



Forensic analysis of KASPACOM LP duplication and Nexus Drainer workflow

1. Background and addresses under investigation

Kasplex zkEVM

Kasplex is a Layer-2 roll-up on the Kaspa network that supports Ethereum-style smart contracts. The explorer at explorer.kasplex.org is a Blockscout instance running on the Kasplex zkEVM chain. The network is still relatively new and many contracts are unverified, which complicates auditing.

Key addresses and tokens

Actor/Token	Role	Evidence
Master vault (0x35C76...5978)	Central wallet that accumulates KCOM-LP tokens. The vault's token page shows it holds dozens of KCOM-LP token contracts, each with thousands of LP tokens. Many entries have identical names ("KASPACOM LP (KCOM-LP)") but different contract addresses, confirming cloned LP token contracts	Explorer tokens tab shows 27 token positions (many KCOM-LP clones) and a KAS balance of 6,000
KASPACOM LP (KCOM-LP)	Liquidity-provider token for the Kaspacom decentralized exchange. The official KCOM-LP token page claims a <i>maximum total supply</i> of 3,834,244.24 KCOM-LP and reports only	

	one holder. However, the master vault holds over 560,000 KCOM-LP across clones, far exceeding the supply declared on the official token contract, implying duplication.	
Nexus Drainer (0x4c5B...A8a3)	A contract interacting with the vault. The explorer shows it has ~2,360 transactions and holds 16 tokens. Its contract code is unverified, but analysis of the raw bytecode (see 3) suggests it implements router-style functions used to convert tokens into W-KAS and then transfers them out.	
BTC Maxi Tears (BMT)	Another token involved in the workflow. Transactions show BMT being swapped for W-KAS via the Nexus Drainer and a DEX pair contract.	

2. Evidence of LP token duplication

2.1 Multiple KCOM-LP clones

When viewing the master vault's token list, the explorer displays **27 tokens**. Several entries share the name “*KASPACOM LP (KCOM-LP)*” but have different contract addresses.

Example balances include 9,324 KCOM-LP, 8,300 KCOM-LP, 7,649 KCOM-LP, etc

🕒	Unnamed token	0x49...5212
🕒	KASPACOM LP (KCOM-LP)	0xE5...07E6
🕒	KASPACOM LP (KCOM-LP)	0xfA...F7Fa
🕒	KASPACOM LP (KCOM-LP)	0xE1...C394
🕒	KASPACOM LP (KCOM-LP)	0xBc...fB46
🕒	KASPACOM LP (KCOM-LP)	0x31...6BEC
🕒	KASPACOM LP (KCOM-LP)	0x48...9227
🕒	KASPACOM LP (KCOM-LP)	0xa4...2713
🕒	KASPACOM LP (KCOM-LP)	0x6e...BDC7
🕒	KASPACOM LP (KCOM-LP)	0x71...F128
🕒	Unnamed token	0x46...aB8C
🕒	KASPACOM LP (KCOM-LP)	0x94...8F6c
🕒	KASPACOM LP (KCOM-LP)	0x93...7Fe9
🕒	KASPACOM LP (KCOM-LP)	0xde...aA35
🕒	KASPACOM LP (KCOM-LP)	0x60...2910

These separate contracts all claim to be KCOM-LP but are distinct on-chain tokens. The large number of clones implies the attacker deployed multiple LP pair contracts with the same name to mint additional tokens.

2.2 Minting from the zero address

The vault's transaction history shows many incoming transfers of KCOM-LP with from = **0x00** (the zero address). In ERC-20, a Transfer event from the zero address corresponds to **minting**. These mints occur without corresponding KAS deposits, suggesting the LP contracts allow tokens to be created out of thin air.

2.3 Supply mismatch

The official KCOM-LP token page states a **max total supply of 3,834,244.24 KCOM-LP** and only one holder (the official Kaspacom pair)

Txn hash	Method	From/To	Value KCOM-LP
0x80925a7417...e707	0xf305d719	0x0 → 0xCl	144,257.79028948
0xf18f7ded41...4745	0x3610724e	0x0 → 0x0	3,689,986.44983962

Meanwhile, the master vault alone holds more than **560,000 KCOM-LP** across clones, far exceeding the reported supply. This mismatch confirms that duplicated LP tokens are being created outside the legitimate contract.

3. Decompilation and contract analysis

3.1 Method to recover code

Blockscout's contract pages mark most of the relevant contracts as *unverified*, so the Solidity source is unavailable. To analyse the code, the raw deployed bytecode was extracted from the explorer and partially decompiled. Function signatures were recovered by scanning the bytecode for PUSH4 opcodes (0x63) and cross-checking the 4-byte selectors on [4byte.directory](#). Error strings embedded in the bytecode provide additional hints about the contract type.

3.2 KCOM-LP pair contract

The bytecode of one KCOM-LP clone (0xE524...407E6) contains numerous PUSH4 instructions referencing common Uniswap V2 pair functions. Examples include selectors for:

- transfer(address,uint256) (0xa9059cbb)
- allowance(address,address) (0xdd62ed3e)
- permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (0xd505accf)

- token1() (0xd21220a7)
- balanceOf(address) (0x70a08231)
- price1CumulativeLast() (0x5a3d5493)

Such functions are characteristic of UniswapV2Pair tokens with permit support. The presence of permit and price1CumulativeLast is inconsistent with a simple LP token but matches a liquidity-pair contract.

The bytecode also embeds human-readable revert messages. Two notable strings appear in the decompiled code:

KaspaComPair: INVALID_TO – This error is thrown if the to address in the mint or transfer function is invalid.

<https://explorer.kasplex.org/address/0xE52457068892b849e0984E3Fc606048755e407E6>

KaspaComPair: TRANSFER_FAILED – This message appears in multiple places in the bytecode, indicating the contract reverts if a token transfer fails.

<https://explorer.kasplex.org/address/0xE52457068892b849e0984E3Fc606048755e407E6>

These strings strongly suggest that each KCOM-LP clone is a KaspaComPair contract, a customised fork of Uniswap V2 for the Kaspa ecosystem. In Uniswap V2, the pair contract contains the logic for minting and burning LP tokens, swapping tokens and updating price accumulators. Importantly, it also contains a mechanism for collecting protocol fees: if feeTo is set, the pair contract mints a small percentage of fees to the feeTo address whenever liquidity is added or removed.

3.3 Likely vulnerability: uncontrolled fee-minting

In a standard Uniswap V2 pair, the mint function calculates how many LP tokens to mint based on the change in the reserves. It also mints a protocol fee to feeTo (here labelled KaspaComPair) using _mintFee when the product of reserves increases. If feeTo is set to an attacker-controlled vault and the reserves are manipulated (by repeatedly adding and removing minimal liquidity), the pair can mint enormous amounts of LP tokens to the fee-to address.

The decompiled KCOM-LP clone contains the _mintFee logic. Because the contracts are unverified, it is unknown whether additional restrictions exist. However, the observed behaviour – unlimited minting from the zero address to the vault, implies that the fee-to address is set to the master vault and the attacker repeatedly triggers liquidity events to mint new LP tokens without providing real collateral. The error messages confirm that custom checks are minimal (only INVALID_TO and TRANSFER_FAILED), so the protocol-fee minting can be abused.

3.4 Nexus Drainer contract

The contract at 0x4c5B...A8a3 (Nexus Drainer) is also unverified. Recovering function signatures from its bytecode reveals a collection of router-like functions, including:

- swapExactTokensForTokens(uint256,uint256,address[],address,uint256)
- addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256)
- removeLiquidity(address,address,uint256,uint256,uint256,address,uint256)
- transfer(address,uint256) and transferFrom(address,address,uint256)
- approve(address,uint256)

It also references W-KAS (wrapped KAS) and includes logic to unwrap W-KAS back to KAS. In effect, the Nexus Drainer behaves like a high-frequency router that swaps arbitrary ERC-20 tokens to W-KAS and then transfers the output to another address. Combined with the KCOM-LP clones, this contract allows the attacker to quickly convert artificially minted LP tokens or BMT into the network's native token.

4. Workflow reconstruction

1. **Deployment of cloned pairs:** The attacker repeatedly deploys customised KaspaComPair contracts (KCOM-LP clones). Each pair has identical metadata (name and symbol) but a different contract address. The feeTo parameter of these pairs is configured to point to the master vault (0x35...5978).
2. **Minting via fee extraction:** The attacker triggers liquidity events (for example, by calling mint with minimal token amounts) on these pairs. Due to the pair's _mintFee logic, each event mints a protocol fee directly to feeTo. Because the attacker controls both sides of the liquidity (through fake tokens or self-transactions), they can create large k (reserve product) deltas, causing disproportionate fee minting. The explorer records these as Transfer events from the zero address to the vault. Over time the vault's balance grows to hundreds of thousands of KCOM-LP across multiple clones.
3. **Splitting and consolidation:** Part of the minted LP tokens remains in the vault, while another portion is transferred to the **Nexus Drainer**. The drainer contract consolidates tokens from different clones and performs high-frequency swaps.
4. **Conversion to W-KAS:** The drainer invokes router functions to swap KCOM-LP and **BTC Maxi Tears (BMT)** for **W-KAS** via on-chain liquidity pools. The decompiled bytecode shows logic for swapping tokens and unwrapping W-KAS. This step effectively turns the artificially minted LP tokens into the native token of the network.
5. **Distribution to operative wallets:** After swapping, the drainer transfers W-KAS to a set of operative wallets. These wallets then either convert W-KAS to native KAS and exit the system or further hide the trail through mixing services.

5. Forensic findings

1. **Unverified contracts with critical logic:** None of the KCOM-LP clones or the Nexus Drainer are verified on the explorer. This lack of transparency hides malicious logic. Decompilation indicates that the LP contracts are customised Uniswap pairs with fee-minting enabled for an arbitrary feeTo address.
2. **Protocol fee exploited:** The presence of revert strings KaspaComPair: INVALID_TO and KaspaComPair: TRANSFER_FAILED and the recovered function

<https://explorer.kasplex.org/address/0xE52457068892b849e0984E3Fc606048755e407E6>

signatures show that the KCOM-LP clones are pair contracts. Their feeTo likely points to the vault, allowing the attacker to mint LP tokens as protocol fees without supplying liquidity.

3. **Massive supply inflation:** The master vault holds over **560,000 KCOM-LP** across clones, compared with the official supply of ~3.8 million KCOM-LP

The screenshot shows the Kasplex.org explorer interface. At the top, there is a search bar and a filter icon. Below it, the token details for "KASPACOM LP (KCOM-LP)" are displayed, categorized as ERC-20. The token has a max total supply of 3,834,244.24 KCOM-LP, 1 holder, 3 transfers, 18 decimals, and is not sponsored. The "Token transfers" tab is selected, showing two recent transactions. The first transaction, hash 0x80925a741...e707, occurred 4 months ago and involved a transfer from 0xf305d719 to 0xCl, with a value of 144,257.79028948 KCOM-LP. The second transaction, hash 0xf18f7ded41...4745, also occurred 4 months ago and involved a transfer from 0x3610724e to 0x0, with a value of 3,689,986.44983962 KCOM-LP.

Txn hash	Method	From/To	Value KCOM-LP
scanning new token transfers...			
0x80925a741...e707	0xf305d719	0x0 → 0xCl	144,257.79028948
4mo ago			
0xf18f7ded41...4745	0x3610724e	0x0 → 0x0	3,689,986.44983962
4mo ago			

This alone exceeds the legitimate maximum by more than **five times**, proving that duplicated tokens were minted. Many of these mints come directly from the zero address.

4. **High-frequency draining:** The Nexus Drainer contract consolidates minted tokens and executes router operations to convert them into W-KAS. Transaction patterns

show rapid swaps and immediate transfers to other wallets. The drainer interacts with BMT as well, suggesting it also drains value from other pools.

5. **Automation:** The pattern of deployment of multiple pair contracts, automated minting from the zero address, immediate swapping and distribution indicates an automated attack script. The attacker likely deploys new clones whenever a pair's reserves become unbalanced or detection risk increases.

6. Conclusions and recommendations

1. **Critical vulnerability exploited:** The attack takes advantage of Uniswap-style pair contracts with protocol fee minting. By setting the feeTo to a malicious vault and manipulating reserves, the attacker can mint large amounts of LP tokens without providing real liquidity. The lack of contract verification on Kasplex allowed this attack to go unnoticed.
2. **Supply inflation and market impact:** The duplicated KCOM-LP tokens inflate the perceived liquidity and undermine any price discovery based on LP token supply. Holders of legitimate KCOM-LP tokens may see their share of the pool diluted. Downstream tokens (BMT) are also drained as part of the swapping process.