

Stop Spoofing My Wallet!

Demystifying Simulation Spoofing Attacks

2025/01/11

slipper  OFFSIDE LABS



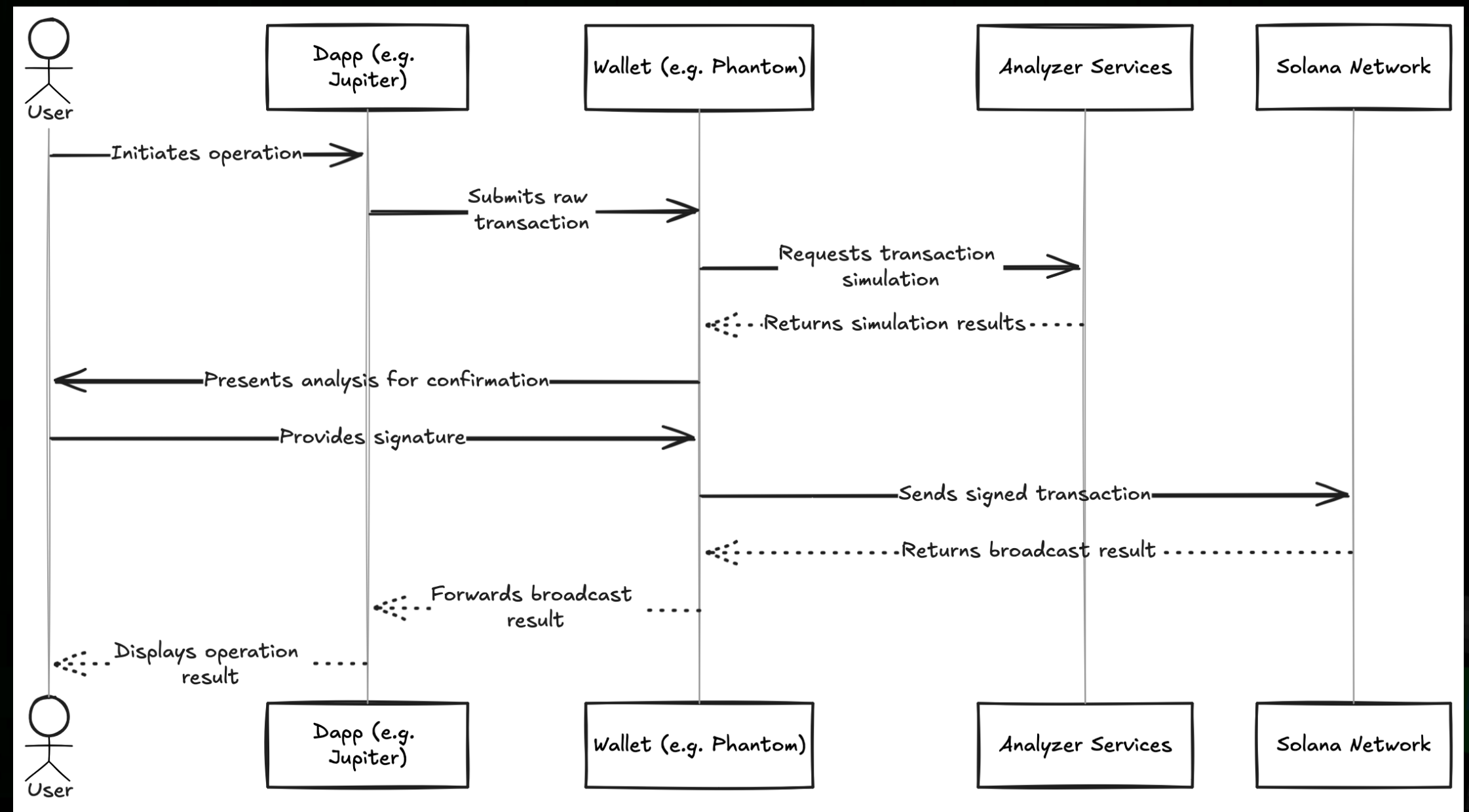
The Growing Threat to Solana Wallets

- Surge in phishing attacks targeting Solana users.
- Attacks bypass even secure wallet protections.
- Users unknowingly authorize malicious activities.



What Happens Behind the Scenes?

- Wallet interactions have hidden complexities.
- A simple "Sign" can authorize dangerous activities.
- Example: Fake Jupiter swap drains user accounts.





Drainer Alert: A Series of Sophisticated Attacks

01 Targets

Solana users using Phantom wallet on Chrome with Windows.

02 Platforms

Jupiter, Raydium, and others.

03 Characteristics

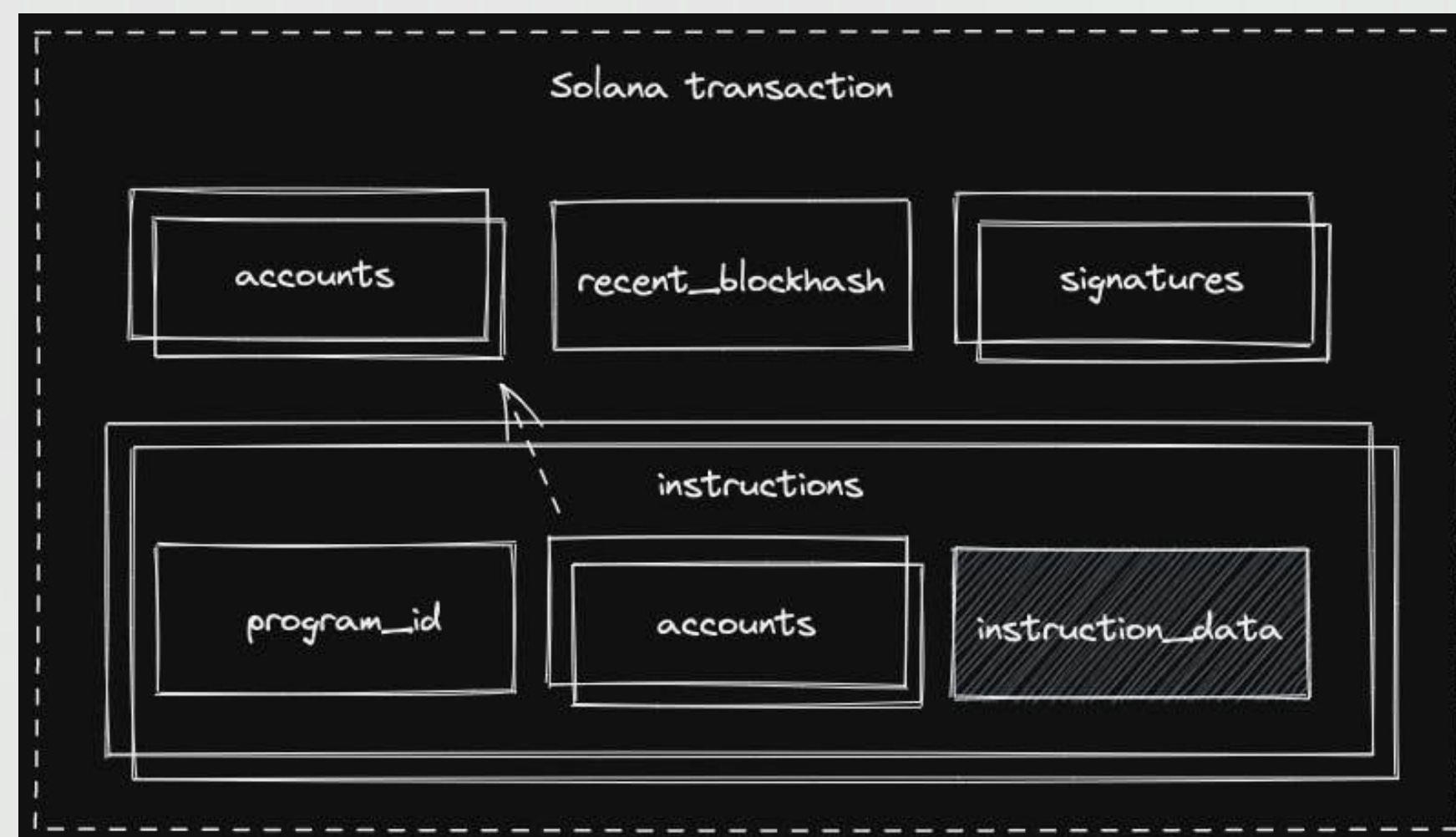
- Not widespread but highly effective.
 - Random victims, not targeting whales.
 - Triggered by user interactions.
-

How the Attack Works

Understanding the Complexity of Solana Transactions

Key Components of a Solana Transaction:

1. Signatures: Cryptographically prove authorization for the transaction.
2. Message: Core content of the transaction, containing: Header/Account Addresses/Recent Blockhash/Instructions



Token Swap Process on Jupiter:

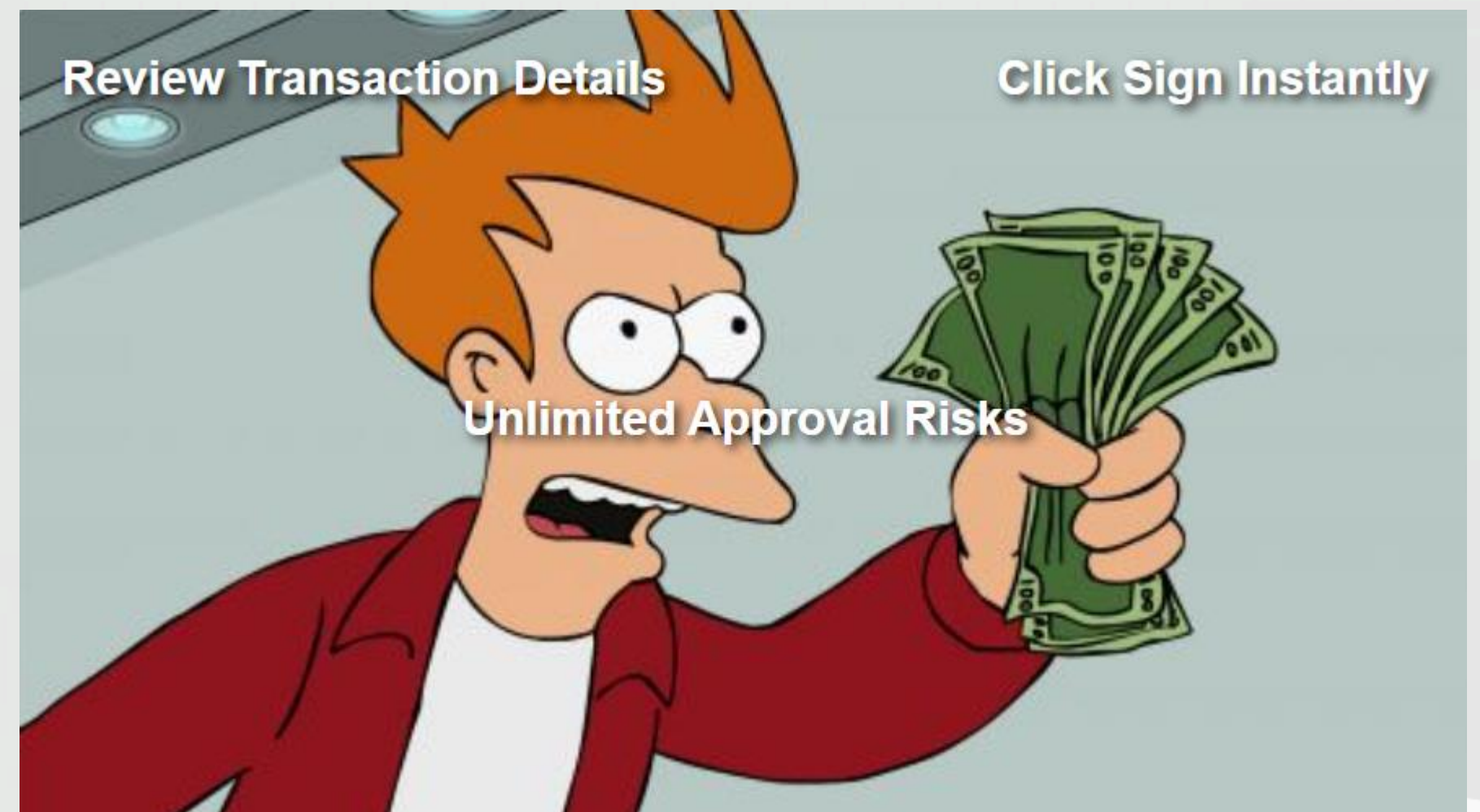
- Fetch token price and determine the **best route**.
- Assemble transaction instructions and send to wallet.
- Wallet simulates, user approves, and transaction is signed & broadcasted.





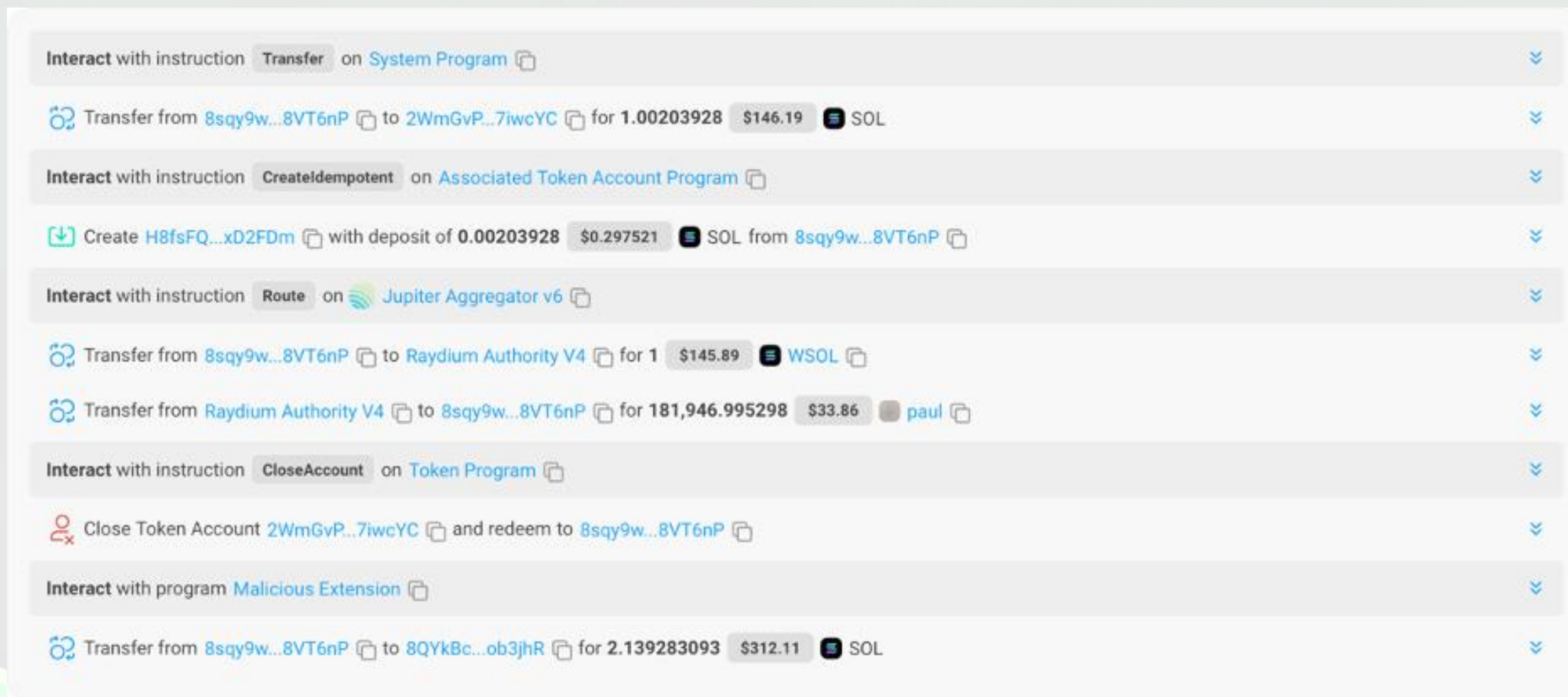
Key Risk: "Unlimited Approval":

- Signing grants **full authority** over accounts in the transaction.
- Can enable **unexpected actions** (e.g., malicious token transfers).
- Users must review transactions carefully.



How the Attack Works

Case Study – Anatomy of a Malicious Transaction



The screenshot displays a transaction log with the following instructions:

- Interact with instruction** `Transfer` on `System Program`
 - Transfer from `8sqy9w...8VT6nP` to `2WmGvP...7iwcYC` for `1.00203928` `$146.19` `SOL`
- Interact with instruction** `CreateIdempotent` on `Associated Token Account Program`
 - Create `H8fsFQ...xD2FDm` with deposit of `0.00203928` `$0.297521` `SOL` from `8sqy9w...8VT6nP`
- Interact with instruction** `Route` on `Jupiter Aggregator v6`
 - Transfer from `8sqy9w...8VT6nP` to `Raydium Authority V4` for `1` `$145.89` `WSOL`
 - Transfer from `Raydium Authority V4` to `8sqy9w...8VT6nP` for `181,946.995298` `$33.86` `paul`
- Interact with instruction** `CloseAccount` on `Token Program`
 - Close Token Account `2WmGvP...7iwcYC` and redeem to `8sqy9w...8VT6nP`
- Interact with program** `Malicious Extension`
 - Transfer from `8sqy9w...8VT6nP` to `8QYkBc...ob3jhR` for `2.139283093` `$312.11` `SOL`



HACKPROVE



OFFSIDE LABS

2025
HACKPROVE WORLD

These malicious instructions were made to steal funds and take control of the victim's token account while pretending to be part of the normal swap process.

#8 - Malicious Extension: Unknown

Interact With

Input Accounts

Instruction Data

Inner Instructions

Malicious Extension - 5UMucMksJweA1AtgyxrK8DJeBXR3DQGEGRs5Kkq2pZjr

- #1 - Account: 8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP Writable Signer Fee Payer
- #2 - Account: 8QYkBcer7kzCtXJGNazCR6jrJS829aBow12jUob3jhr Writable
- #3 - Account: BKW62NtBeQJkbBdh61WNfpfuT6ai34pU1eX4QPjxQyW
- #4 - Account: System Program Program
- #5 - Account: Token Program Program
- #6 - Account: GPjgCcVt3vH7PiMUBVphMTWUpQfVamRnq5Q4LNDJDk8w Writable
- #7 - Account: FkeG2VyH3Hf9m1okKa7GNFYjZaiwZjtyeVUm1ytTRUNf Writable

#8.1 - System Program: Transfer

Interact With

System Program - 11111111111111111111111111111111

Instruction Data

Source: 8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP Writable Signer Fee Payer
Destination: 8QYkBcer7kzCtXJGNazCR6jrJS829aBow12jUob3jhr Writable
Amount: 2.139283093 SOL

#8.2 - Token Program: SetAuthority

Interact With

Token Program - TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA

Instruction Data

Account: GPjgCcVt3vH7PiMUBVphMTWUpQfVamRnq5Q4LNDJDk8w Writable
Authority Type: accountOwner
Multisig Authority: 8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP Writable Signer Fee Payer
New Authority: 8QYkBcer7kzCtXJGNazCR6jrJS829aBow12jUob3jhr Writable
Signers:

```
[ { item
  0 : "8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP"
}]
```

#8.3 - Token Program: SetAuthority

Interact With

Token Program - TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA

Instruction Data

Account: FkeG2VyH3Hf9m1okKa7GNFYjZaiwZjtyeVUm1ytTRUNf Writable
Authority Type: accountOwner
Multisig Authority: 8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP Writable Signer Fee Payer
New Authority: 8QYkBcer7kzCtXJGNazCR6jrJS829aBow12jUob3jhr Writable
Signers:

```
[ { item
  0 : "8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP"
}]
```

How the Attack Works

Default Safeguard: Transaction Simulation

Key Feature:



- **Purpose:** Predict transaction outcomes before execution
- **How it works:**
 - Unsigned transactions sent to **Blowfish** servers for simulation.
 - Identifies threats such as:
 - Unexpected token transfers.
 - Suspicious smart contracts.
 - Rug pulls and abnormal slippage in DEX transactions.

Simulation Limitations:

- **Sophisticated attacks** can bypass checks using clever programming.
- Case studies reveal gaps in detecting advanced malicious behaviors.

Why Simulations Fail

Case Study – Examining a Simulation Log

Key Question: Why didn't the built-in transaction simulation protect the victims?
--- The simulation failed to reveal the malicious behaviors.

1. In the log, the malicious program did not execute long or consume many compute units:

```
“Program 5UMucMksJweA1AtgyxrK8DJeBXr3DQGEGRs5Kkq2pZjr invoke [1]”,  
“Program 5UMucMksJweA1AtgyxrK8DJeBXr3DQGEGRs5Kkq2pZjr consumed 1106 of 102657 compute  
units”,  
“Program 5UMucMksJweA1AtgyxrK8DJeBXr3DQGEGRs5Kkq2pZjr success”
```



[illegible]

So, what happened in this program [5UMucMksJweA1AtgyxrK8DJeBXR3DQGEGRs5Kkq2pZjr](#)?

Unmasking the Malware: Reverse Engineering

Executable Program Accounts on Solana

- Smart contracts are deployed as executable program accounts
- Solana programs are compiled to BPF bytecode (different from Ethereum's EVM bytecode).

Reverse Engineering Steps

- Dump the malicious program using Solana's CLI:
- Generate an ELF file containing BPF bytecode.
- Use OtterSec's eBPF decompiler to convert bytecode into pseudo C code.

The decompiled pseudo c code is like this:

```
Solana ▾ Linear ▾ Pseudo C ▾

int64_t sub_1000005d8(void* arg1, int64_t arg2, void* arg3, int64_t arg4)

1000006c8      *(uint32_t*)r6 = r1_2;
1000006c8      }
100000678      if (((r8 != 0 && r8 != 1) && r8 != 2) && r8 != 3))
100000678      {
1000006a0          r5_2 = ((char*)r7 + 0xc0);
1000006a8          if (r8 != 4)
1000006a8          {
1000006e0              void* r0_1 = *(uint64_t*)((char*)r7 + 0x30);
100000700              if (*(uint64_t*)r0_1 != 0x4d702b828d0f0a6e)
100000700              {
100000718                  label_100000718:
100000718                      r1_2 = 0x1a;
100000720                      goto label_1000006c8;
100000720              }
100000740              if (*(uint64_t*)((char*)r0_1 + 8) != 0x2a8e185ab7b764da)
100000740              {
100000740                  goto label_100000718;
100000740              }
100000740              if (*(uint64_t*)((char*)r0_1 + 0x10) != 0x7752c6d53f4d424b)
100000760              {
100000760                  goto label_100000718;
100000760              }
100000760              if (*(uint64_t*)((char*)r0_1 + 0x18) != 0xef7a7aa1ace0d9)
100000788              {
100000788                  goto label_100000718;
100000788              }
100000788              if (sub_1000003120(((char*)r7 + 0x60)) == 0)
1000007b8              {
1000007b8                  goto label_100000718;
1000007b8              }
1000007b8          }
```




HACKPROVE



OFFSIDE LABS

2025
HACKPROVE WORLD

Key Findings in Decompiled Code

- **Malicious contract uses:**
 - Early exit conditions to avoid detection during simulation.
 - Account balance checks to trigger hidden “backdoors.”
- **Embedded "magic constants" to target specific accounts.**

Interact With: Malicious Extension - 5UMucMksJweA1AtgyxrK8DJeBXr3DQGEGRs5Kkq2pZjr

Input Accounts:

- #1 - Account: 8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP [Writable] [Signer] [Fee Payer]
- #2 - Account: 8QYkBcer7kzCtXJGNazCR6jrRJS829aBow12jUob3jhR [Writable]
- #3 - Account: BKW62NtBeQJkjbBdh61WNfpfuT6ai34pU1eX4QPjxQyW [Writable]
- #4 - Account: System Program [Program]
- #5 - Account: Token Program [Program]
- #6 - Account: GPjgCcVt3vH7PiMUBVphMTWUpQfVamRnq5Q4LNDJDk8w [Writable]
- #7 - Account: FkeG2VyH3Hf9m1okKa7GNFYjZaiwZjtyeVUm1ytTRUNf [Writable]

```
100007b8  if (sub_100003120(((char*)r7 + 0x60)) == 0)
100007b8  {
100007b8  |   goto label_10000710;
100007b8  }
100007c8  int64_t r0_3 = sub_100003120(r7);
100007d0  int64_t var_1f8;
100007d0  int64_t var_180;
100007d0  void var_120;
100007d0  int32_t var_90;
100007d0  void var_60;
100007d0  void var_30;
100007d0  uint64_t r1_38;
100007d0  if (r0_3 != 0)
100007d0  {
100007d0  |   void* r7_4;
100007d0  |   void* r9_4;
100007d0  |   r7_4 = sub_100003120(r7);
100007d0  |   sub_1000004c8(&var_1f8, r7_4);
100007d0  |   sub_1000004c8(&var_180, r7_4);
100007d0  |   void* r1_35 = &var_120;
100007d0  |   sub_1000004c8(r1_35, r7_4);
100007d0  |   sub_10000edd8(sub_100003120(r7));
10000cc0  }
10000cc0  }
10000ce8  sub_1000004c8(&var_1f8, r7);
10000d10  sub_1000004c8(&var_180, r7);
10000d20  void* r1_35 = &var_120;
10000d30  sub_1000004c8(r1_35, r7);
10000d88  sub_10000edd8(sub_100003120(r7));
```

Dereference a field to read 'lamports'

```
0 @ 100003120 void* r1_1 = *(uint64_t*)((char*)arg1 + 8);
1 @ 100003128 int64_t r2 = *(uint64_t*)((char*)r1_1 + 0x10);
2 @ 100003140 if (r2 > 0x7fffffffffffffff)
2 @ 100003140 {
5 @ 100003158 *(uint64_t*)((char*)r1_1 + 0x
6 @ 100003168 **(uint64_t*)((char*)r1_1 +
7 @ 100003170 *(uint64_t*)((char*)r1_1 + 0x
8 @ 100003178 return;
```

Example of Malicious Behavior













- Checks if an account's lamport balance is zero → exits or performs malicious actions.
- Sandwiches malicious instructions between normal transactions to evade detection.

```

17  /// Account information
18  #[derive(Clone)]
19  #[repr(C)]
20  pub struct AccountInfo<'a> {
21      /// Public key of the account
22      pub key: &'a Pubkey,
23      /// The lamports in the account. Modifiable by programs.
24      pub lamports: Rc<RefCell<&'a mut u64>>,
25      /// The data held in this account. Modifiable by programs.
26      pub data: Rc<RefCell<&'a mut [u8]>>,

```

Hide failed transaction(s) ☐

Signature	Block	Time	Instructions	By	Value (SOL)	Fee (SOL)
 Mi6HxScSZX2PZDPX7P5WaMncuRojTX6YhDjpbToi2...	 284324557	10 days ago	transfer 1+	5hiB6cS3ev9LVZYKYcuNo6mbDMC6WPw1PL44ox1ge8gb	 0.00101	 0.00001
 5krgaq2FTZAp9CT1X1ue4dY7JV2rSVYeNLQFgHjWK...	 284324557	10 days ago	route 1+	8sqy9w334wEPwB86i4VWa6p1megZLSu5m7WDje8VT6nP	 1.002005	 0.002005
 424fD81zbhrG2pBopeknip4QAYjbz9YMFZA5gnxvd28...	 284324557	10 days ago	transfer 1+	5hiB6cS3ev9LVZYKYcuNo6mbDMC6WPw1PL44ox1ge8gb	 0.001005	 0.000005

Show 10 per page < Item 1 to 3 >

What We Discovered

Root Cause:

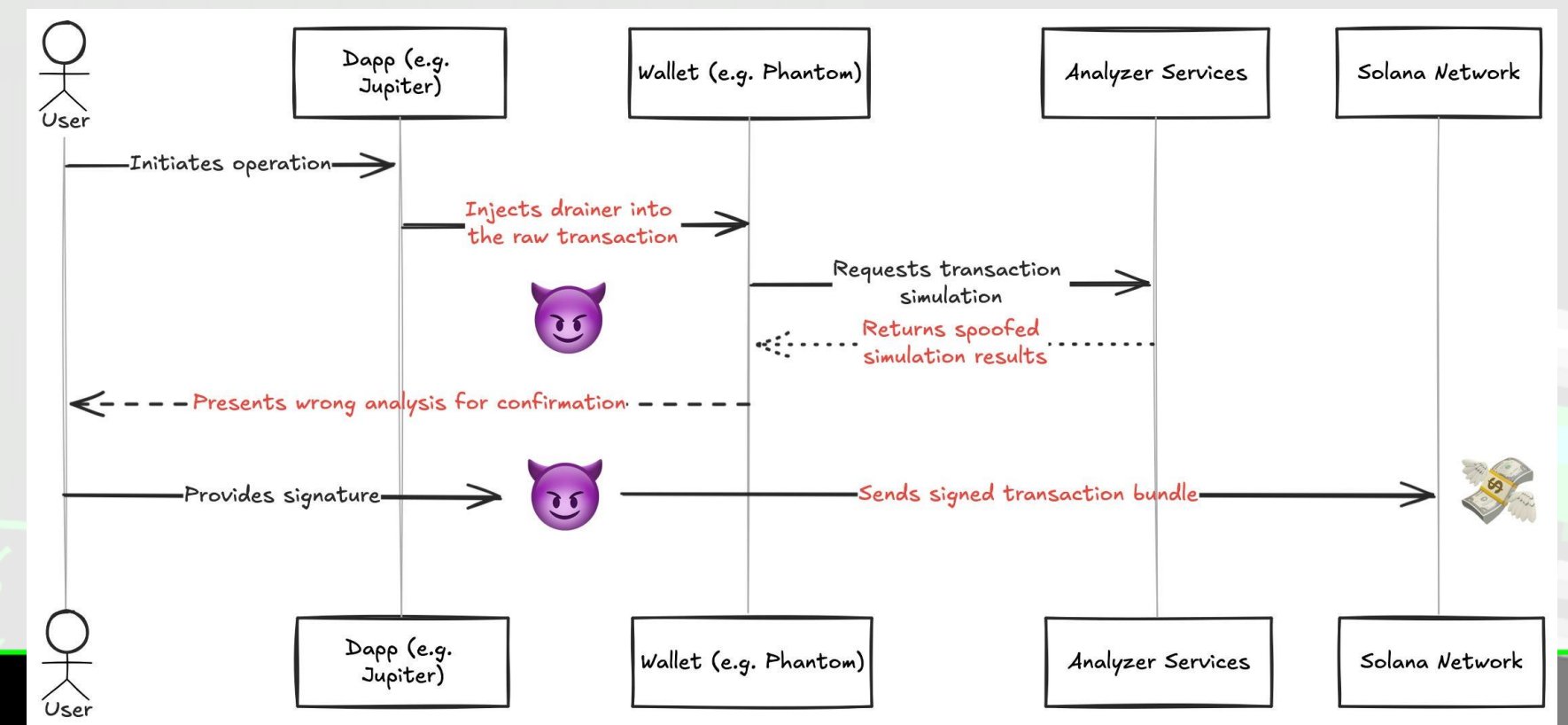
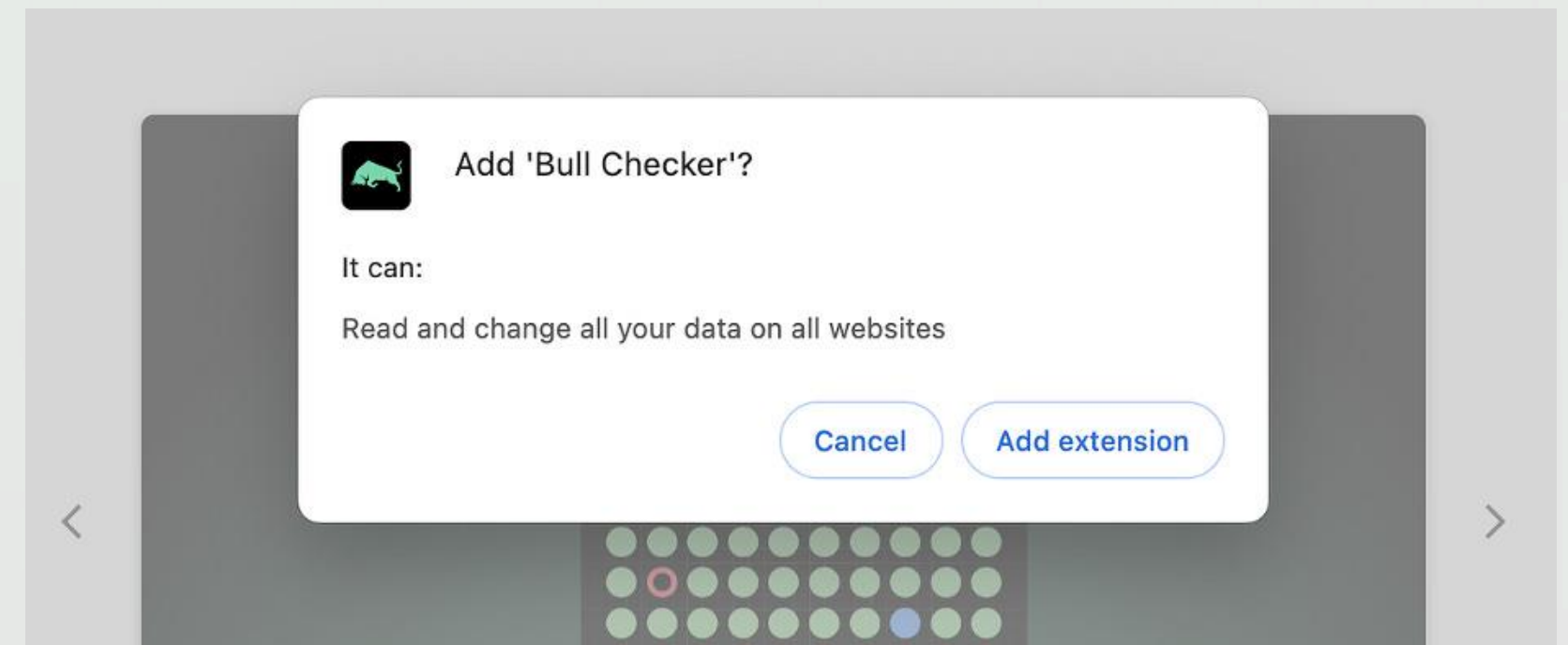
- Malicious browser extension “Bull Checker”.
- Intercepts and modifies wallet adapter functions.

Target Audience:

- Memecoin traders via Reddit and other social channels.

Execution:

- Drainer instructions sandwiched between normal transactions.



How to Stay Safe

Avoid:

- Extensions with excessive permissions (e.g., Bull Checker).
- Unverified tools or recommendations from social media.

Use:

- Wallets with advanced transaction scanning (e.g., Phantom + Blowfish).
- Blowfish's **SafeGuard** feature to prevent simulation spoofing.

Stay Vigilant:

- Always review transactions before signing.
- Verify tools and extensions before use.

THANKS

