Competitive Security Assessment

Star Name Service

Jan 20th, 2023

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	Star Name Service
Platform & Language	Aptos Move
Codebase	Code Provided offline
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

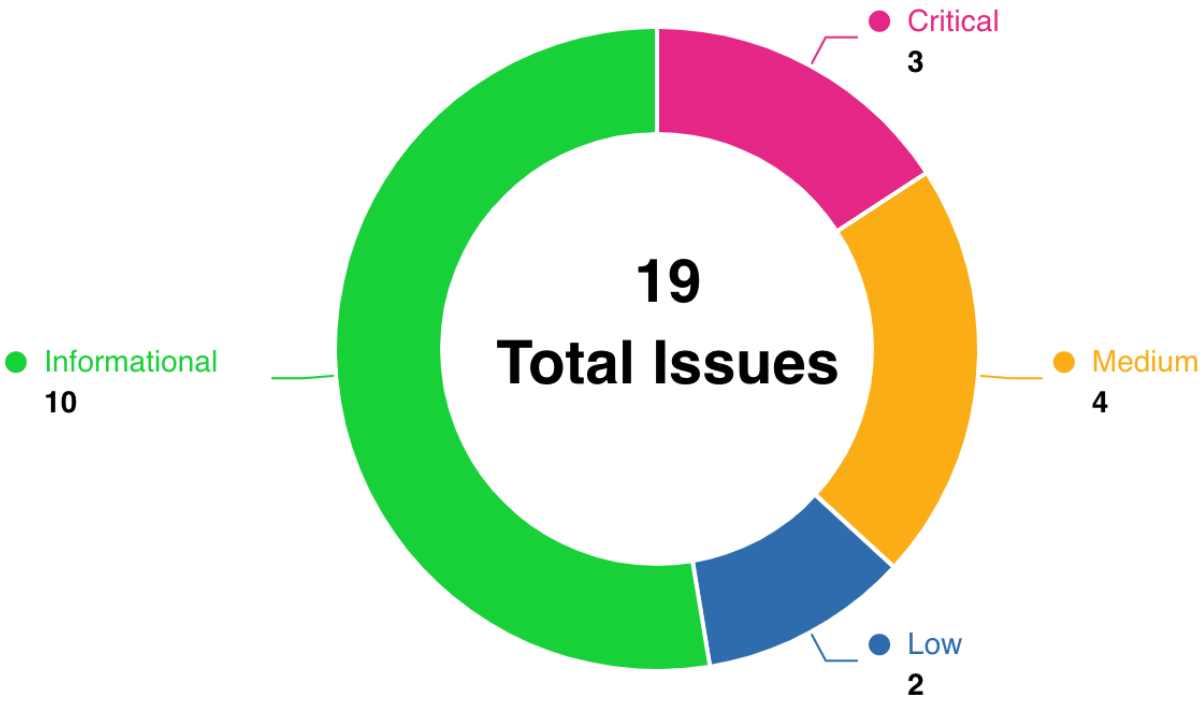
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	3	0	0	3	0	0
Medium	4	0	1	3	0	0
Low	2	0	1	1	0	0
Informational	10	0	1	9	0	0

Audit Scope

File	Commit Hash
sns_nameservice/sources/Airdrop.move	2afba22ffd0d5c62b4d00e7609e6711a8da2d1cbea0e1d1865abbc14be985837
sns_nameservice/sources/Resovler.move	8b897b183e14dd17f873cbb279d8b61f5ace0a9de8884b1421e275acd20ed6c9
sns_nameservice/sources/StarNameService.move	6cc2d4d08c892cf1ed7da7290146f6ad2ca0e56ca321abb42dbe5136bb49fcfb

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
STA-1	Code Style in `Airdrop` contract `consume` function	Code Style	Informational	Fixed	yekong
STA-2	Design flaw of StarNameService NFT and NameServices resource	Logical	Critical	Fixed	Kong7ych3
STA-3	Expired SNS NFTs are not processed	Logical	Medium	Fixed	Kong7ych3
STA-4	Index out of bound.	Logical	Critical	Fixed	jayphbee
STA-5	Input parameter `to` has not been used by the function `airdrop_mint`	Logical	Medium	Fixed	0xac, Kong7ych3

STA-6	Lack of limiting the length of `prefix` in function `check_prefix_length_isLegal`	Logical	Low	Fixed	0xac, alansh, BradMoonUESTC, Hupixiong3
STA-7	No need for ex_expire_time in `StarNameService` contract `renew_name_script` function	Gas Optimization	Informational	Fixed	alansh
STA-8	Redundant codes in multiple functions of `StarNameService` contract	Gas Optimization	Informational	Fixed	yekong, alansh
STA-9	Redundant initialize_token_store operation	Gas Optimization	Informational	Fixed	Kong7ych3
STA-10	Resource operation type usage error	Code Style	Informational	Fixed	Kong7ych3
STA-11	Risk of registered domain being front-run	Race condition	Informational	Acknowledged	Kong7ych3
STA-12	The relationship of two parameters has not been checked	Logical	Medium	Acknowledged	0xac
STA-13	Typographical errors in multiple contracts	Code Style	Informational	Fixed	Hupixiong3, alansh
STA-14	User can free from fee by registering a very short time domain	Logical	Low	Acknowledged	jayphbee, 0xac
STA-15	Wrong main_uri append	Logical	Medium	Fixed	Kong7ych3
STA-16	`Airdrop.airdrop()` incorrect amount logic	Logic	Critical	Fixed	jayphbee, yekong, 0xac, Kong7ych3, BradMoonUESTC, Hupixiong3
STA-17	`StarNameService.LengthRange` is empty and not used	Code Style	Informational	Fixed	BradMoonUESTC
STA-18	resource_account variable is unused in `StarNameService.move` contract `initialize_name_services` function	Code Style	Informational	Fixed	alansh

STA-19	**uri parameter is not used in `StarNameService.move` contract `create_name_script` function**	**Code Style**	**Informational**	**Fixed**	**alansh**

STA-1:Code Style in `Airdrop` contract `consume` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/Airdrop.move#L61	Fixed	yekong

Code

```
61:         assert(balance - amount >= 0,NO_SUFFICIENT_FNUDS);
```

Description

yekong : The usage of assert is not standardized

Recommendation

yekong : Assert is a builtin, macro-like operation provided by the Move compiler.Since the operation is a macro, it must be invoked with the !. reference link:https://move-language.github.io/move/abort-and-assert.html

Client Response

Fixed.

STA-2:Design flaw of StarNameService NFT and NameServices resource

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/sources/StarNameService.move#L227-L241	Fixed	Kong7ych3

Code

```
227:        if (table::contains(&name_services.expire_data, name)) {
228:            let expire_time = table::borrow(&name_services.expire_data, name);
229:
230:            assert!(*expire_time + BUFFER_DURATION < timestamp::now_seconds() ,
NAME_NOT_AVAILABLE);
231:
232:            table::remove(&mut name_services.expire_data, name);
233:            let ex_owner = table::borrow(&name_services.name_owner, name);
234:            let nft_vec = table::borrow_mut(&mut name_services.temp_nft_vec, *ex_owner);
//temp show all nft of owner
235:            token::burn_by_creator(&resource_account_signer, *ex_owner, domain_name, name, 0,
1);
236:            table::remove(&mut name_services.name_owner, name);
237:            let (exist, index): (bool, u64) = vector::index_of(nft_vec, &name);
//temp show all nft of owner
238:            if (exist) {
239:                vector::remove(nft_vec, index);
240:            }
241:        };
```

Description

Kong7ych3 : In the StarNameService module, when we register a new SNS name through the `create_name_script` function, it will cast a corresponding SNS NFT to the registrant, and update the resolver_contract, expire_data, name_owner, nft_controller and other tables in the NameServices resource. However, SNS NFT resources and NameServices resources are independent of each other. Users can transfer SNS NFT at will, but the table records in NameServices resources will not change during the NFT transfer process. So this will lead to a situation where the SNS NFT resource data does not match the NameServices resource data.

E.g: The user transfers his SNS NFT to other users, and when the SNS NFT expires, other users will not be able to use `burn_by_creator` to burn the user's NFT when they re-register through the `create_name_script` function,

because the NFT has already been transferred.

Recommendation

Kong7ych3 : It is recommended that protocol data should be checked with the actual owner of the SNS NFT as the subject. For example: check the actual owner of the expired SNS NFT when performing the create_name_script operation; the read_name_owner operation should also obtain the actual owner of the SNS NFT. You should not rely on the data recorded in NameServices.

Perhaps it is a good solution to customize an SNS-compatible NFT module based on the Token module in the Aptos framework.

Client Response

We add an 'owner_of' function to replace the name_owner, it will query the data from Aptos Token module, which may solved the issue. And we think the expire_data can still work when user transfer his SNS NFT.

STA-3:Expired SNS NFTs are not processed

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/sources/StarNameService.move#L482-L499	Fixed	Kong7ych3

Code

```
482:    public fun read_name_owner(name: string::String): address acquires NameServices{
483:        let name_services = borrow_global_mut(@SNS_address);
484:        assert!(table::contains(&name_services.name_owner, name), DATA_NOT_EXIST);
485:        return *table::borrow(&name_services.name_owner, name)
486:    }
487:
488:
489:    public fun read_name_controller(name: string::String): address acquires NameServices{
490:        let name_services = borrow_global_mut(@SNS_address);
491:        assert!(table::contains(&name_services.nft_controller, name), DATA_NOT_EXIST);
492:        return *table::borrow(&name_services.nft_controller, name)
493:    }
494:
495:    public fun read_name_resolver_contract(name: string::String): address acquires NameServices{
496:        let name_services = borrow_global_mut(@SNS_address);
497:        assert!(table::contains(&name_services.resolver_contract, name), DATA_NOT_EXIST);
498:        return *table::borrow(&name_services.resolver_contract, name)
499:    }
```

Description

Kong7ych3 : The SNS domain name has an expiration time, which is stored under the user's account in the form of SNS NFT resources. But SNS NFT does not have the concept of expiration time, and the expiration time is only stored in the NameServices.expire_data table. Therefore, when a domain name expires and no other users re-register, the expired domain name NFT can still be used normally. The owner of this domain name can still be read through the `read_name_owner` function. This not only allows users to maliciously register a large number of remote domain names,

but also specifies a short duration, so as long as these domain names are not re-registered, they can be used for a long time. This will also lead to unexpected risks for other protocols connected to the SNS domain name service.

Recommendation

Kong7ych3 : The short-term deferred construction solution is to specify the domain name reading interface in the SNS protocol, and check whether the domain name has expired when reading. But this still does not prevent other protocols from doing domain name checks through the SNS NFT TokenStore resource under the user account. Therefore, in the long run, building your own NFT Token to replace Aptos framework Token is a better solution.

Client Response

We add an 'owner_of' function to replace the name_owner, it will query the data from Aptos Token module, which may solved the issue. And we think the expire_data can still work when user transfer his SNS NFT.

STA-4:Index out of bound.

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/sources/StarNameService.move#L246-L252 • code/sources/StarNameService.move#L256-L262 • code/sources/StarNameService.move#L404-L410 • code/sources/StarNameService.move#L414-L420	Fixed	jayphbee

Code

```
246:                 if ( name_length > vector::length(fee_vector)) {
247:                     let fee = *vector::borrow(fee_vector, 0);
248:                     Airdrop::consume(account,fee * duration / 31536000);
249:                  }else {
250:                     let fee = *vector::borrow(fee_vector, name_length);
251:                     Airdrop::consume(account,fee * duration / 31536000);
252:                 }

256:                 if ( name_length > vector::length(fee_vector)) {
257:                     let fee = *vector::borrow(fee_vector, 0);
258:                     coin::transfer(account, @SNS_address, fee * duration / 31536000);
259:                  } else {
260:                     let fee = *vector::borrow(fee_vector, name_length);
261:                     coin::transfer(account, @SNS_address, fee * duration / 31536000);
262:                 }

404:                 if ( name_length > vector::length(fee_vector)) {
405:                     let fee = *vector::borrow(fee_vector, 0);
406:                     Airdrop::consume(owner,fee * duration / 31536000);
407:                  }else {
408:                     let fee = *vector::borrow(fee_vector, name_length);
409:                     Airdrop::consume(owner,fee * duration / 31536000);
410:                 }

414:                 if ( name_length > vector::length(fee_vector)) {
415:                     let fee = *vector::borrow(fee_vector, 0);
416:                     coin::transfer(owner, @SNS_address, fee * duration / 31536000);
417:                 } else {
418:                     let fee = *vector::borrow(fee_vector, name_length);
419:                     coin::transfer(owner, @SNS_address, fee * duration / 31536000);
420:                 }
```

Description

jayphbee : If `name_length == vector::length(fee_vector)`, it will cause execution abort due to index out of bound.

```
let fee = *vector::borrow(fee_vector, name_length);
```

The impact is that all the `name` whose length is equal to the `fee_vector` length when call to `renew_name_script` and `create_name_script` will abort due to index out of bound.

Recommendation

jayphbee : change

```
if ( name_length > vector::length(fee_vector)) {
```

to

```
if ( name_length >= vector::length(fee_vector)) {
```

This can avoid index out of bound abort.

Client Response

Fixed.

STA-5:Input parameter `to` has not been used by the function `airdrop_mint`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/sources/StarNameService.move#L272-L280 • code/sources/StarNameService.move#L326	Fixed	0xac, Kong7ych3

Code

```
272:    public entry fun airdrop_mint(
273:        account: &signer,
274:        to: address,
275:        name: string::String,
276:        domain_name: string::String,
277:        description: string::String,
278:        uri: string::String,
279:        duration: u64
280:        ) acquires NameServices, ResourceAccount, ForbiddenStrings, NameServicesFeeManager{

326:        create_name_script_nft(&resource_account_signer, signer::address_of(account), name,
domain_name, description, main_uri, duration);
```

Description

0xac : The `airdrop_mint` get a parameter `to`, but not use inside the function. It cause that the `to` address could not receive the token minted by the contract.

```
public entry fun airdrop_mint(
        account: &signer,
        to: address,
        name: string::String,
        domain_name: string::String,
        description: string::String,
        uri: string::String,
        duration: u64
        )
```

Kong7ych3 : In the StarNameService module, the airdrop_mint function is used to airdrop SNS NFT to the specified `to` address. However, the wrong receiving address was passed in during the create_name_script_nft operation. It incorrectly filled in the `signer` address for the `to` address that should have obtained the SNS NFT.

Recommendation

0xac : It is recommended to modify the code(Line 326) to `create_name_script_nft(&resource_account_signer, to, name, domain_name, description, main_uri, duration);`. Then it can mint the token to the `to` address in function `create_name_script_nft`(Line 360-365).

```
token::mint_token_to(
            resource_account,
            owner,
            tokendata_id,
            1,
        );
```

Kong7ych3 : It is recommended to replace the address of `signer` with the address of `to`, the following is the repair reference:

```
create_name_script_nft(&resource_account_signer, to, name, domain_name, description, main_uri,
duration);
```

Client Response

Fixed.

STA-6:Lack of limiting the length of `prefix` in function `check_prefix_length_isLegal`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/sources/StarNameService.move#L108-L118	Fixed	0xac, alansh, BradMoonUESTC, Hupixiong3

Code

```
108:    public fun check_prefix_length_isLegal(collection: string::String,prefix: string::String) :
bool acquires NameServicesFeeManager{
109:        let name_service_extension = borrow_global_mut(@SNS_address);
110:        let tmp_table = &mut name_service_extension.domain_to_shortestlength;
111:        let shortest_length = *table::borrow(tmp_table,collection);
112:
113:        if (string::length(&prefix) > shortest_length) {
114:            return true
115:        } else{
116:            return false
117:        }
118:    }
```

Description

0xac : In function `check_prefix_length_isLegal`, the code only ensure that the length of `prefix` is greater than `shortest_length`, but not less than `longest_length`.

```
if (string::length(&prefix) > shortest_length) {
    return true
} else{
    return false
}
```

alansh : `check_prefix_length_isLegal` only checks against shortest length, it should also check against longest length to be complete.

BradMoonUESTC : Function `check_prefix_length_isLegal` not used, also need to concern the length of input param

Hupixiong3 : Incomplete function, `check_prefix_length_isLegal` function is not referenced by other functions, and the check condition is missing, Missing `longst_Length` Check

Recommendation

0xac : Suggest that modifying the function as following:

```
public fun check_prefix_length_isLegal(collection: string::String,prefix: string::String) : bool
acquires NameServicesFeeManager{
    assert!(exists<NameServicesFeeManager>(@SNS_address),ERROR);
    let name_service_extension = borrow_global_mut<NameServicesFeeManager>(@SNS_address);

    let shortest_length_table = &mut name_service_extension.domain_to_shortestlength;
    assert!(table::contains(&shortest_length_table, collection), NOT_EXIST);
    let shortest_length = *table::borrow(shortest_length_table,collection);

    let longest_length_table = &mut name_service_extension.domain_to_longestlength;
    assert!(table::contains(&longest_length_table, collection), NOT_EXIST);
    let longest_length = *table::borrow(longest_length_table,collection);

    if (string::length(&prefix) > shortest_length && string::length(&prefix) < longest_length) {
        return true
    } else{
        return false
    }
}
```

alansh : Also check against `domain_to_longestlength`.

BradMoonUESTC : add Upper limit of input param of prefix collection, use this function or delete it.

Hupixiong3 : Improve function functions

Consider below fix in the `StarNameService.check_ prefix_ length_ isLegal` function

```
if (string::length(&prefix) < longest_length) {
        return true
```

Client Response

The check_prefix_length_isLegal function is not needed in the short term and we remove the function.

STA-7:No need for ex_expire_time in `StarNameService` contract `renew_name_script` function

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/sources/StarNameService.move#L422-L423	Fixed	alansh

Code

```
422:         let ex_expire_time = *table::borrow(&name_services.expire_data, name);
423:         table::upsert(&mut name_services.expire_data, name, ex_expire_time + duration);
```

Description

alansh : `ex_expire_time` is the same as `expire_time`, just use `expire_time` to save some gas.

Recommendation

alansh : Use `expire_time` instead.

Client Response

Fixed.

STA-8:Redundant codes in multiple functions of `StarNameService` contract

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/sources/StarNameService.move#L244-L245 • code/sources/StarNameService.move#L254-L255 • code/sources/StarNameService.move#L396-L397 • code/sources/StarNameService.move#L403-L404 • code/sources/StarNameService.move#L412-L413	Fixed	yekong, alansh

Code

```
244:            assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
245:            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);

254:            assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
255:            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);

396:        assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
397:        let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);

403:            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
404:                if ( name_length > vector::length(fee_vector)) {

412:            assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
413:            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
```

Description

yekong : Redundant code

```
assert!(table::contains(&name_service_extension.gradient_fee_in_domainname, domain_name),
GRADIENT_FEE_NOT_EXIST);
let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname, domain_name);
```

in

```
    if (pay_option == 1){
            assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
                if ( name_length > vector::length(fee_vector)) {
                    let fee = *vector::borrow(fee_vector, 0);
                    Airdrop::consume(account,fee * duration / 31536000);
                 }else {
                    let fee = *vector::borrow(fee_vector, name_length);
                    Airdrop::consume(account,fee * duration / 31536000);
                }
        } else {
            assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
            let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
                if ( name_length > vector::length(fee_vector)) {
                    let fee = *vector::borrow(fee_vector, 0);
                    coin::transfer<AptosCoin>(account, @SNS_address, fee * duration / 31536000);
                 } else {
                    let fee = *vector::borrow(fee_vector, name_length);
                    coin::transfer<AptosCoin>(account, @SNS_address, fee * duration / 31536000);
                }
        };
```

yekong : Duplicate code.The following code has already appeared before the if judgment, and there is no need to repeat the judgment and assignment

```
assert!(table::contains(&name_service_extension.gradient_fee_in_domainname, domain_name),
GRADIENT_FEE_NOT_EXIST);
let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname, domain_name);
```

alansh :

```
assert!(table::contains(&name_service_extension.gradient_fee_in_domainname, domain_name),
GRADIENT_FEE_NOT_EXIST);
let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname, domain_name);
```

These 2 lines are always needed in both branches, no need to write the same code twice.

Recommendation

yekong : Put these two pieces of code before the if judgment
yekong : It is recommended to delete the two codes of the if branch and the else branch
alansh : Put the common code before the branch.

Client Response

Fixed.

STA-9:Redundant initialize_token_store operation

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/sources/StarNameService.move#L213	Fixed	Kong7ych3

Code

```
213:         token::initialize_token_store(account);
```

Description

Kong7ych3 : In the `StarNameService` module, users can register domain names through the `create_name_script` function. It will first create a Token resource for the user through the `initialize_token_store` function, and then set its `direct_transfer` to true through the `opt_in_direct_transfer` function. But in fact, the `initialize_token_store` function is also called in the `opt_in_direct_transfer` function to create resources for the user. So the `initialize_token_store` operation in the `create_name_script` function is redundant.

Below is the implementation of `Token::opt_in_direct_transfer` function.

```
    public entry fun opt_in_direct_transfer(account: &signer, opt_in: bool) acquires TokenStore {
        let addr = signer::address_of(account);
        initialize_token_store(account);
        let opt_in_flag = &mut borrow_global_mut<TokenStore>(addr).direct_transfer;
        *opt_in_flag = opt_in;
    }
```

Recommendation

Kong7ych3 : It is recommended to remove the initialize_token_store operation in the create_name_script function.

Client Response

Fixed.

STA-10:Resource operation type usage error

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/StarNameService.move#L109 • code/sources/StarNameService.move#L476 • code/sources/StarNameService.move#L483 • code/sources/StarNameService.move#L490 • code/sources/StarNameService.move#L496	Fixed	Kong7ych3

Code

```
109:        let name_service_extension = borrow_global_mut(@SNS_address);

476:        let name_services = borrow_global_mut(@SNS_address);

483:        let name_services = borrow_global_mut(@SNS_address);

490:        let name_services = borrow_global_mut(@SNS_address);

496:        let name_services = borrow_global_mut(@SNS_address);
```

Description

Kong7ych3 : In the Move language, there are two ways to reference resources: `borrow_global_mut` and `borrow_global`. The `borrow_global_mut` operation is generally used to modify resource references, and `borrow_global` is generally used to read resources. So you only need to use the borrow_global operator when reading resources. However, in the `check_prefix_length_isLegal`, `read_name_expire`, `read_name_owner`, `read_name_controller` and `read_name_resolver_contract functions` of the `StarNameService` module, only the resource is read, but `borrow_global_mut` is used for the operation.

Recommendation

Kong7ych3 : It is recommended to use `borrow_global` when doing read operations on resources.

Client Response

Fixed.

STA-11:Risk of registered domain being front-run

Category	Severity	Code Reference	Status	Contributor
Race condition	Informational	• code/sources/StarNameService.move#L195-L270	Acknowledged	Kong7ych3

Code

```
195:    public entry fun create_name_script(
196:        account: &signer,
197:        name: string::String,
198:        domain_name: string::String,
199:        description: string::String,
200:        uri: string::String,
201:        duration: u64,
202:        pay_option: u64
203:    )acquires NameServices, ResourceAccount, ForbiddenStrings, NameServicesFeeManager{
204:        let name_service_extension = borrow_global_mut(@SNS_address);
205:        let shortest_length_table =&mut name_service_extension.domain_to_shortestlength;
206:        let shortest_length = *table::borrow(shortest_length_table, domain_name);
207:        let longest_length_table = &mut name_service_extension.domain_to_longestlength;
208:        let longest_length = *table::borrow(longest_length_table,domain_name);
209:
210:        let name_length = string::length(&name);
211:        assert!(name_length >= shortest_length, ILEGAL_NAME_LENGTH);
212:        assert!(name_length <= longest_length,ILEGAL_NAME_LENGTH);
213:        token::initialize_token_store(account);
214:        token::opt_in_direct_transfer(account, true);
215:        assert!(check_forbidden_string(name), ILLEGAL_STRING_EXIST);
216:
217:        let name_services = borrow_global_mut(@SNS_address);
218:        assert!(table::contains(&name_services.nameservice_collections, domain_name),
DOMAINNAME_NOT_EXISTS);
219:
220:        string::append_utf8(&mut name, b".");
221:        string::append(&mut name, domain_name);
222:
223:        let resource_account_capability = borrow_global(@SNS_address);
224:        let resource_account_signer =
account::create_signer_with_capability(&resource_account_capability.account_capability);
225:
226:
227:        if (table::contains(&name_services.expire_data, name)) {
228:            let expire_time = table::borrow(&name_services.expire_data, name);
229:
230:            assert!(*expire_time + BUFFER_DURATION < timestamp::now_seconds() ,
NAME_NOT_AVAILABLE);
```

```
231:
232:             table::remove(&mut name_services.expire_data, name);
233:             let ex_owner = table::borrow(&name_services.name_owner, name);
234:             let nft_vec = table::borrow_mut(&mut name_services.temp_nft_vec, *ex_owner);
//temp show all nft of owner
235:             token::burn_by_creator(&resource_account_signer, *ex_owner, domain_name, name, 0,
1);
236:             table::remove(&mut name_services.name_owner, name);
237:             let (exist, index): (bool, u64) = vector::index_of(nft_vec, &name);
//temp show all nft of owner
238:             if (exist) {
239:                 vector::remove(nft_vec, index);
240:             }
241:         };
242:
243:         if (pay_option == 1){
244:             assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
245:             let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
246:                 if ( name_length > vector::length(fee_vector)) {
247:                     let fee = *vector::borrow(fee_vector, 0);
248:                     Airdrop::consume(account,fee * duration / 31536000);
249:                  }else {
250:                     let fee = *vector::borrow(fee_vector, name_length);
251:                     Airdrop::consume(account,fee * duration / 31536000);
252:                 }
253:         } else {
254:             assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
255:             let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
256:                 if ( name_length > vector::length(fee_vector)) {
257:                     let fee = *vector::borrow(fee_vector, 0);
258:                     coin::transfer(account, @SNS_address, fee * duration / 31536000);
259:                  } else {
260:                     let fee = *vector::borrow(fee_vector, name_length);
261:                     coin::transfer(account, @SNS_address, fee * duration / 31536000);
262:                 }
263:         };
264:
```

```
265:         let main_uri = string::utf8(b"https://api.sns.so/v1/image/");
266:         string::append(&mut main_uri, name);
267:         string::append_utf8(&mut main_uri, b".");
268:         string::append(&mut name, domain_name);
269:         create_name_script_nft(&resource_account_signer, signer::address_of(account), name,
domain_name, description, main_uri, duration);
270:     }
```

Description

Kong7ych3 : In the StarNameService module, users can register a domain name through the create_name_script function. But in Aptos, verifiers can also sort the transactions submitted by users (refer to the transaction life cycle in Aptos official documentation). Therefore, when a user registers for SNS, other users or nodes can pre-empt it to register and sell this SNS at a higher price.

Recommendation

Kong7ych3 : It is recommended to refer to the registration logic in ENS. Users need to commitment hash first to determine that they need to register a domain name, and then check whether the registrant is consistent with the commitment hash during the registration process to avoid this risk.
Ref: https://docs.ens.domains/contract-api-reference/.eth-permanent-registrar/controller

Client Response

TBD

STA-12:The relationship of two parameters has not been checked

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/sources/StarNameService.move#L387-L399	Acknowledged	0xac

Code

```
387:        assert!(exists(@SNS_address),NAMESERVICES_NOT_PUBLISH);
388:        let name_services = borrow_global_mut(@SNS_address);
389:        assert!(table::contains(&name_services.expire_data, name), NAME_NOT_EXIST);
390:        let expire_time = *table::borrow(&name_services.expire_data, name);
391:        assert!(expire_time + BUFFER_DURATION > timestamp::now_seconds(),
NAME_HAS_BEEN_EXPIRED);
392:        let storaged_owner = table::borrow(&name_services.name_owner, name);
393:        assert!( *storaged_owner == signer::address_of(owner), IS_NOT_YOUR_NAME);
394:
395:        let name_service_extension = borrow_global_mut(@SNS_address);
396:        assert!(table::contains(&name_service_extension.gradient_fee_in_domainname,
domain_name), GRADIENT_FEE_NOT_EXIST);
397:        let fee_vector = table::borrow(&name_service_extension.gradient_fee_in_domainname,
domain_name);
398:        //let name_length = string::length(&name) - string::length(&domain_name) - 1;
399:        let name_length = string::length(&name);
```

Description

0xac : The relationship of `name` and `domain_name` has not been checked in function `renew_name_script`. It means that the owner of `name` can input a `domain_name` which is not related to `name`. For example the value of `name` is `abc.com`, but the value of `domain_name` is `xyz`. This can further affect the calculation of fees(Line 410-421).

```
public entry fun renew_name_script(
        owner: &signer,
        name: string::String,
        domain_name: string::String,
        duration: u64,
        pay_option: u64
    )
```

Recommendation

0xac : It is suggest that to check the relationship of `name` and `domain_name`. To make sure `domain_name` is the domain of `name`.

Client Response

TBD

STA-13:Typographical errors in multiple contracts

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/Airdrop.move#L19 • code/sources/Resovler.move#L23-L24 • code/sources/Resovler.move#L35-L36 • code/sources/Resovler.move#L71 • code/sources/Resovler.move#L83	Fixed	Hupixiong3, alansh

Code

```
19:       maxmium_supply: u64

23:        defualt_resolver: Table,
24:        defualt_reverse_resolver: Table,

35:                        defualt_resolver: table::new(),
36:                        defualt_reverse_resolver: table::new(),

71:        table::upsert(&mut resolver_services.defualt_resolver, name, account);

83:        table::upsert(&mut resolver_services.defualt_reverse_resolver ,
StarNameService::read_name_owner(name), name);
```

Description

Hupixiong3 : Spelling mistakes
alansh : `defualt_reverse_resolver` should be `default_reverse_resolver`, the file `Resovler.move` should be renamed to `Resolver.move`

Recommendation

Hupixiong3 : Correct Spelling
Consider below fix in the `Airdrop`

```
struct CreditService has key{
   credit_table: table::Table<address,u64>,
   supplied: u64,
   maximum_supply: u64
}
```

alansh : n: `defualt_reverse_resolver` should be `default_reverse_resolver`, the file `Resovler.move` should be renamed to `Resolver.move`

Client Response

TBD

STA-14:User can free from fee by registering a very short time domain

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/sources/StarNameService.move#L248 • code/sources/StarNameService.move#L251 • code/sources/StarNameService.move#L258 • code/sources/StarNameService.move#L261 • code/sources/StarNameService.move#L406 • code/sources/StarNameService.move#L409 • code/sources/StarNameService.move#L416 • code/sources/StarNameService.move#L419	Acknowledged	jayphbee, 0xac

Code

```
248:                    Airdrop::consume(account,fee * duration / 31536000);

251:                    Airdrop::consume(account,fee * duration / 31536000);

258:                    coin::transfer(account, @SNS_address, fee * duration / 31536000);

261:                    coin::transfer(account, @SNS_address, fee * duration / 31536000);

406:                    Airdrop::consume(owner,fee * duration / 31536000);

409:                    Airdrop::consume(owner,fee * duration / 31536000);

416:                    coin::transfer(owner, @SNS_address, fee * duration / 31536000);

419:                    coin::transfer(owner, @SNS_address, fee * duration / 31536000);
```

Description

jayphbee : There are some fees paid to `SNS_address` by AptosCoin or by consume some airdropped amount with different `pay_option` when calling `StarNameService::create_name_script` or `StarNameService::renew_name_script` functions.

Due to the pricision loss, `fee * duration / 31536000` can be zero, that is to say user can free from fee by calling `create_name_script` and `renew_name_script` periodically as long as he ensures `fee * duration` less than 31536000.

0xac : Based on the `fee * duration / 31536000` formula, the result of calculation is 0 if `fee * duration` less then `31536000`. This formula would influent two functions as following. While the value of `fee` is set less than `31536000`, `owner` of `name` can input a small `duration` (such as 1) to avoid paying.

```
// pay_option == 1
Airdrop::consume(account,fee * duration / 31536000);

// else
coin::transfer<AptosCoin>(account, @SNS_address, fee * duration / 31536000);
```

Recommendation

jayphbee : To prevent user free from fee there should be a minimum fee to pay when fee * duration / 31536000 is zero. Or add `MIN_REGISTRATION_DURATION`.

0xac : It suggests that to ensure the calculating result of `fee * duration / 31536000` is greater than 0.

```
assert!(fee * duration / 31536000 > 0);
```

Client Response

We will set the fee to be greater than or equal to 1 APT, which represents 1 followed by eight zeros in program. Therefore, users will have to pay even if they register a name for a short time because the value of fee is greater than 31536000.

STA-15:Wrong main_uri append

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/sources/StarNameService.move#L268 • code/sources/StarNameService.move#L325	Fixed	Kong7ych3

Code

```
268:        string::append(&mut name, domain_name);

325:        string::append(&mut name, domain_name);
```

Description

Kong7ych3 : In the StarNameService module, the main_uri will be spliced in the create_name_script function to complete the casting of SNS NFT. But it mistakenly spliced domain_name to name repeatedly, resulting in the lack of domain_name in the link of main_uri. And the token_mutate_config parameter of SNS is set to be unchangeable. So once the casting of SNS NFT is completed, the wrong main_uri will not be able to be modified.

The main_uri in the airdrop_mint function also has this issue.

The following is the setting of the `token_mutate_config` parameter in the `StarNameService::create_name_script_nft` function:

`let mutate_setting = vector<bool>[false, false, false, false, false];`

Recommendation

Kong7ych3 : The following code is recommended for fix.

`string::append(&mut main_uri, domain_name);`

Client Response

Fixed.

STA-16: `Airdrop.airdrop()` incorrect amount logic

Category	Severity	Code Reference	Status	Contributor
Logic	Critical	• code/sources/Airdrop.move#L37-L43 • code/sources/Airdrop.move#L40	Fixed	jayphbee, yekong, 0xac, Kong7ych3, BradMoonUESTC, Hupixiong3

Code

```
37:    public entry fun airdrop(account: &signer, to: address, amount: u64) acquires CreditService{
38:        let credit_service = borrow_global_mut(@SNS_address);
39:        assert!(signer::address_of(account) == @SNS_address, ONLY_OWNER);
40:        assert!(amount + credit_service.supplied >=
credit_service.maxmium_supply,SUPPLY_OVERFLOW);
41:        let tmp_table = &mut credit_service.credit_table;
42:        table::upsert(tmp_table,to,amount);
43:    }

40:        assert!(amount + credit_service.supplied >=
credit_service.maxmium_supply,SUPPLY_OVERFLOW);
```

Description

jayphbee : There is an invariant that the to be airdropped `amount` plus the `supplied` should less than or equal to `maxmium_supply`, but this line of code have the reverse logic.

```
assert!(amount + credit_service.supplied >= credit_service.maxmium_supply,SUPPLY_OVERFLOW);
```

The impact is that `Airdop::airdrop` can airdrop unlimitted amount that greater than the `maxmium_supple`.

jayphbee : When `Airdriop::airdrop` finished, `credit_service.supplied` should updated accordingly to indicate how much have been airdropped.

The impact is that more than the `maxmium_supply` amount can be airdropped if `credit_service.supplied` not being updated accordingly.

yekong : The variable 'supplied' is not updated after calling the 'airdrop' function, and the assert judgment is wrong. Therefore, it can cause additional issuance, exceeding the maximum limit

```
    public entry fun airdrop(account: &signer, to: address, amount: u64) acquires CreditService{
        let credit_service = borrow_global_mut<CreditService>(@SNS_address);
        assert!(signer::address_of(account) == @SNS_address, ONLY_OWNER);
        assert!(amount + credit_service.supplied >= credit_service.maxmium_supply,SUPPLY_OVERFLOW);
        let tmp_table = &mut credit_service.credit_table;
        table::upsert(tmp_table,to,amount);
    }
```

0xac : The logic of `assert!(amount + credit_service.supplied >= credit_service.maxmium_supply,SUPPLY_OVERFLOW);` is wrong. This `assert` cause the amount of airdrops exceeds the set maximum `maxmium_supply`. For example, the value of `maxmium_supply` is 100, and `supplied` is 0. The `amount` must be no less than 100.
Kong7ych3 : In the Airdrop module, the airdrop function is used to airdrop Credits to users. It will check that the sum of the required airdrop Credit amount and the current total supply must be less than or equal to `maxmium_supply`. However, in the actual inspection, the `>=` operation symbol is used incorrectly, which will cause airdrop to not work as expected.
BradMoonUESTC : The supplied data is not updated in time after initial executed, so the actual number of supplied may exceed the maxmium_supply
Hupixiong3 : Condition judgment error, `amount + credit_service.supplied >= credit_service.maxmium_supply` It should be `<=`.
Hupixiong3 : `credit_service.supplied` not updated

Recommendation

jayphbee : change the implementation to:

```
  assert!(amount + credit_service.supplied <= credit_service.maxmium_supply,SUPPLY_OVERFLOW);
```

jayphbee : update the `credit_service.supplied` in `Airdriop::airdrop` function.
`credit_service.supplied += amount`

yekong : Update the variable 'supplied' at the end of the function, and modify the assert to be '<='
0xac : It is recommended to modify the code(Line 40) to `assert!(amount + credit_service.supplied <= credit_service.maxmium_supply,SUPPLY_OVERFLOW);`.
Kong7ych3 : It is recommended to change `>=` to `<=`.
Consider below fix in the `Airdrop::airdrop` function
`assert!(amount + credit_service.supplied <= credit_service.maxmium_supply,SUPPLY_OVERFLOW);`

BradMoonUESTC : update the supplied data in initial function

Hupixiong3 : Revise judgment

Consider below fix in the `Airdrop.airdrop` function

```
    public entry fun airdrop(account: &signer, to: address, amount: u64) acquires
CreditService{
        let credit_service = borrow_global_mut<CreditService>(@SNS_address);
        assert!(signer::address_of(account) == @SNS_address, ONLY_OWNER);
        assert!(amount + credit_service.supplied <=
credit_service.maxmium_supply,SUPPLY_OVERFLOW);
        let tmp_table = &mut credit_service.credit_table;
        table::upsert(tmp_table,to,amount);
    }
```

Hupixiong3 : `credit_service.supplied` should be updated

Consider the below fix in the `Airdrop.airdrop` function

```
credit_service.supplied += amount;
```

Client Response

TBD

STA-17: `StarNameService.LengthRange` is empty and not used

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/StarNameService.move#L38-L41	Fixed	BradMoonUESTC

Code

```
38://  todo
39:    struct LengthRange{
40:
41:    }
```

Description

BradMoonUESTC : The struct `LengthRange` is empty and not used.

Recommendation

BradMoonUESTC : Delete it or use it.

Client Response

Fixed.

STA-18:resource_account variable is unused in `StarNameService.move` contract `initialize_name_services` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/StarNameService.move#L156	Fixed	alansh

Code

```
156:        let (resource_account, cap) = account::create_resource_account(creator, seed);
```

Description

alansh : `resource_account` is unused.

Recommendation

alansh : Unused variables should be replaced with `_` instead.

Client Response

Fixed.

STA-19:uri parameter is not used in `StarNameService.move` contract `create_name_script` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/sources/StarNameService.move#L200	Fixed	alansh

Code

```
200:        uri: string::String,
```

Description

alansh : The parameter `uri` is not used.

Recommendation

alansh : Check the usage of `uri`. Delete it or implement it.

Client Response

Fixed.

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.