Earn Contract Exploit

September 2021 Earn Contract Exploit

- Chain id:
- secret-2
- Date:
- 13/09/2021 3am UTC
- Related issues:
- https://forum.scrt.network/t/earn-contract-exploit-post-mortem/4426
-

Description

Hi everyone,

A couple of weeks ago, a vulnerability in the SecretSwap Earn contracts (also known as the SPY contracts) was discovered and exploited. As far as we know, this is the first Rust/WASM-based contract exploit case, which is interesting in and by itself, and specifically, the first one on Secret Network where the interactions with said contract were all private (more on this below).

At this point, it's important to clarify that Secret Network was in no way exploited, neither were the bridges, and that all funds are safe (with the exception of some minor network-upgrade related cases we are actively resolving, accounting for ~50K). Like in any other major smart-contract chain, including ETH, BSC, and others, smart contract-related vulnerabilities are a potential risk. All we can do is mitigate the risk (and improve on our best practices in doing so), but it cannot be completely eliminated. In this case, the vulnerability, as is described below in a quite technical manner, was not an easy one to uncover and was quite sophisticated.

What Happened

The exploit took advantage of a missing input integrity check in the SPY contracts' (=reward pools' contracts) deposit function to arbitrarily generate rights to withdraw assets from the SPY contracts. We'll go over a normal flow of a deposit to a SPY contract, and then how it was exploited. Keep in mind that there are 5 types of contracts involved here:

Secret Tokens - contracts such as sSCRT, sETH, sXMR, etc. Swap Pairs - which handle trading between pairs of Secret Tokens. LP Tokens - which represent liquidity-provider's portion of the liquidity pools in the Swap Pairs. SPY contracts - which allow users to deposit LP tokens in exchange for accumulating SEFI rewards. The Master Contract - which orchestrates the allocation and minting of SEFI to the SPY contracts. A valid flow of depositing assets in SPY contracts works like this:

Alice has eligible LP tokens. Alice executes a send transaction to the SPY contract, with a deposit inner message:

```

Copy exportconstDepositRewards=async(params:{ secretjs:AsyncSender; recipient:string; address:string; amount:string; fee?:StdFee; }):Promise=>{ consttx=awaitSnip20Send({ msg:'eyJkZXBvc2l0Ijp7fX0K',// '{"deposit":{}}' -> base64 ...params, }); console.log(tx) return'yooyoo'; };

```

The Receive handler of the SPY contract receives the message above with the LP funds amount, which is then parsed and handled as described below.

The integrity of the received assets (Alice's locked assets) relies on the integrity of the LP token; we have to trust the LP token to provide an accurate amount of received tokens i.e. we trust amount that is received in the receive call. The LP contract constructs the receive message with the correct information here: (full code section) 3

```

Copy fn try_add_receiver_api_callback( messages:&mut Vec, storage:&S, recipient:&HumanAddr, msg: Option, sender: HumanAddr, from: HumanAddr, amount: Uint128, )->StdResult<()>{ let receiver_hash=get_receiver_hash(storage,recipient); if letSome(receiver_hash) = receiver_hash { letreceiver_hash=receiver_hash?; let receiver_msg=Snip20ReceiveMsg::new(sender,from,amount,msg); letcallback_msg=receiver_msg.into_cosmos_msg(receiver_hash,recipient.clone())?;

messages.push(callback_msg); } Ok(()) }

```

Upon receive, the SPY contract first needs to get the amount of rewards that the Master contract has allocated to it so far.

This information needs to be collected before the other state changes occur (either a deposit or a redeem). Therefore, the SPY contract calls the Master contract with update_allocation. Since there is no ability to call an external contract function inline from another contract function, the SPY contract also provides a callback message (in this case called hook) that the Master, in turn, will send back to the same SPY contract, to proceed with the deposit operation: Building the hook message in the SPY contract: update_allocation(env, config, Some(to_binary(&LPStakingHookMsg::Deposit { from, amount: Uint128(amount), })?),) Wrapping the hook with update_allocation:

```
Copy fnupdate_allocation(env: Env,config: Config,hook: Option)->StdResult{ Ok(HandleResponse { messages: vec![WasmMsg::Execute { contract_addr: config.master.address, callback_code_hash: config.master.contract_hash, msg: to_binary(&MasterHandleMsg::UpdateAllocation { spy_addr:env.contract.address, spy_hash:env.contract_code_hash, hook, })?, send: vec![], } .into()], log: vec![], data: None, }) }
```

That message is handled by the Master contract, and then the hook is sent back to the SPY contract like this:

```
Copy // Notify to the spy contract on the new allocation messages.push( WasmMsg::Execute { contract_addr:spy_address.clone(), callback_code_hash:spy_hash, msg:to_binary(&LPStakingHandleMsg::NotifyAllocation { amount:Uint128(rewards), hook, })?, send:vec![], } .into(), );
```

The Master contract calls the SPY with notify_allocation which contains hook. The SPY contract proceeds to finish the deposit operation. Note that the amount argument here came originally from the receive function, therefore trusted and should be valid.

Exploit Flow

Bob (attacker) executes a transaction that calls directly to the Master contract with update_allocation with an inner deposit message (as hook). Note that update_allocation requires no permissions and can be called by anyone. The Master contract calls notify_allocation on the SPY contract with the provided hook. Since notify_allocation relies on the data originally coming from a receive message, there are no further integrity checks on amount, and the hook is interpreted as a valid deposit message. In the SPY contract, the deposit_hook function is called with the parameters from the deposit, and increments Bob's balance in the SPY contract: user.locked += amount;

Bob's deposit message is successfully processed and he is given a right to withdraw funds equivalent to the amount he provided i.e. Bob can withdraw assets that were not deposited by him.

Resolution

As soon as the exploit became known, the entire Enigma team, many of the network's validators and other members of the community, such as the Secret Foundation, committee members and leads, bridge operators and many others, came together to devise an action plan. Despite the many difficulties in coordinating so many actors in a decentralized ecosystem across many time zones, we were all able to coordinate a network upgrade that corrected the situation. While not an easy decision, given the funds at stake, this course of action was accepted by the majority of validators in the network.

In addition, to prevent funds from flowing out of the network, we communicated with all bridge operators and exchanges to ensure withdrawals outside of the network are temporarily disabled. This again required the interaction of many parties in the community and outside of it, and we are grateful for everyone who participated and assisted.

In particular, I'd like to also use this opportunity to thank my own team (Enigma), for staying up for 40+ hours while ensuring the vulnerability is found and patched, and for taking a leading part in coordinating all the different parties until a successful resolution.

Currently, everything in the network and all of its applications (including SecretSwap and the ETH/BSC bridges) are back to normal activity. We expect the Monero bridge to activate shortly as well, and we can say that the new Earn contracts, which would require migrating liquidity from the old (vulnerable) Earn contracts, are coming soon (next week at the latest). Given the privacy features of the network, it's not possible to easily withdraw unclaimed SEFI from the old rewards contracts. This means that quite a lot of SEFI will in fact be burned. In addition, no new SEFI has been minted in the past few weeks, reducing the effective SEFI supply. Some of that supply will be reintroduced as compensation for liquidity providers who stayed and will migrate to the new contracts, in the form of accelerated rewards in the first few days.

Conclusion And Next Steps

There was a very sophisticated vulnerability in a Secret Contract. The network was never compromised, nor were any of the bridges. Nevertheless, in a collective action, the community came together and performed a network upgrade that ensured funds' safety.

At this point, everything is back to normal operation, with the exception of the new, patched, Earn contracts (and by extension – governance) that will be re-introduced in the next week or so. These will require users to migrate, and will initially over-compensate LP’s unclaimed rewards loss. At the same time, a substantial amount of SEFI were effectively burned, thus reducing its overall supply.

Hope this clarifies the situation. We would like to remind everyone that we have a very generous bug/exploit bounty program, and that we always recommend taking a responsible disclosure course of action (we will make it worthwhile). For those who are interested please e-mail us at info (at) enigma (dot) co.

Best, Guy Enigma CEO