Umbra Smart Contracts

Consensys Diligence

Date	March 2021
Auditors	Nicholas Ward

1 Executive Summary

This report presents the results of our engagement with ScopeLift to review the Umbra Protocol smart contracts.

The review was conducted by Nicholas Ward between March 22nd and March 26th, 2021.

2 Scope

The review focused on the commit hash `fa2e17367d66a85f20c77299ded5942d9ab64fe0`. A cursory review of the ENS Resolver contract for stealth keys was also performed at commit hash `2d7795082308d303eb23c66490579a5b21a1bac9`. The list of files in scope can be found in the Appendix.

Notably, all off-chain libraries and cryptography used in the Umbra Protocol were explicitly excluded from the scope, as were all Gas Station Network integrations.

3 System Overview

This section should not serve as an alternative for documentation, which ScopeLift provides in the project README and in the code itself. In brief:

Umbra allows semi-private payments between two parties using “stealth addresses”. The `Umbra` contract forwards ETH payments to these stealth addresses, emitting an encrypted message provided by the sender. This encrypted message can be used by the recipient to identify payments intended for them and to recover the forwarded funds.

For token payments, the `Umbra` contract accepts tokens from the sender and allows the recipient to withdraw the tokens. Optionally, the recipient of a token payment can withdraw their tokens via a signed message submitted to the contract by a “sponsor”. Token payment withdrawals can also include a call to a hook receiver contract, which can be used forward deposits into another privacy solution, interact with other on-chain protocols, or take some arbitrary action upon receipt of a payment. Crucially, nonce-based signature replay protection is *NOT* provided by the `Umbra` contract under the expectation that stealth addresses are used only once.

Users should note that the privacy gained from use of the Umbra Protocol can be directly impacted by their own privacy hygience as well as that of their counterparties.

4 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 Reuse of `CHAINID` from contract deployment Minor ✓ Fixed

Resolution
This is addressed in ScopeLift/umbra-protocol@`7cfdc81`.

Description

The internal function `_validateWithdrawSignature()` is used to check whether a sponsored token withdrawal is approved by the owner of the stealth address that received the tokens. Among other data, the chain ID is signed over to prevent replay of signatures on other EVM-compatible chains.

contracts/contracts/Umbra.sol:L307-L329

```
function _validateWithdrawSignature(
  address _stealthAddr,
  address _acceptor,
  address _tokenAddr,
  address _sponsor,
  uint256 _sponsorFee,
  IUmbraHookReceiver _hook,
  bytes memory _data,
  uint8 _v,
  bytes32 _r,
  bytes32 _s
) internal view {
  bytes32 _digest =
    keccak256(
      abi.encodePacked(
        "\x19Ethereum Signed Message:\n32",
        keccak256(abi.encode(chainId, version, _acceptor, _tokenAddr, _sponsor, _sponsorFee, address(_hook), _data))
      )
    );

  address _recoveredAddress = ecrecover(_digest, _v, _r, _s);
  require(_recoveredAddress != address(0) && _recoveredAddress == _stealthAddr, "Umbra: Invalid Signature");
}
```

However, this chain ID is set as an immutable value in the contract constructor. In the case of a future contentious hard fork of the Ethereum network, the same `Umbra` contract would exist on both of the resulting chains. One of these two chains would be expected to change the network's chain ID, but the `Umbra` contracts would not be aware of this change. As a result, signatures to the `Umbra` contract on either chain would be replayable on the other chain.

This is a common pattern in contracts that implement EIP-712 signatures. Presumably, the motivation in most cases for committing to the chain ID at deployment time is to avoid recomputing the EIP-712 domain separator for every signature verification. In this case, the chain ID is a direct input to the generation of the signed digest, so this should not be a concern.

Recommendation

Replace the use of the `chainId` immutable value with the `CHAINID` opcode in `_validateWithdrawSignature()`. Note that `CHAINID` is only available using Solidity's inline assembly, so this would need to be accessed in the same way as it is currently accessed in the contract's constructor:

contracts/contracts/Umbra.sol:L68-L72

```
uint256 _chainId;

assembly {
  _chainId := chainid()
}
```

5 Recommendations

5.1 Use separate mappings for keys in StealthKeyResolver

Description

The `StealthKeyResolver` currently stores keys in a mapping `bytes32 => uint256 => uint256` that maps `nodes => prefixes => keys`. The prefixes are offset in the `setStealthKeys()` function to differentiate between viewing public keys and spending public keys, and these offsets are reversed in the `stealthKeys()` view function.

contracts/profiles/StealthKeyResolver.sol:L37-L56

```
function setStealthKeys(bytes32 node, uint256 spendingPubKeyPrefix, uint256 spendingPubKey, uint256 viewingPubKeyPrefix, uint256 viewingPubKey) external authorised(node)
    require(
        (spendingPubKeyPrefix == 2 || spendingPubKeyPrefix == 3) &&
        (viewingPubKeyPrefix == 2 || viewingPubKeyPrefix == 3),
        "StealthKeyResolver: Invalid Prefix"
    );

    emit StealthKeyChanged(node, spendingPubKeyPrefix, spendingPubKey, viewingPubKeyPrefix, viewingPubKey);

    // Shift the spending key prefix down by 2, making it the appropriate index of 0 or 1
    spendingPubKeyPrefix -= 2;

    // Ensure the opposite prefix indices are empty
    delete _stealthKeys[node][1 - spendingPubKeyPrefix];
    delete _stealthKeys[node][5 - viewingPubKeyPrefix];

    // Set the appropriate indices to the new key values
    _stealthKeys[node][spendingPubKeyPrefix] = spendingPubKey;
    _stealthKeys[node][viewingPubKeyPrefix] = viewingPubKey;
}
```

This manual adjustment of prefixes adds complexity to an otherwise simple function. To avoid this, consider splitting this into two separate mappings – one for viewing keys and one for spending keys. For clarity, also specify the visibility of these mappings explicitly.

5.2 Document potential edge cases for hook receiver contracts

Description

The functions `withdrawTokenAndCall()` and `withdrawTokenAndCallOnBehalf()` make a call to a hook contract designated by the owner of the withdrawing stealth address.

contracts/contracts/Umbra.sol:L289-L291

```
if (address(_hook) != address(0)) {
  _hook.tokensWithdrawn(_withdrawalAmount, _stealthAddr, _acceptor, _tokenAddr, _sponsor, _sponsorFee, _data);
}
```

There are very few constraints on the parameters to these calls in the `Umbra` contract itself. Anyone can force a call to a hook contract by transferring a small amount of tokens to an address that they control and withdrawing these tokens, passing the target address as the hook receiver. Developers of these `UmbraHookReceiver` contracts should be sure to validate both the caller of the `tokensWithdrawn()` function and the function parameters. There are a number of possible edge cases that should be handled when relevant. These include, but are not limited to, the following:

- The `_amount` may not have been transferred to the hook receiver itself.
- All four addresses passed to `tokensWithdrawn()` could be the same. Most of these address parameters could also be any arbitrary address. This includes the token contract address, the address of the hook receiver, or the address of the `Umbra` contract itself.
- The token received may be valueless.
- The token received may be malicious. The only requirements are that the token contract address contains code and accepts calls to the ERC20 methods `transfer()` and `transferFrom()`.

While it is difficult to determine a feasible exploit without knowledge of what hook receiver contracts may do in the future, a slightly contrived example follows.

Suppose a user builds a hook receiver contract that accepts an arbitrary token, `TOK`, and immediately provides liquidity to the `ETH-TOK` Uniswap pair when `tokensWithdrawn()` is called by the `Umbra` contract. An attacker could create a malicious token that can not be transferred out of its own Uniswap Pair contract and force a call to the hook receiver contract from `Umbra`. The hook receiver would be able to provide liquidity to the pool but would be unable to remove it, losing any ETH that was provided.

5.3 Document token behavior restrictions

As with any protocol that interacts with arbitrary ERC20 tokens, it is important to clearly document which tokens are supported. Often this is best done by providing a specification for the behavior of the expected ERC20 tokens and only relaxing this specification after careful review of a particular class of tokens and their interactions with the protocol.

In the absence of this, the following is a necessarily incomplete list of some known deviations from “normal” ERC20 behavior that should be explicitly noted as *NOT* supported by the Umbra Protocol:

- Deflationary or fee-on-transfer tokens: These are tokens in which the balance of the recipient of a transfer may not be increased by the amount of the transfer. There may also be some alternative mechanism by which balances are unexpectedly decreased. While these tokens can be successfully sent via the `sendToken()` function, the internal accounting of the `Umbra` contract will be out of sync with the balance as recorded in the token contract, resulting in loss of funds.
- Inflationary tokens: The opposite of deflationary tokens. The `Umbra` contract provides no mechanism for claiming positive balance adjustments.
- Rebasing tokens: A combination of the above cases, these are tokens in which an account’s balance increases or decreases along with expansions or contractions in supply. The contract provides no mechanism to update its internal accounting in response to these unexpected balance adjustments, and funds may be lost as a result.

5.4 Add an address parameter to withdrawal signatures ✓ Fixed

Resolution

This is addressed in ScopeLift/umbra-protocol@`d6e4235`, which replaces the `version` parameter with `address(this)` in the signature encoding.

Description

As discussed above, the `_validateWithdrawSignature()` function checks the signer of a digest consisting of the keccak-256 hash of the following preimage:

```
abi.encodePacked(
    "\x19Ethereum Signed Message:\n32",
    keccak256(abi.encode(chainId, version, _acceptor, _tokenAddr, _sponsor, _sponsorFee, address(_hook), _data))
)
```

Consider adding the address of the contract itself to this signed message. Currently, it is possible to deploy any number of contracts with the same `version` to the same chain, and signatures would be replayable across all of these contracts. While users are likely to only have balances for the same stealth address in one of these contracts, adding an address parameter provides some additional replay protection. Because the contract can not be self-destructed, a given address can only ever contain a single version of the Umbra contract.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
Umbra.sol	bb9fc1f58c7c1246aa8533161153533392042 0b8
IUmbraHookReceiver.sol	f8c1835a62a82c9129318aa05f77cee6e4176d93
StealthKeyResolver.sol	f27bf5e6c29bfd3b516352ca15d0704c3899b65c

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment

advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.