs. A buyer retrieves a `tokenURI` through `Read'` for every `TID` retrieved previously through `Browse`. Intercepting and modifying the `tokenURI` can cause the buyer to fetch unexpected assets in the subsequent calls to `Retrieve` or `Retrieve'`. The buyer is thus tricked into associating a low-value, attacker-owned NFT with the asset of a more valuable NFT. If the buyer has untampered access to the ownership registry, it can detected the attack by comparing the retrieved `tokenURI` against that stored in the registry.

$\mathcal{A}$**8:** `Retrieve'`(`tokenURI`→ `asset`). This attack also tricks the buyer into associating an attacker-owned NFT with another asset. It is functionally equivalent to $\mathcal{A}$**2** and $\mathcal{A}$**7**, except that it targets NFTMs that cache assets on the MS instead of centralized asset storage.

$\mathcal{A}$**9:** `Purchase`({`TID`, `ask`, `buyer`} →). We describe two attacks against the `Purchase` interaction. The first deceives the buyer into purchasing an attacker-chosen NFT instead of the intended one. The second redirects the funds intended for purchasing an NFT to the attacker.

To purchase an NFT, the buyer signs a buy order blockchain transaction whose parameters {`TID`, `ask`, `buyer`} are provided by the MS, similarly to the sell order in $\mathcal{A}$**4**. By changing the `TID` parameter to that of an NFT sold by the attacker, signing the buy order causes the buyer to purchase the attacker's NFT.

Each blockchain transaction has a destination address as an additional parameter, which in our scenario is provided by the MS as the NFTM smart contract address. In the second attack, the attacker intercepts and replaces this address with that of its own smart contract. This causes the buyer to unintentionally send the buy transaction with the included funds to the attacker.

A buyer can detect both attacks by careful audit the transactions.

$\mathcal{A}$**10:** `Bid`({`TID`, `bid`, `buyer`} →). We describe four attacks against NFT auctions. The first attack tricks the buyer into bidding on an attacker-offered NFT, the second into placing unnecessarily high bids, the third into redirecting funds

intended for purchasing an NFT to the attacker, and the fourth into bidding an attacker-chosen amount.

The first attack is functionally identical to the first attack of $\mathcal{A}$**9**, except that the attacker exploits the `Bid` function instead of `Purchase`.

The second attack increases the current highest bid amount retrieved by the buyer from the MS. This coerces the buyer into bidding more than necessary to win the auction. Detection is only possible if a copy of the current bids is stored on-chain, against which the buyer can compare the amount provided by the MS.

The third and fourth attack are complementary, and they target NFTMs implementing `Bid` as blockchain transactions or as interactions with the MS respectively. In both cases, `Bid` parameters {`TID`, `bid`, `buyer`} are provided by the MS. If `Bid` is a blockchain transaction, the funds are transferred to the NFTM as part of the transaction and held there for the duration of the auction. Wallets prominently display the amount of cryptocurrency attached to the transaction, making modifications easy to detect. The attacker can, however, stealthily modify the blockchain destination address as outlined in second attack of $\mathcal{A}$**9** and thus extract the funds. If `Bid` is not a blockchain transaction, it is merely a signed commitment by the buyer to purchase the token upon winning the auction. As no cryptocurrency is transferred, the amount is not prominently displayed by wallets. Intercepting and increasing the `bid` tricks the buyer into bidding more than intended. Careful manual auditing can detect the attack.

$\mathcal{A}$**11:** `Delegate`({`TID`, `Mkt`} $\rightarrow$). In bidding protocols where the funds are only transferred upon winning the auction, the NFTM must have access to the bidder's account to execute the winning bid. This is achieved by the buyer delegating tokens equal in value to the bid amount to the NFTM. On Ethereum, wrapped Ether (wETH), a fungible ERC-20 token, is commonly used. Intercepting and replacing `Mkt` provided by the MS with the attacker's address tricks the buyer into delegating the tokens to the attacker. Detection requires manual auditing and knowledge of the expected `Mkt` value.

4.4 Architecture Type IV: Centralized Ownership Registry

For this architecture, we only consider an on-chain ownership registry accessed through a BaaS CI, as by design NFT records should be stored on a blockchain.

$\mathcal{A}$**12:** `Read`(`TID`$\rightarrow$ `tokenURI`). This attack is functionally equivalent to $\mathcal{A}$**7**, except that it targets a centrally accessed ownership registry instead of an NFTM.

5 Attack Validation

We have validated our attacks on real systems used in today's NFT ecosystem. We focus on the case of centralized NFTMs, because (1) they are common in the real world, (2) they normally subsume the functions of other CIs like BaaS and IPFS as discussed earlier, and (3) they allow us to simulate complete attack procedures from the preparation (deploying fraudulent contracts, creating and listing bait NFTs, etc.) to the production of final outcomes (see Table 1).

```python
class ChangeHTTPCode:
    def request(self, flow: http.HTTPFlow) -> None:
        graphql_id = "useCollectionFormEditMutation"
        request_url = flow.request.pretty_url
        raw_payload = flow.request.get_text(strict = False)
        if not ("graphql" in request_url and graphql_id in raw_payload):
            return
        new_creator_fees = [{"address": attacker_account, "basisPoints":500}]
        payload = json.loads(raw_payload)
        payload['variables']['input']['collectionInput']['creatorFees'] = new_creator_fees
        new_payload = json.dumps(payload)
        flow.request.set_text(new_payload)
addons = [ChangeHTTPCode()]
```

Fig. 3: Python code for changing royalty payout address with `mitmproxy`.

Setup. We simulate the attacks using OpenSea—the largest NFTM today—on the Ethereum Rinkeby testnet. For user-side software, we use the Firefox browser with the MetaMask crypto wallet. To simulate a man-in-the-middle attacker, we use the `mitmproxy` tool and install a self-signed CA certificate into Firefox's trust store. This setup allows us to route all Firefox traffic to `mitmproxy`, where we can intercept and decrypt the HTTPS requests or responses and modify different data fields according to the simulated attacks.

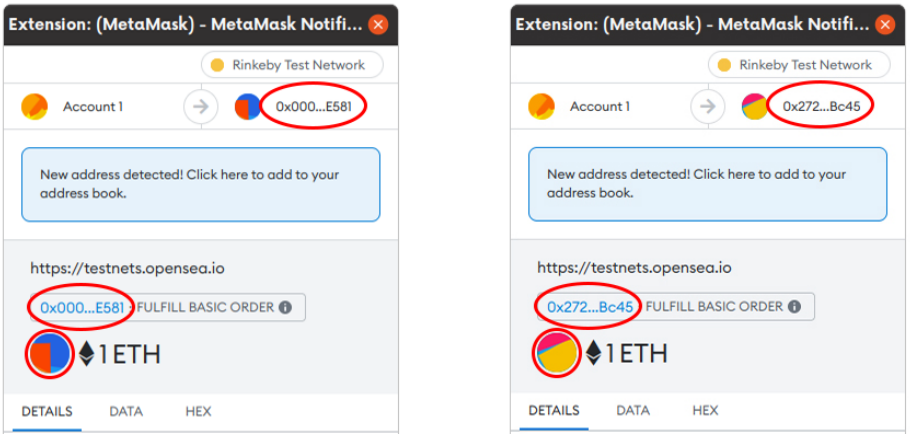
Validation Details. Our attacks can be conducted in two ways: (1) directly modify data exchanged between the victim and the MS, or (2) exploit third-party JavaScript (JS) loaded in the NFTM storefront. While the first approach targets the victim's connection to the MS, the second approach targets the victim's connection to third-party JS providers. By intercepting and maliciously modifying JS loaded from these providers, the attacker can execute arbitrary JS in the victim's browser. The attacks against centralized NFTMs are further categorized based on their methodologies.

- **Royalty payout address change:** $\mathcal{A}3$
- **Change order or transaction parameters:** $\mathcal{A}4$, $\mathcal{A}5$, $\mathcal{A}9$, $\mathcal{A}10$, $\mathcal{A}11$
- **Wrong data fetched from NFTM:** $\mathcal{A}6$, $\mathcal{A}7$, $\mathcal{A}8$

We briefly explain the validation of attacks $\mathcal{A}3$ and $\mathcal{A}9$, which are representative for their respective attack category. We have also successfully validated the third attack category via straightforward modifications to data fetched from the MS.

Royalty payout address change ($\mathcal{A}3$). In attack $\mathcal{A}3$, the attacker targets NFTMs that store the royalty payout address on their MS and replaces the creator's address with its own. In our validation, the creator submits a new payout address through OpenSea's user portal, and this is encoded as a GraphQL HTTPS request. We can intercept the request using `mitmproxy` and modify the address. Figure 3) shows the python code to implement the interception, including finding the request for changing the royalty payout address (lines 3-7) and replacing the address with an attacker's controlled account (lines 8-12).

Change order or transaction parameters ($\mathcal{A}9$). We have validated both versions of $\mathcal{A}9$. The first version causes the buyer to purchase an attacker-chosen



(a) Untampered transaction (b) Tampered transaction

Fig. 4: Screenshots of MetaMask showing a genuine and a fradulent transaction. The latter claims to be from the NFTM and has the correct function name. Signing this transaction causes 1 ETH sent to the attacker.

NFT, and the second directs the buyer's buy-transaction to a malicious smart contract that extracts the attached cryptocurrency.

For the first attack, we simulate a buyer's request for an NFT through the OpenSea website, which generates an HTTPS request containing the sell-order ID `orderId` to the GraphQL endpoint on the MS. We can intercept this request using `mitmproxy` and change `orderId` to another attacker-controlled order. The MS will respond with the unsigned buy transaction corresponding to the malicious `orderId` and thus cause the buyer to inadvertently buy the attacker's NFT. The modification of `orderId` is minor and hard to notice, unless the victim carefully compares the NFT information displayed on the OpeaSea website and the parameters contained in the transaction to sign, which requires manual decoding of the raw, hex-encoded transaction data in MetaMask.

For the second attack, the attacker first deploys a malicious smart contract which extracts cryptocurrency from received transactions. When the buyer requests to buy an NFT, the attacker intercepts and replaces the `destination` field in the transaction data provided by the MS with that of its malicious smart contract. As seen in Figure 4, the attack only causes visible changes to the destination address field and the icon associated with the address (both are pseudorandom values). MetaMask still displays the genuine OpenSea URL and the prominently displayed cryptocurrency amount is unchanged.

Ethical Consideration. Our experiments create test smart contracts and NFTs on a public blockchain intended for testing, including security research. They incur no real costs, even if tokens are accidentally purchased by other users. The test assets uploaded to IPFS disappear after some time if they are not cached by any node. Note that our attacks do not exploit the service providers'

Table 2: The susceptibility of the top 10 NFTMs (according to DappRadar.com) to prefix hijack attack. "Decentralized" architecture allow trading without MS. We mark MSes loading JavaScript from JavaScript providers vulnerable to subprefix hijacking as "JS." An aggregator NFTM collects sell order from other NFTMs. Its security thus depends on the security of the queried NFTMs.

NFTM	Architecture	Interception Possible	Vulnerable JS Provider	Vulnerability
OpenSea	MS, off-chain	No	-	-
CryptoPunks	Decentralized MS, on-chain	No	-	-
LooksRare	MS, off-chain	JS	cdn.jsdelivr.net	Max-Len
X2Y2	-	No		
Rarible	MS, off-chain	JS	static.klaviyo.com	No ROA
SuperRare	Decentralized Central MS	JS	cdn.heapanalytics.com	Max-Len
Foundation	Decentralized MS, on-chain	JS	cdn.segment.com	Max-Len
Decentraland	-	JS	cdn.segment.com	Max-Len
Element	Aggregator	No		
Golom	-	Yes/JS		Max-Len

systems but generic network vulnerabilities in centralized architectures. Service providers often consider man-in-the-middle attacks beyond their responsibility. We reported our findings through OpenSea's bug bounty program [2] and received the confirmation that our attacks are not within its scope.

6 Vulnerabilities of Real-world Entities

We analyze the susceptibility of popular real-world NFTMs and CIs to our attacks, by examining the BGP security of their networks. Specifically, we inspect whether they are vulnerable to subprefix hijacking, a practical and highly effective form of BGP hijacking attack that happens frequently in today's Internet. Using an open Internet data platform RIPEstat, we collected relevant information such as publicly announced IP prefix, source AS, and route origin attestation (ROA). The IP prefix of an entity is deemed vulnerable to subprefix hijacking if: (1) no valid ROA exists and the prefix length is less than 24, or (2) a valid ROA exists but the max-length field in the ROA is strictly greater than the prefix length and the prefix length is less than 24.

Our investigation results for NFTMs are shown in Table 2. One popular marketplace (Golom) has its MS directly originating from an IP prefix susceptible to subprefix hijacking. Six NFTMs rely on JavaScript code originating from risky IP prefixes; once these external JavaScript providers are hijacked by our network attacker, so could be the NFTMs. This suggests that NFTMs should carefully

Table 3: Popular CIs' susceptibility to prefix hijack attack

Centralized service	Vulnerable AS	Reason
IPFS Gateways		
4everland.io	AS16509	Max-Len
hardbin.com	AS14061	Max-Len
ipfs.eth.aragon.network	AS24940	Max-Len
jorropo.net	AS14061	No ROA
ipfs.runfission.com	AS14618	Max-Len
BaaS Gateways		
mainnet.infura.io	AS14618	Max-Len

audit their external dependencies to minimize their attack surface. We also identified 5 IPFS gateways and one major BaaS provider (Infura) that are subject to subprefix hijacking, as shown in Table 3. There are likely more in the wild.

Note that even if NFT service providers reside in secure networks that are resistant to BGP hijacking, our attacks can still be launched if users or their DNS servers locate in vulnerable ASes, or these servers are subject to DNS cache poisoning attacks [13]. A comprehensive demographic study of victim users is an interesting avenue for future research.

Most NFT service providers deploy TLS to secure their communication with users. A successful attack thus requires the attacker to obtain a fraudulent TLS certificate to impersonate a service provider's domain. This has been demonstrated to be practical in many ways, especially by attacking the domain validation process during certificate issuance [29,6].

7 Conclusion

This paper makes a step in uncovering the security risks of centralization in the booming Web3 ecosystem. We focus on the case of NFT, a central application of Web3, and perform a systematic study of architecture-level vulnerabilities regarding the interactions between users and centralized entities. Our results confirm that centralization increases the overall attack surface by a wide margin. This is worrisome given the variety and practicality of such attacks, and the large financial investments in NFTs. Some of these attacks are relatively easy to detect if users take caution to audit blockchain transactions before signing them; the others are less so, requiring the shift to a truly decentralized architecture or extensive end-to-end data authentication.

Our findings also underscore the importance of secure Internet infrastructures for inter-domain routing and name resolution, which would prevent our attacks in the first place. A promising research direction is to evaluate how existing security extensions to the Internet as well as emerging clean-slate solutions [9,16] can improve the Web3 ecosystem's resilience to attacks.

References

1. Official IPFS gateway. https://ipfs.io.
2. OpenSea Bug Bounty Program. https://hackerone.com/opensea.
3. Anders, Lance, and Shrug. EIP-4907: Rental NFT, an Extension of EIP-721. Available: https://eips.ethereum.org/EIPS/eip-4907, March 2022.
4. Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing attacks on cryptocurrencies. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2017.
5. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A Survey of Attacks on Ethereum Smart Contracts SoK. In *Proc. of the International Conference on Principles of Security and Trust (POST)*, 2017.
6. Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. Bamboozling Certificate Authorities with BGP. In *Proc. of the USENIX Security Symposium*, 2018.
7. Zach Burks, James Morgan, Blaine Malone, and James Seibel. EIP-2981: NFT Royalty Standard. Available: https://eips.ethereum.org/EIPS/eip-2981, September 2020.
8. Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. SoK: Blockchain Light Clients. In *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*, 2022.
9. Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. *The Complete Guide to SCION. From Design Principles to Formal Verification.* Springer International Publishing AG, 2022.
10. Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *Proc. of the USENIX Security Symposium*, 2017.
11. Catalin Cimpanu. DNS hijacks at two cryptocurrency sites point the finger at GoDaddy, again. https://therecord.media/two-cryptocurrency-portals-are-experiencing-a-dns-hijack-at-the-same-time/. Accessed 01/10/2022.
12. Catalin Cimpanu. KlaySwap crypto users lose funds after BGP hijack. https://therecord.media/klayswap-crypto-users-lose-funds-after-bgp-hijack/. Accessed 01/10/2022.
13. Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. From IP to Transport and beyond: Cross-Layer Attacks against Applications. In *Proc. of the ACM SIGCOMM Conference*, 2021.
14. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2020.
15. Dipanjan Das, Priyanka Bose, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. Understanding Security Issues in the NFT Ecosystem. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2022.
16. Huayi Duan, Rubén Fischer, Jie Lou, Si Liu, David Basin, and Adrian Perrig. Rhine: Robust and high-performance internet naming with e2e authenticity. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

17. William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. EIP-721: Non-Fungible Token Standard.
18. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.
19. Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. Are We There Yet? On RPKI's Deployment and Security. Cryptology ePrint Archive, Paper 2016/1010, 2016. https://eprint.iacr.org/2016/1010.
20. James Grimmelmann, Yan Ji, and Tyler Kell. EIP-5218: NFT Rights Management. Available: https://eips.ethereum.org/EIPS/eip-5218, July 2022.
21. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *Proc. of USENIX Security*, 2015.
22. Bo Jiang, Ye Liu, and Wing Kwong Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proc. of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.
23. David Johnston, Sam Onat Yilmaz, Jeremy Kandah, Nikos Bentenitis, Farzad Hashemi, Ron Gross, Shawn Wilkinson, and Steven Mason. The General Theory of Decentralized Applications, DApps. Technical report, 2014.
24. Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service. In *Proc. of the Symposium on Network and Distributed Systems Security (NDSS)*, 2021.
25. Shaurya Malwa. Two Polygon, Fantom Front Ends Hit by DNS Attack. https://www.coindesk.com/tech/2022/07/01/two-polygon-fantom-front-ends-hit-by-dns-attack/. Accessed 01/10/2022.
26. Moxie Marlinspike. My first impressions of web3. https://moxie.org/2022/01/07/web3-first-impressions.html, January 2022. Accessed 01.10.2022.
27. Anton Permenev, Dimitar Dimitrov, Petar Tsankov, Dana Drachsler-Cohen, and Martin Vechev. VerX: Safety Verification of Smart Contracts. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2020.
28. Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three Attacks on Proof-of-Stake Ethereum. In *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*, 2022.
29. Lorenz Schwittmann, Matthäus Wander, and Torben Weis. Domain Impersonation is Feasible: A Study of CA Domain Validation Vulnerabilities. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.
30. Liya Su, Xinyue Shen, Xiangyu Du, Xiaojing Liao, XiaoFeng Wang, Luyi Xing, and Baoxu Liu. Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications. In *Proc. of USENIX Security*, 2021.
31. Verified Market Research (VMR). Non-Fungible Tokens Market Size And Forecast. Technical report, 2022.
32. Dabao Wang, Hang Feng, Siwei Wu, Yajin Zhou, Lei Wu, and Xingliang Yuan. Penny Wise and Pound Foolish: Quantifying the Risk of Unlimited Approval of ERC20 Tokens on Ethereum. In *Proc. of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022.
33. Ziwei Wang, Jiashi Gao, and Xuetao Wei. Do NFTs' Owners Really Possess Their Assets? A First Look at the NFT-to-Asset Connection Fragility. In *Proc. of the ACM Web Conference (WWW)*, 2023.
34. Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-Frequency Trading on Decentralized On-Chain Exchanges. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2021.