BRC20 Snipping Attack

Minfeng Qi[1,*], Qin Wang[2,*], Ningran Li[3], Shiping Chen[2], Tianqing Zhu[1]
[1]*City University of Macau*, China
[2]*CSIRO Data61* | [3]*University of Adeleide*, Australia

ABSTRACT

In this paper, we introduce and implement *BRC20 sniping attack*. Our attack manipulates the BRC20 token transfers in open markets and disrupts the fairness among bidding participants. The long-standing principle of "highest bidder wins" is rendered ineffective.

Typically, open BRC20 token markets rely on Partially Signed Bitcoin Transactions (PSBT) to broadcast selling intents and wait for buying auctions. Our attack targets the BRC20 buying process (i.e., transfer) by injecting a front-running transaction to complete the full signature of the PSBT. At its core, the attack exploits the mempool's fee-based transaction selection mechanism to snipe the victim transaction, replicate metadata, and front-run the legesmate transction. This attack applies to platforms using PSBT for BRC20 token transfers, including popular Bitcoin exchanges and marketplaces (e.g., Magic Eden, Unisat, Gate.io, OKX).

We implemented and tested the attack on a Bitcoin testnet (regtest), validating its effectiveness through multiple experimental rounds. Results show that the attacker consistently replaces legitimate transactions by submitting higher-fee PSBTs. We have also made responsible disclosures to the mentioned exchanges.

1 INTRODUCTION

Since mid-2024, the Bitcoin network has witnessed [1] a sharp rise in so-called *mempool snipping*, a fee-based form of transaction front-running that threatens to disrupt ordinary users' ability to participate in Ordinals [2] and BRC20 trading [3–8]. At its core, mempool snipping relies on the attacker's capacity to monitor unconfirmed transactions and outbid legitimate buyers by submitting higher-fee alternatives. No direct loss of funds is reported for users who experience this phenomenon, while the damage manifests in a degraded marketplace. Participants may be repeatedly outbid and ultimately deterred from engaging in token purchases.

According to data shared by community members, this practice is no longer a rare occurrence. The Wizards of Ord "Snipes" Discord channel [9] recorded 220 daily snipes in one instance, amounting to 3.9 BTC (approximately $247,000), and similar activity levels have been reported consistently over the course of several weeks. High-profile figures have encountered similar setbacks. For instance, on July 10, an individual affiliated with the Bored Ape Yacht Club engaged in what community members termed a "sniping battle," ultimately forfeiting over $5,800 in fees without succeeding in the attempted purchase of an Ape Hoodie Ordinal [10]. Another indicative incident occurred during the highly anticipated OrdiBots mint [11], which had garnered substantial attention among Ordinals enthusiasts. The enthusiasm quickly soured when numerous participants experienced mempool sniping throughout the launch.

Despite widespread awareness of mempool sniping, the technical underpinnings of Ordinals and BRC20 transfers have yet to be fully examined in the context of fee-driven front-running. At the heart of these trading workflows lies the partially signed Bitcoin transaction (PSBT) mechanism, which enables sellers to broadcast sale data and buyers to finalize the purchase. This convenience, however, introduces an opening for adversaries to intercept, modify, or outbid legitimate transactions before they reach confirmation, a phenomenon we term the *BRC20 sniping attack*.

> **Evidence:** A poll [12] conducted by Magisat's founder on June 12 revealed that, out of 229 respondents, **73%** confirmed having been sniped at least once while attempting to secure Ordinal assets.

In this paper, we present a deep dive into *how BRC20 token transfers via PSBT are susceptible to sniping.*

① **We conduct the first formal treatment of the PSBT-based BRC20 transfer process, which is the key process for building open marketplaces within the Bitcoin ecosystem.**

BRC20 tokens, unlike Ethereum-based tokens [13–15], lack native smart contract support. Instead, BRC20s rely on Bitcoin's UTXO (Unspent Transaction Output) model. This necessitates a structured transfer mechanism to handle token movements while preserving the system integrity. The PSBT standard (BIP-174 [16]) provides a means to separate transaction creation and signing across multiple parties, making it a cornerstone for implementing BRC20 token marketplaces. However, this introduces vulnerabilities as incomplete transactions are visible and manipulable before their finalization.

Our analysis formalizes the BRC20 transfer workflow through PSBT by decomposing its phases: seller initialization, buyer signing, and network broadcast. We introduce a mathematical model capturing the dependencies between transaction inputs, outputs, and the metadata inscribed in PSBTs. By detailing how token transfer inscriptions (e.g., JSON-based metadata) are integrated into the Bitcoin blockchain, we identify the system's reliance on mempool transparency and fee-driven prioritization.

② **We propose a new attack we term *BRC20 sniping attact*, where partial/incompleted PSBTs can be replicated with a higher fee, displacing legitimate transactions within the mempool from malicious buyers.**

Our sniping attack exploits the transparent nature of the mempool and the predictable workflow of PSBT-based transactions. In our attack, the adversary monitors the mempool for unconfirmed BRC20 transactions, identifies partially signed PSBTs containing token transfer instructions, and crafts a competing transaction. By replicating the legitimate transaction's core details and outbidding it with a higher fee, the attacker ensures their transaction is prioritized for block inclusion, effectively invalidating the original.

Our attack not only disrupts the fairness of BRC20 marketplaces but also undermines user confidence in their reliability. More importantly, the damage can amplify significantly as our attack is widely applicable (details in Table 2).

---

* Equal contribution.

It is important to note that we do **NOT** claim novelty in the concept of mempool sniping, as extensively discussed in prior works. However, our study is the first to investigate the sniping attack specifically within the context of BRC20 token transfers and the PSBT protocol. The complexities introduced by these additional protocols pose greater analytical challenges compared to simpler mempool injection or traditional front-running attacks. Our attack, despite such complexity, demonstrates a fully operational implementation. For better understanding, we also summarize relevant (mempool) attack vectors in Table 1.

③ **We implement and validata our attack in a controlled environment. We conduct a detailed local experiment that simulates real-world conditions of network congestion, fee competition, and opportunistic adversaries.**

We developed a controlled environment using Bitcoin's regtest mode. We emulate real-world scenarios, including mempool congestion, varying transaction fees, and adversarial behaviors. We created a local marketplace for BRC20 tokens and implemented the attack by crafting sniping transactions with varying fee levels.

Our experiments show that attackers with minimal resources can consistently outbid legitimate buyers, achieving block inclusion for their malicious transactions. We measured the success rate of the attack under different network conditions and fee strategies, confirming its reliability. Additionally, we tracked the impact on token balances, demonstrating how the attacker successfully redirected tokens to their control while rendering the legitimate transaction invalid. These findings validate the attack's practicality.

④ **We present our suggestion for mitigation**.

We propose three mitigation strategies: (i) adding mempool protection, (ii) enabling dynamic fee escalation, and (iii) implementing a fee-locking mechanism. For instance, the advanced fee-locking mechanism embeds a maximum fee commitment within the PSBT, preventing attackers from arbitrarily escalating fees. This mechanism leverages cryptographic commitments to ensure that only transactions adhering to the agreed-upon fee structure are accepted by the network. We also explore the feasibility of integrating dynamic fee adjustments, where users can pre-authorize higher fees in case their initial transaction is outbid.

In short, our contributions are:
- formalisation for the PSBT-based BRC20 transfer process (§3);
- the new BRC20 sniping attack (§4, §6);
- implementation and validation of the attack (§5);
- mitigation strategies (§7).

> **Responsible disclosure.** We have notified our attack findings to four leading BRC20 token platforms and marketplaces, i.e., *Magic Eden*, *Unisat*, *Gate.io*, *OKX*, which are potentially affected by this vulnerability.

2 TECHNICAL WARMUPS

Bitcoin's UTXO. Bitcoin's UTXO (Unspent Transaction Output) [42, 43] is a fundamental concept in its transaction model, representing outputs of transactions that have not yet been spent. Each UTXO is tied to a Bitcoin address and controlled by a private key, allowing the holder to spend it. UTXOs are indivisible, meaning they must be fully consumed in a transaction, with any leftover value returned

as a new UTXO to the sender. The Bitcoin network maintains a global UTXO set, which tracks all unspent outputs and forms the basis for validating transactions. A transaction spends UTXOs as inputs and creates new UTXOs as outputs, ensuring a clear lineage for every Bitcoin. We present a formalized version in §3.

The UTXO model enhances security, efficiency, and transparency. UTXOs prevent double-spending since each can only be spent once and ensure traceability through their transactional history. Unlike the account model used in blockchains like Ethereum [44, 45], UTXOs eliminate the need to manage balances directly, instead relying on the creation and consumption of outputs.
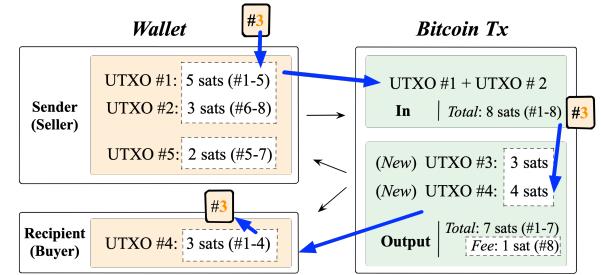

Figure 1: Transferring BRC20/Inscriptions via the UTXO model

Bitcoin's inscription. Creating an inscription transaction involves two distinct phases [29, 46]:

- **Commit transaction.** This phase begins with the creation of a taproot output [47, 48] that cryptographically commits to a script referencing the inscription data. Importantly, this commitment is designed to reference the data without directly revealing its content, ensuring confidentiality until the appropriate stage. The commit transaction is then broadcast to the Bitcoin network, where it gets included in a block on the blockchain. This process securely anchors the commitment on-chain, forming the foundation for the subsequent phase.

- **Reveal transaction.** In this phase, the output from the commit transaction is spent in a new transaction known as the reveal transaction. This transaction includes the actual inscription data embedded within its script, fully disclosing the committed content. Once the reveal transaction is confirmed on the blockchain, the inscription data becomes permanently stored on-chain, ensuring public accessibility, verifiability, and immutability. This phase completes the inscription process, allowing the data to be associated with the Bitcoin network permanently.

BRC20. The BRC20 protocol [4, 5, 46] builds upon inscription technology to embed personalized content into each unique satoshi, the smallest unit of Bitcoin. The inscribed data is assigned a P2TR (Pay-to-Taproot) script type and stored within the SegWit witness section [49] of a transaction, ensuring compatibility with the Bitcoin network. Inscriptions can take diverse formats, including JSON-based text, images, audio, and even highly compressed video files. For text-based BRC20 tokens, such as ORDI [50], the on-chain inscription includes key details: the protocol name (brc20), operation type (e.g., transfer), token name (e.g., ordi), total token supply (max), maximum tokens minted per round (lim), and the quantity to be minted or transferred (amt). This structured approach ensures accurate and permanent recording of token metadata on-chain.

Table 1: Mempool-Related Attacks / Exploits

Aspect	Target	Objective	Key Exploit	Mechanism	Platform
Frontrunning [17–20]	Pending transactions	Displace transaction	Fee bidding	Fee escalation or reduction (backrunning)	EVM-compatible
MEV arbitrage [21–23]	Transaction orderings	Extract unfair value	Miner prioritization	Order manipulation/censorship	EVM-compatible
Sniping bots [24–26]	Token/NFT auctions	Capture mints	Gas auctions	Automated gas bidding in mints	EVM-compatible
Imitation attack [27]	Pending tx/contracts	Extract contract value	Contract cloning	Copy and frontrun (see above) contracts	EVM-compatible
RBF exploits [28–30]	Low-fee transactions	Accelerate confirmation	Fee prioritization	Replace transactions	Bitcoin (UTXO)
Tx malleability [31, 32]	Transaction IDs	Invalidate/hijack Tx	Signature alteration	Alter tx IDs before confirmation	Bitcoin (UTXO)
Pinning attacks [29, 30, 33]	Inscriptions/BRC20s	Drain out liquidity	Locking UTXOs	Fee escalation + full withdraw order	Bitcoin (UTXO)
Timejacking attack [34]	Node timestamps	Disrupt transaction order	False timestamp	Alter node time to influence validation	All blockchains
Double-spend [35–37]	UTXOs	Spend coins twice	Conflicting Tx/branch	Broadcast multiple tx with same inputs	All blockchains
DDoS [38–41]	Mempool capacity	Block valid tx propagation	Spam tx flooding	Overload mempool with low-priority tx	All blockchains
Ours	**PSBT/Inscription/BRC20**	**Hijack token transfers**	**Metadata in PSBT**	**Fee escalation + PSBT metadata replication**	**Bitcoin (UTXO)**

The protocol leverages Ordinal Theory [2], which functions as a numbering system to index every indivisible satoshi, enabling them to be uniquely tracked and transferred as individual entities. This indexing mechanism ensures that each satoshi carrying an inscription, such as BRC20 metadata, becomes non-fungible, effectively differentiating it from others. This contrasts with Ethereum, which inherently supports smart contracts for token creation and management, making the BRC20 protocol a distinct and innovative approach to tokenization within the Bitcoin ecosystem.

Transferring BRC20. BRC20 tokens lack built-in smart contracts and instead operate using Bitcoin's UTXO model and transaction inscriptions. Inscriptions only store plaintext messages about BRC-20 actions, not executable logic. Real-time balance updates depend on off-chain indexers for data retrieval. We explain more below.

- **On-chain storage.** BRC-20 tokens are fully recorded on-chain, with their movements tied to Bitcoin's native transactions. The Transfer operation corresponds to the sending and receiving of satoshis on the layer-one Bitcoin network. However, the Bitcoin network only logs transaction data, not the real-time status of BRC-20 tokens or balances. Unlike account-based systems, advanced features like balance tracking are offloaded to external layers, keeping the on-chain ledger lightweight and efficient.
- **Off-chain retrievals.** Off-chain indexers [46, 51] handle real-time data for BRC-20 tokens. These services track token minting, determine the cessation points, and map token trades to specific wallets. They only observe on-chain transactions without modifying them, ensuring data integrity. If an indexer fails, token data remains secure, as it can be reconstructed by reprocessing blockchain transactions. Many wallets, like Ordinals Wallet [52], UniSat [53], and Binance's hot wallet [54], integrate indexer functionalities seamlessly for users.

3 FORMALISING BRC20 TRANSFER ON PSBT

3.1 Basics

Transaction model. In the Bitcoin network, transactions are the fundamental units[1] that facilitate the transfer of value, operating under the UTXO model. A Bitcoin transaction $\mathcal{T}$ can be formally defined as a tuple $\mathcal{T} = (\text{txid}, \text{In}, \text{Out}, \text{Witness})$, where:

- **txid** $= \mathcal{H}(\text{serialize}(\mathcal{T}))$ is a unique identifier for transactions, derived from the double SHA-256 hash function $\mathcal{H}$ applied to the serialized transaction data.
- **In** $= \{I_1, I_2, \ldots, I_n\}$ is the set of inputs. Each input I_i is a tuple: $I_i = (\text{txid}^{(i)}, \text{vout}^{(i)}, \text{scriptSig}^{(i)}, \text{sequence}^{(i)})$ where $\text{txid}^{(i)}$ and $\text{vout}^{(i)}$ reference the originating transaction and the specific output index being consumed, respectively. The scriptSig$^{(i)}$ provides the necessary cryptographic proof to authorize the spending of the referenced UTXO. sequence$^{(i)}$ enables features such as *Replace-By-Fee* (RBF, to be explained soon) and facilitating relative lock-time constraints.
- **Out** $= \{O_1, O_2, \ldots, O_m\}$ is the set of outputs. Each output O_j is defined as: $O_j = (\text{amount}_j, \text{scriptPubKey}_j)$ where amount_j specifies the value being transferred, and scriptPubKey_j contains the locking script that sets the conditions under which the output can be spent in future transactions. There are several commonly used output types, including *P2PKH (Pay-to-PubKey-Hash)*, *P2SH (Pay-to-Script-Hash)*, *P2WPKH (Pay-to-Witness-PubKey-Hash)*, *P2WSH (Pay-to-Witness-Script-Hash)*, and *P2TR (Pay-to-Taproot)*.
- **Witness** is the data structure introduced by Segregated Witness (SegWit) to hold the unlocking data for witness-based outputs like *P2WPKH*, *P2WSH*, and also *P2TR*. Conceptually, the witness serves as the new location for the data that would otherwise appear in the scriptSig of legacy transactions. Unlike scriptSig, the witness section does not employ Bitcoin script opcodes directly; rather, it is composed solely of data pushes. Each data push corresponds to a piece of unlocking data (e.g., signatures, public keys, or entire witness scripts) required to satisfy the output's conditions.

UTXO. The UTXO set $\mathcal{U}$ is a finite collection of all unspent outputs (**txid**, **vout**, amount, scriptPubKey). The state transition equation below illustrates how $\mathcal{U}$ evolves with each transaction $\mathcal{T}$.

$$\mathcal{U}' = \mathcal{U} \setminus \left\{ \left(\text{txid}^{(i)}, \text{vout}^{(i)} \right) \mid I_i \in \text{In} \right\}$$
$$\cup \left\{ \left(\text{txid}, j, \text{amount}_j, \text{scriptPubKey}_j \right) \mid O_j \in \text{Out} \right\}.$$

When a transaction $\mathcal{T}$ is processed, it consumes specific UTXOs referenced by its inputs **In**. These consumed UTXOs are removed from the current UTXO set $\mathcal{U}$. Simultaneously, the transaction generates new UTXOs through its outputs **Out**. These new UTXOs are added to form the updated UTXO set $\mathcal{U}'$.

---

[1]We denote Bitcoin unit as ₿ (i.e., BTC) and BRC20 token unit as ฿ .

3.2 Transaction Replacement

Transaction Fee. Transaction fees serve as an incentive for miners to include transactions in the next block and act as a mechanism to prevent network abuse by adding a small cost to each transaction. The transaction fee (f) is determined by the difference between the input and output values, which can be calculated as:

$$f = \sum_{i=1}^{n} \text{amount}(I_i) - \sum_{j=1}^{m} \text{amount}(O_j). \qquad (1)$$

Miners collect this difference as compensation for processing the transaction and securing the network. The fee is typically measured in satoshis per byte (sat/byte), and the exact fee needed for a transaction depends on network congestion and competition for block space at the time. Fig.6 depicts the average Bitcoin transaction fee in USD (left y-axis) and the number of Bitcoin transactions per day (right y-axis) over time, spanning from January 2022 to October 2024. The transaction fee remained relatively low and stable, fluctuating around $20 until early 2023. After this point, there was a noticeable increase in volatility, with transaction fees spiking multiple times, reaching above $120 at several points in 2024. The creation of the BRC20 standard and the Bitcoin halving event appear to be associated with a significant spike in both the average transaction fee and the number of transactions.

Transaction replacement. The transaction replacement mechanism allows users to replace an unconfirmed transaction (*i.e.*, $\mathcal{T}$) with a new one (*i.e.*, $\mathcal{T}'$) that offers a higher fee. It mandates that the input sets remain identical, $\textbf{In}' = \textbf{In}$, ensuring that both transactions consume the exact same UTXOs from the current UTXO set $\mathcal{U}$. However, the outputs $\textbf{Out}'$ in $\mathcal{T}'$ are required to differ from those in $\mathcal{T}$. Specifically, $\mathcal{T}'$ typically lowers the output amounts to assign a higher transaction fee, which can be mathematically expressed as:

$$\sum_{j=1}^{m'} \text{amount}(O_j') < \sum_{j=1}^{m} \text{amount}(O_j),$$

where m and m' denote the number of outputs in $\mathcal{T}$ and $\mathcal{T}'$, respectively. The increased fee $f' = \sum_{i=1}^{n} \text{amount}(I_i) - \sum_{j=1}^{m'} \text{amount}(O_j')$ in $\mathcal{T}'$ serves as an incentive for miners to prioritize the replacement transaction over the original one.

> **Example**: The Bitcoin history (cf. link) shows that a transaction (i.e., txid.fc7c) was replaced with a new transaction (i.e., txid.e744) with a fee rate of 12.8 sat/vB, higher than the initial 9.04 sat/vB.

3.3 PSBT Protcol

PSBT. PSBT, short for partially signed Bitcoin transactions, is a Bitcoin standard (BIP-174 [16]) for Bitcoin transactions designed to separate the construction and signing processes. Formally, a PSBT can be represented as a tuple $\mathcal{P} = (\mathcal{T}, \textbf{In}_{\text{PSBT}}, \textbf{Out}_{\text{PSBT}})$, where $\mathcal{T}$ denotes the underlying Bitcoin transaction $\mathcal{T} = (txid, In, Out, Witness)$, and $\textbf{In}_{\text{PSBT}}$ and $\textbf{Out}_{\text{PSBT}}$ encapsulate the additional metadata required for partially signing the transaction.

Each input $I_i \in \textbf{In}$ within the transaction $\mathcal{T}$ corresponds to an input in $\textbf{In}_{\text{PSBT}}$, augmented with supplementary information such as the full UTXO details ($\textbf{txid}^{(i)}, \textbf{vout}^{(i)}, \text{amount}^{(i)}, \text{scriptPubKey}^{(i)}$),

the non-witness UTXO if applicable, and any partial signatures $\sigma^{(i)}$. Mathematically, the PSBT input can be denoted as:

$$I_i^{\text{PSBT}} = \left(I_i, \text{UTXO}^{(i)}, \sigma^{(i)} \right)$$

where $\text{UTXO}^{(i)}$ provides the necessary context for signing, and $\sigma^{(i)}$ is the partial signature obtained from one or more signers.

Similarly, each output $O_j \in \textbf{Out}$ is associated with $\textbf{Out}_{\text{PSBT}}$, which may include additional information such as the redeem script or witness script required for spending conditions. This can be expressed as:

$$O_j^{\text{PSBT}} = \left(O_j, \text{RedeemScript}_j \right)$$

ensuring that all participants have access to the necessary scripts to validate and sign the transaction.

The PSBT protocol operates through a series of stages:

- **Creation**: An initial PSBT is generated, containing the transaction structure $\mathcal{T}$, the input references $\textbf{In}_{\text{PSBT}}$, and the output definitions $\textbf{Out}_{\text{PSBT}}$. At this stage, the PSBT includes all the information required to fully construct the transaction, except for the signatures.
- **Signature propagation**: The PSBT is shared among the involved multiple signers. Each participant reviews the PSBT, adds their partial signatures $\sigma^{(i)}$, and updates the PSBT accordingly. This process can be iterated until all signatures are collected.
- **Finalization**: Once all required signatures are present, the PSBT is finalized by embedding the complete signature data into the transaction $\mathcal{T}$, producing the fully signed transaction $\mathcal{T}'$. $\mathcal{T}'$ then is serialized and broadcasted to the Bitcoin network for inclusion in a block.

Formally, the transformation from a partially signed PSBT $\mathcal{P}$ to a fully signed transaction $\mathcal{T}'$ can be depicted as:

$$\mathcal{T}' = \textit{Finalize}(\mathcal{P})$$

where the *Finalize* function aggregates all partial signatures $\sigma^{(i)}$ and integrates them into the corresponding inputs of $\mathcal{T}$, resulting in a valid and broadcast-ready transaction $\mathcal{T}'$.

3.4 BRC20 Transfer via PSBT

Building upon the standards of BRC20 and PSBT, we formalize the lifecycle of a PSBT-based BRC20 transfer. Let

$$\Omega = \{\text{p, op, tick, amt}\}$$

denote the core inscription metadata that specifies the protocol name (p = brc-20), the operation type (op), the token name (tick), and the transfer amount (amt). In particular, for a transfer operation, we might have:

$$\Omega = \{\text{p : brc-20, op : transfer, tick : ordi, amt : 1000}\}.$$

- **Seller's operation.** A seller begins by *inscribing* a satoshi with metadata Ω that encodes the intention to transfer amt = 1000 BRC20 ₿ (e.g., tick = ordi). Formally, we can regard this inscription event as creating a tuple

$$\mathcal{I} = (\textbf{txid}_s, \textbf{vout}_s, \Omega),$$

where $\textbf{txid}_s$ and $\textbf{vout}_s$ identify the unspent output hosting the inscribed satoshi. The existence of $\mathcal{I}$ indicates the seller's willingness to exchange these tokens for a certain Bitcoin amount (e.g., 0.2 ₿).

- **Creation of PSBT.** Next, the seller assembles a partially signed transaction $\mathcal{P}$ that integrates $\mathcal{I}$ as one of the inputs. Concretely, the seller designates an output

$$O_s = (\ 0.2\ ₿,\ \text{scriptPubKey}_s)$$

to reflect the requested price. In other words, by offering an output crediting 0.2 ₿ to one of the seller's addresses, the seller stipulates that any buyer must fund this output in exchange for the amt = 1000 ₿ inscribed on the satoshi from $\textbf{txid}_s$, $\textbf{vout}_s$. The rest of the transaction structure (including any change outputs to the buyer or additional $\texttt{witness}$ fields referencing Ω is specified but not fully signed.

- **Publishing the PSBT.** The seller then publishes $\mathcal{P}$ to a marketplace or similar off-chain platform, making it visible to prospective buyers. The marketplace now holds a partially signed transaction

$$\mathcal{T} = \mathcal{P}\big(\textbf{In}_{\text{mkt}}, \textbf{Out}_{\text{mkt}}\big),$$

which includes the vital metadata Ω describing the token type and amount. At this stage, no complete signature for the spending input $\mathcal{I}$ is present, or only a partial signature belonging to the seller exists (depending on multi-sig policies).

- **Buyer's operation.** A buyer who wishes to acquire 1000 ordi₿ must *finalize* $\mathcal{P}$ by providing the necessary 0.2 ₿ input(s) to the transaction and appending their own signatures. Suppose the buyer's contribution is a UTXO $\big(\textbf{txid}_b, \textbf{vout}_b, \text{amount}_b\big)$ such that $\text{amount}_b \geq 0.2$ ₿. The buyer modifies

$$\textbf{In}_{\text{PSBT}}\ \cup\ \{(\textbf{txid}_b, \textbf{vout}_b)\}$$

to incorporate this new input, ensuring the resulting transaction covers the required payment to scriptPubKey_s. The buyer then attaches any partial signatures σ_b necessary to authorize the spending of $\big(\textbf{txid}_b, \textbf{vout}_b\big)$.

- **Finalizing the PSBT.** When all inputs are properly signed—i.e., the seller's signature on $(\textbf{txid}_s, \textbf{vout}_s)$ plus the buyer's signature on $(\textbf{txid}_b, \textbf{vout}_b)$—the PSBT is transformed into a valid raw transaction. One may express this as

$$\mathcal{T}'\ =\ \textit{Finalize}\big(\mathcal{P}\big),$$

where $\mathcal{T}'$ denotes the fully signed Bitcoin transaction. Once the buyer broadcasts $\mathcal{T}'$ to the network, nodes incorporate it into the mempool and, upon successful validation and block inclusion, the transaction becomes irrevocable.

- **Off-chain state updates.** Subsequent to on-chain confirmation, off-chain ledgers or token-management layers update the final token states $\mathcal{S}$. Let

$$\begin{cases} \mathcal{S}_{\text{buyer}} \leftarrow \mathcal{S}_{\text{buyer}} + 1000₿, \\ \mathcal{S}_{\text{seller}} \leftarrow \mathcal{S}_{\text{seller}} - 1000₿, \end{cases}$$

denote the change in ordi ₿ balance, assuming an off-chain indexer or marketplace logic tracks the BRC20 token state. At the same time, the seller's on-chain balance increases by 0.2 ₿ , while the buyer's on-chain balance decreases accordingly. In other words, the cryptographic settlement on Bitcoin triggers

an atomic exchange of the BRC20 tokens for BTC, ensuring $\mathcal{T}'$ captures the binding transaction logic.

It is worth noting that the protocol necessitates two on-chain transactions to finalize the Transfer operation. The first transaction is to signify the action, while the second is an actual transaction on-chain (detailed mechanism refers to [29]). Our attack focuses on the second transaction.

4 BRC20 SNIPPING ATTACK

We present a vulnerability tied to the way BRC20 transactions are published using PSBT, termed *BRC20 snipping attack*.

4.1 Attack Description

A snipping attack in the context of BRC20 tokens on Bitcoin is a focused form of fee-driven front-running that targets a PSBT before it is fully confirmed. Its defining feature involves intercepting the partially signed data and crafting a competing transaction that replicates the essential transfer details. Unlike standard front-running methods, which typically exploit fee escalations, a snipping attack seizes upon the unique conditions of partially signed workflows. By broadcasting a version of the transaction with a higher fee than the legitimate proposal, the attacker ensures that miners will prioritize the malicious transaction, thus rendering the original transaction invalid or unconfirmable.

4.2 Motivation

The snipping attack typically manifests in a marketplace environment where BRC20 tokens are traded using partially signed transactions. In such a setting, sellers initiate token transfers by creating PSBTs that outline the details of the transaction, including the amount of tokens to be transferred and the associated Bitcoin inputs and outputs. These PSBTs are often shared among multiple participants, such as buyers, sellers, and possibly escrow services, to gather the necessary signatures before the transaction is finalized and broadcasted to the network. This collaborative workflow, while facilitating multi-party authorization, inadvertently exposes the PSBT to potential interception by malicious actors.

Within this context, multiple buyers may concurrently compete to finalize a token transfer, each aiming to have their transaction confirmed promptly to secure the desired tokens. The motivations driving an attacker to execute such an attack are multifaceted.

- The attacker seeks financial gain by redirecting valuable BRC20 tokens to their own address, stealing assets intended for legitimate buyers.
- By invalidating the original transaction, the attacker can cause a denial of service, disrupting the intended transfer and potentially causing economic losses for both buyers and sellers.
- The attacker may also aim to manipulate the market by affecting token prices or eroding buyer trust. Repeated successful snipping attacks can lead to a loss of confidence in the marketplace and destabilize the overall BRC20 token economy.

4.3 Attack Methodology

Fig.2 and the steps below illustrate how an attacker with minimal access can exploit PSBT transactions. The workflow assumes that

the attacker can observe or intercept the PSBT metadata and that they possess UTXOs sufficient to cover the total requested output value plus a higher fee.

A crucial element of this strategy is timing. The attacker must insert the malicious transaction into the mempool while the legitimate one remains unconfirmed. The attacker's high-fee option prompts miners to drop or invalidate the original partial transaction due to conflicting inputs. Additionally, by retaining the core BRC20 inscription data, the attacker ensures that the on-chain record still recognizes the same token transfer parameters, allowing them to channel the tokens to an address of their choosing.

Figure 2: Workflow of Our Snipping Attack

- **Step 1: Intercept the legitimate PSBT.** By observing unconfirmed BRC20 transactions (often embedding JSON-like data within the witness field) in the mempool, the attacker is able to identify the legitimate buyer's pending PSBT $\mathcal{P}$.
- **Step 2: Parse BRC20 transfer details.** Extract essential fields from $\mathcal{P}$:
  (1) The spending inputs: $\{(\mathbf{txid}^{(i)}, \mathbf{vout}^{(i)})\}$;
  (2) The receiving address(es) and amounts for the BRC20 context, e.g., $\mathsf{p : brc20, op : transfer, tick, amt, \ldots}$;
  (3) The on-chain fee estimate f_{legit} if deducible from the input-output difference.
- **Step 3: Construct the malicious transaction.** Using attacker-owned UTXOs $\mathcal{A}$, prepare a new PSBT $\mathcal{P}_{\text{atk}}$ structure referencing equivalent BRC20 metadata. Let

$$\mathcal{P}_{\text{atk}} = (\mathbf{In}_{\text{atk}}, \mathbf{Out}_{\text{atk}}, \mathbf{Witness}_{\text{atk}}).$$

  (1) Copy the core inscription data (e.g., *tick, amt, op*) to maintain validity for the same BRC20 token;
  (2) Allocate outputs so that at least one matches the seller's address and amount (if the aim is to mimic the buyer's payment);
  (3) Reserve minimal change for the attacker's address, ensuring a higher fee $f_{\text{atk}} > f_{\text{legit}}$.

- **Step 4: Sign and finalize.** Use the attacker's private key(s) to sign $\mathcal{P}_{\text{atk}}$ fully. Ensure the transaction meets all constraints (e.g., scriptPubKey validations, input availability). Once signed,

$$\mathcal{T}_{\text{atk}} \leftarrow \text{ finalizePSBT}(\mathcal{P}_{\text{atk}}).$$

- **Step 5: Broadcast malicious transaction.** Send $\mathcal{T}_{\text{atk}}$ to the mempool before the legitimate PSBT is confirmed. If the fee surpasses the original transaction's fee, miners will likely prioritize $\mathcal{T}_{\text{atk}}$ over the legitimate one.
- **Step 6: Observe mempool and block inclusion.** Monitor the Bitcoin mempool for confirmation. If $\mathcal{T}_{\text{atk}}$ is included in the next block, the legitimate transaction becomes invalid due to input conflicts, causing it to be removed from the mempool.

5 LOCAL EXPERIMENT

5.1 Experimental Setup

Hardware setup. The experiment was conducted on a desktop computer equipped with an Intel Core i7-9700K processor running at 3.60 GHz, 32 GB of DDR4 RAM, and a 1 TB NVMe SSD. The operating system used was Ubuntu 20.04 LTS.

Software setup. The core of our experimental setup relied on Bitcoin Core version 26.0, configured to operate in Regtest mode. Regtest provides a private Bitcoin environment that allows for rapid block generation and fine-grained control over network conditions, making it ideal for testing purposes. Specifically, we initiated a Bitcoin Core node, configured with the following parameters: *-regtest* to enable Regtest mode and *-txindex=1* to maintain a complete transaction index.

We utilized Bitcoin Core's built-in PSBT functionalities (e.g., `createpsbt`, `finalizepsbt`, etc.) to manage the creation and signing of BRC20 token transactions. Furthermore, we implemented the BRC20 protocol by building upon the existing open-source Oridinal [52] project. To achieve accurate and real-time tracking of BRC20 token states, we deployed locally running indexing services (writing in Python scripts) that parse and monitor the JSON-based inscriptions embedded within transaction *witness* fields.

5.2 Threat Model

In our local experiment, We assume the attacker can monitor or obtain transaction data in the following ways:

- **Access to PSBT:** The attacker is able to observe or intercept PSBTs before they are fully signed and broadcast. Such interception could occur through compromised communication channels (e.g., local file sharing).
- **Mempool visibility:** For already completed PSBT transactions, the attacker can inspect the local mempool. This visibility allows the attacker to detect any transaction referencing the same UTXOs. In a real-world scenario, similar transparency might arise from a public Bitcoin data explorer (i.e., mempool.space [55]).

Additionally, we also assume that the attacker controls at least one UTXO with enough bitcoin to create a higher-fee transaction than legitimate buyers. If the attacker needs to complete a partial signature, they can also generate the necessary private keys for their own transaction inputs.

Algorithm 1 BRC20 Snipping Attack Methodology

Global Parameters:
$\mathcal{I}_{\text{buyer}}$, $\mathcal{I}_{\text{atk}}$, Attacker's fee rate f_{atk}, Buyer's fee rate f_{buyer}, Mempool C_m.

1: **Step-①: Attacker monitors the mempool.**
2: Attacker $\mathcal{A}$ monitors mempool for unconfirmed BRC20 transactions.
3:    Attacker observes PSBT $\mathcal{P}$ with input $\mathcal{I}_{\text{buyer}} = (\textbf{txid}_b, \textbf{vout}_b)$ and output $O_s = (0.5\,\text{₿}, \text{scriptPubKey}_s)$.
4: Attacker identifies fee rate f_{buyer} associated with the transaction.

5: **Step-②: Attacker creates sniping PSBT.**
6: Attacker $\mathcal{A}$ prepares sniping transaction $\mathcal{P}_{\text{atk}}$ targeting $\mathcal{I}_{\text{buyer}}$.
7:    Attacker uses their own UTXO $\mathcal{I}_{\text{atk}} = (\textbf{txid}_a, \textbf{vout}_a)$ as input to match $\mathcal{I}_{\text{buyer}}$.
8:    Attacker sets a higher fee $f_{\text{atk}} = 0.28125\,\text{₿} > f_{\text{buyer}} = 0.27985\,\text{₿}$.
9: Create sniping output $O_{\text{atk}} = (0.5\,\text{₿}, \text{scriptPubKey}_s)$, ensuring the payment goes to the seller's address.
10: Add additional change output to attacker's address $O_{\text{change}} = (0.00005\,\text{₿}, A_{\mathcal{A}})$.
11: Broadcast $\mathcal{P}_{\text{atk}}$ to mempool.

12: **Step-③: Mempool conflict resolution.**
13: Mempool processes $\mathcal{P}_{\text{atk}}$ and compares fee rates.
14: **if** Attacker's fee $f_{\text{atk}} > f_{\text{buyer}}$ **then**
15:    $\mathcal{T}_{\text{atk}}$ is prioritized for inclusion in the next block.
16:    *Legitimate transaction $\mathcal{T}_{\text{buyer}}$ is invalidated due to input conflict and removed from mempool.*
17: **else**
18:    *Transaction not included due to low fee.*
19: **end if**

20: **Step-④: Transaction Confirmation.**
21: Confirm the inclusion of $\mathcal{T}_{\text{atk}}$ in the next block.
22:    Verify that $\mathcal{T}_{\text{atk}}$ is confirmed, while the original $\mathcal{T}_{\text{buyer}}$ remains unconfirmed or discarded.

23: **Step-⑤: Verify Attack Success.**
24: Attacker's wallet ($A_{\mathcal{A}}$) should now hold the BRC20 tokens transferred through the sniping attack.
25:    Confirm the amount of ak47 tokens in $A_{\mathcal{A}}$ matches the amount originally inscribed in $\mathcal{I}_{\text{buyer}}$.
26:    Confirm that $\mathcal{T}_{\text{buyer}}$ is no longer valid in the mempool.

5.3 Attack Strategy

Attack goal. The attacker's primary goal in our local experiment is to hijack an in-progress BRC20 token transfer by exploiting vulnerabilities in the PSBT workflow. In other words, the attacker invalidates or preempts the original transaction, "snipping" the intended BRC20 token transfer.

Attack method. Shown in Algorithm 1, to execute this attack, the attacker first obtains the PSBT by monitoring mempool. Because the local node logs incomplete PSBT data or temporarily loads it into memory while awaiting additional signatures, the attacker can parse those logs or memory dumps. Next, they modify the transaction metadata (e.g., UTXO information and output address), most critically, increasing the miner fee (i.e., decreasing the change sent back to the sender) beyond what the legitimate sender has offered. The attacker then completes partial signatures to render the malicious transaction valid. Finally, they re-broadcast this high-fee variant to the local mempool, where the node's incentive mechanism causes mining simulators in the regtest environment to prioritize it over the legitimate transaction.

Success Criteria. We deem the snipping attack successful if the attacker's malicious transaction both confirms before (and thereby invalidates) the legitimate transaction *and* delivers the targeted BRC20 tokens to the attacker's control.

Concretely, two conditions must be met:

- *Block inclusion.* The manipulated transaction authored by the attacker is accepted in a newly mined block prior to the genuine transaction, causing the genuine transaction to remain unconfirmed or be evicted from the mempool due to input conflicts.
- *Token receipt.* Upon confirmation of the attacker's transaction, a query to local indexers indicates that the BRC20 tokens, originally intended for the legitimate buyer, have been credited to an address controlled by the attacker. Verifying an increase in the attacker's token balance confirms the successful appropriation of the asset.

5.4 Execution of the Attack

Seller address setup. To initiate our experiment, we first created a seller wallet and an address within the *bitcoind-regtest* network, specifically adr.0dd8. We then used the built-in mining functions in regtest mode to generate blocks, designating the newly created address to receive the corresponding block rewards. After the mining stage, a balance check on the seller's wallet confirmed that it held exactly 1.5625 ₿, which would serve as the principal for creating and issuing BRC20 tokens, as well as covering transaction fees.

Token deploy and mint. We used the same seller address to mint a BRC20 token on the regtest network. Following the BRC20 protocol, we prepared a JSON-based inscription containing the necessary parameters: {p : brc20; op : deploy; tick : ak47; max : 2100000; lim : 1000}. This metadata was then serialized into hexadecimal form, yielding hex.307d.

Subsequently, we constructed a raw transaction on our regtest network using the createrawtransaction command, consuming the output from a prior transaction (txid.240c) as input, and specifying the hexadecimal-encoded inscription (hex.307d) as *"data"* in the transaction output. After finalizing and signing the transaction with our seller wallet, we broadcast it to the local network and mined a new block to confirm it. The transaction was successfully included in block.a2e2, marking the on-chain deployment of the BRC20 *ak47* ₿.

In our local regtest environment, BRC20 tokens default to belonging to the deployer's address once the "deploy" transaction confirms. Consequently, the seller's wallet automatically acquired a total of *21,000,000 ak47* ₿ without requiring a separate mint operation.

Token transfer via PSBT. To initiate a subsequent transfer, the seller prepared a JSON inscription reflecting the transfer operation: {p : brc20; op : transfer; tick : ak47; amt : 1000}. Converting this metadata to a hexadecimal string yielded: hex.227d.

Using the createpsbt function, the seller constructed a new partially signed transaction. The relevant input referenced a previous transaction output (txid: tx.641d, vout: 0) held by the seller, while the output was defined to include the transfer inscription in its "data" field. Executing createpsbt produced the following PSBT string: psbt.r85P.

Next, we used walletprocesspsbt to sign this PSBT. However, as only one signature was provided and the PSBT did not yet meet all necessary signing requirements for a complete transaction, the walletprocesspsbt result indicated "complete": false. This status signifies that, under our experimental setup, additional signatures would be needed before broadcasting the final transaction.

Buyer & attacker addresses setup. We introduced three additional addresses to represent potential buyers and attackers. The

ordinary buyer's address was set to adr.qa8r, while two attacker addresses, adr.v6r9 and adr.xffg, were designated to simulate different fee-bidding strategies. Specifically, one attacker was intended to surpass the ordinary buyer's fee, while the other remained lower, helping us observe various outcomes in our snipping experiments.

Buyer signs PSBT. First, we assumed that the ordinary buyer discovered the seller's partially signed BRC20 transaction. From the buyer's perspective, the crucial element of interest was the transaction's data field, which encodes the BRC20 transfer details in hexadecimal format. Using these details, the buyer constructed a new and fully formed PSBT with the following command (simplified for illustration):

```
1   bitcoin-cli -regtest createpsbt
2   '[{"txid": "6ec491e4928d9b6af81b0ebd5040e1a44834f
3   2576e1085ecb79d5fddc4fbb2ea", "vout": 0}]'
4   '[
5     {"bcrt1qkckuparv0d8nxadvzxgl0m0fmsn0ym6xfq0dd8":
          0.5},
6     {"bcrt1qackm4gsk5szqmylr0k5fq2n6v9jugz4473qa8r":
          0.1},
7     {
8       "data": "7b22703a226272632d3230222c226f70223
9       a227472616e73666572222c227469636b223a22616b34
10      37222c22616d74223a2231303030227d"
11    }
12  ]'
```

Listing 1: Constructing a new PSBT from buyer's view

In this command:

- tx.b2ea (vout=0) is the UTXO the buyer intends to spend (with the amount of $0.87985\text{\textBitcoin}$).
- adr.0dd8 (the seller's address) is allocated $0.5\text{\textBitcoin}$.
- adr.qa8r (the buyer's address) is allocated $0.1\text{\textBitcoin}$.
- The data field includes the hexadecimal-encoded BRC20 transfer instruction, representing the inscription {p : brc20; op : transfer; tick : ak47; amt : 1000}.

The input UTXO provides $0.87985\text{\textBitcoin}$ in total, while the outputs allocate $0.5\text{\textBitcoin}$ and $0.1\text{\textBitcoin}$, respectively. According to Formula 1, the resulting fee $\mathcal{F}$ is:

$$\mathcal{F}_b = 0.87985\text{\textBitcoin} - (0.5\text{\textBitcoin} + 0.1\text{\textBitcoin}) = 0.27985\text{\textBitcoin}.$$

After creating this new PSBT, the bitcoin-cli command returned a base64-encoded string (psbt.+Gqb) indicating the partially constructed transaction. The buyer used the walletprocesspsbt command to sign this PSBT with their private key, returning hex-encoded transaction data (hex.e140). A response looks like:

```
1   {
2   "psbt": "cHNidP8BALQCAAAAAeqy+8
        TdX5237IUQblfyNEik4UBQvQ4b+
        GqbjZLkkcRuAAAAAAD9////A4Dw+
        gIAAAAAFgAUti3A9Gx7TzN1rBGR9+3
        p3Cbyb0bgIgIAAAAAABYAFOZzVbqGcZy..."
3   "complete": true
4   "hex": "02000000000101eab2fbc4dd5f9db7ec85106e57f
5   23448a4e140...8200000000"
6   }
```

Listing 2: Buyer-signed PSBT Example

"complete": true now indicates the transaction has all required signatures for this scenario, meaning it can be broadcast and confirmed without additional inputs.

Broadcast the completed PSBT. The final step is to broadcast the fully signed transaction to the local regtest network. Having obtained the hex-encoded transaction data (hex.e140) from the completed PSBT, the buyer invokes the sendrawtransaction command, supplying the raw transaction in hexadecimal form and returning a transaction id txid.c936. This submission passes the transaction into the local node's mempool for validation. Once validated, the node broadcasts the transaction to the rest of the regtest environment, enabling block producers to mine it. Under non-adversarial circumstances, this broadcasted transaction would then be included in a newly mined block, completing the token transfer from the seller to the buyer. However, in our subsequent snipping attack experiments, the attacker will attempt to outbid this transaction, thereby invalidating the transfer.

Adverisal buyers (attackers) snip PSBT. In our experimental setup, the attacker aims to invalidate the buyer's PSBT by crafting and broadcasting an alternative PSBT with a higher fee. Two attacker addresses were introduced to conduct a comparative experiment: one attempting to outbid the ordinary buyer's fee, and the other offering a lower fee to illustrate a failed snipping attempt.

(1) Snipping with a higher fee. The attacker's address (adr.v6r9) had a UTXO valued at $0.7813\text{\textBitcoin}$. This UTXO provides enough funds to both pay the seller the same amount ($0.5\text{\textBitcoin}$) that the buyer intended to pay, and simultaneously offer a fee slightly exceeding the legitimate transaction fee of $0.27985\text{\textBitcoin}$. By allocating only $0.00005\text{\textBitcoin}$ back to the attacker's own address, the resulting fee is calculated to be $0.28125\text{\textBitcoin}$, ensuring that this snipping transaction becomes more appealing to local miners or block producers on regtest.

$$\mathcal{F}_h = 0.7813\text{\textBitcoin} - (0.5\text{\textBitcoin} + 0.00005\text{\textBitcoin}) = 0.28125\text{\textBitcoin}.$$

The attacker mirrors the buyer's steps of using createpsbt but replaces the inputs and outputs with their own addresses and amounts. In particular, they preserve the essential details that must remain consistent—namely sending $0.5\text{\textBitcoin}$ to the seller's address (adr.0dd8) and embedding the same BRC20 transfer inscription in the data field. An invocation is shown below:

```
1   bitcoin-cli -regtest createpsbt
2   '[{"txid": "6
        ec491e4928d9b6af81b0ebd5040e1a44834f2576
3   e1085ecb79d5fddc4fbb2ea", "vout": 0}]'
4   '[
5     {"bcrt1qkckuparv0d8nxadvzxgl0m0fmsn0ym6xfq0dd8":
          0.5},
6     {"bcrt1qtpq478leuks3wcg9zgr7rqhz6enszpsmt7v6r9":
          0.00005},
7     {
8       "data": "7b22703a226272632d3230222c226f70223a
9       227472616e73666572222c227469636b223a22616b343
10      7222c22616d74223a2231303030227d"
11    }
12  ]'
```

Listing 3: Constructing the Attacker's PSBT with a Higher Fee

This step yields a base64-encoded PSBT string (psbt.UBQv). The attacker proceeds to sign this PSBT (through walletprocesspsbt). A successful signature produces a valid, fully signed transaction in hexadecimal form (hex.ec85).

Finally, the attacker issues `sendrawtransaction` to broadcast this signed raw transaction to the local regtest mempool. Upon success, the node returns a `txid` txid.f0bb.

(2) Snipping with a lower fee. In a parallel experiment, we tested a scenario in which the attacker attempts to front-run the buyer's transaction but uses a notably lower fee than the buyer. This second attacker address, here referred to as adr.xffg, holds a UTXO worth 0.700001 ₿. Following the same basic approach, the attacker preserves the essential fields from the original transfer transaction—namely sending 0.5 ₿ to the seller's address adr.0dd8 and embedding the identical BRC20 transfer inscription in the `data` field. However, they allocate 0.2 ₿ as change to their own address adr.xffg, leaving a transaction fee of only 0.000001 ₿. This amount is far below the ordinary buyer's fee.

$$\mathcal{F}_l = 0.700001\,\cancel{B} - (0.5\,\cancel{B} + 0.2\,\cancel{B}) = 0.000001\,\cancel{B}.$$

The attacker constructs this low-fee transaction by issuing:

```
1  bitcoin-cli -regtest createpsbt
2  '[{"txid": "6ec491e4928d9b6af81b0ebd5040e1a44834f
3  2576e1085ecb79d5fddc4fbb2ea", "vout": 0}]'
4  '[
5    {"bcrt1qkckuparv0d8nxadvzxgl0m0fmsn0ym6xfq0dd8":
       0.5},
6    {"bcrt1quee4tw5xwxw236yuw8zyuw83r36dtnztsuxffg":
       0.2},
7    {
8      "data": "7b22703a226272632d3230222c226f7022
9  3a227472616e73666572222c227469636b223a22616b3437
10 222c22616d74223a2231303030227d"
11   }
12 ]'
```

Listing 4: Constructing the Attacker's PSBT with a Lower Fee

Subsequently, the attacker signs the returned PSBT to generate a fully signed transaction in hexadecimal form and broadcasts this signed transaction, returning a transaction ID (txid.2323).

Mempool observation. Following the attacker's snipping attempts, we inspected the local mempool using the command `bitcoin-cli -regtest getrawmempool`. As expected, we discovered three unconfirmed transactions residing in the mempool: the legitimate PSBT transaction broadcast by the ordinary buyer, and the two malicious PSBT transactions issued by the attacker.

Specifically, the mempool listing presents the chronological order of transaction submission. The Ordinary buyer's transaction (txid.c936) is encountered first, immediately followed by the high-fee attacker transaction (txid.f0bb), and culminating with the low-fee attacker transaction (txid.2323).

Validation of the snipping attack. To ascertain whether the attacker's snipping attempt succeeded, we proceeded to mine a new block in our local regtest environment by issuing:

```
1  bitcoin-cli -regtest generatetoaddress
      bcrt1qt30mvqww2cq9nvq4zlefj0l330y4k2ulqtdyks
```

This command instructs the regtest node to generate one block and direct its mining reward to the specified address (adr.tdks). Upon successful block creation, the node returned a newly mined block hash block.d848. Next, we inspected the block's contents with the returned block hash:

(a) **Details of Buyer-signed PSBT:** The Base64-encoded string of the buyer's signed PSBT (psbt.+Gqb) contains the necessary data to complete the transaction. The highlighted transaction fee shows the calculated fee of **0.27985000 ₿**.

(b) **Scenario 1: Attacker-signed PSBT with higher TxFee:** It displays the Base64-encoded string of the attacker-signed PSBT (psbt.UBQv) with a higher transaction fee. In this scenario, the transaction fee is marked as **0.28125000 ₿**, which is higher compared to 3(a), indicating a malicious modification by the attacker.

(c) **Scenario 2: Attacker-signed PSBT with lower TxFee:** It shows the Base64-encoded string of the attacker-signed PSBT (psbt.b+Gq) with a lower transaction fee. In this scenario, the transaction fee is significantly reduced to **0.0000010 ₿**, which indicates a malicious attempt as a comparison with 3(b).

(d) **Block details after our attack:** There is only one confirmation transaction in the packed block. The transaction is highlighted in the `tx` field, which is the one (txid.f0bb) with the higher transaction fee. This confirms that the attacker's attack was effective.

Figure 3: Screenshots during our attack

```
1    bitcoin-cli -regtest getblock
2    18be98d75b24c4f03e58f17bb9adf098b96c02d39456b
3    74e9217ff4bcbd8d848
```

(1) First condition - block inclusion. In the resulting JSON output (Fig.3(d)), the `tx` field revealed a single relevant transaction ID: **tx.f0bb** which matched the higher-fee PSBT created by the attacker.

To confirm this observation, we rechecked the mempool status via `getrawmempool` command. At this stage, the previously visible transactions, namely the ordinary buyer's PSBT and the low-fee attacker transaction, had disappeared from the mempool. This disappearance occurred because those transactions became invalid once the attacker's higher-fee PSBT transaction was included in the newly mined block. In other words, by consuming the same input PSBT at a higher fee, the attacker's higher-fee PSBT transaction superseded the other two, causing them to be dropped from the mempool due to a conflict.

(2) Second condition - token receipt. In addition to the confirmation of the attacker's transaction in the block, we also validate the success of the attack by examining the distribution of BRC20 tokens. Specifically, we check whether the attacker's wallet has received the expected ak47₿, confirming that the attack was successful in redirecting the tokens from the legitimate buyer to the attacker.

To accomplish this, we queried the BRC20 token balances of the attacker's wallet address (tx.f0bb) using a custom script that we developed for tracking token balances across different addresses. The script extracts transaction data from the regtest network and calculates the balance of each address by aggregating the token transfers. Upon performing this check, we observed, as anticipated, that the attacker's address had received an additional 1000 AK47₿. Furthermore, we verified the token balance of the seller's address (tx.0dd8), which showed a corresponding decrease of 1000 AK47₿.

Reproducibility of attack results. To ensure the reliability and reproducibility of our findings, we conducted two additional rounds of experiments beyond the initial trial (shown in Fig. 4). The objective was to further validate whether the observed results were consistent and not due to chance.

In the second round of experiments, we reversed the order of the low-fee and high-fee attacker transactions. This was done to examine whether the sequence in which the transactions were submitted had any impact on the attack's success. Despite the change in order, the results were consistent with the initial experiment: the transaction with the highest fee was always confirmed in the block, while the lower-fee transaction and the legitimate transaction were discarded. The outcome of the attack was solely dependent on the fee rate but not the order of transaction submission.

In the third round, we increased the complexity of the experiment by introducing multiple sniping attempts. In this case, we set up four different attack transactions with varying fee rates (e.g., 9 sats/vB, 20 sats/vB, 35 sats/vB, and 50 sats/vB). We ran the experiment again under these conditions, and as expected, the transaction with the highest fee (50 sats/vB) was always the one included in the block, while the others were removed from the mempool due to conflicts. This third round demonstrated that the sniping attack could consistently be replicated, even when multiple transactions with different fees were used.
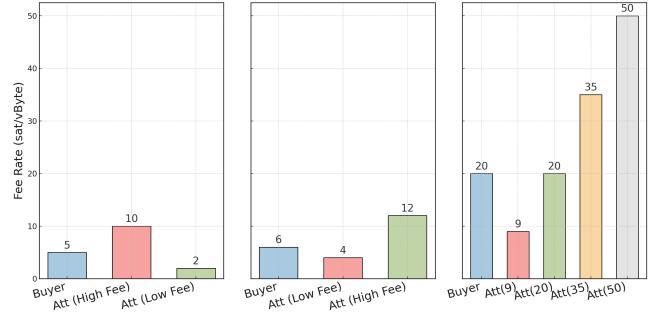

Figure 4: Fee rate comparison in three experiments: The chart compares the fee rates of transactions across three experimental rounds, showing the fee rates of legitimate buyer transactions and those of attackers using high and low fees.

6 ATTACKING ADAPTABILITY

We discuss the adaptability of our attack and examine its applicability to various platforms.

6.1 Platform Requirements

For a BRC20 snipping attack to be possible, the following platform conditions must be met:

PSBT support. The platform must utilize or support the creation, signing, and broadcasting of PSBTs. In our attack model, these transactions are broadcasted with incomplete signatures, leaving room for a buyer to sign and complete the transaction.

Unconfirmed transactions in mempool. The platform must allow transactions to stay in an unconfirmed state long enough for an attacker to access them. This is essential because the snipping attack exploits the fee-based transaction replacement mechanism, where transactions in the mempool can be replaced by those with higher fees before being included in a block. If a platform processes transactions too quickly or confirms them instantly, it would limit the opportunity for an attacker to execute this attack.

Fee transparency. In many marketplaces, BRC20 tokens are traded with a transparent fee structure, where both the buyer and the seller are aware of the transaction's proposed fee. If the platform does not provide sufficient visibility into transaction fees or hides this information from participants, it could reduce the effectiveness of the snipping attack, as the attacker would not be able to adjust their fee to outbid the legitimate buyer.

Third-party API support. Attackers often rely on the ability to query or subscribe to the mempool using third-party APIs or other tools that allow them to observe pending transactions. Platforms that enable API access will make it easier for attackers to spot legitimate BRC20 transactions.

6.2 Token Requirements

BRC20 tokens are a specific implementation of token standards built on the Bitcoin, typically through the use of inscriptions that embed JSON-like data directly into satoshis. The success of the snipping attack depends on several features of the BRC20 token standard and how it integrates with Bitcoin's transaction system.

However, the attack's applicability could vary depending on the particularities of the token standard and its underlying structure.

First, BRC20 tokens, like other Bitcoin-based tokens, rely on Bitcoin's UTXO model for transaction handling. For the snipping attack to work, the BRC20 token must be able to function within this model, meaning that the tokens must be transferred as part of a valid Bitcoin transaction. This means that token transfers must rely on inputs and outputs in the Bitcoin transaction that can be partially signed, and these signatures must be modifiable by the buyer and the attacker.

Second, a key element in BRC20 token transactions is the embedding of metadata (e.g., token type, amount, operation) in the Bitcoin transaction's `witness` or an `OP_RETURN` field. This ensures that each token transfer includes a data payload that can be manipulated or read by an attacker looking for transactions with high fees. If a token standard uses different mechanisms for data storage or transaction processing, such as different witness formats or proprietary methods for encoding token data, this could limit the applicability of the attack. Therefore, the attack is well-suited to tokens that are inscribed in the manner of BRC20, where metadata is clearly visible and can be captured in the mempool.

Furthermore, BRC20 tokens, like most Bitcoin transactions, depend on the finality provided by block confirmation. The attack assumes that the target transaction is unconfirmed and still in the mempool, allowing the attacker to insert a higher-fee transaction that replaces the original. This finality characteristic can vary based on how the transactions are managed. For example, if the token platform uses alternative finality mechanisms that do not rely on Bitcoin's block confirmation times (e.g., through a secondary layer or off-chain settlements), the snipping attack may not be applicable.

6.3 Applicability to Other Tokens

Our snipping attack is primarily designed for BRC20 tokens. However, the general principles of the attack could be adapted to other token standards, depending on how they are integrated with Bitcoin's base layer and whether the platform employs similar transaction mechanisms. We examine the potential for applying the attack to other token systems like Rune in Table 2.

Rune, another token standard designed for use within the Bitcoin ecosystem, could also be vulnerable to a form of snipping attack. Although Runes also leverage Bitcoin's core framework of satoshi-level inscription, they adopt a distinct approach compared to BRC20 by splitting the inscription data and functional definitions differently. Runes often separate key state-tracking details from the operational constraints of token transfers, potentially distributing them across multiple inscription contexts. Nevertheless, like any Bitcoin-based inscription method, Runes relies on unspent outputs and transaction finalization through block inclusion. Consequently, if a Rune transaction is broadcast as a partially signed or unconfirmed transaction visible in the mempool, it remains vulnerable to the snipping attack.

7 MITIGATION STRATEGIES

Mitigating the risk of snipping attacks in a BRC20 marketplace involves measures that either anticipate the possibility of fee-driven replacement or swiftly respond once a transaction is at risk of

Table 2: Attacking Applicability to Tokens in Various Platforms

Tokens	M.Cap	Protocol	Network	Binance	Unisat	Magic Eden	Gate.io	OKX
Ordi	$711M	BRC20	Bitcoin	✓	✓	✓	✓	✓
Sats	$612M	BRC20	Bitcoin	✓	✓	✗	✓	✓
Rats	$115M	BRC20	Bitcoin	✓	✓	✗	✓	✓
LeverFi	$74M	BRC20	Bitcoin	✓	✓	✗	✓	✓
PIZZA	$61M	BRC20	Bitcoin	✓	✓	✗	✓	✓
WZRD	$57M	BRC20	Bitcoin	✓	✓	✗	✓	✓
PUPS	$48M	BRC20	Bitcoin	✓	✓	✗	✓	✓
TEXO	$46M	BRC20	Bitcoin	✓	✓	✗	✓	✓
Multibit	$33M	BRC20	Bitcoin	✓	✓	✗	✓	✓
TRAC	$28M	BRC20	Bitcoin	✓	✓	✗	✓	✓
Atomicals	$27M	ARC20	Bitcoin	✗	✗	✗	✗	✗
Piin	$11.27M	BRC20	Bitcoin	✓	✓	✗	✓	✓
Orange	$8M	BRC20	Bitcoin	✓	✓	✗	✓	✓
BNSx	$1.22M	BRC20	Bitcoin	✓	✓	✗	✓	✓
cats	$1.19M	BRC20	Bitcoin	✓	✓	✗	✓	✓
Runes	$541,682	Rune	Bitcoin	✓	✓	✓	✓	✓
ETHS	$19M	ETHS20	Ethereum	✗	✗	✗	✗	✗
DOGI	$17M	DRC20	Dogecoin	✗	✗	✗	✗	✗
Gram	$9.89M	TON20	Arbitrum	✗	✗	✗	✗	✗
BSCS	$1.15M	BSC20	BNB Chain	✗	✗	✗	✗	✗
SOLS	$90,278	SPL20	Solana	✗	✗	✗	✗	✗
							Attacking Applicable?	

Binance: The largest centralized exchange worldwide for trading blockchain tokens.
Unisat: A user-facing wallet and marketplace specifically tailored for BRC20 tokens.
Magic Eden: A prominent multi-chain NFT and token marketplace. Some tokens are *not available for trading* or are not part of the marketplace's featured offerings. They only serve as (*NFT collection*).
Gate.io: A centralized exchange where trading is typically managed internally.
OKX: A centralized exchange offering digital asset trading.

being overtaken. From our observations, there are two primary strategies that have been implemented in certain platforms (e.g., Magic Eden) to secure token transfers: *Adding Mempool Protection* and *Increasing Fees* after submission. Apart from that, we proposed a more fundamental mitigation that can be pursued at the protocol, termed as *Advanced Fee-Locking Mechanism*.

7.1 Adding Mempool Protection

The first approach centers on a proactive layer of defense termed partial mempool protection. This mechanism operates by automatically broadcasting a series of higher-fee transactions on behalf of the user if the original transaction becomes vulnerable to being replaced or sniped. Once the user opts in, they can specify a maximum fee multiplier relative to the prevailing network fee rate. When proceeding to make a purchase, three transaction signatures are typically required. The first transaction uses the base network fee (for instance, 9 sats/vB), while the second and third pre-authorize higher fee rates that can be broadcast if the initial attempt is supplanted. Concretely, as shown in Fig.5(a), a user might sign an initial purchase at 9 sats/vB, then authorize the platform to escalate the fee to 95 sats/vB if the first transaction is sniped, and finally approve a peak fee of 180 sats/vB should the second transaction also fail. This tiered arrangement enables the user to maintain control of the maximum fee while still protecting against multiple rounds of replacement by an adversary.

7.2 Increasing Fees after Submission

A second mitigation strategy allows users to manually heighten their transaction fees once the purchase is pending in the mempool (shown in Fig.5(b)). After submitting an order, the user can track its status on a dedicated "Pending Orders" page. If the transaction remains valid and unconfirmed, a "Top Fee" status indicates that it has not been overridden; however, if the status shows "Getting Replaced," this signals that a snipping attempt may be underway, prompting the user to confirm a higher fee rate. By selecting the "Increase Fee" option, users can override their original fee with a new, higher rate exceeding any recommended minimum. Should the mempool or the platform detect that the user's transaction is still active, this re-issuance with a higher fee can successfully restore priority and secure the purchase. In effect, this second approach grants users direct control over re-broadcasting their transaction at an elevated fee without relying on a prearranged tiered schedule.

7.3 Advanced Fee-Locking Mechanism

While adding mempool protection and increasing fees after submission offer user-level responses to snipping threats, we propose *advanced fee-locking mechanism* at the protocol to prevent external parties from arbitrarily escalating fees on PSBTs. At its core, our approach involves cryptographic commitments that embed a maximum allowable fee within the partially signed data, ensuring that no external actor can alter that fee.

Commitment scheme. A commitment-based design requires users to commit to a maximum fee (e.g., f_{max}) when they first assemble the PSBT. This commitment might look like:

$$\text{commitFee}(f_{max}) = \mathcal{H}\big(\text{encode}(f_{max})\|\text{nonce}\big),$$

where $\mathcal{H}$ is a secure hash function, and the random nonce conceals the exact value of the user's maximum fee until finalization. Once the user has committed to this fee lock, the transaction's subsequent steps must not exceed f_{max}, or else the commitment becomes invalid. When the transaction is fully signed and ready for broadcast, the user reveals the nonce and the specific fee level, ensuring all nodes can verify that the fee remains within the committed bound.

Mechanics of finalization and broadcast. In practice, the scheme would proceed as follows: (i) The user computes a commitment to f_{max} and embeds it into an extra output script or the PSBT witness field; (ii) The user partially signs the transaction, ensuring that the script logic rejects any attempt to set fees above f_{max}; (iii) The final signature phase reveals the pre-image of $\text{commitFee}(f_{max})$, thereby binding the on-chain transaction to a specific actual fee; (iv) The broadcast transaction includes this revealed fee, allowing miners to validate that the transaction satisfies its original commitment.

8 RELATED WORK

Transaction replacement. Transaction replacement allows for substituting a previously sent but unconfirmed transaction with a new one. In Bitcoin, this is commonly done using replace-by-fee (RBF, outlined in BIP-125 [56]. RBF adds a layer of strategy to adjusting transaction orders [28], yet it necessitates cautious handling to sidestep security risks such as pinning attacks [29, 30]. Importantly, RBF is initiated by the asset owners themselves.

Transaction replacement can also be initiated by miners, a practice known as miner-extracted value (MEV) [17, 23, 57]. MEV is prevalent in the Ethereum ecosystem and has spread to other blockchain platforms. This practice can disadvantage users (e.g., reorg attacks) [17, 21, 22]), leading to potential losses and compromising the fairness of transactions [28, 58].

Blockchain sniper. A sniper bot is an automated tool designed to outpace all participants in purchasing tokens, leveraging blockchain mempool activity for malicious purposes. These activities include targeting short-lived tokens (e.g., 1-day rug pulls) on EVM token-issuing platforms [24–26] and exploiting MEV arbitrage opportunities in DeFi protocols [21–23]. Instead of directly deploying such bots, we adopt a similar strategy by monitoring the mempool to intercept the initial transaction containing a buy request, replacing it with a falsified one to disrupt its execution. Our attack contrasts with mainstream frontrunning attacks [17, 18, 20], which can either manipulate transactions, miners' blocks, or miner rotations.

BRC20/inscription. The concepts of BRC20 and inscriptions were widely introduced [3–5, 59] and explored across various dimensions, including social sentiment [46], on-chain fee fluctuations [60], off-chain indexers [61], component enhancements [51], and security analyses [29, 62]. Inscirption-based techniques can empower Bitcoin with more functionalities to further establish layer-two solutions [63–65]. At the moment, they remain in infancy.

Mempool-related attacks (Table 1). Blockchain mempool attacks include manipulating or exploiting the unconfirmed transactions within the mempool to achieve benefits. Mempool attacks include transaction malleability attack [31, 32] (altering the digital signature of a transaction to change its ID before confirmation), frontrunning or sandwitching attack [17–20] (placing a transaction with a higher/lower fee to precede/follow another transaction for profit), double spend attack [35–37] (sending two conflicting transactions or braches to favor one over the other), timejacking attack [34] (manipulating node timestamps to affect unconfirmed transaction timing), imitation attack [27] (replicating a victim's transaction or contract logic for financial gain) and pinning attacks [29, 30], DDos attacks [38–41] (overloading the mempool to disrupt transaction processing and mempool functionality).

9 CONCLUSION

In this paper, we present the BRC20 sniping attack, a growing threat to the open market for BRC20 tokens. BRC20 open marketplace relies on partially signed transactions (PSBT), which we exploit by manipulating transaction priorities. By allocating unfair fees, malicious actors can front-run legitimate transactions. Our attack is applicable to most mainstream BRC20 trading platforms.

We implemented and evaluated the attack. Experimental results show that even attackers with minimal resources can invalidate a buyer's transaction by simply paying a higher fee. We responsibly report our attack to related platforms. We hope this security analysis of BRC20 will lay the groundwork for protocol-level improvements and better marketplace designs in the Bitcoin ecosystem.

Table 3: Abbreviations (i.e., type.id) in our attack

	Type	ID	Full ID	Notes
Address	adr.	0dd8	bcrt1qkckuparv0d8nxadvzxgl0m0fmsn0ym6xfq0dd8	Seller's wallet address for selling BRC20 tokens
	adr.	qa8r	bcrt1qackm4gsk5szqmylr0k5fq2n6v9jugz4473qa8r	Buyer's wallet address for receiving BRC20 tokens
	adr.	v6r9	bcrt1qtpq478leuks3wcg9zgr7rqhz6enszpsmt7v6r9	Attacker's wallet address for snipping BRC20 tokens with *high* fee
	adr.	xffg	bcrt1quee4tw5xwxw236yuw8zyuw83r36dtnztsuxffg	Attacker's wallet address for snipping BRC20 tokens with *low* fee
	adr.	tdks	bcrt1qt30mvqww2cq9nvq4zlefj0l330y4k2ulqtdyks	Miner's wallet address for mining block and receive mining reward
Transaction	txid	240c	1d153468fbb27178ed84709fcdb22d6fef4140d67fa887166451b63a24a9240c	Transaction of the *seller* linked to an UTXO for *deploy*
	txid	641d	3aad7f5f7d3e39bf7a98741550de767ff191d1ebfe9326177825890ec92d641d	Transaction of the *seller* linked to an UTXO for *createpsbt*
	txid	b2ea	6ec491e4928d9b6af81b0ebd5040e1a44834f2576e1085ecb79d5fddc4fbb2ea	Transaction of the *buyer* linked to an UTXO for *createpsbt*
	txid	c936	1d269e83d7a415197038372196ad0c19fc380861b51e788cd3ec0fced777c936	Transaction from the buyer broadcasts the completed PSBT
	txid	f0bb	e5be4872b8dd95369df0d15846e5099b5f20662e9809e65348151a9a257af0bb	Transaction from the attacker broadcasts the completed PSBT (*high fee*)
	txid	2323	3abe4872b8dd95369df0d15846e5099b5f20662e9809e65342323aodsa2323	Transaction from the attacker broadcasts the completed PSBT (*low fee*)
Block	block	a2e2	6ffa7cb6d94727f6a619d4a33c60f0f65b8b7f72dde3e5fd3fd9e94753c6a2e2	Block containing a BRC20 deploy transaction
	block	d848	18be98d75b24c4f03e58f17bb9adf098b96c02d39456b74e9217ff4bcbd8d848	Block containing brc20 transfer pbst transaction
Data Hex	hex.	307d	7b22703a20226272632d3230222c20226f70223a20226465706c6c...13030307d	Hexadecimal-encoded string of BRC20 token deploy data
	hex.	227d	7b22703a226272632d3230222c226f70223a227472616e7366657...03030227d	Hexadecimal-encoded string of BRC20 token transfer data
	hex.	e140	02000000000101eab2fbc4dd5f9db7ec85106c57f23448a4e140...8200000000	Buyer created completed psbt hex-encoded string
	hex.	ec85	02000000000101eab2fbc4dd5f9db7ec85...15b96648c1ea8ba588200000000	Attacker created completed psbt hex-encoded string (with *high* fee)
	hex.	4050	02000000000101eab2fbc4dd5f9db7ec85106c57f23448a4e14050...0000000	Attacker created completed psbt hex-encoded string (with *low* fee)
PSBT	psbt	r85P	cHNidP8BAHYCAAAAAR1kLckOiSV4FyaT/uvRkfF/dt5QFXSYer85P...0AAAAAAAAA	Seller created psbt base64-encoded string
	psbt	+Gqb	cHNidP8BALQCAAAAAeqy+8TdX5237IUQblfyNEik4UBQvQ4b+Gqb...AAgAAAAAA	Buyer created psbt base64-encoded string
	psbt	UBQv	cHNidP8BAJUCAAAAAeqy+8TdX5237IUQblfyNEik4UBQv...AAIAAAAAAAgAAAAAA	Attacker created psbt base64-encoded string (with *high* fee)
	psbt	b+Gq	cHNidP8BALQCAAAAAeqy+8TdX5237IUQblfyNEik4UBQvQ4b+Gq...AAABAAAAAAA=	Attacker created psbt base64-encoded string (with *low* fee)

REFERENCES

[1] Chainalysis. The Chainalysis 2024 Crypto Crime Report. *https://go.chainalysis.com/crypto-crime-2024.html*. Accessed: Sep, 2024, 2024.

[2] Ordinal Theory Authors. Ordinal theory handbook. *https://docs.ordinals.com/*. Accessed: April, 2024, 2024.

[3] Binance Research. A new era for Bitcoin. *https://research.binance.com/static/pdf/a-new-era-for-bitcoin.pdf*, 2023.

[4] Binance Research. BRC-20 tokens: A primer. *https://research.binance.com/static/pdf/BRC-20%20Tokens%20-%20A%20Primer.pdf*, 2023.

[5] Binance Research. Navigating the inscriptions landscape. *https://public.bnbstatic.com/static/files/research/navigating-the-inscriptions-landscape.pdf*, 2024.

[6] Binance Research. The future of Bitcoin #1: The halving & what's next. *https://public.bnbstatic.com/static/files/research/the-future-of-bitcoin-1-the-halving-and-whats-next.pdf*, 2024.

[7] Binance Research. The future of Bitcoin #2: Tokens. *https://public.bnbstatic.com/static/files/research/the-future-of-bitcoin-2-tokens.pdf*, 2024.

[8] Binance Research. The future of Bitcoin #3: Scaling Bitcoin. *https://public.bnbstatic.com/static/files/research/the-future-of-bitcoin-3-scaling-bitcoin.pdf*, 2024.

[9] Snipe Discord Channel. *https://discord.com/channels/1131000554005467206/1258112452453728316/*. Accessed: Sep, 2024, 2024.

[10] Sniping Battle in Ape Hoodie Ordinal. *https://x.com/Xeer/status/1811232129692704787/*. Accessed: Sep, 2024, 2024.

[11] Phoenixx Down. Mempool Sniping Exploit Plagues Ordibots Mint. *https://www.influencive.com/mempool-sniping-exploit-plagues-ordibots-mint/*. Accessed: Sep, 2024, 2023.

[12] A Poll Asking Users Whether They had been Sniped. *https://x.com/const_quary/status/1802999453017632785/*. Accessed: Sep, 2024, 2024.

[13] Vogelsteller Fabian and Buterin Vitalik. Ethereum ERC-20 token standard. *Accessible: https://eips.ethereum.org/EIPS/eip-20*, 2015.

[14] Entriken William, Shirley Dieter, Evans Jacob, and Sachs Nastassia. Eip-721: ERC-721 non-fungible token standard. *Accessible: https://eips.ethereum.org/EIPS/eip-721*, 2018.

[15] Qin Wang et al. Non-fungible token (NFT): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*, 2021.

[16] Ava Chow. BIP-174: Partially signed bitcoin transaction format. *https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki*. Accessed: Jan, 2025, 2020.

[17] 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 *IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[18] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE Symposium on Security and Privacy (SP)*, pages 428–445. IEEE, 2021.

[19] Xinrui Zhang et al. Frontrunning block attack in PoA clique: A case study. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–3. IEEE, 2022.

[20] Qin Wang, Rujia Li, Qi Wang, Shiping Chen, and Yang Xiang. Exploring unfairness on proof of authority: Order manipulation attacks and remedies. In *ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 123–137, 2022.

[21] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent dishonesty: Front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 170–189. Springer, 2020.

[22] Lioba Heimbach and Roger Wattenhofer. SoK: Preventing transaction reordering manipulations in decentralized finance. In *Proceedings of the ACM Conference on Advances in Financial Technologies (AFT)*, pages 47–60, 2022.

[23] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. SoK: MEV countermeasures. In *Proceedings of the Workshop on Decentralized Finance and Security*, pages 21–30, 2024.

[24] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in Ethereum and in the Binance smart chain (BNB). In *USENIX Security Symposium (USENIX Sec)*, pages 3349–3366, 2023.

[25] Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Francesco Sassi. Ready, aim, snipe! analysis of Sniper bots and their impact on the DeFi ecosystem. *Companion Proceedings of the ACM Web Conference (WWW)*, pages 1093–1102, 2023.

[26] Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Francesco Sassi. The blockchain warfare: Investigating the ecosystem of sniper bots on Ethereum and BNB smart chain. *ACM Transactions on Internet Technology (TOIT)*, 2024.

[27] Kaihua Qin, Stefanos Chaliasos, Liyi Zhou, Benjamin Livshits, Dawn Song, and Arthur Gervais. The blockchain imitation game. In *USENIX Security Symposium (USENIX Sec)*, pages 3961–3978, 2023.

[28] Rujia Li, Xuanwei Hu, et al. Transaction fairness in blockchains, revisited. *Cryptology ePrint Archive*, 2023.

[29] Minfeng Qi, Qin Wang, Zhipeng Wang, Lin Zhong, Tianqing Zhu, Shiping Chen, and William Knottenbelt. BRC20 pinning attack. *arXiv preprint arXiv:2410.11295*, 2024.

[30] Pinning attacks in lightning network. *https://github.com/t-bast/lightning-docs/blob/master/pinning-attacks.md#attacks-on-lightning*, 2024.

[31] Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and MtGox. In *European Symposium on Research in Computer Security (ESORICS)*, pages 313–326. Springer, 2014.

[32] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. On the malleability of Bitcoin transactions. In *International Conference on Financial Cryptography and Data Security: (FC) Workshops*, pages 1–18. Springer, 2015.

[33] Bitcoin Optech. Transaction pinning. *https://bitcoinops.org/en/topics/transaction-pinning/*. Accessed: Sep, 2024, 2024.

[34] Xinrui Zhang, Rujia Li, et al. Time-manipulation attack: Breaking fairness against proof of authority Aura. In *Proceedings of the ACM Web Conference (WWW)*, pages 2076–2086, 2023.

[35] Kevin Alarcón Negy, Peter R Rizun, and Emin Gün Sirer. Selfish mining re-examined. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 61–78. Springer, 2020.

[36] Chen Feng and Jianyu Niu. Selfish mining in Ethereum. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1306–1316. IEEE, 2019.

[37] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[38] Shuangke Wu, Yanjiao Chen, Minghui Li, Xiangyang Luo, Zhe Liu, and Lan Liu. Survive and thrive: A stochastic game for DDoS attacks in Bitcoin mining pools. *IEEE/ACM Transactions On Networking (ToN)*, 28(2):874–887, 2020.

[39] Muhammad Saad, Laurent Njilla, Charles A Kamhoua, Kevin Kwiat, and Aziz Mohaisen. Shocking blockchain's memory with unconfirmed transactions: New DDoS attacks and countermeasures. *Blockchain for Distributed Systems Security*, page 205, 2019.

[40] Muhammad Saad, Laurent Njilla, Charles Kamhoua, Joongheon Kim, DaeHun Nyang, and Aziz Mohaisen. Mempool optimization for defending against DDoS attacks in PoW-based blockchain systems. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019.

[41] Marie Vasek, Micah Thornton, and Tyler Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *International Conference on Financial Cryptography and Data Security (FC) Workshops*, pages 57–71. Springer, 2014.

[42] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Satoshi Nakamoto*, 2008.

[43] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. *Journal of the ACM*, 71(4):1–49, 2024.

[44] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[45] Qin Wang and Shiping Chen. Account abstraction, analysed. In *IEEE International Conference on Blockchain (Blockchain)*, pages 323–331. IEEE, 2023.

[46] Qin Wang and Guangsheng Yu. Understanding BRC-20: Hope or hype. *Available at SSRN 4590451*, 2023.

[47] Wuille Pieter, Nick Jonas, and Towns Anthony. BIP-341: Taproot: Segwit version 1 spending rules. *https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki. Accessed: Jan. 2025*, 2020.

[48] Bitcoin Community. BIP 342: Tapscript. *https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki. Accessed: April, 2025*, 2021.

[49] Pieter Wuille et al. BIP-141: Segregated witness (consensus layer). *https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Accessed: Jan. 2025*, 2017.

[50] BRC20-ORDI. *https://ordinalswallet.com/collection/brc20-ORDI. Accessed: April, 2024*, 2024.

[51] Qin Wang, Guangsheng Yu, and Shiping Chen. Bridging BRC-20 to Ethereum. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2024.

[52] Ordinal wallet. *https://ordinalswallet.com/brc20*, 2023.

[53] Unisat wallet: Open source chrome extension for Bitcoin ordinals & BRC-20. *https://unisat.io/brc20?ref=bankless.ghost.io*, 2023.

[54] Binance Web3 Wallet. *https://www.binance.com/en/web3wallet. Accessed: Jan. 2025*, 2024.

[55] Mempool - Bitcoin Explorer. *https://mempool.space/. Accessed: Sep, 2024*, 2024.

[56] Harding David and Todd Peter. BIP-125: Opt-in full replace-by-fee signaling. *https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki. Accessed: Jan. 2025*, 2015.

[57] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[58] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In *Annual International Cryptology Conference (CRYPTO)*, pages 451–480. Springer, 2020.

[59] Ningran Li, Minfeng Qi, et al. Bitcoin inscriptions: Foundations and beyond. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2024.

[60] Louis Bertucci. Bitcoin ordinals: Determinants and impact on total transaction fees. *Research in International Business and Finance*, page 102338, 2024.

[61] Hongbo Wen, Hanzhi Liu, Shuyang Tang, Shuhan Cao, Yu Feng, et al. Modular indexer: Fully user-verified execution layer for meta-protocols on bitcoin. *Cryptology ePrint Archive*, 2024.

[62] Hanzhi Liu, Jingyu Ke, Hongbo Wen, Robin Linus, Lukas George, Manish Bista, Hakan Karakuş, Junrui Liu, Yanju Chen, Yu Feng, et al. Push-button verification for BitVM implementations. *Cryptology ePrint Archive*, 2024.

[63] Minfeng Qi, Qin Wang, Zhipeng Wang, Manvir Schneider, Tianqing Zhu, Shiping Chen, William Knottenbelt, and Thomas Hardjono. SoK: Bitcoin layer two (L2). *arXiv preprint arXiv:2409.02650*, 2024.

[64] BitVM. BitVM: Compute anything on Bitcoin. *https://bitvm.org/bitvm.pdf. Accessed: April, 2024*, 2024.

[65] Robin Linus. BitVM2: Opening up the playing field. *Bitcoin Magazine*, 2024.

(a) Adding Mempool Protection Mechanism

(b) Increasing Fees After Submission
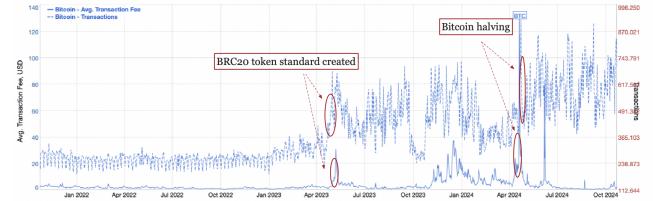
Figure 5: Screenshots for Mitigation Strategies

Figure 6: Bitcoin Avg. Transaction Fee vs. Transactions Per Day historical chart (data source: [66])

[66] Bitinfocharts. *https://bitinfocharts.com/comparison/transactionfees-transactions-btc.html#3y. Accessed: April, 2024*.

A   BITCOIN HISTORICAL PRICES

Fig.6 illustrates the correlation between the average Bitcoin transaction fee (in USD) and the number of transactions per day from 2022 to 2024. We observe that the average transaction fee exhibits fluctuations, reflecting changes in network demand and activity. The number of daily transactions shows periodic spikes, often coinciding with specific events or external factors influencing the Bitcoin network.

In early 2023, the creation of the BRC20 token standard marks a significant event, as highlighted in the first red circle. This innovation introduced a new layer of activity to the Bitcoin blockchain by enabling token issuance directly on-chain. The subsequent increase in activity led to network congestion, as seen in the spike in transaction fees and daily transactions during this period. Higher fees result from users bidding to have their transactions prioritized in the increasingly crowded mempool.

Later in the chart, the second red circle aligns with Bitcoin's halving event. The halving reduced miner rewards, directly impacting miner incentives. This likely increased transaction fees temporarily, as miners prioritized high-fee transactions to compensate for reduced block subsidies. This event also correlated with a spike in daily transactions, possibly reflecting increased trading activity and network adjustments around the halving.