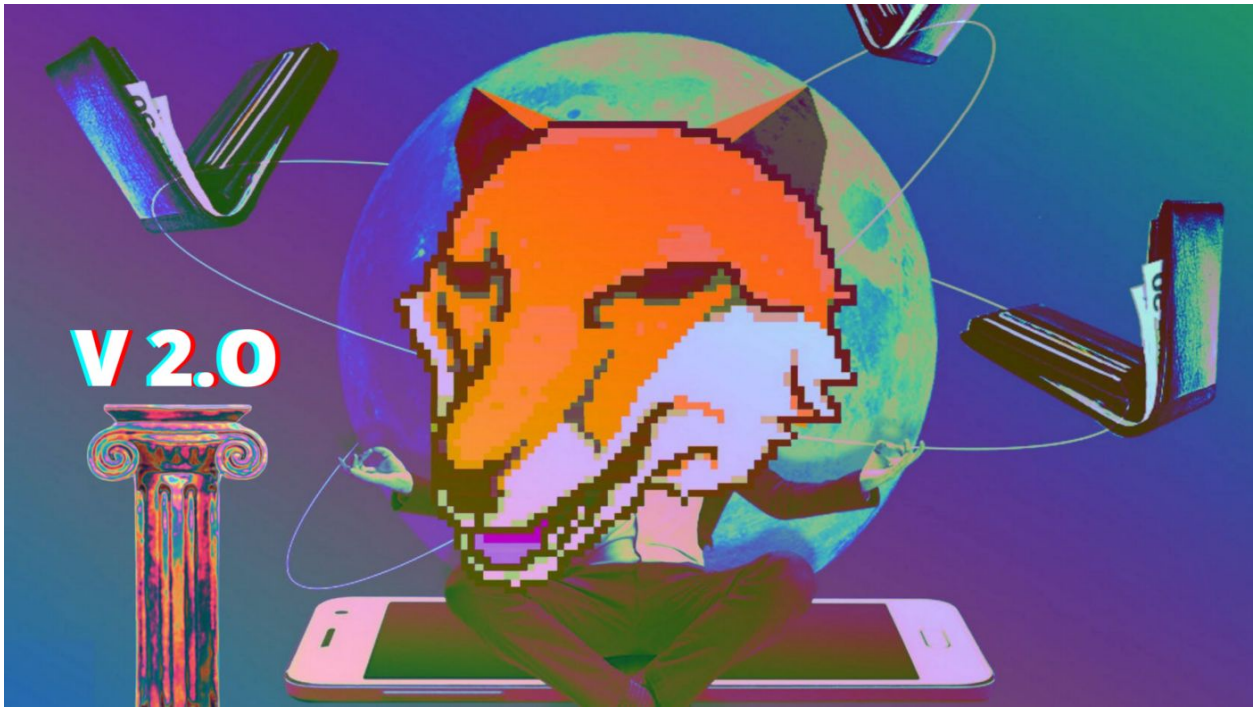
What Are the Blue Buttons of Death?



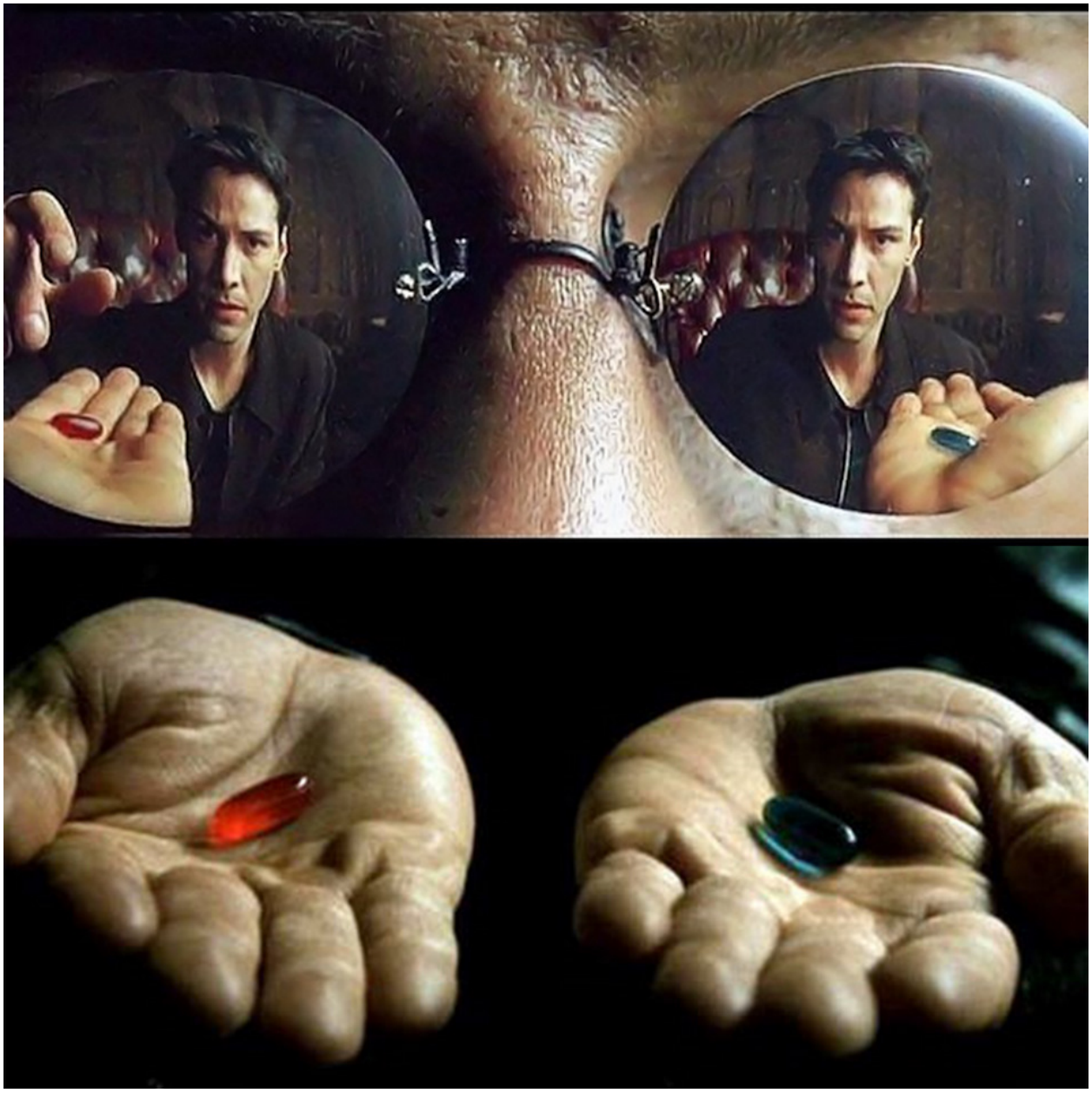
Authors: twitter.com/ortomichDev, twitter.com/officer_cia

Today, we will deal with what we in our arthouse understanding of the web3 security call the "blue button of death", and then Ortem will talk about this type of attack using the signature function, which uses EIP-712—which, in our subjective opinion, is a very underestimated danger!

First, I would like you to study these 2 sources to get an understanding of the issue before reading the article:

- **typefully.com/korpi87/iHknFMq—Important for everyone to read**
- **officercia.mirror.xyz/M0QAuwwbppAFj2KWZV02DEC8CtrxaX3R47kdpcTspvE—Especially important if you are into NFTs**

Do not confuse it with an allowance approve scam (to prevent it, you can use revoke.cash / unrekt.net) which targets ERC20 tokens, but not Ethers. (1, 2, 3, 4).

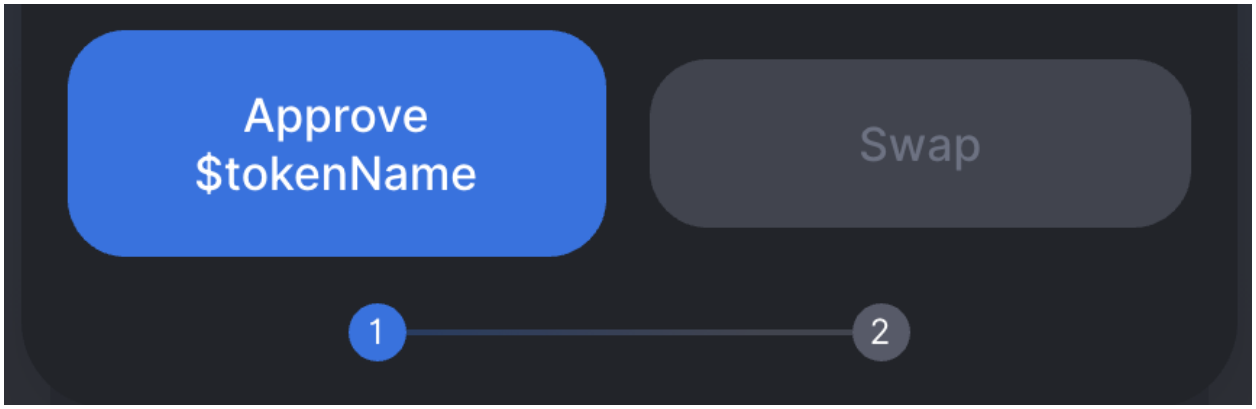


#I - Approve Button

- **Source:** medium.com/mycrypto/bad-actors-abusing-erc20-approval-to-steal-your-tokens-c0407b7f7c7c

A lot of tokens on Ethereum use the [ERC/EIP20 standard](). This is a standard interface for the token contract that details the contract properties, functions, function arguments, and function return types.

One of these standard functions is function approve(address _spender, uint256 _value) public returns (bool success). This allows a third party to send tokens from your account on your behalf. You'll recognize this pattern if you've ever done a swap through an AMM (like Uniswap), as you'll have to approve the AMM to spend your tokens before you swap.



Bad actors have learned how to exploit this function, as it's harder for unsuspecting users to avoid when they're only expecting scams that outright ask for their private keys.

Exploiting token approvals is a clever approach because users generally think: "If they don't have my key, then they can't sign a transaction, so they cannot steal my assets." Whilst generally accurate, this is not true when we talk about ERC20 tokens and other token standards.

If you or someone you know may have approved unnecessary amounts of tokens on any product, fix it now with revoke.cash or app.unrekt.net !

#II - EIP-712: PoC

First, a little preface. This EIP allows you to approve tokens by signing, that is, if you are already used to all approves being given as a separate transaction, you won't believe it. You can also approve with a simple signature, which will save you some gas.

Signature Request

Uniswap

https://app.uniswap.org

0xd79f47...391eb167

Message

owner: 0xD79f479f56Dac4E4102f4189Ae550adc391eb167

spender: 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45

value: 9000000000000000000

nonce: 2

deadline: 1662549856

CANCEL

SIGN

This EIP is implemented in some ERC20 tokens, among them $USDC, $UNI, and others. Everybody got used to the fact that the signature does not carry any danger, but then we showed you how you can steal all the money from your wallet with the eth_sign function:

- **officercia.mirror.xyz/M0QAuwwbppAFj2KWZV02DEC8CtrxaX3R47kdpcTspvE**

But that method had one special feature — a big red warning sign from the MetaMask that warned you about the risks. In this case, if an attacker forges a signature for EIP-712, the MetaMask will not warn you in any way, and you will not even know that you have been scammed!

https://youtu.be/8mzo9odDCVU

To understand the next paragraph, you need to learn for yourself how signatures work and have an idea of the structure of a smart contract.

Next, we need to make that very signature. The EIP-712 type can differ from token to token, so ideally, you should look for a different signature approach for each token.

Signature Request

Uniswap

https://app.uniswap.org

0xd79f47...391eb167

Message

owner: 0xD79f479f56Dac4E4102f4189Ae550adc391eb167

spender: 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45

value: 9000000000000000000

nonce: 2

deadline: 1662549856

CANCEL

SIGN

Then we have to take v, r, and s from our signature and pass them and a few other parameters to the permit function on the token contract we want to access.

```
function permit(address holder, address spender, uint256 nonce, uint256 expiry,
                bool allowed, uint8 v, bytes32 r, bytes32 s) external
{
    bytes32 digest =
        keccak256(abi.encodePacked(
            "\x19\x01",
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH,
                                 holder,
                                 spender,
                                 nonce,
                                 expiry,
                                 allowed))
    ));

    require(holder != address(0), "Dai/invalid-address-0");
    require(holder == ecrecover(digest, v, r, s), "Dai/invalid-permit");
    require(expiry == 0 || now <= expiry, "Dai/permit-expired");
    require(nonce == nonces[holder]++, "Dai/invalid-nonce");
    uint wad = allowed ? uint(-1) : 0;
    allowance[holder][spender] = wad;
    emit Approval(holder, spender, wad);
}
```

As we can see, at the very end of the function, after all the checks have been passed, we see a call to the allowance function, which is responsible for approving.

All information is presented strictly for introductory purposes! Do not commit crimes!

For those who want to understand the work of this mechanism in detail — we've prepared a concept scam site; here is a link to it, all set to work with $DAI on Rinkeby!

- **github.com/ortomich/scam_with_sign_2**

#III - Fantastic Beasts and How to Protect from Them

First of all, I want to say that you can meet this scam anywhere - you can get such a site with a spam token, accidentally get into such an attack if they hack into a legitimate Web3 resource that you use or deliver it to you in some other way, but the whole point is the same - to make you click on a link and do something.

If, in the case of approving, you can save by using revoke.cash or cointool.app and follow approve hygiene - when you have to cancel all approves every time, in the case of signing, I

advise you to use the basic tips from my Guide, use the separation of devices and never do what you do not understand!

Follow the 25 rules in this set, the first 10 rules relate to personal security and the rest to corporate security. Also, keep an eye on the latest trends in crypto OpSec, that always makes sense. Don't be afraid of links, you don't need all of them but you should be able to pick up which will interest you the most for your own Pathway.

- **DarkNet-DeepWeb OpSec Guide**
- **ThreatModeling**
- **Read about Timing Attack | Attack via a Representative Sample**

Use extensive measures when working with files, and always keep an eye on the latest security trends even if your area is far from it. Take this subreddit and this awesome old & trusted resource as the first step. In our dangerous world, anyone can become a target, especially in crypto.

Forewarned is forearmed! Stay safe!

- **iples of storing crypto, cold wallet security**
- **2 violent attack vectors in Crypto: a closer look**
- **A CIA Agent's Guide to Steganography, Fooling the KGB, and Protecting Your Crypto**
- **OpSec in Crypto & Web3.0: Thoughts**
- **A View on OpSec Through the Prism of Time**
- **All known smart contract-side and user-side attacks and vulnerabilities in Web3.0, DeFi, NFT and Metaverse**

That said, it doesn't really matter what industry you're in. If you have any sensitive, proprietary information at all, then you could very well be a target. This is a good thing to always keep in mind.

Learn the latest attack techniques, white-hat cheatsheets, and defense methods, and join hacker communities - because only with knowledge can we defeat the knowledge of hackers. In this intellectual battle, the most prepared will win, and I believe that it will be you, Anon. It sounds scary but it is possible, the main thing is to always think ahead.

Support is very important to me, with it, I can spend less time at work and do what I love - educating DeFi & Crypto users!

- **Check out my GitHub**
- **Track all my activities**
- **All my Socials**
- **Join my TG channel**

If you want to support my work, you can send me a donation to the address:

- **0xB25C5E8fA1E53eEb9bE3421C59F6A66B786ED77A or officercia.eth — ETH, BSC, Polygon, Optimism, Zk, Fantom, etc**
- **17Ydx9m7vrhnx4XjZPuGPMqrhw3sDviNTU - BTC**

- **4AhpUrDtfVSWZMJcRMJkZoPwDSdVG6puYBE3ajQABQo6T533cVvx5vJRc5fX7sktJe67mXu1CcDmr7orn1CrGrqsT3ptfds - Monero XMR**

Also published [here.]