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 the first systematic analysis of the threats posed by centralized entities in the current Web3 ecosystem. Our findings are concerning: *almost every interaction of a user with a centralized entity can be exploited to hijack NFTs or cryptocurrencies from the user*. Moreover, all attacks we discovered can be launched through network attacks practical today. Our measurement results further show that many big players in the NFT market are vulnerable to the attacks, placing large financial investments at risk. While our analysis of NFTs has been fruitful, it is just a starting point to reveal the pervasive centralization issues in the vast Web3 space.

1 Introduction

We are witnessing an increase in popularity of re-decentralizing web services provided by big corporations today, which is portrayed as the transition from Web2 to Web3. In the envisioned Web3 paradigm, all web services or applications are hosted by decentralized infrastructures, in particular blockchains supporting programmable smart contracts. A defining feature of such decentralized applications (DApps) is that they do not rely on any single entity for their governance and operation [30]; as a result, users can access DApps without trusting any single entity. However, practical DApps usually deviate from this idealized model and, somewhat inevitably, employ centralized platforms for reasons of cost, performance, and usability. Many real-life incidents have shown that those centralized components are inviting targets for attackers [36,26,31,16,15].

These attacks pose serious threats to the growing Web3 ecosystem, but they have not received equal attention from the research community compared with vulnerabilities in blockchain protocols [17,43,38,25] or smart contracts [11,33,35], 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, the complex interactions of them with other parties, and hence the potential massive attack surface therein.

We initiate a systematic study of the security issues *induced by centralization* in Web3, focusing on the sub-ecosystem around non-fungible tokens (NFTs). Restricting the scope is necessary for in-depth analysis because Web3 encompasses numerous aspects, and the reasons for choosing NFTs are three-fold. First, NFTs are among the most popular Web3 concepts with a multi-billion dollar market [46]. Second, NFTs establish the fundamental and ubiquitous notion of asset ownership for Web3, and so they will persist even if high market valuations should decline. For instance, after the initial standard [24], the Ethereum community has a series of proposals to bring NFTs closer to the practical realm from usability [42,9] and legal perspectives [27]. Third, the NFT sub-ecosystem is sufficient to demonstrate common centralization issues, as it involves different centralized entities. The introduction of these entities essentially changes the ways users interact with decentralized infrastructures. Thus, we are interested in new attack vectors arising from these interactions rather than vulnerabilities in the entities themselves. More specifically, we consider practical network adversaries capable of intercepting these new interactions that are otherwise non-existent in a fully decentralized architecture. This can reveal, instead of security issues pertinent to specific system design or implementation, generic architecture-level problems due to centralization.

Our work starts by defining an abstract model that captures the essential entities in today's NFT ecosystem and the interactions between them for the creation, tracking, and trading of NFTs. By instantiating this model with concrete architectures employing varying forms of centralization, we systematically examine each interaction and the potential exploits therein. 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 powering the NFTs. We have validated all but one families of attacks we discovered with practical systems. Furthermore, our measurement of major players in the NFT market shows that many of them, including 6 out of 10 largest marketplaces, are susceptible to our hijacking attacks.

Our use of NFTs as a slice of the Web3 universe to investigate centralization risks is just a starting point. The methodology we developed in this work can also be applied to analyze DApps beyond NFTs. To sum up, we make the following contributions in this paper.

– A **taxonomy of off-chain hijacking attacks** for the NFT ecosystem.
– **Empirical validation** of these attacks' practicality and applicability.
– **Measurement** of vulnerable entities in the NFT ecosystem.

2 Related Work

We review security research on NFTs and DApps as well as the underlying blockchain and smart contract platform, followed by a brief discussion of practical network attacks that underpin our NFT hijacking attacks.

Research on NFT security. In a blog post [37], Moxie Marlinspike provides his reflection on Web3 by experimentally building DApps that allow users to mint and exchange NFTs. Through the experiments, he points out the fact that most existing DApps are not as decentralized as claimed due to their reliance on centralized servers. These servers can return arbitrary data associated with NFTs to users, and marketplaces like OpenSea can unilaterally remove these 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. A recent study by Das et al. analyzes today's NFT ecosystem and security issues therein [22], for example insufficient user authentication and unverifiable smart contracts by NFT marketplaces, lack of persistent asset data storage by external entities, and malpractices by NFT traders. Kapoor et al. analyze how social media can influence the valuation of NFTs [32].

To the best of our knowledge, we are the first to perform an in-depth analysis of the vulnerabilities introduced by various forms of centralization in the NFT ecosystem. Unlike previous work [37,22] 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 behave correctly and honestly.*

Attacks on DApps. Su et al. [45] perform an extensive measurement of DApps attack traces from the Ethereum transaction history. They identify common attack patterns and develop a methodology to automatically discover exploits of a DApp from related transactions. These attacks make use of design and implementation flaws in smart contracts and thus are orthogonal to what we consider in this paper.

Representing a major class of DApps, decentralized finance (DeFi) aims to do away with traditional financial institutions like banks and exchanges. However, the transparency and high transaction latency of the blockchain platform makes DeFi services susceptible to manipulation, for example front running [21] and sandwiching attacks [48]. Similar to the case of NFTs, practical DeFi platforms [8,1] rely on centralized components like web servers and blockchain gateways. Therefore, our attacks on NFTs can also apply to DeFi services.

Recently, Wang et al. [47] quantify the security risks of unlimited approval of ERC20 tokens (aka cryptocurrencies, which are fungible) when used by DApps. Some of our attacks also exploit the fact that the trade of NFTs requires their owners to approve the delegation of control to marketplaces.

Li et al. [34] 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 angle.

Blockchain and smart contract security. Many attacks on blockchain consensus protocols have been found [17,43,38,25]. Attacks on blockchains' network layer [39] or execution layer [40] also exit. In comparison, our work explores a new class of security threats arising from external entities which are not part of a blockchain network but widely exist for practical reasons.

Programming errors in smart contracts are common sources of exploits [11,33]. Different tools have been developed to discover security bugs in smart contracts via static [35,41] or dynamic analysis [29]. Such tools, however, cannot detect our attacks because we do not exploit any flaws in smart contracts themselves.

Practical network attacks. Many fundamental Internet protocols, including Border Gateway Protocol (BGP) for routing and Domain Name System (DNS) for naming, are not secure by design. An *off-path* network adversary can maliciously take over different Internet resources (IP addresses, domain names, certificates, etc.) through BGP or DNS hijacking [19], leading to attacks on a wide range of critical Internet services and applications [18]. While there are prior studies of attacks against blockchain networks [10,28,23], we for the first time investigate practical network-based attacks in the realm of Web3.

3 Abstract Model for NFT Functionality

The NFT ecosystem contains different entities revolving around the creation, tracking, and trading of NFTs. Despite their variety (e.g., over 50 active marketplaces exist [6]), the design of practical systems to realize the functionality of NFTs follows a common set of patterns. We define an abstract model to capture this essential functionality and later use instantiations of it to conduct fine-grained security analyses. Figure 1 depicts our model. It consists of two classes of entities: users (in shaded area) and service providers. They interact with each other through predefined interfaces, which comply with the widely implemented Ethereum standard EIP-721 [24]. In real systems, some of the interactions may be merged into a single action, and new interactions may be introduced. Our model captures the typical life circle of NFTs as seen today.

3.1 Data and Interfaces

Our model has three types of service providers: ownership registry, asset storage, and NFT marketplace (NFTM). We elaborate their interfaces and data (illustrated in the dashed boxes in Figure 1) one by one.

Ownership registry. NFTs are essentially ownership records of some digital or physical assets. An ownership registry permanently stores these records and offers operations on them. Each record is defined by 4 fields. The first and foremost is `TID` that uniquely identifies an NFT; in practice, this is implemented through the pair of a globally unique smart contract address and a locally unique token index. The other three fields are: `OID` identifying the token's owner, `tokenURI` which is a pointer to the underlying asset, and `delegatee` which is an entity who can control the token on behalf of its owner. The registry expose four interfaces: (1) `Register` to create a new NFT 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`, (3) `Delegate` to set the delegatee for a token, and (4) `Read` to retrieve a record given a `TID`. Note that in practice `tokenURI` can be accompanied by other metadata like textual description of the NFT. We assume

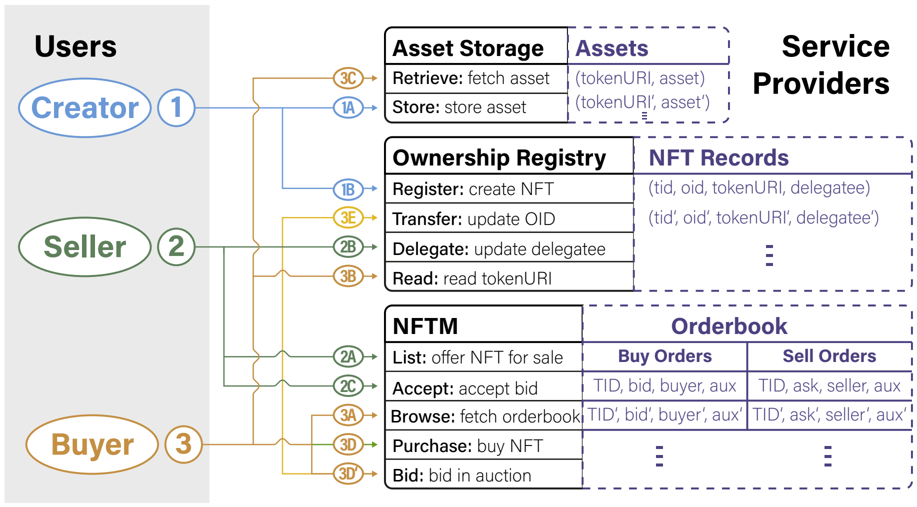


Fig. 1: Our abstract model to capture the essential functionality of NFTs.

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

Asset storage. The assets associated with NFTs are maintained in an asset storage. Here we consider two basic interfaces to store and retrieve the asset of a given `TID`. Unlike an ownership registry that is hosted on a blockchain by default, asset storage systems are almost always off-chain and come in various forms, as we discuss in the next section.

NFTM. NFTs must be tradable to create value. This is enabled by an NFTM that connect buyers and sellers. The most important data maintained by the NFTM is an orderbook, which keeps track of the current sell and buy orders. Here 5 interfaces are exposed to users: (1) `List` for a seller to offer a token for sale, (2) `Accept` for a seller to accept a buyer order or 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 for a token in auction. The NFTM updates it 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 in our model can take three roles: creator, seller, and buyer. We describe a typical NFT life cycle through their interactions with service providers. We use the notation Ⓧ `action(in` $\rightarrow$ `out)` to represent the invocation of an interface `action`, which takes `in` as input and returns `out` to the caller, in an interaction. For ease of exposition, we omit non-critical data in some interactions.

Creation. The creator of an NFT can vary, for example the 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 the asset through $\textcircled{1A}$ `Store`(`asset` $\rightarrow$ `tokenURI`). Then, it creates a token by calling $\textcircled{1B}$ `Register`({`OID`, `tokenURI`} $\rightarrow$ `TID`), which results in a new record stored by the ownership registry.

Listing. The owner of a token offers it for sale through $\textcircled{2A}$ `List`({`TID`, `OID`, `ask`} $\rightarrow$). The invoked marketplace needs permission to transfer the token without the seller's further involvement. This is done via $\textcircled{2B}$ `Delegate`({`TID`, `Mkt`} $\rightarrow$), where the `delegatee` of token `TID` is set to 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 $\textcircled{3A}$ `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 $\textcircled{3B}$ `Read`(`TID`$\rightarrow$ `tokenURI`) from the ownership registry and then fetches the asset by calling $\textcircled{3C}$ `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 $\textcircled{3D}$ `Purchase`({`TID`, `ask`, `buyer`} $\rightarrow$). In the latter case, the buyer places a bid via $\textcircled{3D'}$ `Bid`({`TID`, `bid`, `buyer`} $\rightarrow$), which results in a buy order stored in the orderbook. Upon a successful sale, the NFTM transfers the token to the new owner by calling $\textcircled{3E}$ `Transfer`({`TID`, `buyer`} $\rightarrow$) without the seller's involvement. Note that this is possible because the NFTM has been approved by the seller before.

4 System Architecture & Attack Taxonomy

We consider four instantiations of our abstract model and describe concrete attacks present in three of them. We first present a fully decentralized architecture where every service provider is implemented on, and accessed through, a decentralized system. In each subsequent architecture, one service is hosted centrally, or accessed through a centralized intermediary (CI). Table 1 summarizes the attack potential created by centralized entities in different architectures.

Adversary 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 [19]. Even if the communication is secured by the now widely deployed Transport Layer Security (TLS) protocol, the attacker can still acquire a TLS certificate to impersonate the victim domain [13]. We assume that any data hosted by decentralized infrastructures, including blockchains and decentralized storage systems, is tamper proof and always verified by their nodes.

4.1 Architecture Type I: Fully Decentralized

In a fully decentralized and ideal architecture, the ownership registry and the NFTM reside on a blockchain in the form of smart contracts. Users interact with the service providers (e.g. to mint an NFT or to create a new listing) by sending blockchain transactions to these smart contracts. Parameters such as

Table 1: A summary of potential NFT hijacking attacks in different architectures. Detectability 1 requires careful auditing of parameters to be signed, 2 requires local merkle-tree file verification, 3 requires additional, untampered accesses to the blockchain or IPFS, and 4 means undetectable.

Attack family	Architecture	Detectability	Outcome
$\mathcal{A}$**1**: 1A	Type II	2/4*	NFT with attacker-controlled asset minted
$\mathcal{A}$**2**: 3C	Type II	2/4**	Wrong NFT bought by buyer
$\mathcal{A}$**3**: 1C	Type III	4	Royalty payments sent to attacker
$\mathcal{A}$**4**: 2A	Type III	1	NFT sold to attacker for a too low amount NFT transferred to attacker
$\mathcal{A}$**5**: 3A	Type III	3/4***	Buyer more likely to but attacker's NFT Buyer places higher bids than needed
$\mathcal{A}$**6**: 3B'	Type III	3	Wrong NFT bought by buyer
$\mathcal{A}$**7**: 3C'	Type III	3	Wrong NFT bought by buyer
$\mathcal{A}$**8**: 3D	Type III	1	Funds stolen from buyer Wrong NFT bought by buyer
$\mathcal{A}$**9**: 3D'	Type III	1	Funds stolen from buyer Buyer bids in attacker's auction
$\mathcal{A}$**10** 3B:	Type IV	4	Wrong NFT bought by buyer

* 2 if stored on IPFS, 4 if stored in centralized storage without integrity checks
** 2 if IPFS, 4 if Blockchain or centralized storage
*** 4 if centralized access or off-chain orderbook, 3 if on-chain orderbook

the list price of an NFT are encoded within the transaction data. Asset storage can reside either on the blockchain or on a decentralized storage network such as the Interplanetary Filesystem (IPFS) [4]. In IPFS, files are split into shards which together with a checksum form the leaf nodes of a Merkle-DAG [12]. The content identifier used to query the IPFS network references the Merkle-DAG's root node, which in turn enables data integrity verification. Since IPFS is the most popular storage solution in today's NFT ecosystem, we will use it as the representative for all decentralized storage networks.

In this architecture, users operate their own blockchain nodes or light clients, and if the assets storage resides on IPFS, also IPFS nodes. This allows users to benefit from the data integrity guarantees provided by these decentralized infrastrucures. To access an on-chain NFTM, users need specialized explorer software that renders the relevant data (token listings, market orders, digital assets, etc.) retrieved from local blockchain or IPFS nodes. They do not need to interact with any centralized entities.

Security. This model enables end-to-end validation of all actions performed by the users. The creator in 1A is ensured that the asset is uploaded correctly to the asset storage and that the returned pointer actually references the asset. The `Register` transaction sent in 1B is signed with the creator's blockchain private key and is therefore cryptographically protected against tampering. The seller's interactions 2A and 2B are transactions signed with the seller's blockchain private key and are thus also protected against tampering. When the buyer browses the NFTM's orderbook in 3A and retrieves the `tokenURI` in 3B, all data is read from the local, consistent blockchain state if the buyer uses a full node, or if the buyer operates a light client, it is received in a verifiable way from supporting full nodes. The `tokenURI` is queried in 3C which either involves reading the blockchain state or a `Retrieve` call to the IPFS network. Because a local IPFS node verifies that the retrieved asset matches the queried asset reference, the user is guaranteed to also retrieve the correct asset if IPFS is used. Buying and bidding in 3D and 3D' are again signed blockchain transactions and are thus protected. We conclude that in this idealized architecture, an adversary cannot hijack NFTs without breaking the underlying decentralized mechanisms.

4.2 Architecture Type II: Centralized Asset Storage

Asset storage can be centralized either by accesses to it being centralized, or by the data itself being stored centrally. For centralized access, we differentiate between Blockchain as a service (BaaS) providers if the asset is stored on-chain, and IPFS gateways if the asset is stored on IPFS. These CIs run their own infrastructure nodes and provide easy-to-use interfaces for users to interact with blockchains and IPFS without joining the networks themselves. Both HTTP(S) and WebSocket-based CIs are common. We discuss two attacks which apply to both cases and we highlight important differences in the detectability of these attacks for each setting.

Centralized Access. Although the assets storage is implemented on a decentralized system, users interact with the asset storage through a CI such as the BaaS provider Infura and the IPFS gateway gateway.ipfs.io [3]. Because the connection between the CI and the user is not part of the blockchain or IPFS network, it is not protected by their respective integrity mechanisms. IPFS is special because the asset identifier used to query the IPFS gateway is the same reference used by IPFS internally and is thus a function of that file's Merkle-DAG root. Users can therefore, after downloading an asset, locally compute its Merkle-DAG and compare it with the asset reference used in the query to the IPFS gateway. This verification function however requires dedicated software, as it is not built into commodity browsers such as Chrome or Firefox.

Centralized Storage. Asset storage can alternatively also be implemented entirely on centralized infrastructure such as cloud providers like AWS or Azure, or file hosters like Mega. From a security point of view, retrieving assets from such an asset storage system is somewhat equivalent to fetching data from a regular web server. The asset provider has to implement their own integrity mechanism to ensure similar integrity guarantees as in decentralized asset storage.

Attacks. The loss of verifiability of interactions with the asset storage enables attacks against interactions 1A and 3C. We highlight the data being modified by each family of attacks in red.

$\mathcal{A}$1: `Store`(`asset` $\rightarrow$ `tokenURI`). Assets are uploaded to the asset storage through a CI. If the asset storage is on-chain, the `Store` command is implemented as a call to a smart contract and because it changes the blockchain state, requires a transaction. This transaction is cryptographically signed with the creator's private key and is thus, despite the existence of a CI, protected against tampering by the attacker under our attacker model. If the asset is stored off-chain on IPFS or on a centrally stored asset storage provider, no cryptographic signatures are required when uploading the asset. Thus, an attacker under our attacker model can hijack the connection and modify the uploaded asset, resulting in a different, attacker-chosen, asset (e.g. image) being uploaded. The returned `tokenURI` references the attacker-modified asset. If the creator then queries the `tokenURI` to check whether the correct file was uploaded, the attacker can, again, by hijacking the connection modify the returned asset to look like the intended asset. The only way for a creator to notice the modification is when IPFS was used, and the creator locally compared the Merkle-DAG of the genuine file with the content identifier returned by the CI.

After uploading the asset, the actual NFT is created by calling `Register`({`OID`, `tokenURI`} $\rightarrow$ `TID`). If the preceding `Store` operation was hijacked, the asset referenced by the `tokenURI` was chosen by an attacker. Note that the actual `Register` call was not tampered with by the attacker.

$\mathcal{A}$2: `Retrieve`(`tokenURI`$\rightarrow$ `asset`). Upon a call to retrieve, the asset storage provider returns the requested asset. On-chain asset storage is queried through a BaaS as a CI, which returns the asset to the requester. The returned data is not authenticated by the blockchain, the channel between the BaaS and requester is relied upon for authentication. IPFS asset storage similarly suffers from a reliance on the communication channel between the IPFS gateway CI and the requester. By using the offline integrity check capability of IPFS, the requester can manually verify the file using the `tokenURI` used to query it. If the asset itself is stored centrally, the any integrity checks provided by the asset storage provider and the authentication of the channel protect the returned asset.

By hijacking the connection and modifying the returned asset, an attacker can trick the buyer into believing it is looking at a more valuable NFT than is actually the case. This attack can be used to trick a buyer into buying a less valuable NFT offered by the attacker for the price of a more valuable asset. Detectability depends on the asset storage system used. In IPFS, and for centrally stored asset storage if the provider offers integrity guarantees, the attack could in principle be detected.

4.3 Architecture Type III: Centralized NFTM

Marketplaces can be centralized in a similar fashion: they can either be stored in a decentralized infrastructure which is accessed through a CI, or by storing part of the marketplace off-chain. The latter allows an attacker to tamper with almost all interactions between users and marketplaces.

Centralized Access. In this architecture, the marketplace is implemented on a blockchain as a smart contract. Users interact with it through the same NFTM

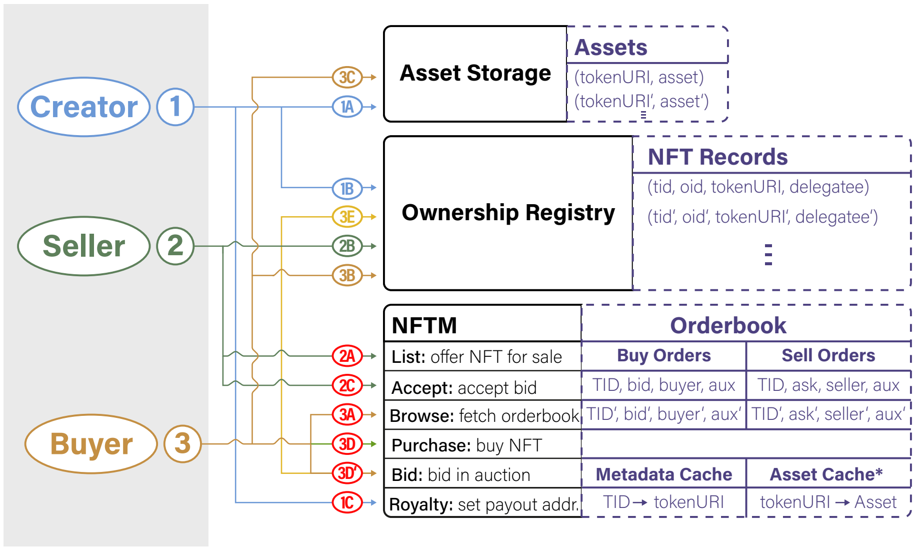


Fig. 2: Centrally hosted NFTM Architecture. The Asset Cache* is optional.

explorer software described in the fully decentralized architecture of Sec. 4.1, except that it does not interact with the user's local blockchain node and instead with a CI. The architecture is otherwise identical to the fully decentralized case.

Centralized Storage. We consider a marketplace that operates an off-chain marketplace server (MS) which stores the NFTM's orderbook, caches the `tokenURI` of NFTs currently for sale, provides users with parameters to be included in blockchain transactions and in buy/sell orders, and hosts a storefront with which users can interact using a web browser. Transaction and order parameters are provided by the MS because browsers, unlike NFTM explorer software, do not have information like NFTM smart contract addresses hard coded. This is also observed in real-world marketplaces like OpenSea and Rarible where the MS provides all transaction and order parameters. Optionally, the marketplace can already cache the asset itself. This is present in marketplaces like OpenSea, Rarible, Foundation, etc.

By moving these capabilities off-chain onto the MS, interactions (3B), and if the assets are cached then also interaction (3C), of the abstract model in Fig. 1 are removed. The buyer instead retrieves the `tokenURI` of NFTs currently for sale from the MS in interaction (3B'). If the MS also caches the assets, the buyer fetches them in interaction (3C') from the MS as opposed to from the asset storage in interaction (3C).

We further differentiate two cases: (i) marketplace keeps a copy of the orderbook on-chain, and (ii) marketplace only stores the orderbook in the MS. Both instantiations exist in practice, the marketplace Foundation keeps the orderbook on-chain while OpenSea and Rarible store the orderbook exclusively off-chain.

The NFTM however still has a smart contract deployed on the blockchain which implements the matching of buy and sell orders and which issues the transfer of ownership once a token has been sold.

Attacks. Centralized access to a marketplace server enables attacks $\mathcal{A}$**5** against interaction (3A). All further attacks are possible in the presence of a centralized MS.

$\mathcal{A}$**3:** `SetRoyalties({addr, amount} →)`. Creators can benefit from secondary sales of their NFTs through royalties. Upon a secondary sale, the NFTM transfers a fraction of the ask price to a creator-chosen wallet. Several popular NFTMs such as OpenSea and Rarible store the royalty payout address centrally on their MS. Updates to the payout address do not require the creator's signature, the creator only has to be logged into the MS. A creator setting its payout address can fall victim to an attacker modifying the payout address sent to the MS.

$\mathcal{A}$**4:** `List`({`TID`, `OID`, `ask`} →). Unlike in previous architectures, where function call parameters are generated locally, MSes supply the user with these parameters. This additional interaction is not shown in Fig. 2 for brevity but it is present in all further attacks involving an MS.

By sending the seller's terms of sale, it notifies the MS about the intent to sell an NFT. The parameters {`TID`, `OID`, `ask`} returned by the MS are susceptible to being tampered by an attacker which can cause the seller to list an NFT with a lower than intenden `ask` amount. The `List` call itself can not be modified because it is signed by the seller. The parameters to `Delegate` the permission to transfer an NFT are also provided by the MS. In a similar attack, the `Mkt` parameter included in `Delegate`({`TID`, `Mkt`} →) can be changed to instead make the attacker a `delegatee`. Although the data included in the `List` and `Delegate` calls are shown to the seller for auditing, they are often not decoded properly and are thus difficult to verify.

$\mathcal{A}$**5:** `Browse`(→ {`TID`, `ask`, `seller`}). By increasing the `ask` amount returned by the MS in competing sell orders by legitimate sellers, an attacker gets an unfair advantage because its listings appear cheaper than others. For auctions, `Browse` additionally fetches bids {`TID`, `bid`, `buyer`}. Increasing the `bid` amount can cause the buyer to bid a higher than necessary amount to win the auction. This attack is also possible against on-chain NFTMs accessed through a CI. To detect this attack, the buyer needs secure, untampered access to the orderbook. Detection for NFTMs without on-chain orderbooks is difficult unless they provide read access to their off-chain orderbook which includes cryptographic authentication of the sellorders.

$\mathcal{A}$**6:** `Read`(`TID`→ `tokenURI`). Modifications to the `tokenURI` by the attacker cause the buyer to erroneously fetch an attacker-chosen asset in the subsequent interaction (3C) or (3C'). Detecting this modifications requires the buyer to fetch the `tokenURI` from the blockchain in an untampered way.

$\mathcal{A}$**7:** `Retrieve`(`tokenURI`→ `asset`). Some marketplaces also provide a centralized asset cache. By modifying the returned asset, the attacker is able to trick the buyer into believing it is looking at a more valuable NFT than is actually the case. Detecting this attack requires untampered access to the original asset storage.

$\mathcal{A}$8: `Purchase`({`TID`, `ask`, `buyer`} →). Two attacks involving interaction 3D are possible. By modifying the `TID`, the attacker is able to trick the buyer into purchasing a different, attacker-owned, asset. In addition to the parameters for the `Purchase` call, the MS also provides a destination to where this function should be sent. In concrete instantiations, this is the smart contract address of the NFTM. By changing the destination of the `Purchase` function, the buyer is tricked into sending the `Purchase` transaction to the attacker who is then able to extract the `ask` amount.

$\mathcal{A}$9: `Bid`({`TID`, `bid`, `buyer`} →). Some marketplaces such as foundation implement bidding protocols which require the buyer to deposit the bid amount already within `Bid`. Such bid protocols are susceptible to the same redirection attack mentioned in $\mathcal{A}$8.

If the `Bid` does not include a deposit, as is the case in OpenSea, the attacker can modify `bid` to trick the buyer into bidding more in a hard to detect manner. NFTMs using these bidding protocols enforce the winning bid by becoming delegatee of surrogate tokens for the native cryptocurrency. Concretely, in the case of OpenSea and Rarible on Ethereum, if for example a bid is placed for 1ETH, the buyer has to delegate the right over at least 1wETH, a tokenized version of 1ETH, to the NFTM. This requires a call `Delegate`({`TID`, `Mkt`} →), the parameters for which are provided by the MS. Modifying the `Mkt` value causes the attacker to become delegatee over these surrogate tokens.

4.4 Architecture Type IV: Centralized Ownership Registry

Unlike in the previous two architectures, here we only consider the case of centralized access and ignore a centralized ownership registry, as it contradicts the core idea of NFTs being stored in a blockchain.

$\mathcal{A}$10: `Read`(`TID`→ `tokenURI`). By modifying the `tokenURI`, an attacker is able to trick the buyer into believing it is looking at a more valuable NFT than is actually the case. This attack can be used to trick a buyer into buying a less valuable NFT offered by the attacker for the price of a more valuable asset.

5 Attack Validation

We have validated our attacks with practical systems. The validation consists of two major parts based on the centralized entities involved:

- **Attacks involving a BaaS or IPFS gateway:** $\mathcal{A}$1, $\mathcal{A}$2, $\mathcal{A}$10
- **Attacks involving a centralized NFTM:** $\mathcal{A}$3–$\mathcal{A}$9

Setup. We use OpenSea as an NFTM with a centrally implemented marketplace; and use Ethereum's Goerli testnet as a blockchain, which mimics the real Ethereum mainnet and is used by OpenSea for testing purposes. Ethereum makes up 76% of the total NFT sale volume across the top 10 blockchains. OpenSea is the largest NFTM accounting for 70% of the total NFT sale volume on Ethereum across the top 15 marketplaces[2]. NFT trade on OpenSea accounts for roughly 53% of the entire sale volume. For the gateways to the blockchain and IPFS, we use the Infura blockchain gateway and the official IPFS gateway. For the IPFS pinning service, we use Pinata which offers a web interface for uploading files into IPSF. Both the gateways and Pinata are considered industry-standard.

All the attacks are implemented in Python 3.7.9 and carried out on a laptop running Firefox 102.0 and Metamask 10.14.7 on Ubuntu 20.04.4 LTS. To simulate the attacker, we install an CA certificate in Firefox's trust store for the impersonation capability and use mitmproxy 8.0.0 to hijack website connections. This allows us to modify HTTPs requests and responses.

Ethical Consideration. Our experiments create smart contracts and NFTs on a blockchain that is intended for testing purposes including security analysis. They do not incur real monetary costs even if a user accidentally buys the tokens. The test asset files we upload to IPFS disappear after some time when they are not cached by any node, and so their impact to the network is minimal.

Our attacks do not exploit the systems of service providers themselves, but rather architecture-level vulnerabilities induced by centralization and general network-based attacks. Service providers often do not consider attacks requiring man-in-the-middle (MITM) capabilities, which is our case, within their responsibility. We submitted a report through OpenSea's official bug bounty program [7] and their reply confirms that our attacks are outside the program's scope.

5.1 Attacks Involving a BaaS or IPFS Gateway

A gateway service exposes APIs for users to fetch data from a decentralized infrastructure without running nodes by themselves. We validate that it is possible to intercept the connection and modify the data in our attacks.

Change the asset reference returned by a BaaS Gateway ($\mathcal{A}10$). To simulate the attack, we first created an NFT, of which its asset reference could be returned by the `tokenURI()` function according to ERC-721. Then as the user, we tried to get the asset reference by using a common tool `curl` to invoke the function through the Infura blockchain gateway. Meanwhile, as the attacker, we used `mitmproxy` to intercept the gateway's response and managed to replace the returned asset reference with an arbitrary string.

Modify the asset to or from an IPFS gateway ($\mathcal{A}1$, $\mathcal{A}2$). To simulate the attack $\mathcal{A}1$, as the user, we uploaded a file to IPFS through the IPFS pinning service Pinata. Meanwhile, as the attacker, we used mitmproxy to intercept the request and managed to change the file to be uploaded. For the attack $\mathcal{A}2$, as the user, we tried to browse the asset of an NFT via the official IPFS gateway. Simultaneously, as the attacker, we used `mitmproxy` to intercept the gateway's response and managed to change the returned asset.

5.2 Attacks involving a Centralized Marketplace

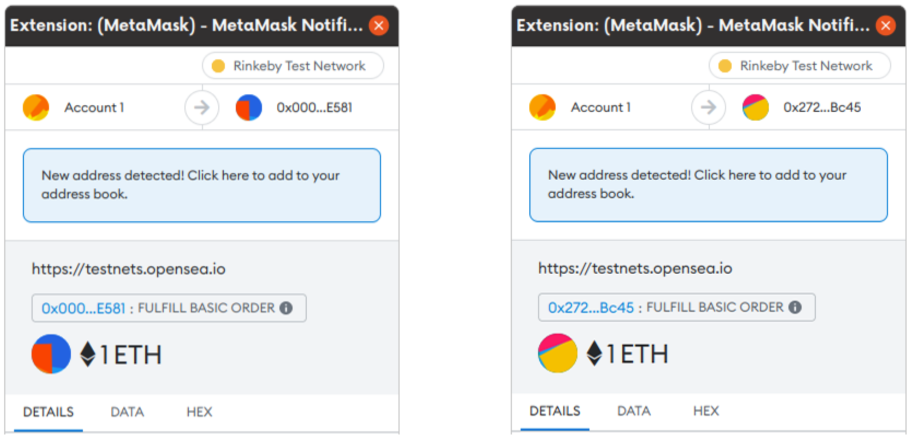
We have identified two methods to attack a user's interactions with an MS. First, an attacker can hijack the connection between the MS and the user. This allows direct modification of the information exchanged between the MS and the user. Second, an attacker can hijack the connection between the user and a third-party JavaScript provider of the MS. Most marketplace websites include third-party scripts for example to enhance the user experience or for analytics purposes. Unless a script is sandboxed within an iframe, it has access to the entire web page and can perform virtually arbitrary modifications. We briefly discuss our validated attacks below and leave further details to Appendix A.

Royalty payout address change ($\mathcal{A}3$). To simulate the attack, as the user, we tried to change the royalty payout address on the OpenSea website. Meanwhile, as the attacker, we used `mitmproxy` to intercept the request and managed to set the new royalty payout address to the attacker's account.

Wrong data fetched from NFTM ($\mathcal{A}5$, $\mathcal{A}6$, $\mathcal{A}7$). To simulate the attack $\mathcal{A}5$, as the buyer, we tried to browse the bids for an NFT on the OpenSea website. Meanwhile, as the attacker, we used `mitmproxy` to intercept the buyer's request for fetching the bids from MS and managed to increase the price of the bids. To validate the attack $\mathcal{A}6$, as the buyer, we browsed an NFT on the OpenSea website. At the same time, as the attacker, we used `mitmproxy` to intercept the buyer's request for fetching the NFT image from MS. We managed to change the image URI in the request, causing the buyer to receive another NFT's image. Validating the attack $\mathcal{A}7$ was the same except that as the attacker, we intercepted the MS's response to the buyer with `mitmproxy` and changed the returned image.

Change order or transaction parameters ($\mathcal{A}4$, $\mathcal{A}8$, $\mathcal{A}9$). To simluate the attack $\mathcal{A}4$, as the seller, we listed an NFT for sale via the OpenSea website. In the first case that the seller lists an NFT from a smart contract new to the NFTM, the MS asked us to send a transaction that approves the NFTM smart contract to transfer the NFT. As the attacker, by using the malicious JavaScript injected to the seller's webpage, we mananged to replace the account to be approved with the attacker's account. In the other case, we used `mitmproxy` to intercept the seller's request for listing the NFT. We managed to make the NFT be sold cheaper by decreasing the selling price in the request.

For the attack $\mathcal{A}8$ and $\mathcal{A}9$, we mimicked a buyer that buys an NFT via purchase and bidding on the Opensea website respectively. In the purchase case ($\mathcal{A}8$), as the buyer, we were required by the MS to send a transaction that invokes the NFTM smart contract to finish the purchase. As the attacker, we used `mitmproxy` to intercept the buyer's request and managed to set another NFT as the one to buy. We were also able to intercept the NFTM's response and changed the transaction destination address, causing the buyer to call a malicious contract that stole all the ETH attached to the transaction. Changing the destination address causes minor visual changes in MetaMask, as seen in Fig. 3. Crucially, MetaMask still shows that the transaction is associated with OpenSea. For the bidding case ($\mathcal{A}9$), as the attacker, we similarly used `mitmproxy` and managed to


Extension: (MetaMask) - MetaMask Notifi...
Rinkeby Test Network
Account 1 → 0x000...E581
New address detected! Click here to add to your address book.
https://testnets.opensea.io
0x000...E581 : FULFILL BASIC ORDER
1 ETH
DETAILS DATA HEX

Extension: (MetaMask) - MetaMask Notifi...
Rinkeby Test Network
Account 1 → 0x272...Bc45
New address detected! Click here to add to your address book.
https://testnets.opensea.io
0x272...Bc45 : FULFILL BASIC ORDER
1 ETH
DETAILS DATA HEX

(a) Untampered transaction (b) Tampered transaction

Fig. 3: 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.

make the buyer bid for another NFT or bid more. In addition, for a buyer that has not approved the NFTM smart contract to transfer its ERC20 tokens before, we were able to apply the same trick in the attack $\mathcal{A}4$ to give the attacker's account the transfer right.

6 Measurements

In this section, we first investigate the MSes hosted by the top ten NFTMs[3] and their susceptibility to BGP hijacks. Then, we analyze CIs' susceptibility to such attacks. Finally, we discuss the prevalence of route origin validation (ROV) in ASes serving NFTM users.

To investigate the susceptibility to routing attacks, we check the IP prefixes from which the MSes and CIs originate. A detailed analysis of IP prefixes is possible using RIPEstat, a tool provided by RIPE, which for an IP provides the announced IP prefix, source AS, route origin attestation (ROA), in addition to other metrics. We specifically investigate the service providers' susceptibility to BGP subprefix hijacking, a very effective form of BGP hijacking. We consider an IP prefix vulnerable to a subprefix hijack if: (i) no valid ROA exists and prefix length is less than 24, (ii) valid ROA exists but max-length field in ROA is stricly larger than prefix length and prefix length is less than 24.

The results of our analysis on the susceptibility of NFTMs is displayed in Table 2. One of the top ten NFTMs, Golom, has its MS originate in an IP prefix susceptible to subprefix hijacking. Five of the top ten marketplaces however

[3] According to `https://dappradar.com`

Table 2: NFTMs' susceptibility to prefix hijack attack. "Decentralized" architecture allow trading without MS interaction. MSes loading JavaScript from JavaScript providers vulnerable to subprefix hijacking are marked 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

included JavaScript from a JavaScript provider originating from such a vulnerable IP prefix. Thus, five of the top ten NFTMs are likely vulnerable to the JavaScript-based versions of our attacks. We were also able to identify several popular IPFS gateways[4] to be vulnerable to subprefix hijacks. Infura, a popular BaaS provider used by many Web3 applications and wallets such as MetaMask, has also been found to be vulnerable. Table 3 summarizes the vulnerable CIs.

Even if service providers themselves are not vulnerable to network attacks, as long as an attacker can obtain a fraudulent certificate for a service provider, and a user locates in a vulnerable AS or uses a DNS resolver located in a vulnerable AS, then our attacks can still be launched. Many practical CAs are still subject to network attacks [13,20] and therefore can be tricked to issue fraudulent certificates for arbitrary domains. Moreover, many ASes still do not enable ROV properly. For example, British Telecom, the largest cable provider in the UK with over 33% marketshare, does not perform ROV [44]. Similarly, Bell Canada and Rogers, two of the largest ISPs in Canada, have not implemented, or are just in the process of implementing ROV. None of the three main ISPs in Austria, A1, Hutchinson Drei, and Magenta, implement ROV [5]. To sum up, our attacks can affect a large number of users across the Internet.

[4] According to https://ipfs.github.io/public-gateway-checker/

Table 3: 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

7 Conclusion

This paper makes the first step in uncovering the security risks of centralization in the booming Web3 ecosystem. We focus on players around NFTs, one of the ecosystem's most important parts, and perform a systematic study of the architecture-level vulnerabilities induced by centralized entities. Our results confirm that centralization increases the overall attack surface by a wide margin. This is worrisome given the practicality and variety of the attacks, and the large financial investments in NFTs. We hope that our work will raise more awareness about the harm of centralization in the Web3 community.

References

1. Compound. https://compound.finance/. Accessed on 10.10.2022.
2. Dappradar. https://dappradar.com/nft/marketplaces/protocol/ethereum. Accessed 26/07/2022.
3. Infura. https://www.infura.io/.
4. Ipfs: Interplanetary file system. https://ipfs.tech.
5. Is bgp safe yet? https://isbgpsafeyet.com/. Accessed 19/10/2022.
6. List of nft marketplaces. https://dappradar.com/nft/marketplaces.
7. Opensea bug bounty program. https://hackerone.com/opensea.
8. Uniswap. https://uniswap.org/. Accessed on 10.10.2022.
9. Anders, Lance, and Shrug. Eip-4907: Rental nft, an extension of eip-721. Available: https://eips.ethereum.org/EIPS/eip-4907, March 2022.
10. Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.
11. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, 2017.
12. Juan Benet. Ipfs - content addressed, versioned, p2p file system (draft 3). https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

13. Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. Bamboozling certificate authorities with bgp. In *Proceedings of the USENIX Security Symposium*, 2018.
14. Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. *RHINE: Secure and Reliable Internet Naming Service*, pages 431–459. Springer International Publishing, Cham, 2022.
15. 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.
16. 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.
17. Mauro Conti, E. Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys Tutorials*, 20(4):3416–3452, 2018.
18. Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. From ip to transport and beyond: Cross-layer attacks against applications. In *Proceedings of the ACM SIGCOMM Conference*, 2021.
19. Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. The hijackers guide to the galaxy: Off-path taking over internet resources. 2021.
20. Tianxiang Dai, Haya Shulman, and Michael Waidner. Let's downgrade let's encrypt. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1421–1440, New York, NY, USA, 2021. Association for Computing Machinery.
21. 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 *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.
22. Dipanjan Das, Priyanka Bose, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. Understanding security issues in the nft ecosystem. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.
23. Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Impact of man-in-the-middle attacks on ethereum. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, 2018.
24. William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. Eip-721: Non-fungible token standard. Available: https://eips.ethereum.org/EIPS/eip-721, January 2018.
25. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.
26. DAN GOODIN. How 3 hours of inaction from amazon cost cryptocurrency holders $235,000. https://arstechnica.com/information-technology/2022/09/how-3-hours-of-inaction-from-amazon-cost-cryptocurrency-holders-235000/. Accessed 01/10/2022.
27. James Grimmelmann, Yan Ji, and Tyler Kell. Eip-5218: Nft rights management. Available: https://eips.ethereum.org/EIPS/eip-5218, July 2022.
28. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the USENIX Security Symposium*, 2015.

29. Bo Jiang, Ye Liu, and Wing Kwong Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.
30. David Johnston, Sam Onat Yilmaz, Jeremy Kandah, Nikos Bentenitis, Farzad Hashemi, Ron Gross, Shawn Wilkinson, and Steven Mason. Thegeneraltheoryofdecen-tralizedapplications, dapps. 2014.
31. Kacherginsky. Celer bridge incident analysis. https://www.coinbase.com/blog/celer-bridge-incident-analysis, September 2022. Accessed 28/09/2022.
32. Arnav Kapoor, Dipanwita Guhathakurta, Mehul Mathur, Rupanshu Yadav, Manish Gupta, and Ponnurungam Kumaraguru. Tweetboost: Influence of social media on nft valuation. In *Proceedings of WWW*, 2022.
33. Johannes Krupp and Christian Rossow. teEther: Gnawing at ethereum to automatically exploit smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
34. 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 *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2021.
35. Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2016.
36. 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.
37. Moxie Marlinspike. My first impressions of web3. https://moxie.org/2022/01/07/web3-first-impressions.html, January 2022. Accessed 01.10.2022.
38. Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017.
39. Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys Tutorials*, 21(1):838–857, 2019.
40. Daniel Perez and Benjamin Livshits. Broken metre: Attacking resource metering in EVM. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2020.
41. Anton Permenev, Dimitar Dimitrov, Petar Tsankov, Dana Drachsler-Cohen, and Martin Vechev. Verx: Safety verification of smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.
42. Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, and Ronan Sandford. Eip-1155: Multi token standard. Available: https://eips.ethereum.org/EIPS/eip-1155, June 2018.
43. Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *Proceedings of Financial Cryptography and Data Security (FC)*, 2022.
44. Statista. Market share of telecommunications operators in the united kingdom (uk) from 2007 to 2021, by fixed broadband subscribers. https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/, 2022. Accessed 19/10/2022.
45. 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 *Proceedings of the USENIX Security Symposium*, 2021.

```
"id": "CollectionCreateOrUpdatePageEditMutation",
"query": "...",
"creatorFees": [
  { // address to be set to the attacker's account
    "address": "0x9e9675b276a3695163b230fb14cd6decf2364497",
    "basisPoints": 1000
  }
]
```

Listing 1: Selected parameters in the request for changing royalty payout address

46. Verified Market Research (VMR). Non-fungible tokens market size and forecast. Technical report, 2022.
47. 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 *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022.
48. Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

A Attacks Involving a Centralized Marketplace

Royalty payout address change ($\mathcal{A}3$). To simulate the attack, as the user, we tried to change the royalty payout address on OpenSea. After we entered the new payout address and confirmed the change, the browser sent a request, i.e. a GraphQL query to the MS. The request was to set the new royalty payout address according to the query parameter *address* shown in Listing 1. Meanwhile, as the attacker, we used mitmproxy to intercept the request and set the paramter *address* to the attacker's account. Then the royalty payout address was changed accordingly.

Wrong data fetched from NFTM ($\mathcal{A}5$, $\mathcal{A}6$, $\mathcal{A}7$). To simulate the attack $\mathcal{A}5$, as the buyer, we tried to browse the bids for an NFT on OpenSea. For simplicity, we only considered one bid. To fetch the bid, the browser sent a request, i.e. a GraphQL query to the MS. In the MS's response to the query, the parameter *perUnitPriceType* specifies the bid's price. As the attacker, we used mitmproxy to intercept the MS's response. We made each field of the parameter *perUnitPriceType* larger, causing the buyer to see a bid with a higher price.

For the attack $\mathcal{A}6$, as the buyer, we browsed an NFT on OpenSea. The browser fetched the NFT image from a central source :i.seadn.io/gae/<asset identifier>, where the asset identifier refers to the identifier of the NFT image. As the attacker, we used mitmproxy to intercept the request before it reached MS. We changed the asset identifier to that of another NFT, causing the buyer

```
"maker": "...",
"taker": "...",
"perUnitPriceType": {
  // each field to be set to a larger value
  "eth": "0.0225",
  "usd": "29.242800000000003",
  "unit": "0.0225"
}
```

Listing 2: Selected parameters in the response for fetching existing bids

```
"id": "CreateListingActionModalQuery",
"query": "...",
"price": {
  "paymentAsset": "...",
  "amount": "1" // to be set to a larger value
},
"recipient": null // to be set to the attacker's account
```

Listing 3: Selected parameters in the request for listing an NFT for sale

to receive the wrong image. The attack $\mathcal{A}7$ was validated in the same except that as the attacker, we used mitmproxy to intercept the MS's response and directly changed the image to be received by the buyer. As a result, the buyer may wrongly believe to inspect a highly valuable NFT sold for a cheap price. This attack is distinct from counterfeit NFTs which just copy assets of expensive NFTs because, unlike for counterfeits, the contract address displayed on OpenSea is still that of the genuine NFT collection.

Change order or transaction data ($\mathcal{A}4$, $\mathcal{A}8$, $\mathcal{A}9$). To simluate the attack $\mathcal{A}4$, as the seller, we listed an NFT for sale via OpenSea. In the first case that the seller lists an NFT from a smart contract new to the NFTM, after we clicked the button for completing listing, we were required to send a `Delegate` transaction that approves the NFTM smart contract to transfer the NFT. The transaction parameters were supposed to be fetched from the MS with the browser sending a request i.e. a GraphQL query to the MS. As the attacker, we used the malicious JS injected into the seller's web page to bind another function to the button. The seller was still required to send a transaction but the transaction parameters were provided by us. The attacker's account was instead approved for the transfer.

In the other case that the seller lists an NFT from a smart contract known to the NFTM, the step for approval is skipped. We were directly required to create a sell order, which would be stored on the MS and used when a buyer purchases the NFT. For this, the browser first sent a request, i.e. a GraphQL query to the MS. This request contained the parameter *amount* and *recipient*

```
"source": null,
"destination": {
  // to be set to the address of the malicious smart contract
  "value": "0x2722E1ADC11760A4C9Cca21d6311C55d5e6ABc45"
},
"value": "200000000000000",
"data": "..."
```

Listing 4: Selected parameters in the response for purchasing an NFT

```
"id": "BulkPurchaseActionModalQuery",
"query": "...",
"ordersToFill": [
  { // order to be set to the sell order of another NFT
    "order": "T3JkZXJWM1R5cGU6ODQ5NDkz",
    "itemFillAmount": "1"
  }
]
```

Listing 5: Selected parameters in the request for purchasing an NFT

which specify the price and the intended buyer respectively as shown in Listing 3. The MS replied with the order information for the seller to create the order. As the attacker, we used mitmproxy to intercept the seller's request and changed the parameter *amount* and *recipient* to a lower value and the attacker's account respectively. Then only we could buy the NFT on OpenSea, and we managed to buy it at a cheaper price.

For the attack $\mathcal{A}8$ and $\mathcal{A}9$, we mimicked a buyer that buys an NFT via purchase and bidding on OpenSea respectively. In the purchase case ($\mathcal{A}8$), as the buyer, we were required to send a transaction that invokes the NFTM smart contract to finish the purchase. For this, the browser sent a request, i.e. a GraphQL to the MS. The parameter *order* in the request specifies the sell order identifier of the NFT as shown in Listing 5. The MS replied with transaction parameters for the buyer to create the transaction. The parameter *value* inside *destination* specifies the transaction destination address. As the attacker, when we chose to intercept the request, we changed the parameter *order* to that of another NFT, causing the buyer to buy another one. When we chose to intercept the response, we changed the parameter *value* to the address of a malicious smart contract, stealing the ETH that the buyer attached to the transaction.

In the bidding case ($\mathcal{A}9$), we were required to create a buy order. The browser sent a request, i.e. a GraphQL to the MS for getting the parameters to create the order. As the attacker, we used mitmproxy to intercept the request. As shown in Listing 6, on our first try, we changed the parameter *asset* to that of another NFT, causing the buyer to bid another NFT; on our second try, we changed the

```
"id": "CreateOfferActionModalQuery",
"query": "...",
"item": { // asset to be set to that of another NFT
  "asset": "QXNzZXRUeXBlOjEzODk5MjU5NQ==",
  "quantity": "1"
},
"price": {
  "paymentAsset": "UGF5bWVudEFzc2V0VHlwZToONA==",
  "amount": "0.0003" // to be set to a larger value
}
```

Listing 6: Selected parameters in the request for bidding an NFT

parameter *amount* to a larger value, causing the buyer to bid more. Furthermore, for bidding on OpenSea, a buyer is required to use ERC20 tokens. If a buyer does not approve the NFTM smart contract to transfer its ERC20 tokens before, the buyer is required to sent a transaction for the approval. As the attacker, like what we did in $\mathcal{A}4$, we managed to gain the transfer right by using the malicious JS injected into the buyer's webpage.

B Attack Mitigation

Secure Internet Routing. Our attacks exploit known vulnerabilities of today's Internet to attack Web3. Secure inter-domain routing protocols such as SCION [14] can be used in place of the vulnerable BGP to protect against prefix hijacking attacks.

Application Layer Authentication. To protect against attacks targeting transaction parameters sent by the MS to the user, the wallet can establish shared keys with the server without relying on the web PKI but instead on the NFTM's blockchain public key. Blockchain keys are tied on an entity's identity and are only seldomly changed. It is thus reasonable to bake a list of public keys into the wallet to enable marketplace authentication. When a user visits the marketplace, a handshake between the wallet and the MS is performed. The resulting keys allow efficient authentication of transaction parameters. Cryptographic keys are meant to only be used for a single purpose, so great care has to be given to not introduce further attacks by using a blockchain keypair also for key exchange.

Improving User Interface. Wallets such as Metamask do not effectively display transaction information. Improving the interface with clearer representations and explanations can help users identify potential attacks and enhance the security of NFT transactions.