Got stolen after signing a message? Those who have used Uniswap, please be careful! Let Beosin demystify Permit2 phishing.

Beosin · Follow
11 min read · 5 days ago

Listen Share More

This article is completed by Bocai Bocai (twitter@wzxznl), independent researcher, and Sivan, Beosin security researcher.

Hackers scare everyone in the Web3 ecosystem. For crypto projects, they are afraid of writing a wrong line of code when developing because of the open-source nature of Web3. Once a security incident occurs, the consequences will be difficult to bear.

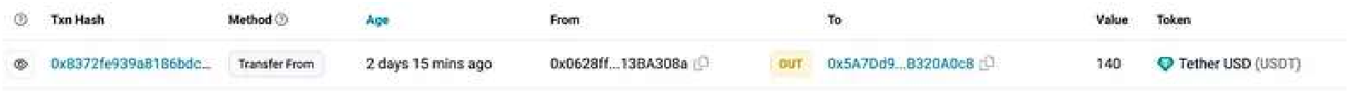
Personally, every on-chain interaction or signature that you make has the potential to have your assets stolen if you don't understand what you're doing. Therefore, security issues have always been one of the most troublesome issues in Web3. Due to the characteristics of the blockchain, once assets are stolen, there is almost no way to recover them so it is especially important for users to have security knowledge.

Bocai Bocai discovered a new phishing method that has been active in the past two months. As long as you sign a message, your assets are stolen, which is extremely hidden and difficult to prevent. Addresses that have interacted with Uniswap may be exposed to risk. In this article, Bocai and Beosin will analyze this signature phishing and try to help everyone avoid asset losses.

The following is a retelling of Bocai's personal experience:

Recently, a friend asked Bocai for help after his assets were stolen. Unlike the usual way of asset theft, the victim did not disclose his private key and did not interact with the contract of the phishing website, so Bocai started to investigate the incident.

In the blockchain explorer, we can see that the stolen USDT from the victim is transferred through the TransferFrom function. When we transfer tokens on Ethereum, we actually call the Transfer function of a token's smart contract. The difference between the two functions is simply that Transfer function is called by the asset owner to transfer the token to another address, while TransferFrom function is a third party's transfer of the token in an address to other addresses. This also means that the stolen asset was transferred by another address.



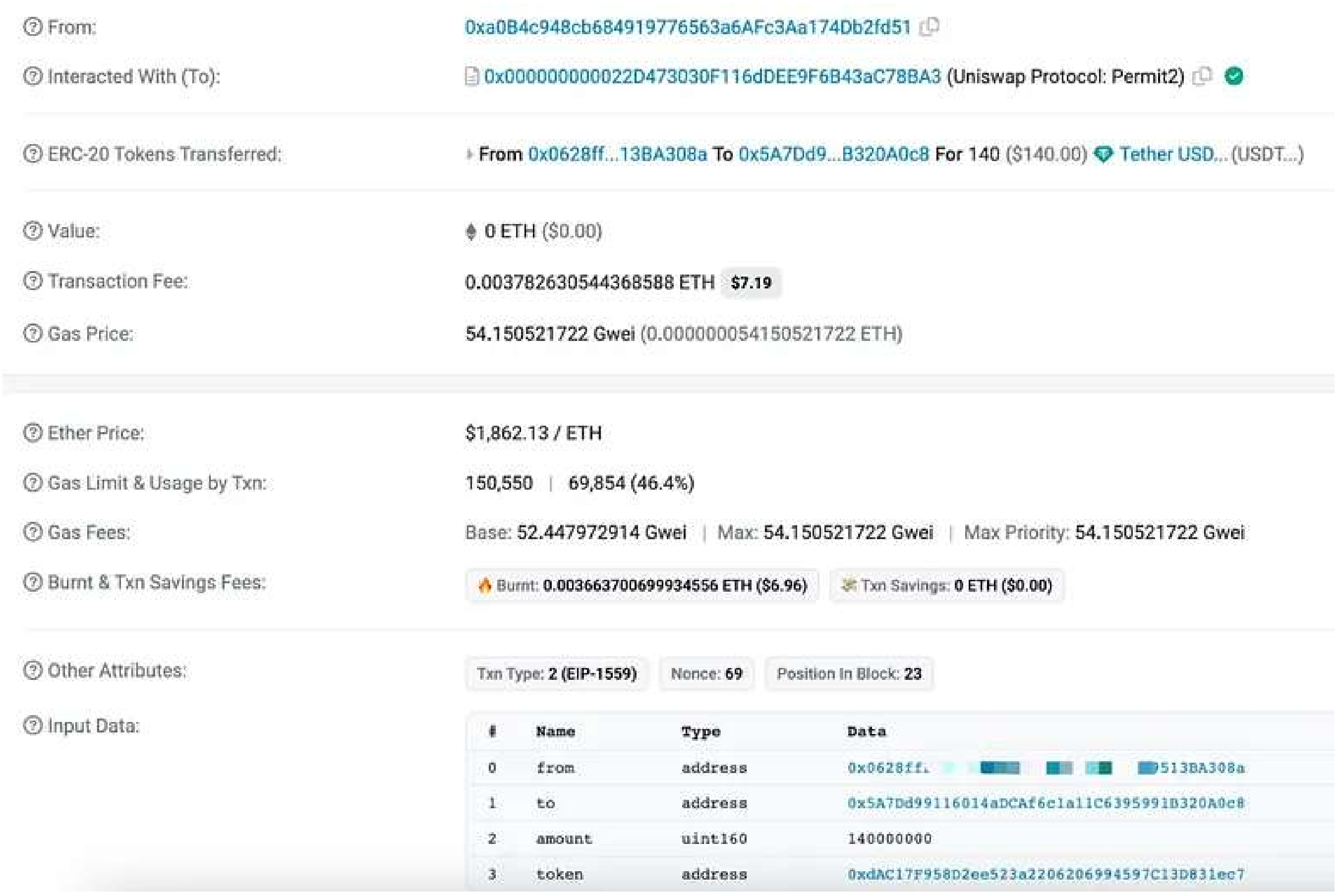
	Txn Hash	Method	Age	From		To	Value	Token
	0x8372fe939a8186bdc...	Transfer From	2 days 15 mins ago	0x0628ff...13BA308a	OUT	0x5A7Dd9...B320A0c8	140	Tether USD (USDT)

By looking into the transaction details, we can find some key clues:

● The address ending in fd51 transfers the victim's assets to the address ending in a0c8.

● This operation interacted with Uniswap's Permit2 contract.

So here comes a doubt, how did the address ending in fd51 get permission for this asset? Why is it related to Uniswap?

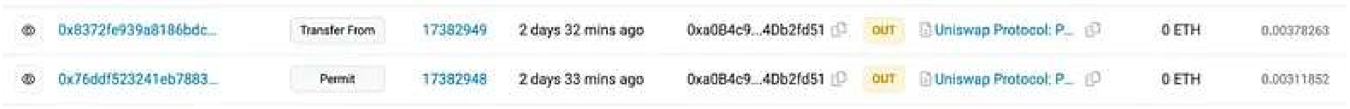


From:	0xa0B4c948cb684919776563a6AFc3Aa174Db2fd51
Interacted With (To):	0x000000000022D473030F116dDEE9F6B43aC78BA3 (Uniswap Protocol: Permit2)
ERC-20 Tokens Transferred:	From 0x0628ff...13BA308a To 0x5A7Dd9...B320A0c8 For 140 ($140.00) Tether USD...(USDT...)
Value:	0 ETH ($0.00)
Transaction Fee:	0.003782630544368588 ETH $7.19
Gas Price:	54.150521722 Gwei (0.000000054150521722 ETH)
Ether Price:	$1,862.13 / ETH
Gas Limit & Usage by Txn:	150,550 | 69,854 (46.4%)
Gas Fees:	Base: 52.447972914 Gwei | Max: 54.150521722 Gwei | Max Priority: 54.150521722 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.003663700699934556 ETH ($6.96) Txn Savings: 0 ETH ($0.00)
Other Attributes:	Txn Type: 2 (EIP-1559) Nonce: 69 Position In Block: 23
Input Data:	

#	Name	Type	Data
0	from	address	0x0628ff[redacted]513BA308a
1	to	address	0x5A7Dd99116014aDCAf6c1a11C6395991B320A0c8
2	amount	uint160	140000000
3	token	address	0xdAC17F958D2ee523a2206206994597C13D831ec7

First of all, we need to know that the prerequisite for successfully calling the TransferFrom function is that the caller needs to have approval of a token. When we use some Dapps, we need to perform an authorization (approve) operation first, so that Dapps' contract has the right to transfer our assets.

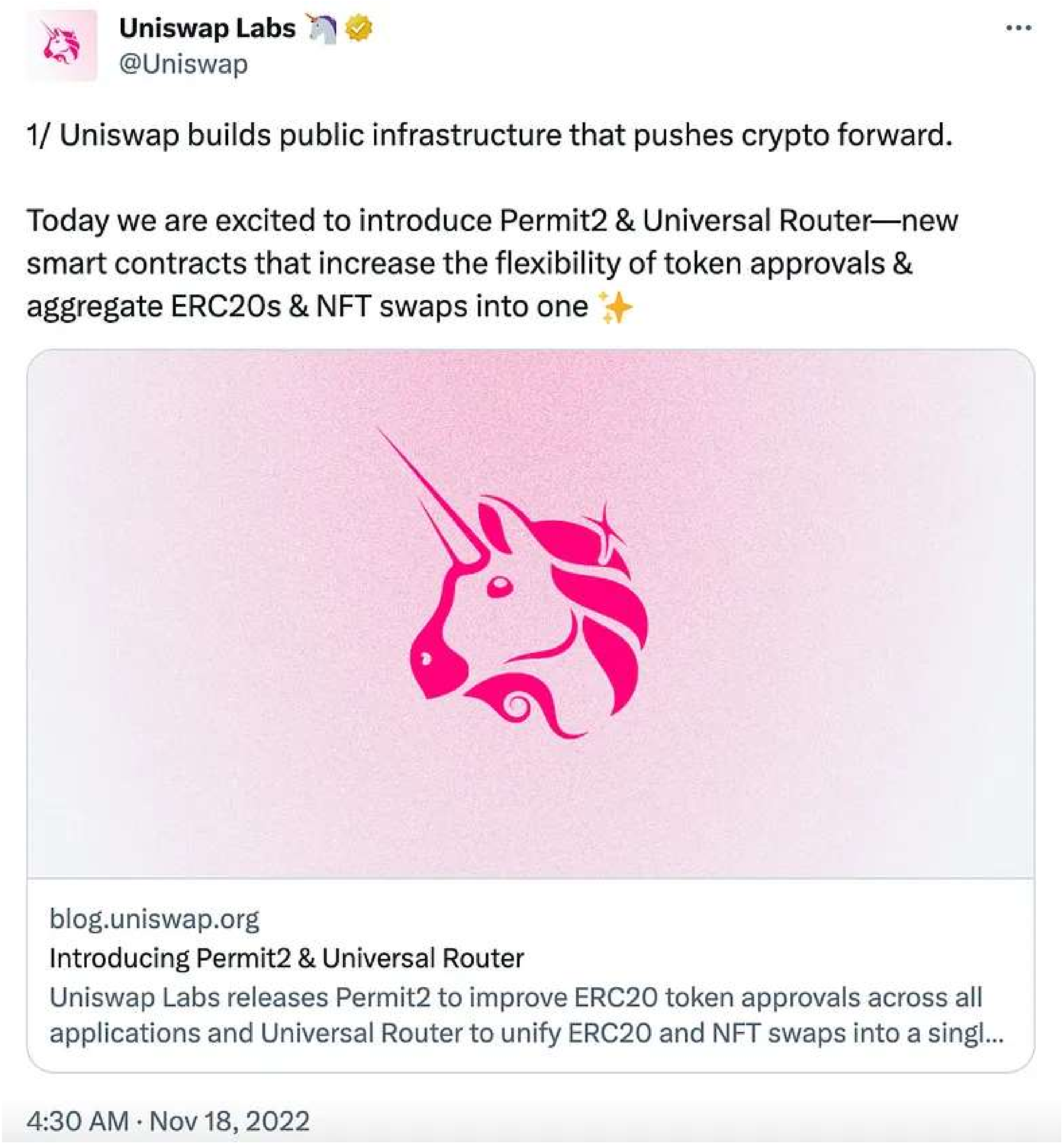
Before the address ending in fb51 called TransferFrom to transfer A's assets, you can see that this address has also called a Permit operation, and the interaction object of these two operations is Uniswap's Permit2 contract. What are this Permit function and Uniswap Permit2?



0x8372fe939a8186bdc...	Transfer From	17382949	2 days 32 mins ago	0xa0B4c9...4Db2fd51	OUT	Uniswap Protocol: P...	0 ETH	0.00378263
0x76ddf523241eb7883...	Permit	17382948	2 days 33 mins ago	0xa0B4c9...4Db2fd51	OUT	Uniswap Protocol: P...	0 ETH	0.00311852

The Uniswap Permit2 contract is a new smart contract launched by Uniswap at the end of 2022. According to the official statement, it is a token approval contract that allows token authorization to be shared and managed in different applications, creating a more unified, more cost-effective, and more secure user experience.

In the future, as more and more projects integrate with Permit2, Permit2 can achieve standardized token approval in all applications. Permit2 will improve user experience by reducing transaction costs while increasing the security of smart contracts.


Uniswap Labs 🦄
@Uniswap

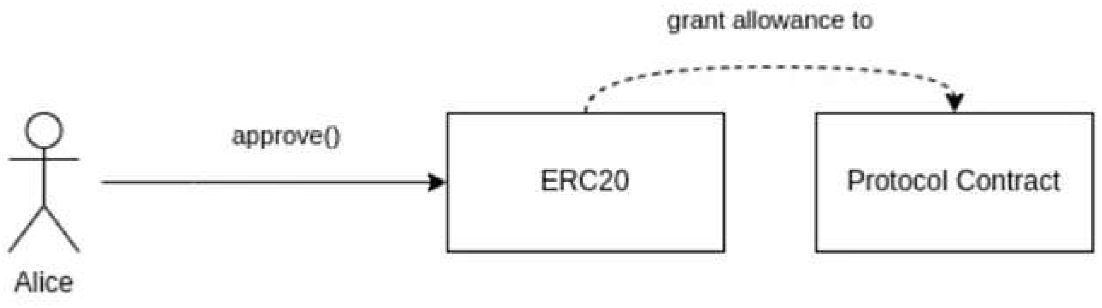
1/ Uniswap builds public infrastructure that pushes crypto forward.

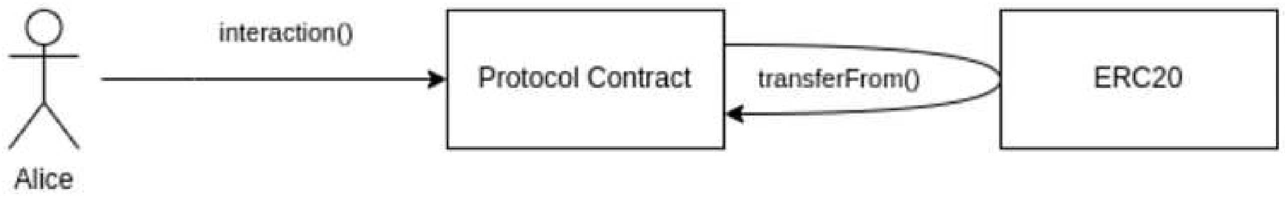
Today we are excited to introduce Permit2 & Universal Router—new smart contracts that increase the flexibility of token approvals & aggregate ERC20s & NFT swaps into one ✨

blog.uniswap.org
Introducing Permit2 & Universal Router
Uniswap Labs releases Permit2 to improve ERC20 token approvals across all applications and Universal Router to unify ERC20 and NFT swaps into a singl...

4:30 AM · Nov 18, 2022

Let's first explain why Uniswap launched Permit2. Let's assume a scenario. When we want to perform a Swap operation on a Dex, the traditional interaction method is that we need to authorize (approve) the Dex before performing Swap. We usually need to pay some gas fees twice, which is a great friction cost for us. I believe everyone has such an experience.

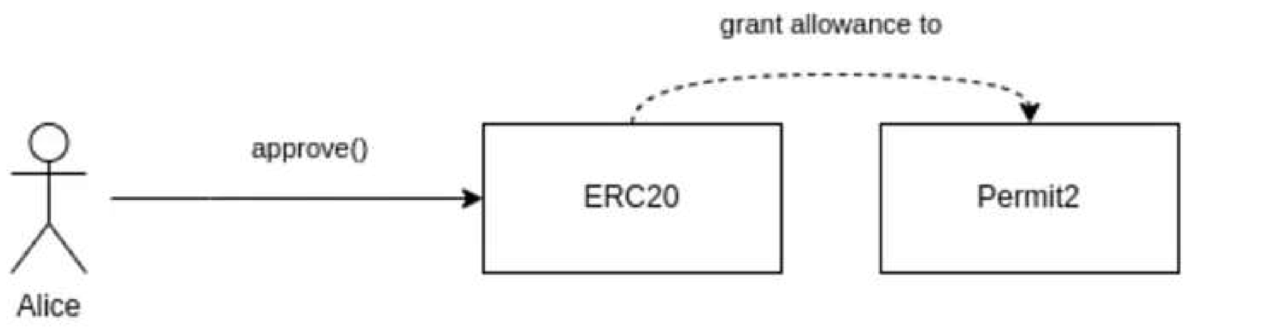


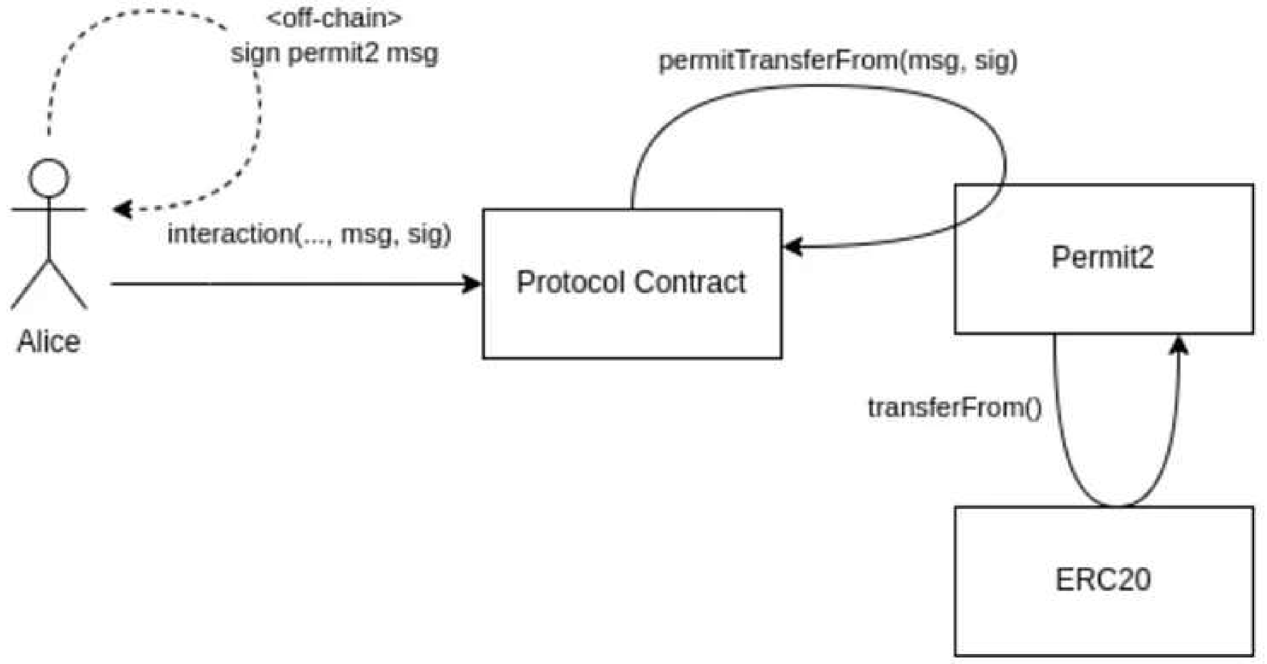


The launch of Permit2 may change the rules of the entire Dapp ecosystem. Simply put, the traditional method is that you need to authorize every time you interact with a Dapp for asset transfer, and Permit2 can save this step. This can effectively reduce user interaction costs and bring a better user experience.

The solution is that Permit2 acts as an intermediary between the user and the Dapp. The user only needs to authorize the permission of a token to the Permit2 contract. All Dapps that integrate the Permit2 contract can share this approved amount. For users, it reduces interaction costs and improves user experience. For Dapp, the improvement of user experience brings more users and funds, which is a win-win situation. At the same time, it can also be a double-edged sword, and the problem lies in the way of interaction with Permit2.

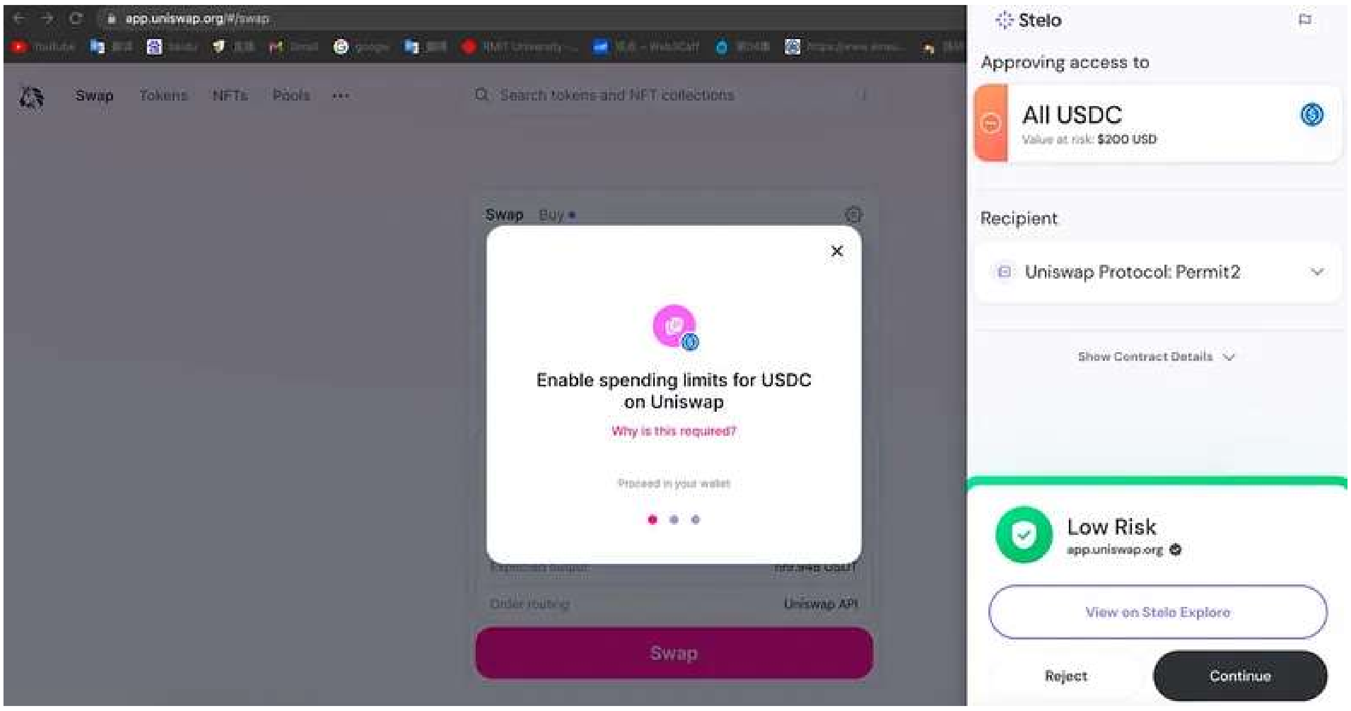
In the traditional interaction mode, whether it is authorization or transfer of funds, it is an on-chain interaction for users. Permit2 turns the user's operation into an off-chain signature, and all operations on the chain are completed by intermediate roles (such as the Permit2 contract and the projects integrating Permit2, etc.). The role of interaction is transferred from users to intermediate roles. Even if the user does not have ETH in the wallet, the user can use other Tokens to pay the gas fee or be fully covered by the intermediate role, depending on the choice of the intermediate role.



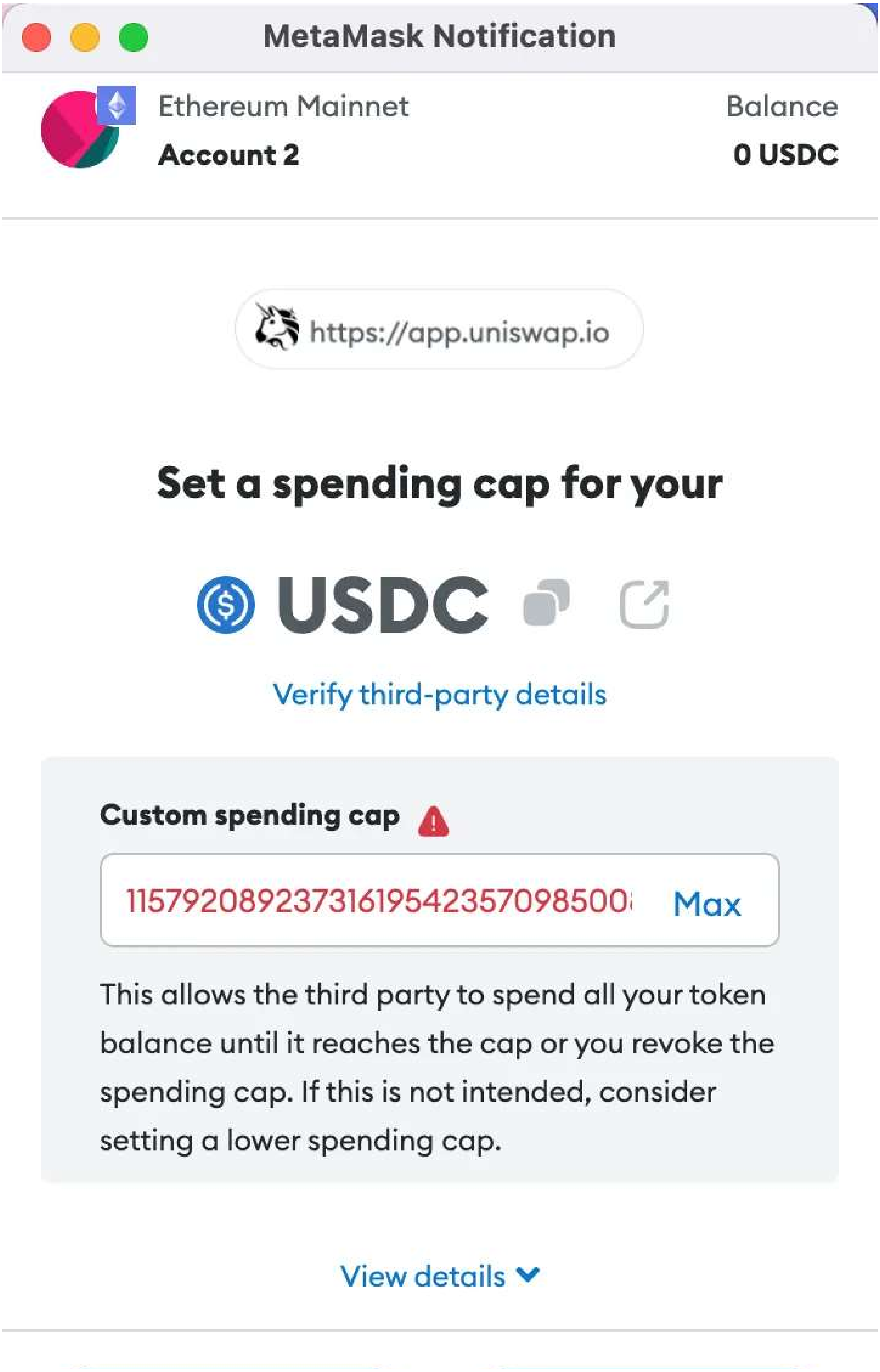


Although the launch of Permit2 may change the rules of Dapps in the future, it can be a strong double-edged sword. For users, an off-chain signature is the easiest step to put down their defenses. For example, when we use wallets to log in some Dapps, a signature is required to connect, and most people do not carefully check the content of the signature and do not understand the content of the signature.

After understanding the Permit2 contract, we will understand why the stolen assets interact with the Permit2 contract when we go back to the above incident. Let Bocai reproduce the Permit2 phishing. First of all, a crucial prerequisite is that the phished wallet needs to have a token authorized to Uniswap's Permit2 contract. Bocai found that as long as swap is performed on the Dapps integrated with Permit2 or Uniswap, it needs to be authorized to the Permit2 contract (Bocai uses a security plug-in in the picture below).



Another risky point is that no matter how much you want to swap, Uniswap's Permit2 contract will allow you to authorize the entire balance of a token by default. Although MetaMask will allow you to customize the input amount, I believe most people will click directly on the maximum or default value and the default value of Permit2 is unlimited.



MetaMask Notification

Ethereum Mainnet

Account 2

Balance

0 USDC

https://app.uniswap.io

Set a spending cap for your

USDC

Verify third-party details

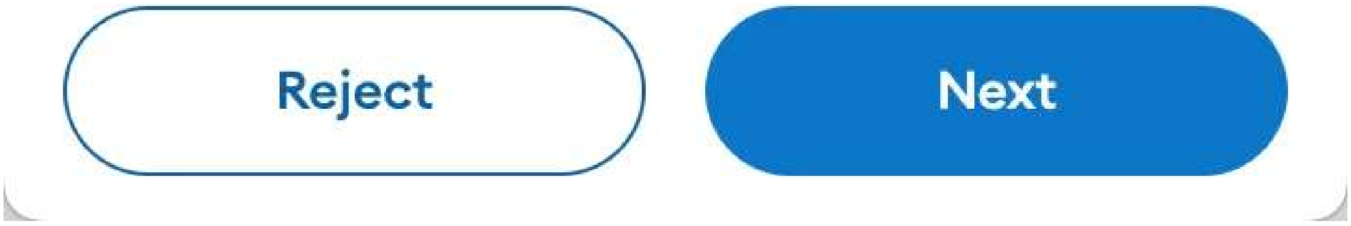
Custom spending cap

115792089237316195423570985008

Max

This allows the third party to spend all your token balance until it reaches the cap or you revoke the spending cap. If this is not intended, consider setting a lower spending cap.

View details

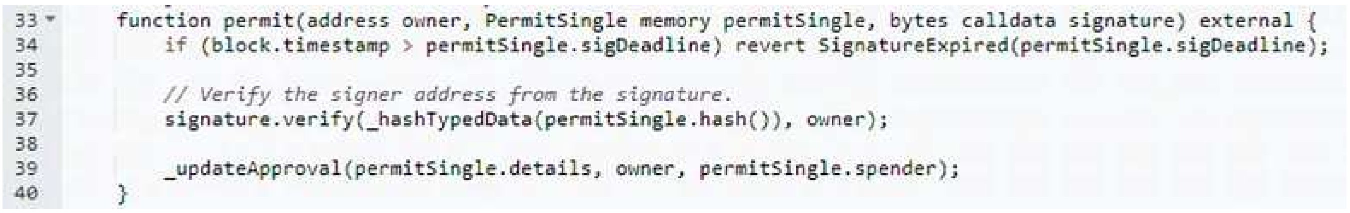


This means that as long as you have interacted with Uniswap and authorized the amount to the Permit2 contract, you will be exposed to the risk of this phishing scam.

The focus is the Permit function that interacted with the Permit2 contract in the address ending in fd51. This function simply uses your wallet to transfer the token amount that you authorized to the Permit2 contract to another address. That is to say, as long as hackers attain your signature, they can get permission for the token in your wallet and transfer your assets away.

Event Detailed Analysis

permit function:



```
function permit(address owner, PermitSingle memory permitSingle, bytes calldata signature) external {
    if (block.timestamp > permitSingle.sigDeadline) revert SignatureExpired(permitSingle.sigDeadline);

    // Verify the signer address from the signature.
    signature.verify(_hashTypedData(permitSingle.hash()), owner);

    _updateApproval(permitSingle.details, owner, permitSingle.spender);
}
```

You can think the Permit function as a way to sign a contract online. This function allows you to pre-sign a “contract” that allows someone else (spender) to spend some of your tokens in the future.

At the same time, you also need to provide a signature, just like signing a paper contract, to prove that this “contract” is really signed by you.

So how does this function work?

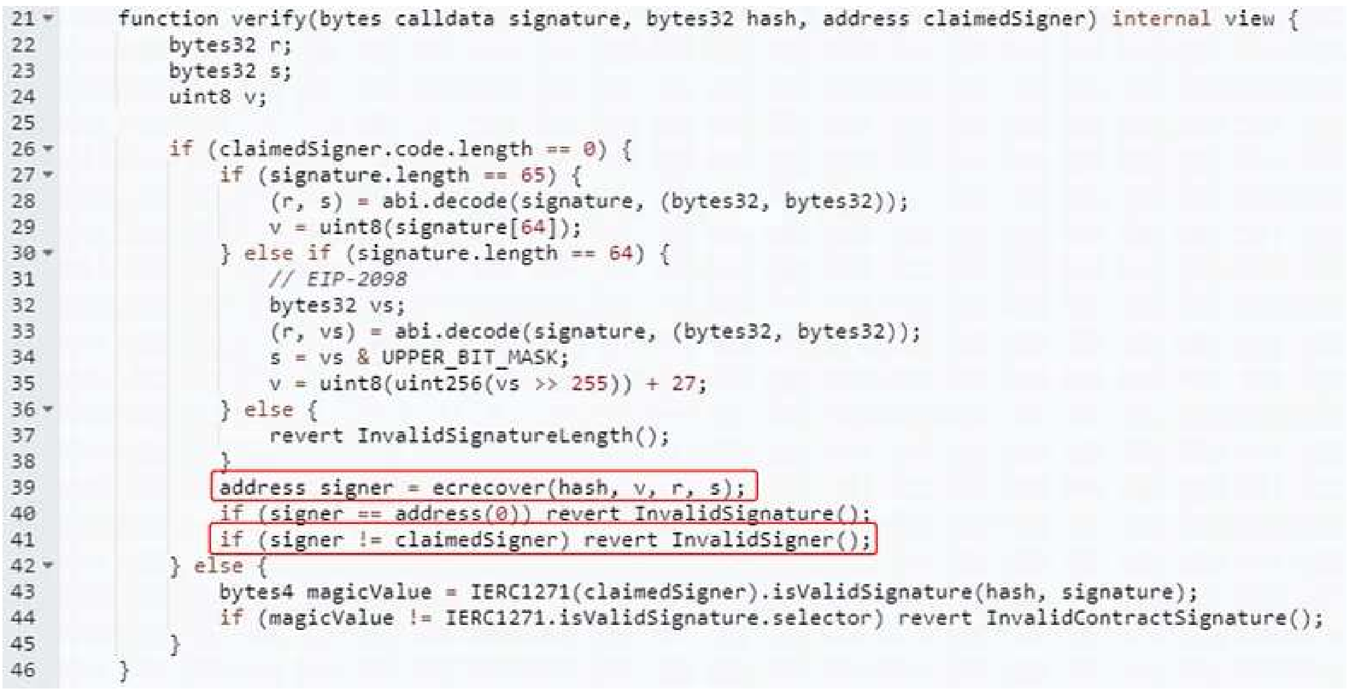
● First, it checks whether the current time exceeds the expiration date of your signature (sigDeadline). Just like the contract you signed has an expiration date, if the current time exceeds the expiration date, then this “contract” can no longer be used and the program will stop directly.

● Next, it checks if your signature is really yours. The program will use a special method (signature.verify) to check the signature to ensure that the signature is really signed by you and has not been forged by others.

● Finally, if the checks pass, the program updates the record to note that you have allowed others to use some of your tokens.

The focus is mainly on the verify function and the _updateApproval function.

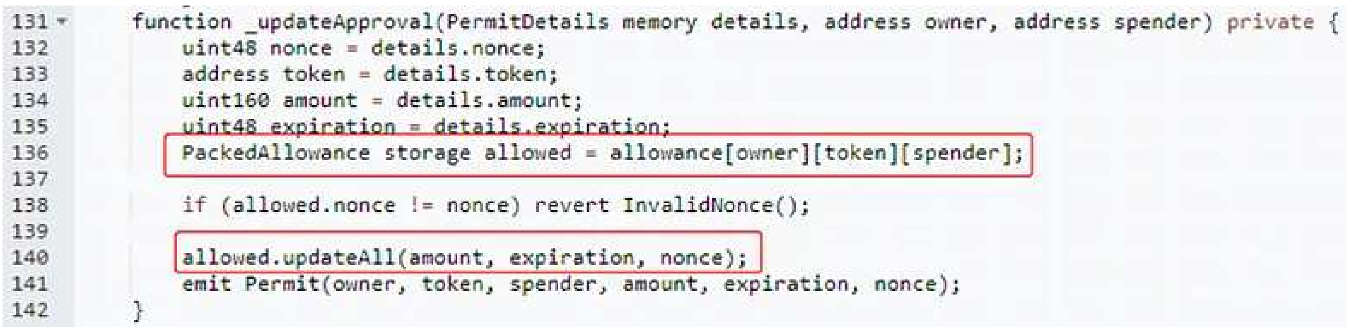
verify function:



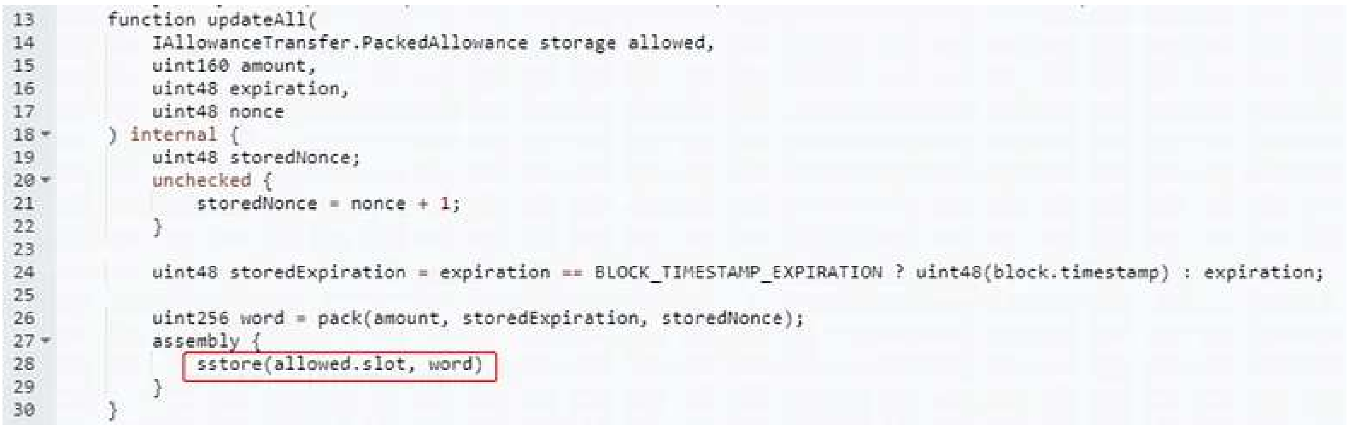
```
function verify(bytes calldata signature, bytes32 hash, address claimedSigner) internal view {
    bytes32 r;
    bytes32 s;
    uint8 v;

    if (claimedSigner.code.length == 0) {
        if (signature.length == 65) {
            (r, s) = abi.decode(signature, (bytes32, bytes32));
            v = uint8(signature[64]);
        } else if (signature.length == 64) {
            // EIP-2098
            bytes32 vs;
            (r, vs) = abi.decode(signature, (bytes32, bytes32));
            s = vs & UPPER_BIT_MASK;
            v = uint8(uint256(vs >> 255)) + 27;
        } else {
            revert InvalidSignatureLength();
        }
        address signer = ecrecover(hash, v, r, s);
        if (signer == address(0)) revert InvalidSignature();
        if (signer != claimedSigner) revert InvalidSigner();
    } else {
        bytes4 magicValue = IERC1271(claimedSigner).isValidSignature(hash, signature);
        if (magicValue != IERC1271.isValidSignature.selector) revert InvalidContractSignature();
    }
}
```

It can be seen that the verify function will obtain the three data v, r, and s from the signature parameter. v, r, and s are the values of the transaction signature. They can be used to restore the address of the transaction signature. You can see that after the contract restores the address of the transaction signature, it compares the restored address with the token owner's address. If they are the same, the verification is passed. The call of the _updateApproval function is continued. If they are different, the transaction is rolled back.

_updateApproval function:

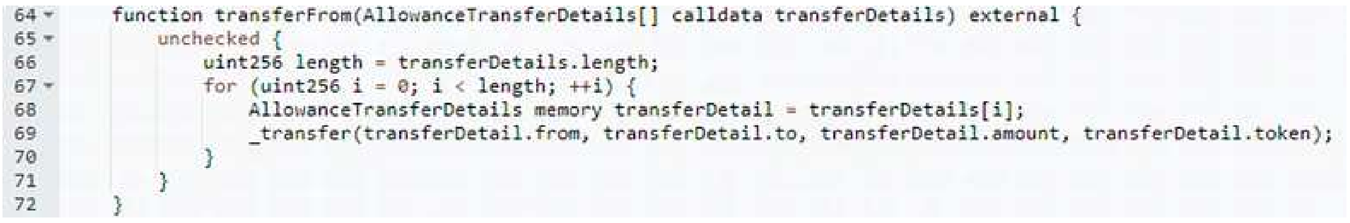


```
function _updateApproval(PermitDetails memory details, address owner, address spender) private {
    uint48 nonce = details.nonce;
    address token = details.token;
    uint160 amount = details.amount;
    uint48 expiration = details.expiration;
    PackedAllowance storage allowed = allowance[owner][token][spender];

    if (allowed.nonce != nonce) revert InvalidNonce();

    allowed.updateAll(amount, expiration, nonce);
    emit Permit(owner, token, spender, amount, expiration, nonce);
}
```

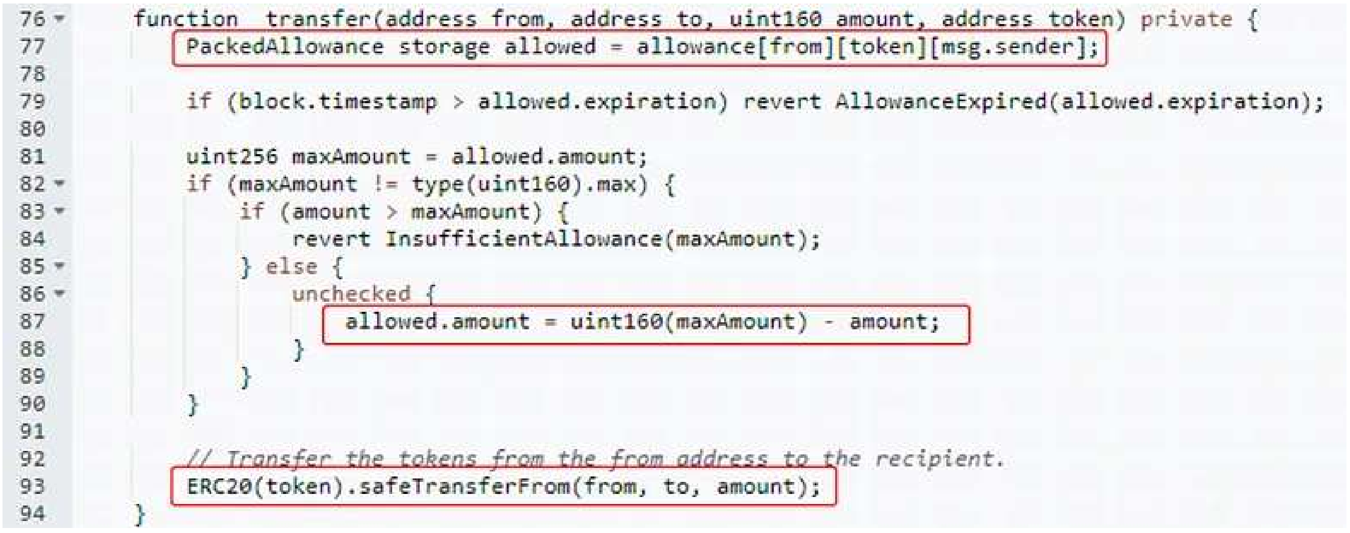


```
    function updateAll(
        IAllowanceTransfer.PackedAllowance storage allowed,
        uint160 amount,
        uint48 expiration,
        uint48 nonce
    ) internal {
        uint48 storedNonce;
        unchecked {
            storedNonce = nonce + 1;
        }

        uint48 storedExpiration = expiration == BLOCK_TIMESTAMP_EXPIRATION ? uint48(block.timestamp) : expiration;

        uint256 word = pack(amount, storedExpiration, storedNonce);
        assembly {
            sstore(allowed.slot, word)
        }
    }
```

When the signature verification is passed, the _updateApproval function will be called to update the authorization value, which means that your permissions have been transferred. At this time, it is convenient to call the TransferFrom function to transfer tokens to the specified address after being authorized, as shown in the code below.



```
    function transferFrom(AllowanceTransferDetails[] calldata transferDetails) external {
        unchecked {
            uint256 length = transferDetails.length;
            for (uint256 i = 0; i < length; ++i) {
                AllowanceTransferDetails memory transferDetail = transferDetails[i];
                _transfer(transferDetail.from, transferDetail.to, transferDetail.amount, transferDetail.token);
            }
        }
    }
```



```
    function _transfer(address from, address to, uint160 amount, address token) private {
        PackedAllowance storage allowed = allowance[from][token][msg.sender];

        if (block.timestamp > allowed.expiration) revert AllowanceExpired(allowed.expiration);

        uint256 maxAmount = allowed.amount;
        if (maxAmount != type(uint160).max) {
            if (amount > maxAmount) {
                revert InsufficientAllowance(maxAmount);
            } else {
                unchecked {
                    allowed.amount = uint160(maxAmount) - amount;
                }
            }
        }

        // Transfer the tokens from the from address to the recipient.
        ERC20(token).safeTransferFrom(from, to, amount);
    }
```

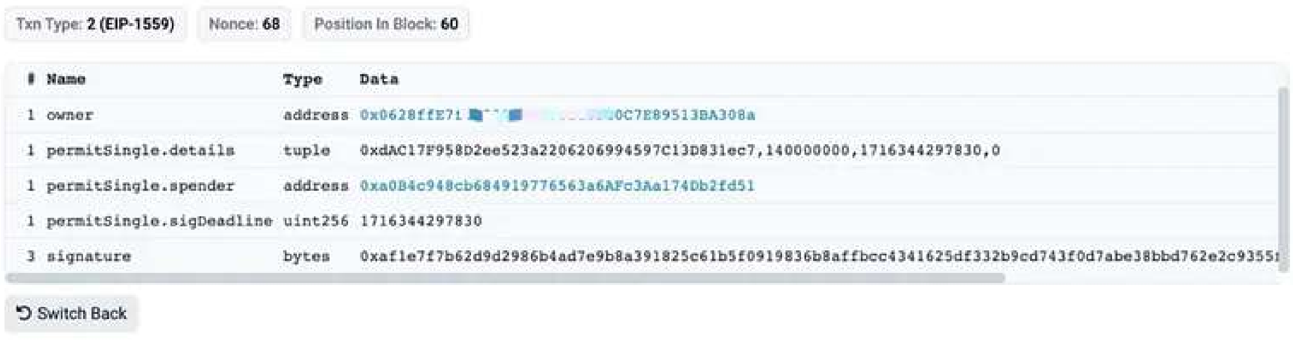
After explaining the permit function, let’s take a look at the real transactions on the chain. We can find out the details of this interaction:

Owner is the victim’s wallet address (end number 308a).

In Details section, we can see the authorized token contract address (USDT) and amount and other information.

Spender is the hacker address ending in fd51.

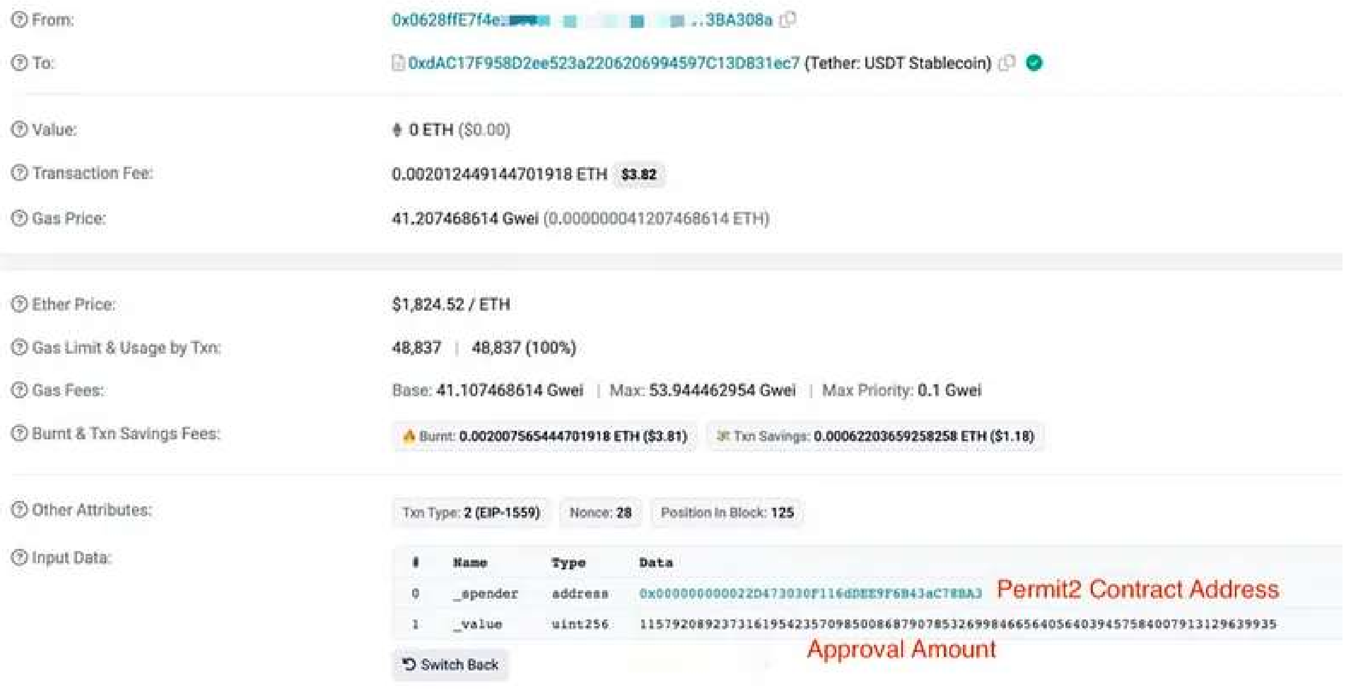
sigDeadline is the effective time of the signature, and signature is the signature information of the victim


Txn Type: **2 (EIP-1559)** Nonce: **68** Position In Block: **60**

#	Name	Type	Data
1	owner	address	0x0628ffE71[redacted]0C7E89513BA308a
1	permitSingle.details	tuple	0xdAC17F958D2ee523a2206206994597C13D831ec7,140000000,1716344297830,0
1	permitSingle.spender	address	0xa0B4c948cb684919776563a6AFc3Aa174Db2fd51
1	permitSingle.sigDeadline	uint256	1716344297830
3	signature	bytes	0xaf1e7f7b62d9d2986b4ad7e9b8a391825c61b5f0919836b8affbcc4341625df332b9cd743f0d7abe38bbd762e2c9355

Switch Back

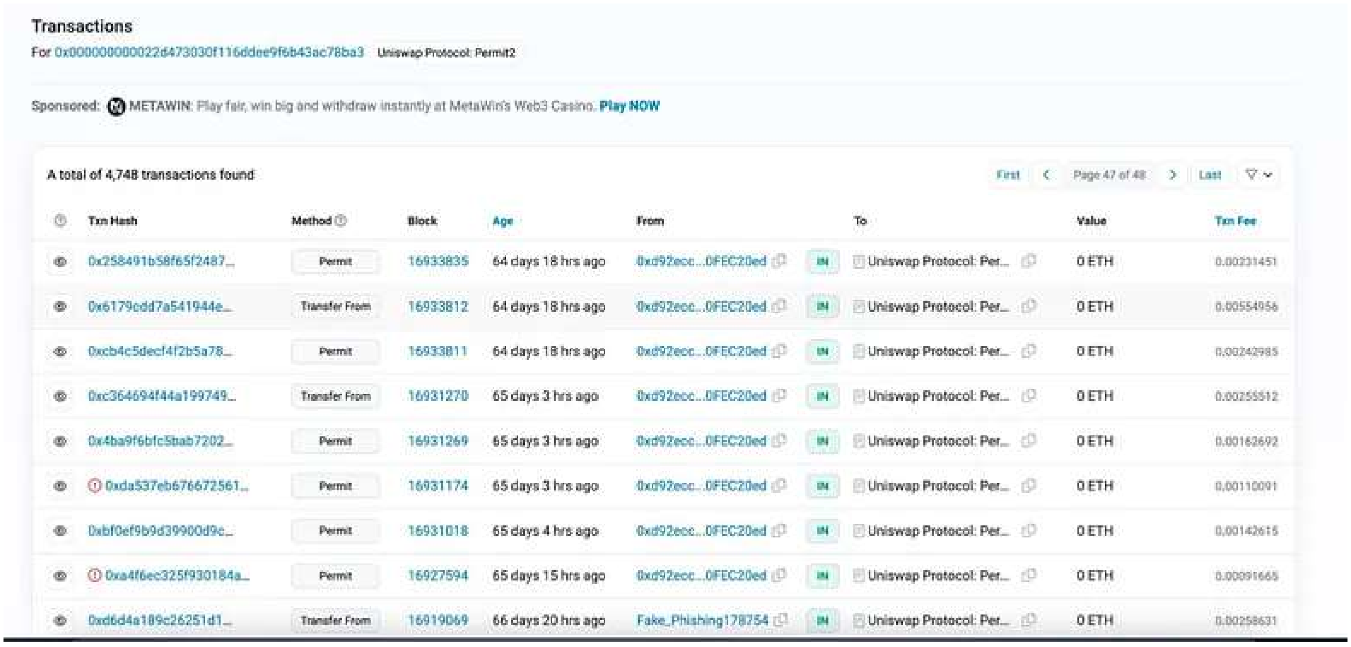
Looking back at the interaction records of the victim, we can find that when he used Uniswap before, he clicked on the default authorization amount, which is almost unlimited.


From:	0x0628ffE7f4e[redacted]..3BA308a
To:	0xdAC17F958D2ee523a2206206994597C13D831ec7 (Tether: USDT Stablecoin)
Value:	0 ETH ($0.00)
Transaction Fee:	0.002012449144701918 ETH $3.82
Gas Price:	41.207468614 Gwei (0.000000041207468614 ETH)
Ether Price:	$1,824.52 / ETH
Gas Limit & Usage by Txn:	48,837 | 48,837 (100%)
Gas Fees:	Base: 41.107468614 Gwei | Max: 53.944462954 Gwei | Max Priority: 0.1 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.002007565444701918 ETH ($3.81) Txn Savings: 0.00062203659258258 ETH ($1.18)
Other Attributes:	Txn Type: 2 (EIP-1559) Nonce: 28 Position In Block: 125
Input Data:	

#	Name	Type	Data
0	_spender	address	0x000000000022D473030F116dDEE9F6B43aC78BA3
1	_value	uint256	115792089237316195423570985008687907853269984665640564039457584007913129639935

Switch Back

A simple recap shows that the victim authorized Uniswap Permit2 to use unlimited USDT when using Uniswap before. Then he accidentally fell into the Permit2 signature phishing trap designed by hackers when performing wallet operations, and the hackers got his signature and used the signature to perform two operations, Permit and TransferFrom, in the Permit2 contract to transfer the victim’s assets away. What Bocai has observed is that Uniswap’s Permit2 contract has become a phishing haven. Permit2 phishing appears to be active about two months ago.



Transactions

For 0x000000000022d473030f116ddee9f6b43ac78ba3 Uniswap Protocol: Permit2

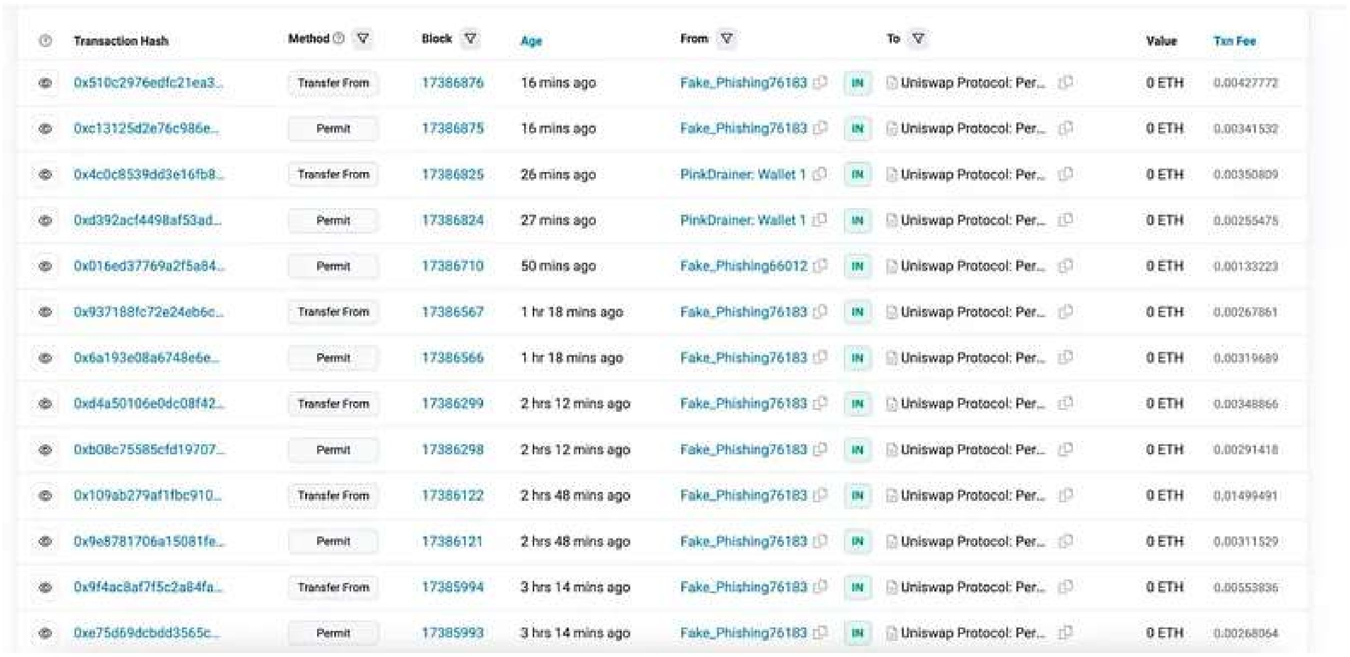
Sponsored: METAWIN: Play fair, win big and withdraw instantly at MetaWin's Web3 Casino. Play NOW

A total of 4,748 transactions found

First < Page 47 of 48 > Last

Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
0x258491b58f65f2487...	Permit	16933835	64 days 18 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00231451
0x6179cdd7a541944e...	Transfer From	16933812	64 days 18 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00554956
0xcb4c5decf4f2b5a78...	Permit	16933811	64 days 18 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00242985
0xc364694f44a199749...	Transfer From	16931270	65 days 3 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00255512
0x4ba9f6bfc5bab7202...	Permit	16931269	65 days 3 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00162692
ⓘ 0xda537eb676672561...	Permit	16931174	65 days 3 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00110091
0xbf0ef9b9d39900d9c...	Permit	16931018	65 days 4 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00142615
ⓘ 0xa4f6ec325f930184a...	Permit	16927594	65 days 15 hrs ago	0xd92ecc...0FEC20ed	IN	Uniswap Protocol: Per...	0 ETH	0.00091665
0xd6d4a189c26251d1...	Transfer From	16919069	66 days 20 hrs ago	Fake_Phishing178754	IN	Uniswap Protocol: Per...	0 ETH	0.00258631

In the interaction records, it can be found that almost most of them are marked phishing addresses (Fake_Phishing) and there are more and more victims.



Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x510c2976edfc21ea3...	Transfer From	17386876	16 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00427772
0xc13125d2e76c986e...	Permit	17386875	16 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00341532
0x4c0c8539dd3e16fb8...	Transfer From	17386825	26 mins ago	PinkDrainer: Wallet 1	IN	Uniswap Protocol: Per...	0 ETH	0.00350809
0xd392acf4498af53ad...	Permit	17386824	27 mins ago	PinkDrainer: Wallet 1	IN	Uniswap Protocol: Per...	0 ETH	0.00255475
0x016ed37769a2f5a84...	Permit	17386710	50 mins ago	Fake_Phishing66012	IN	Uniswap Protocol: Per...	0 ETH	0.00133223
0x937188fc72e24eb6c...	Transfer From	17386567	1 hr 18 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00267861
0x6a193e08a6748e6e...	Permit	17386566	1 hr 18 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00319689
0xd4a50106e0dc08f42...	Transfer From	17386299	2 hrs 12 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00348866
0xb08c75585cfd19707...	Permit	17386298	2 hrs 12 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00291418
0x109ab279af1fbc910...	Transfer From	17386122	2 hrs 48 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.01499491
0x9e8781706a15081fe...	Permit	17386121	2 hrs 48 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00311529
0x9f4ac8af7f5c2a84fa...	Transfer From	17385994	3 hrs 14 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00553836
0xe75d69dcbdd3565c...	Permit	17385993	3 hrs 14 mins ago	Fake_Phishing76183	IN	Uniswap Protocol: Per...	0 ETH	0.00268064

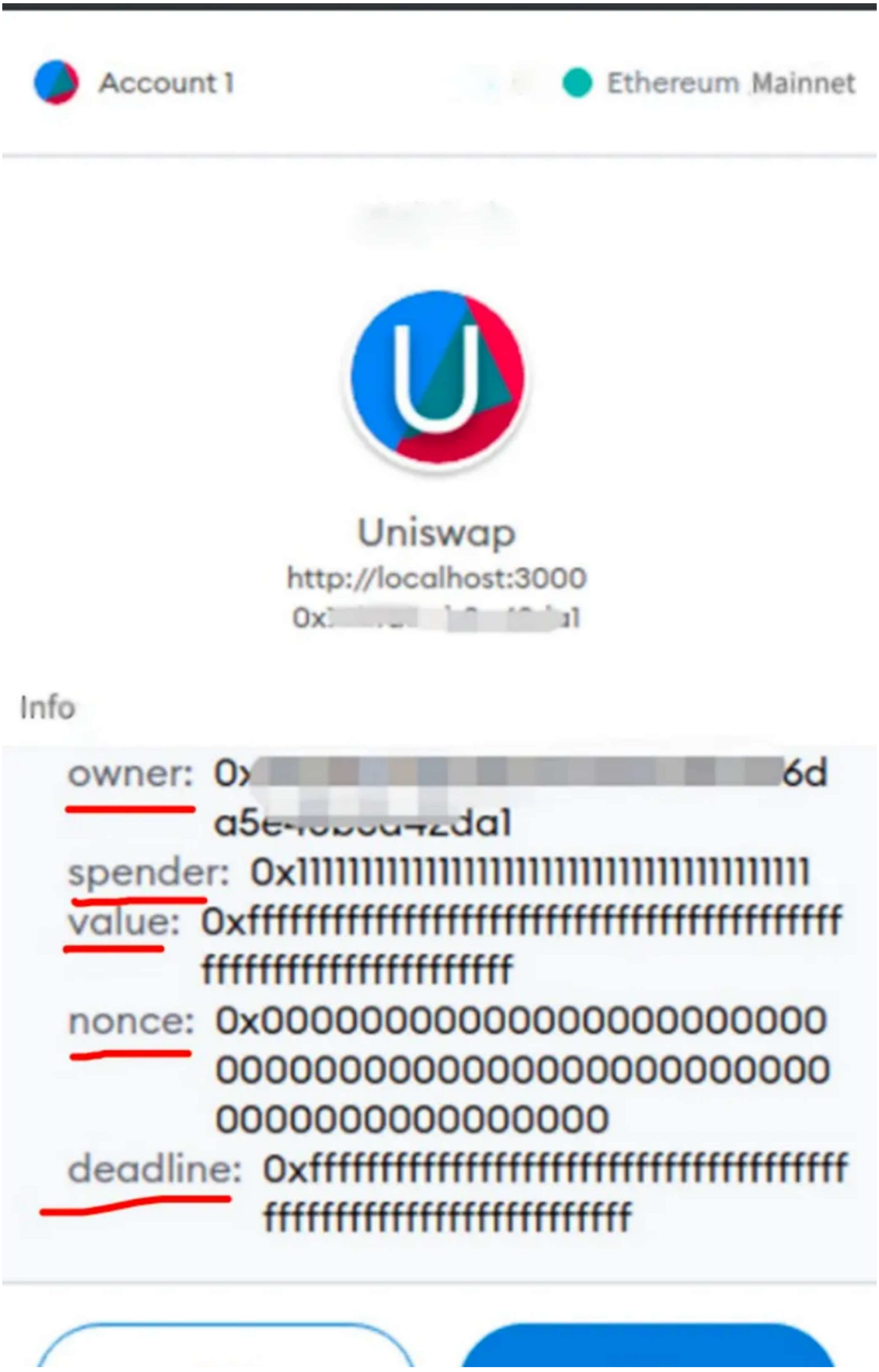
How to prevent it?

Considering that the Uniswap Permit2 contract may become more popular in the future and more projects will integrate the Permit2 contract for authorization sharing, some effective prevention methods for Permit2 phishing are as follows:

1. Understand and identify signature content:

The signature format of Permit usually includes several keys such as Owner, Spender, value, nonce, and deadline. If you want to enjoy the convenience and low cost brought by

Permit2, you must learn to recognize this signature format. (Downloading a security plugin is a good option)



[redacted]

Uniswap

http://localhost:3000

0x[redacted]a1

Info

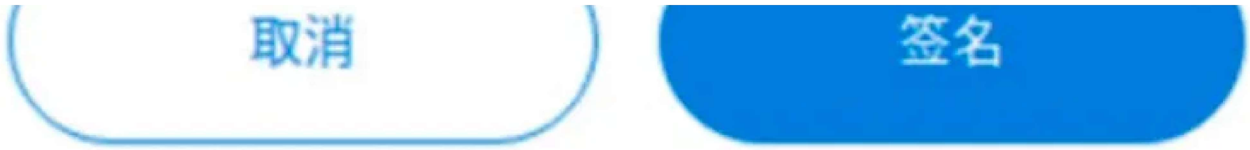
owner: 0x[redacted]6d
a5e[redacted]2da1

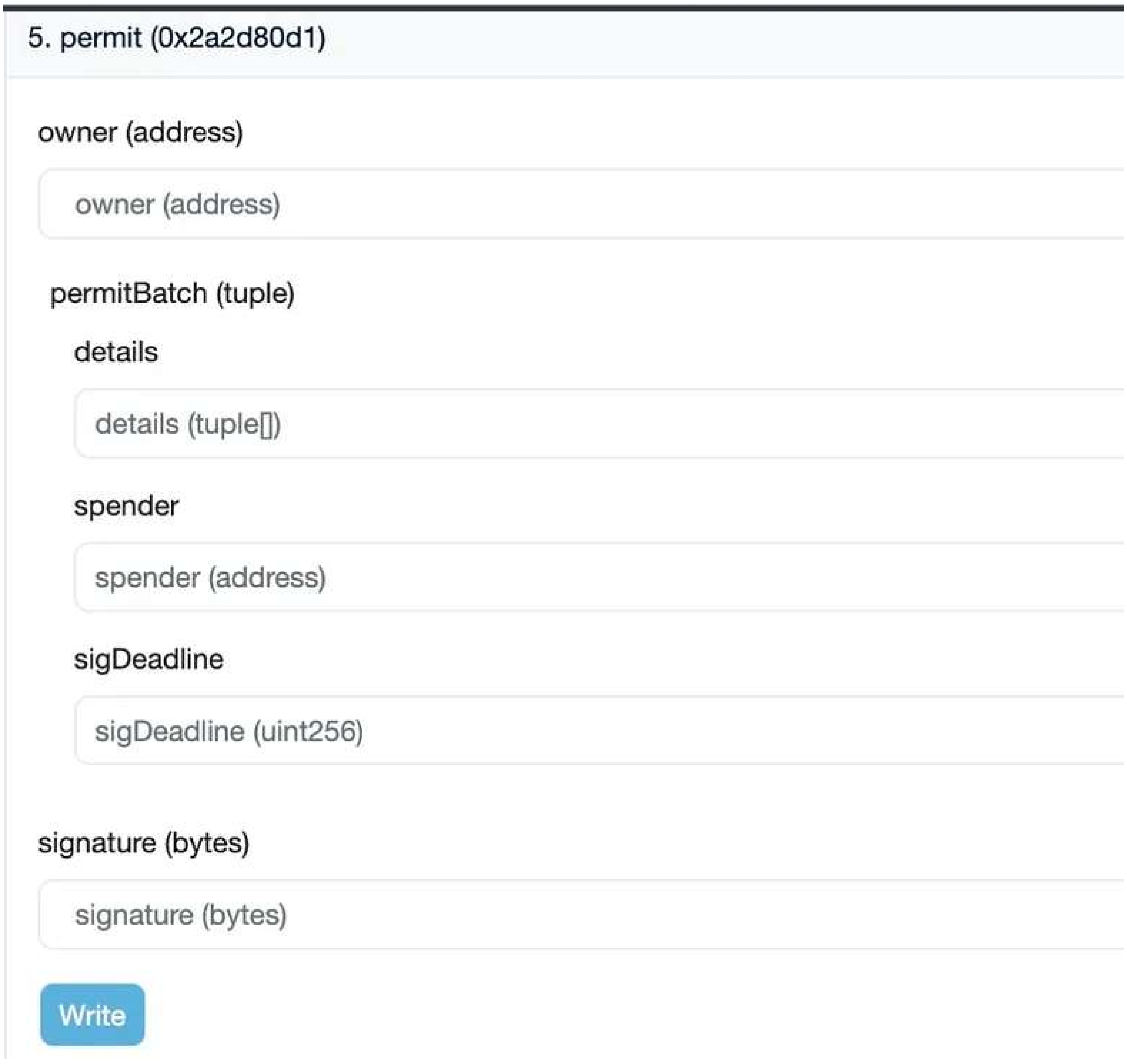
spender: 0x1111111111111111111111111111111111111111

value: 0xffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffff

nonce: 0x000000000000000000000000
0000000000000000000000000
000000000000000

deadline: 0xfffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffff





5. permit (0x2a2d80d1)

owner (address)

owner (address)

permitBatch (tuple)

details

details (tuple[])

spender

spender (address)

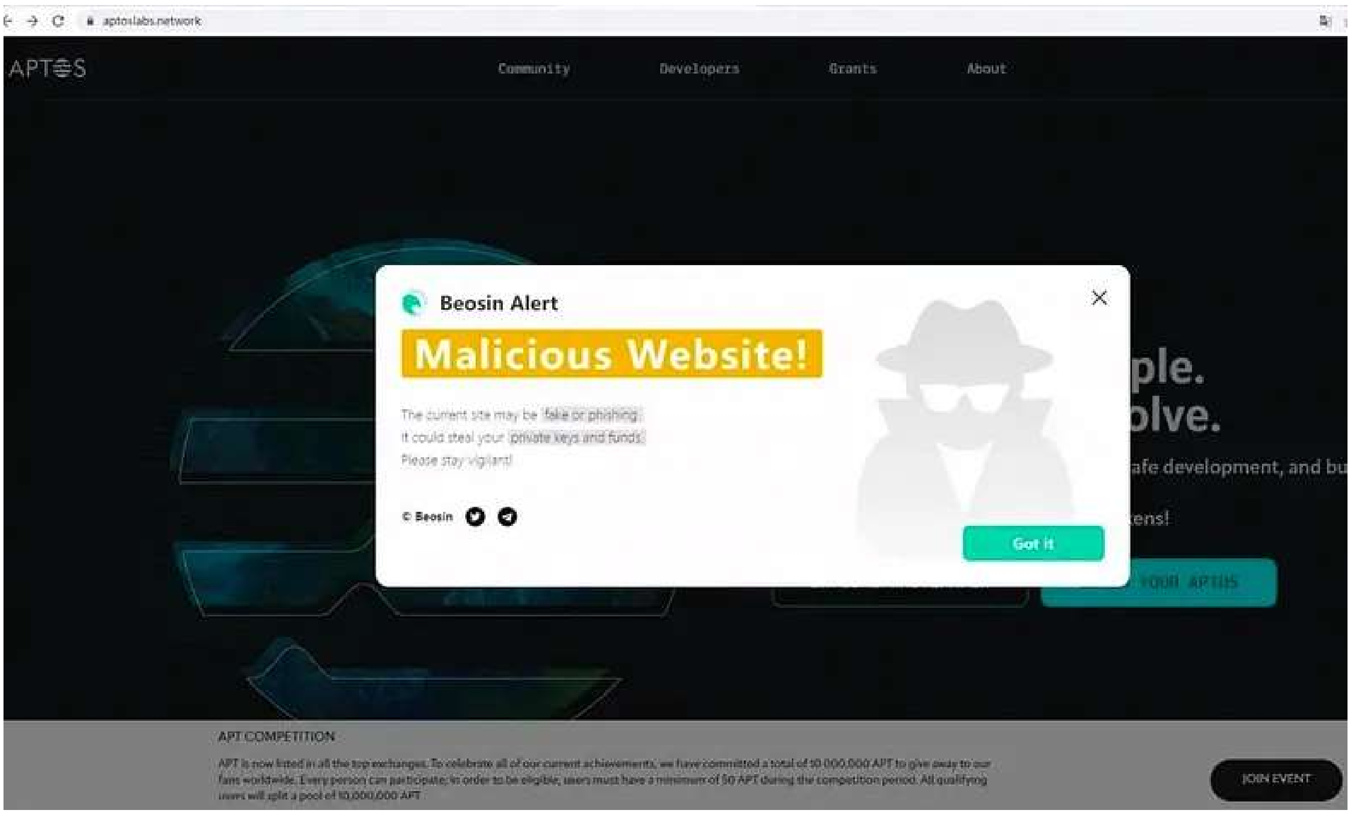
sigDeadline

sigDeadline (uint256)

signature (bytes)

signature (bytes)

We recommend Beosin Alert, an anti-phishing extension that can identify most phishing websites in Web3 and protect users' wallet and asset security.



The current site may be fake or phishing.
It could steal your private keys and funds.
Please stay vigilant!

APT COMPETITION

APT is now listed in all the top exchanges. To celebrate all of our current achievements, we have committed a total of 10,000,000 APT to give away to our fans worldwide. Every person can participate; in order to be eligible, users must have a minimum of 50 APT during the competition period. All qualifying users will split a pool of 10,000,000 APT

https://chrome.google.com/webstore/detail/beosin-alert/lgbhcpagiobjacpmcgckfgodjeogceji?hl=en

2. Wallets for assets storage and interaction can be separated:

If you have a large number of assets, it is recommended to put these assets in a cold wallet and put a small number of funds in a wallet for interactions, which can greatly reduce the loss in case of phishing scams.

3. Do not authorize a great amount to the Permit2 contract/cancel the authorization:

When you swap on Uniswap, you can only authorize the amount that you want to interact with so that although each interaction needs to be re-authorized with more interaction costs, it can avoid the signature phishing of Permit2. If you have already authorized a great amount, you can find the corresponding security plug-ins to cancel the authorization.

4. Check whether a token supports the permit function:

In the future, more and more ERC20 tokens may realize the permit function. Users need to pay attention to whether the token you hold supports this function. If it supports, then you must be careful and check whether each unknown signature is a signature of the permit function.

5. If there are assets stored on other platforms after being hacked, a comprehensive rescue plan is needed:

When you find that your assets have been transferred out by hackers and there are some tokens stored on other platforms by locking or staking, you need to withdraw and transfer them to a safe address. At this time, hackers may monitor your wallet all the time. Since they have your signature, as long as the tokens are in your stolen wallet, hackers can transfer the funds away directly. It is necessary to seek professional security company for help. The two processes of tokens withdrawal and tokens transfer need to be executed in one block. MEV transfers can be used, which requires some blockchain knowledge and coding skills. Victims can ask a professional security company such as the Beosin team to use frontrun script to save your assets.

Case reading: https://medium.com/@Beosin_com/a-crypto-influencer-is-under-a-sweeper-bot-attack-how-can-beosin-help-recover-his-funds-1db672c51ace

It is believed that there will be more and more phishing based on Permit2 in the future. This phishing method is extremely hidden and difficult to prevent. With the wider application of Permit2, more and more addresses will be exposed to risks. It will be helpful to share this article to others to prevent more people from being stolen.