Competitive Security Assessment

Tonka_Finance_AMM

Feb 5th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

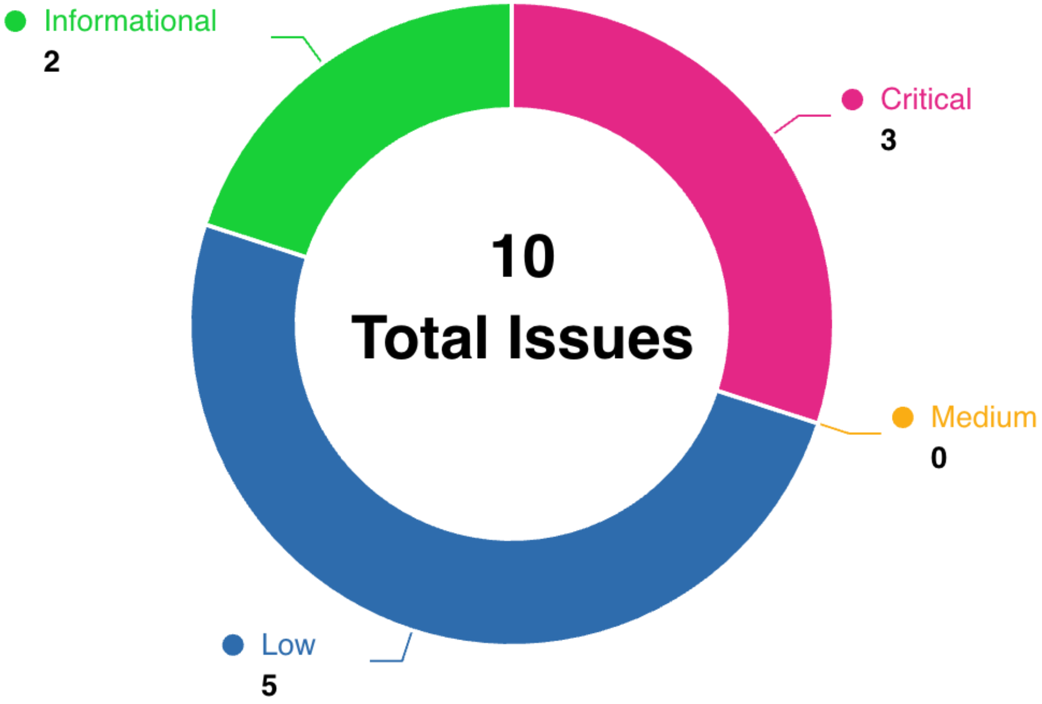
Project Detail

Project Name	Tonka_Finance_AMM
Platform & Language	Solidity
Codebase	• https://github.com/Tonka-Finance/Tonka-Contracts • audit commit - bcd5b93e7005005dbabda1f95511f441a48cf8dc • final commit - b5898856a505df9915214690fe95822a4037ac57
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/Swap/EsSwapPair.sol	**dc11bc5c076859d19594ffb1a3aa6d8d3aa62a5533a342423a394135588caf72**
./contracts/Swap/EsSwapERC20.sol	**974322c8866a8119c5289a6be365f83d492b361721a1e3499789d951cad8992a**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
TFA-1	Incorrect calculation of `balance1Adjusted`	Logical	Critical	Fixed	ginlee, Yaodao
TFA-2	Incorrect calculation in function `getAmountIn()`	Logical	Critical	Fixed	ginlee, Yaodao
TFA-3	Inconsistency between `amount` and `reseve`	Logical	Critical	Fixed	Yaodao
TFA-4	Recommendation use of `safeTransferFrom()`	Logical	Low	Fixed	ravikiran_web3, Yaodao

TFA-5	**Lack of checking if the `token0` is not an `estoken`**	**Logical**	**Low**	**Fixed**	**danielt**
TFA-6	**Ownership change should use two-step process**	**Logical**	**Low**	**Fixed**	**Yaodao, n16h7m4r3**
TFA-7	**Missing Zero Address Check**	**Logical**	**Low**	**Fixed**	**danielt**
TFA-8	**The `EsSwapPair.initialize()` function can be called multiple times by the owner**	**Logical**	**Low**	**Fixed**	**0xac, n16h7m4r3**
TFA-9	**Unused parameter**	**Code Style**	**Informational**	**Fixed**	**Yaodao**
TFA-10	**Switching between 1, 2 instead of 0, 1 is more gas efficient**	**Gas Optimization**	**Informational**	**Fixed**	**n16h7m4r3**

TFA-1:Incorrect calculation of `balance1Adjusted`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	ginlee, Yaodao

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L229

```
229:uint balance1Adjusted = balance1 * 1000 - amount1In - 3;
```

Description

ginlee :

```
uint balance1Adjusted = balance1 * 1000 - amount1In - 3;
```

The line of code uint balance1Adjusted = balance1 * 1000 - amount1In - 3; appears to be intended to adjust balance1 by multiplying it by a factor (1000, which is a common scaling factor in financial calculations to avoid floating points) and then adjusting it based on the input amounts amount0In and amount1In.

The adjusted balance for balance0 is correctly calculated by uint balance0Adjusted = balance0 * 1000 - amount0In * 3; where amount0In is multiplied by 3. However, a similar operation for balance1Adjusted incorrectly subtracts 3 from amount1In, which seems to be a typographical error.

Yaodao : The calculation of `balance1Adjusted` should be `balance1 * 1000 - amount1In * 3` instead of `balance1 * 1000 - amount1In - 3`.

```
            uint balance0Adjusted = balance0 * 1000 - amount0In * 3;
            uint balance1Adjusted = balance1 * 1000 - amount1In - 3;
```

Recommendation

ginlee : The correct calculation should multiply amount1In by 3 to maintain consistency with balance0Adjusted and the apparent intent of the function. The corrected line of code should be:

```
uint balance1Adjusted = balance1 * 1000 - amount1In * 3;
```

Yaodao : Recommend updating the calculation of `balance1Adjusted`.

Client Response

Fixed: - to * commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/33f9ee429811aca51c04cef7e7c69b5f05ac65d0

TFA-2:Incorrect calculation in function `getAmountIn()`

Category	Severity	Client Response	Contributor
Logical	**Critical**	**Fixed**	**ginlee, Yaodao**

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L316-L322
- code/contracts/Swap/EsSwapPair.sol#L320

```
316:function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public pure returns (uint amountIn) {
317:        require(amountOut > 0, "EsSwapLibrary: INSUFFICIENT_OUTPUT_AMOUNT");
318:        require(reserveIn > 0 && reserveOut > 0, "EsSwapLibrary: INSUFFICIENT_LIQUIDITY");
319:        uint numerator = reserveIn * amountOut * 1000;
320:        uint denominator = reserveOut - amountOut * 997;
321:        amountIn = (numerator / denominator) + 1;
322:    }

320:uint denominator = reserveOut - amountOut * 997;
```

Description

ginlee :

```
  uint denominator = reserveOut - amountOut * 997;
```

For the same implementation in Uniswap V2, denominator is calculated as reserveOut.sub(amountOut).mul(997), instead of "reserveOut - amountOut * 997", it is supposed to be calculated as (reserveOut - amountOut) * 997, by changing the sequence of calculation, a wrong denominator will lead to wrong amountIn, which will result in loss of fund in the protocol

Yaodao : The contract uses the 0.8.0+ solidity version and updates the math calculation.

According to the codes in UNISWAP, the `denominator` in the `getAmountIn()` should be `(reserveOut - amountOut) * 997`. However, in the contract `EsSwapPair` updated to be `reserveOut - amountOut * 997`.

```
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public pure returns (uint
amountIn) {
        require(amountOut > 0, "EsSwapLibrary: INSUFFICIENT_OUTPUT_AMOUNT");
        require(reserveIn > 0 && reserveOut > 0, "EsSwapLibrary: INSUFFICIENT_LIQUIDITY");
        uint numerator = reserveIn * amountOut * 1000;
        uint denominator = reserveOut - amountOut * 997;
        amountIn = (numerator / denominator) + 1;
    }
```

Reference: https://github.com/Uniswap/v2-periphery/blob/master/contracts/libraries/UniswapV2Library.sol#L53C14-L59

Recommendation

ginlee : Modify the calculation of denominator in the getAmountIn function to use (reserveOut - amountOut) * 997 instead

```
  uint denominator = (reserveOut - amountOut) * 997;
```

Yaodao : Recommend fixing the incorrect calculation.

Client Response

Fixed: add() commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/70eb2238e51b7cf1c6247a639b461b61ca7093ba

TFA-3:Inconsistency between `amount` and `reseve`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Yaodao

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L270-L283
- code/contracts/Swap/EsSwapPair.sol#L285-L298

```
function swapExactTokensForEsTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountOut) {
        (uint reseveIn, uint reserveOut, ) = getReserves();
        amountOut = getAmountOut(amountIn, reseveIn, reserveOut);
        require(amountOut >= amountOutMin, "EsSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT");
        IERC20(token1).transferFrom(msg.sender, to, amountIn);

        _swap(amountOut, 0, to);
    }

function swapTokensForExactEsTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountIn) {
        (uint reseveIn, uint reserveOut, ) = getReserves();
        amountIn = getAmountIn(amountOut, reseveIn, reserveOut);
        require(amountIn <= amountInMax, "EsSwapRouter: EXCESSIVE_INPUT_AMOUNT");
        IERC20(token1).transferFrom(msg.sender, to, amountIn);

        _swap(amountOut, 0, to);
    }
```

Description

Yaodao : The functions `swapExactTokensForEsTokens()` and `swapTokensForExactEsTokens()` are used to swap `token1` to `EsToken`.

However, the amount and reseve used to calculate are inconsistent.

For example, for `swapExactTokensForEsTokens()`

```
    function swapExactTokensForEsTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external ensure(deadline) returns (uint amountOut) {
        (uint reseveIn, uint reserveOut, ) = getReserves();
        amountOut = getAmountOut(amountIn, reseveIn, reserveOut);
        require(amountOut >= amountOutMin, "EsSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT");
        IERC20(token1).transferFrom(msg.sender, to, amountIn);

        _swap(amountOut, 0, to);
    }
```

The `reseveIn` get by `getReserves()` is for `EsToken`, and the `amountIn` is for `token1`. In the call of `getAmountOut()`, the `amountIn` and `reseveIn` are both used for `token1`.

As a result, the calculation of `amountOut` is incorrect.

Recommendation

Yaodao : Recommend using correct parameters to call `getAmountOut()`.

For example:

```
        amountOut = getAmountOut(amountIn, reserveOut, reserveIn);
```

Client Response

Fixed: to (uint reserveOut, uint reserveIn,) = getReserves() commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/3f5179eefe80eb354a9b4d87386f8b72c8b7ffba

TFA-4:Recommendation use of `safeTransferFrom()`

Category	Severity	Client Response	Contributor
Logical	**Low**	**Fixed**	**ravikiran_web3, Yaodao**

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L108-L121
- code/contracts/Swap/EsSwapPair.sol#L280
- code/contracts/Swap/EsSwapPair.sol#L295

```
108:function removeLiquidity(
109:        uint liquidity,
110:        uint amountAMin,
111:        uint amountBMin,
112:        address to,
113:        uint deadline
114:    ) public ensure(deadline) returns (uint amountA, uint amountB) {
115:        (address tokenA, address tokenB) = (token0, token1);
116:        transferFrom(msg.sender, address(this), liquidity); // send liquidity to pair
117:        (amountA, amountB) = burn(to);
118:
119:        require(amountA >= amountAMin, "EsSwapRouter: INSUFFICIENT_A_AMOUNT");
120:        require(amountB >= amountBMin, "EsSwapRouter: INSUFFICIENT_B_AMOUNT");
121:    }

280:IERC20(token1).transferFrom(msg.sender, to, amountIn);

295:IERC20(token1).transferFrom(msg.sender, to, amountIn);
```

Description

ravikiran_web3 : In the removeLiquidity(), the liquidity is transferred from caller's account to the contract address.

To perform the above transfer, the contract calls transferFrom() of ERC20 base class. It is important that the return value of transferFrom is check to confirm that the transfer went find successfully. This is a norm and best practices when moving ERC20 tokens.

In the removeLiquidity(), the return value for transferFrom() was ignored, meaning failures of the underlying transactions are ignored and other processing in the removeLiquidity() function continues to execute. This will result in inaccurate accounting of tokens.

```
function removeLiquidity(
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public ensure(deadline) returns (uint amountA, uint amountB) {
        (address tokenA, address tokenB) = (token0, token1);
  ===>      transferFrom(msg.sender, address(this), liquidity); // send liquidity to pair
        (amountA, amountB) = burn(to);

        require(amountA >= amountAMin, "EsSwapRouter: INSUFFICIENT_A_AMOUNT");
        require(amountB >= amountBMin, "EsSwapRouter: INSUFFICIENT_B_AMOUNT");
    }
```

Yaodao : In the functions `swapExactTokensForEsTokens()` and `swapTokensForExactEsTokens()`, the function `transferFrom()` is used to transfer `token1`.

It is recommended to use `safeTransferFrom()` to transfer `token1`.

Recommendation

ravikiran_web3 : It is recommended to check the return value for transferFrom().

As an alternative, using safeTransfer functions from openzepplien could be used.

```
bool sent =   transferFrom(msg.sender, address(this), liquidity);
require(sent, "Token transfer failed");
```

Yaodao : Recommend using `safeTransferFrom()`.

Client Response

Fixed: use SafeTransferFrom commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/d5680426dfe9b661625435500a3ad3400953d541

TFA-5:Lack of checking if the `token0` is not an `estoken`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	danielt

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L56-L59
- code/contracts/Swap/EsSwapPair.sol#L337-L342

```
56:function initialize(address _token0, address _token1) external onlyOwner {
57:        token0 = _token0;
58:        token1 = _token1;
59:    }

337:function _transferFromEsToken(address _from, address _to, uint256 _amount) internal {
338:        address _token0 = token0;
339:        // when failing to burn or mint, token0 should revert
340:        IEsTokaToken(_token0).burn(_from, _amount);
341:        IEsTokaToken(_token0).mint(_to, _amount);
342:    }
```

Description

danielt : The `EsSwapPair` contract manages pairs composed of the `estoken`, which can be minted and burnt by the `EsSwapPair` contract. If the `token0` is not an `estoken`, the `_transferFromEsToken` function will revert. The point is that the check, to validate if the `token0` is an `estoken`, is better to be completed in the `initialize` function, in order to prevent the following error. Example validation:

- burns 0 `estoken` and mint 0 `estoken` in the `initialize` function.

Recommendation

danielt : Recommend checking if the `token0` is an `estoken` in the `initialize` function to prevent the future error.

Client Response

Fixed: add check for EsToken commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/09d30d1f070af62af33e6b086c87c3b7aa88f523

TFA-6: Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Logical	**Low**	**Fixed**	**Yaodao, n16h7m4r3**

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L14-L16
- code/contracts/Swap/EsSwapPair.sol#L16

```
14:import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
15:
16:contract EsSwapPair is IEsSwapPair, Ownable, EsSwapERC20 {

16:contract EsSwapPair is IEsSwapPair, Ownable, EsSwapERC20 {
```

Description

Yaodao : The contract uses the openzeppelin's `Ownable` contract to manage owners.

It is possible that the `onlyOnwer` role mistakenly transfers ownership to the wrong address, resulting in the loss of the `onlyOnwer` role.

n16h7m4r3 : `Ownable2Step` prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

Recommendation

Yaodao : Recommend implementing a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOnwership()` function for the transfer of the ownership to fully succeed.

n16h7m4r3 : Use Ownable2Step instead of Ownable.

Client Response

Fixed: use Ownable2Step commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/b5898856a505df9915214690fe95822a4037ac57

TFA-7:Missing Zero Address Check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	danielt

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L56-L59

```
56:function initialize(address _token0, address _token1) external onlyOwner {
57:        token0 = _token0;
58:        token1 = _token1;
59:    }
```

Description

danielt : The `initialize` function should be executed and only executed once, which can be done by using the `initializer` modifier of Openzeppelin's `Initializable` contract or adding checks that ensure `token0` and `token1` are zero addresses before initializing. Example:

- what if the private key of the owner is leakage due to the owner being attacked by the social engineering attacks, then the hacker could call the `initialize` function again lead to the DoS for the `EsSwapPair`, and permanently lock the pair's assets.

Recommendation

danielt : Recommend applying the `initializer` modifier of Openzeppelin's `Initializable` contract to the `initialize` function or adding checks that ensure `token0` and `token1` are zero addresses before initializing.

Client Response

Fixed: initialize token0 and token1 by constructor commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/52859091c3d1b32755e3b1d5477f7fb49d9af8d1

TFA-8:The `EsSwapPair.initialize()` function can be called multiple times by the owner

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xac, n16h7m4r3

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L55-L59
- code/contracts/Swap/EsSwapPair.sol#L56-L59

```
55:// called once by the factory at time of deployment
56:    function initialize(address _token0, address _token1) external onlyOwner {
57:        token0 = _token0;
58:        token1 = _token1;
59:    }

56:function initialize(address _token0, address _token1) external onlyOwner {
57:        token0 = _token0;
58:        token1 = _token1;
59:    }
```

Description

0xac : The initialize function can be called multiple times by the owner instead of just once. The owner can replace token0 and token1 in the pair, putting the funds in the pair in an unsafe situation. For example, replace the logic of the token0 and token1 contracts and transfer the funds in the pair.

n16h7m4r3 : The token pair's contract addresses can be re-initialized by the factory, If the token pair is reinitialized, any existing data associated with the previous token pair would not be taken into consideration. This could affect the accuracy and integrity of the smart contract's functionality.

Recommendation

0xac : It is recommended to use an initializer to ensure that this function is only called once.
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol

n16h7m4r3 : Consider adding checks to ensure that the function `initialize()` can only be executed once.

Client Response

Fixed: initialize token0 and token1 by constructor commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/52859091c3d1b32755e3b1d5477f7fb49d9af8d1

TFA-9:Unused parameter

Category	Severity	Client Response	Contributor
Code Style	**Informational**	**Fixed**	**Yaodao**

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L273
- code/contracts/Swap/EsSwapPair.sol#L288

```
273:address[] calldata path,

288:address[] calldata path,
```

Description

Yaodao : The parameter `path` is declared in the functions `swapExactTokensForEsTokens()` and `swapTokensForExactEsTokens()` but never be used.

Recommendation

Yaodao : Recommend removing the useless parameter.

Client Response

Fixed: remove unused paras commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/a22f35b4c0fb348c611335ebfbb822b45fcd8912

TFA-10:Switching between 1, 2 instead of 0, 1 is more gas efficient

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	n16h7m4r3

Code Reference

- code/contracts/Swap/EsSwapPair.sol#L35-L40

```
modifier lock() {
        require(unlocked == 1, "EsSwap: LOCKED");
        unlocked = 0;
        _;
        unlocked = 1;
    }
```

Description

n16h7m4r3 : `SSTORE` from 0 to 1 (or any non-zero value), the cost is 20000; `SSTORE` from 1 to 2 (or any other non-zero value), the cost is 5000.

By storing the original value once again, a refund is triggered (https://eips.ethereum.org/EIPS/eip-2200).

Since refunds are capped to a percentage of the total transaction's gas, it is best to keep them low, to increase the likelihood of the full refund coming into effect.

Recommendation

n16h7m4r3 : Replace 0,1 with 1, 2 to optimize the gas usage.

Client Response

Fixed: switch between 1, 2 for gas efficiency commit: https://github.com/Tonka-Finance/Tonka-Contracts/commit/afe77b47d7157939b36638a064cb8fd7578e41c5

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.