Forgotten Runes Warrior Guild contest Findings & Analysis Report

2022-08-04

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Forgotten Runes Warrior Guild smart contract system written in Solidity. The audit contest took place between May 3—May 5 2022.

Wardens

102 Wardens contributed reports to the Forgotten Runes Warrior Guild contest:

1. AuditsAreUS
2. BowTiedWardens (BowTiedHeron, BowTiedPickle, m4rio_eth, Dravee, and BowTiedFirefox)
3. IIIIIII
4. pedroais
5. defsec
6. Ruhum
7. sorrynotsorry
8. rfa
9. VAD37

10. MaratCerby
11. shenwilly
12. WatchPug (jtp and ming)
13. teddav
14. leastwood
15. GimelSec (rayn and sces60107)
16. throttle
17. 0xDjango
18. unforgiven
19. reassor
20. hake
21. shung
22. fatherOfBlocks
23. dipp
24. 0x1f8b
25. dirk_y
26. rajatbeladiya
27. Kulk0
28. hyh
29. broccolirob
30. Dinddle
31. joestakey
32. horsefacts
33. hickuphh3
34. robee
35. 0xliumin
36. ilan
37. p4st13r4 (0x69e8 and 0xb4bb4)
38. TrungOre

39. z3s
40. rotcivegaf
41. hubble (ksk2345 and shri4net)
42. berndartmueller
43. tintin
44. cccz
45. m9800
46. peritoflores
47. 0xkatana
48. FSchmoede
49. Czar102
50. kenzo
51. TerrierLover
52. catchup
53. kenta
54. 0x4non
55. marximimus
56. pauliax
57. delfin454000
58. kebabsec (okkothejawa and FlameHorizon)
59. 0xf15ers (remora and twojoy)
60. 0v3rf10w
61. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
62. ellahi
63. minhquanym
64. oyc_109
65. Picodes
66. eccentricexit
67. Funen

68. hansfriese
69. Hawkeye (0xwags and 0xmint)
70. M0ndoHEHE
71. samruna
72. simon135
73. Cr4ckM3
74. sseefried
75. 0x52
76. csanuragjain
77. cryptphi
78. plotchy
79. saian
80. 0xc0ffEE
81. 0xNazgul
82. antonttc
83. Cityscape
84. DavidGialdi
85. MiloTruck
86. slywaters
87. Tadashi
88. 0xProf
89. ACai
90. AlleyCat
91. noobie
92. RoiEvenHaim

This contest was judged by gzeon. The judge also competed in the contest as a warden, but forfeited their winnings.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 6 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 75 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 73 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Forgotten Runes Warrior Guild contest repository, and is composed of 5 smart contracts written in the Solidity programming language and includes 712 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

Medium Risk Findings (6)

[M-01] IERC20.transfer does not support all ERC20 token

Submitted by VAD37, also found by AuditsAreUS, IIIIIII, MaratCerby, rfa, and sorrynotsorry

ForgottenRunesWarriorsGuild.sol#L173-L176
ForgottenRunesWarriorsMinter.sol#L627-L630

Token like USDT known for using non-standard ERC20. (Missing return boolean on transfer).

Contract function forwardERC20 will always revert when try to transfer this kind of tokens.

Impact

Cannot withdraw some special ERC20 token through contract call. Unexpected contract functionality = Medium severity

Recommended Mitigation Steps

Use SafeTransferLib.safeTransfer instead of IERC20 transfer. This accepts ERC20 token with no boolean return like USDT.

cryppadotta (Forgotten Runes) confirmed and commented:

> Ah nice. You learn something new every day. Thanks!

KenzoAgada (warden) commented:

> Description is pretty much invalid as "forwardERC20 will always revert when try to transfer this kind of tokens" is simply not true. Same with impact - "Cannot withdraw some special ERC20 token through contract call" - that's not the impact, using SafeERC20's transfer will not help to transfer tokens. It will just revert on failure. But generally the issue of not using SafeERC20 is kinda-correct. Duplicate of #2.

gzeon (judge) commented:

> This is not a duplicate of #2. #2 describes the silent failure of ERC20 transfer, while this describes a ERC20 that return void instead of bool. The call will revert even if the transfer is successful because Solidity expected a return value. Judging as Med Risk because unlike #2, here you can actually do something to fix the function.

KenzoAgada (warden) commented:

> I apologize, my mistake.

[M-02] Contract may not have enough fund to cover refund

Submitted by gzeon, also found by AuditsAreUS, BowTiedWardens, pedroais, Ruhum, shenwilly, and teddav

Owner of the contract can call `withdrawAll` before the refund process is done to send all ETH to the vault. Since there are no payable receive function in `ForgottenRunesWarriorsMinter`, the owner won't be able to replenish the contract for the refund process.

Proof of Concept

ForgottenRunesWarriorsMinter.sol#L616-L619

```
function withdrawAll() public payable onlyOwner {
    require(address(vault) != address(0), 'no vault');
    require(payable(vault).send(address(this).balance));
}
```

Recommended Mitigation Steps

Only allow owner to call `withdrawAll` after refund period.

cryppadotta (Forgotten Runes) confirmed and commented:

> This is a great point. It would be annoying to accidentally do this and have to make a new contract for refunds.

gzeon (judge) commented:

> Sponsor confirmed, submitted by contest judge.

[M-03] Critical variables shouldn't be changed after they are set

Submitted by pedroais, also found by AuditsAreUS, BowTiedWardens, defsec, GimelSec, gzeon, IIIIIII, leastwood, and WatchPug

ForgottenRunesWarriorsMinter.sol#L564
ForgottenRunesWarriorsMinter.sol#L571
ForgottenRunesWarriorsMinter.sol#L557
ForgottenRunesWarriorsMinter.sol#L571

The price for the dutch auction could be altered.

Proof of Concept

I previously sent an issue about start time being settable more than once. This also happens for many other variables. Since they are many I will send them all in a single issue.

The following functions should only be called once to ensure trustlessness and integrity of the dutch auction:
setDaPriceCurveLength
setStartPrice
setLowestPrice
setDaDropInterval

The price in the dutch auction is computed by this formula:
uint256 dropPerStep = (startPrice - lowestPrice) /
(daPriceCurveLength / daDropInterval);

By making dapriceCurveLength or daDropInterval equal to 0 the owner could stop the auction. This could benefit the owner since the price lowers with time and everyone pays the final lower price. If the auction does well at the beginning the owner could stop the auction to stop the price from being lower. This works against the integrity of the dutch auction.

Also changing the Start Price or the Lowest price in the middle of the auction could allow the owner to manipulate the price.

Recommended Mitigation Steps

To each of these setter functions add require (variable == 0) to ensure they are set once in a permanent way. Also, the Lowest price < startPrice should be required.

gzeon (judge) commented:

> While centralization risk is acknowledged by the team, I agree that these variable should not be able to be changed during sale since it may lead to loss of functionality. Consolidating all similar issue for different variables here.

[M-04] Many unbounded and under-constrained variables in the system can lead to unfair price or DoS

Submitted by throttle, also found by 0xDjango, BowTiedWardens, defsec, dipp, fatherOfBlocks, gzeon, hake, reassor, shung, unforgiven, and WatchPug

Unbounded and under-constrained variables.

Proof of Concept

1. `dsStartTime` | `daPriceCurveLength` | `daDropInterval`

The team can change the above variables during sale. It will either increase or decrease the price of an NFT. Or it can make `currentDaPrice()` revert.

```
uint256 dropPerStep = (startPrice - lowestPrice) / (daPriceCurve

uint256 elapsed = block.timestamp - daStartTime;
```

```
    uint256 steps = elapsed / daDropInterval;
    uint256 stepDeduction = steps * dropPerStep;

    // don't go negative in the next step
    if (stepDeduction > startPrice) {
        return lowestPrice;
    }
    uint256 currentPrice = startPrice - stepDeduction;
```

ForgottenRunesWarriorsMinter.sol#L275-L297

2. `dsStartTime` | `mintlistStartTime` | `publicStartTime` | `claimsStartTime` `selfRefundsStartTime`

The team can change the above variables. It can result in the wrong sale phases order. For example, the public sale can end up being before every other phase due to accidentally setting it to 0.

Recommended Mitigation Steps

Possible mitigation:

1. Bound and constrain variables.
   For example, daDropInterval should be less than daPriceCurveLength
   Another example: The total sum of each supply phase should not be bigger than `MAX_SUPPLY` in the NFT smart contract.

wagmiwiz (Forgotten Runes) commented:

> This is true but is a low operational risk and can be undone.

gzeon (judge) commented:

> Decided to consolidate all issues regarding missing validation of the listed variables here (M-04).

[M-05] Use of `.send()` May Revert if The Recipient's Fallback Function Consumes More Than 2300 Gas

Submitted by leastwood, also found by 0xliumin, berndartmueller, cccz, Czar102, gzeon, hickuphh3, horsefacts, ilan, IllIIll, joestakey, m9800, p4st13r4, peritoflores, reassor, rfa, robee, sorrynotsorry, tintin, TrungOre, VAD37, WatchPug, and z3s

ForgottenRunesWarriorsMinter.sol#L610
ForgottenRunesWarriorsMinter.sol#L618
ForgottenRunesWarriorsGuild.sol#L164

The `.send()` function intends to transfer an ETH amount with a fixed amount of 2300 gas. This function is not equipped to handle changes in the underlying `.send()` and `.transfer()` functions which may supply different amounts of gas in the future. Additionally, if the recipient implements a fallback function containing some sort of logic, this may inevitably revert, meaning the vault and owner of the contract will never be able to call certain sensitive functions.

Recommended Mitigation Steps

Consider using `.call()` instead with the checks-effects-interactions pattern implemented correctly. Careful consideration needs to be made to prevent reentrancy.

gzeon (judge) commented:

> Determined the stake is high here and therefore Medium Risk.

[M-06] The owner can mint all of the NFTs.

Submitted by Kulk0, also found by 0x1f8b, 0xDjango, BowTiedWardens, broccolirob, defsec, Dinddle, dirk_y, hyh, rajatbeladiya, Ruhum, throttle, and unforgiven

ForgottenRunesWarriorsMinter.sol#L257

In ForgottenRunesWarriorsMinter.teamSummon() the owner can mint unrestricted amount of NFTs. This is more of a design issue than an actual bug in my opinion.

Proof of Concept

If the private keys were compromised during the launch the attacker could mint almost all of the NFTs. Normally I wouldn't say this is an issue but from your documentation, I understand that you are not planning to use a multi-sig wallet for the owner of the contracts. I definitely don't want to say that you are incompetent and you can't store your private keys safely but private keys are getting compromised very often in this space.

Recommended Mitigation Steps

Limit how many NFTs can the owner mint. So even if the private keys were compromised the attacker couldn't destroy the entire set by minting thousands of the NFTs to himself making the entire set worth nothing.

I also think this will help with the trust of the protocol since the buyers will know exactly how many NFTs can the Dev Team mint for themselves.

cryppadotta (Forgotten Runes) acknowledged and commented:

> This is true, but by design. It's a risk for minters, but it would be obvious, so we're economically disincentivized to do this. Acknowledged, but not changing it.

gzeon (judge) marked as Invalid and commented:

> Sponsor acknowledged centralization risk in README.

dmitriia (warden) commented:

> Centralization risk in general is one thing, the ability for unlimited mint, which is easily fixable, is another.

> A kind of a boundary state here in my opinion, having 'acknowledged' and 'invalid' flags in the same time poses some contradiction.

gzeon (judge) reassessed as Medium severity and commented:

> Judging this as Med Risk since there are specified amounts of teamSummon in the doc

> Forgotten Council DAO Creators Fund (teamSummon): ~333
> Team & Partners (teamSummon): ~325
> Community Honoraries and Contests (teamSummon): ~50

> which is not enforced in the `teamSummon` function.

Low Risk and Non-Critical Issues

For this contest, 75 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **defsec** received the top score from the judge.

The following wardens also submitted reports: reassor, lllllll, hubble, rotcivegaf, hake, horsefacts, throttle, AuditsAreUS, berndartmueller, BowTiedWardens, hickuphh3, hyh, joestakey, robee, Ruhum, shung, sorrynotsorry, sseefried, leastwood, pedroais, TerrierLover, VAD37, WatchPug, 0xDjango, catchup, delfin454000, ilan, kebabsec, kenta, MaratCerby, p4st13r4, pauliax, rfa, shenwilly, tintin, 0x1f8b, 0x52, 0xf15ers, 0xkatana, 0xliumin, cccz, csanuragjain, dirk_y, eccentricexit, fatherOfBlocks, Funen, GimelSec, hansfriese, Hawkeye, kenzo, rajatbeladiya, teddav, TrungOre, unforgiven, z3s, 0v3rf10w, 0x4non, broccolirob, CertoraInc, Cr4ckM3, cryptphi, ellahi, Kulk0, M0ndoHEHE, m9800, marximimus, minhquanym, oyc_109, peritoflores, Picodes, plotchy, samruna, simon135, *and* gzeon.

ISSUE LIST

[01]: Missing events for only functions that change critical parameters - Non Critical
[02] : Critical changes should use two-step procedure - Non Critical
[03] : Pragma Version - Non Critical
[04] : Missing zero-address check in the setter functions and initializers - Low
[05] : transferOwnership should be two step - Non critical
[06] : Bump OZ packages to ^4.5.0. - Non critical
[07] : Use safeTransfer/safeTransferFrom consistently instead of transfer/transferFrom - Non critical
[08] : Front-running is possible over the bidding mechanism - Low

[01] Missing events for only functions that change critical parameters

The afunctions that change critical parameters should emit events. Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying on-chain contract state for such changes is not considered practical for most users/usages.

Missing events and timelocks do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in liquidity which could negatively impact protocol TVL and reputation.

Proof of Concept

Navigate to the following contracts.

ForgottenRunesWarriorsMinter.sol#L441
ForgottenRunesWarriorsMinter.sol#L448
ForgottenRunesWarriorsMinter.sol#L455
ForgottenRunesWarriorsMinter.sol#L462
ForgottenRunesWarriorsMinter.sol#L469
ForgottenRunesWarriorsMinter.sol#L480

See similar High-severity H03 finding OpenZeppelin's Audit of Audius (https://blog.openzeppelin.com/audius-contracts-audit/#high) and Medium-severity M01 finding OpenZeppelin's Audit of UMA Phase 4 (https://blog.openzeppelin.com/uma-audit-phase-4/)

Recommended Mitigation Steps

Add events to all functions that change critical parameters.

[02] Critical changes should use two-step procedure

The critical procedures should be two step process.

Proof of Concept

Navigate to the following contracts.

ForgottenRunesWarriorsMinter.sol#L441
ForgottenRunesWarriorsMinter.sol#L448
ForgottenRunesWarriorsMinter.sol#L455
ForgottenRunesWarriorsMinter.sol#L462
ForgottenRunesWarriorsMinter.sol#L469
ForgottenRunesWarriorsMinter.sol#L480

Recommended Mitigation Steps

Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two step procedure on the critical functions.

[03] Pragma Version

In the contracts, floating pragmas should not be used. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Proof of Concept

https://swcregistry.io/docs/SWC-103

```
All Contracts
```

Recommended Mitigation Steps

Lock the pragma version: delete pragma solidity 0.8.10 in favor of pragma solidity 0.8.10.

[04] Missing zero-address check in the setter functions and initializers

Missing checks for zero-addresses may lead to infunctional protocol, if the variable addresses are updated incorrectly.

Proof of Concept

Navigate to the following contracts.

ForgottenRunesWarriorsMinter.sol#L544
ForgottenRunesWarriorsMinter.sol#L528

Recommended Mitigation Steps

Consider adding zero-address checks in the discussed constructors: require(newAddr != address(0));.

[05] transferOwnership should be two step

The owner is the authorized user in the solidity contracts. Usually, an owner can be updated with transferOwnership function. However, the process is only completed with single transaction. If the address is updated incorrectly, an owner functionality will be lost forever.

Proof of Concept

Navigate to the following contracts.

ForgottenRunesWarriorsMinter.sol#L15
ForgottenRunesWarriorsGuild.sol#L14

Recommended Mitigation Steps

Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two step procedure on the critical functions.

[06] Bump OZ packages to ^4.5.0.

Line Reference: package.json#L68

Description

I can verify that the installed version is 4.2.0 by executing the following commands:

```
yarn install
yarn list @openzeppelin/contracts
```

Recommended Mitigation Steps

Update the versions of @openzeppelin/contracts and @openzeppelin/contracts-upgradeable to be the latest in package.json. I also recommend double checking the versions of other dependencies as a precaution, as they may include important bug fixes.

[07] Use safeTransfer/safeTransferFrom consistently instead of transfer/transferFrom

It is good to add a require() statement that checks the return value of token transfers or to use something like OpenZeppelin's safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in contract.

Reference: This similar medium-severity finding from Consensys Diligence Audit of Fei Protocol: https://consensys.net/diligence/audits/2021/01/fei-protocol/#unchecked-return-value-for-iweth-transfer-call

Proof of Concept

1. Navigate to the following contract.
2. transfer/transferFrom functions are used instead of safe transfer/transferFrom on the following contracts.

ForgottenRunesWarriorsGuild.sol#L175

Recommended Mitigation Steps

Consider using safeTransfer/safeTransferFrom or require() consistently.

[08] Front-running is possible over the minting process

During the code review, it has been noticed that to bidding mechanism is vulnerable to front-running. The bidding mechanism can have EOA check on the contract.

Proof of Concept

1. Navigate to the following contract.
2. The contract does not check for the External Owned Accounts. Without the check, any contract can interact with the function.

ForgottenRunesWarriorsMinter.sol#L120

Recommended Mitigation Steps

Consider to check EOA at the beginning of the function.

```
msg.sender == tx.origin && !isContract(msg.sender)
```

Gas Optimizations

For this contest, 73 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **BowTiedWardens** received the top score from the judge.

The following wardens also submitted reports: joestakey, FSchmoede, defsec, 0xkatana, horsefacts, hickuphh3, WatchPug, reassor, lllllll, rotcivegaf, kenzo, 0x4non, 0xliumin, catchup, kenta, marximimus, rfa, robee, shung, TerrierLover, saian, sorrynotsorry, 0v3rf10w, 0x1f8b, 0xc0ffEE, 0xDjango, 0xf15ers, 0xNazgul, antonttc, CertoraInc, Cityscape, DavidGialdi, ellahi, GimelSec, hake, ilan, MiloTruck, minhquanym, oyc_109, Picodes, slywaters, throttle, TrungOre, VAD37, Kulk0, M0ndoHEHE, p4st13r4, pauliax, samruna, simon135, Tadashi, 0xProf, ACai, AlleyCat, Cr4ckM3, delfin454000, Dinddle, dirk_y, eccentricexit, fatherOfBlocks, Funen, hansfriese, Hawkeye, kebabsec, MaratCerby, noobie, rajatbeladiya, RoiEvenHaim, shenwilly, unforgiven, z3s, *and* gzeon.

Table of Contents

- Unchecking arithmetics operations that can't underflow/overflow
- Unnecessary `initialize()` function
- `ForgottenRunesWarriorsGuild.forwardERC20s()` and `ForgottenRunesWarriorsMinter.forwardERC20s()` : Unnecessary require statements
- `ForgottenRunesWarriorsMinter`: `bidSummon()` and `publicSummon()` : Unnecessary require statement
- Boolean comparisons
- `> 0` is less efficient than `!= 0` for unsigned integers (with proof)
- `ForgottenRunesWarriorsMinter.currentDaPrice()` : `>` should be `>=`
- Splitting `require()` statements that use `&&` saves gas
- `++i` costs less gas compared to `i++` or `i += 1`
- Increments can be unchecked
- Public functions to external
- No need to explicitly initialize variables with default values
- Upgrade pragma to at least 0.8.4
- Use `msg.sender` instead of OpenZeppelin's `_msgSender()` when meta-transactions capabilities aren't used
- Use Custom Errors instead of Revert Strings to save Gas

[G-01] Caching storage values in memory

The code can be optimized by minimising the number of SLOADs. SLOADs are expensive (100 gas) compared to MLOADs/MSTOREs (3 gas). Here, storage values should get cached in memory (see the `@audit` tags for further details):

```
contracts/ForgottenRunesWarriorsGuild.sol:
  100:         require(numMinted < MAX_WARRIORS, 'All warriors h
  102:         uint256 tokenId = numMinted; //@audit gas: numMir
  104:         numMinted += 1;  //@audit gas: numMinted SLOAD 3

contracts/ForgottenRunesWarriorsMinter.sol:
  136:         require(numSold < maxDaSupply, 'Auction sold out'
  137:         require(numSold + numWarriors <= maxDaSupply, 'Nc
```

```
154:         numSold += numWarriors; //@audit gas: numSold SLO
156:         if (numSold == maxDaSupply) { //@audit gas: numSo
177:         require(numSold < maxForSale, 'Sold out'); //@auc
193:         numSold += 1; //@audit gas: numSold SLOAD 2 (equi
207:         require(numSold < maxForSale, 'Sold out'); //@auc
208:         require(numSold + numWarriors <= maxForSale, 'Not
219:         numSold += numWarriors; //@audit gas: numSold SLO
234:         require(numClaimed < maxForClaim, 'No more claims
248:         numClaimed += 1; //@audit gas: numSold SLOAD 2 (e
279:         if (block.timestamp >= daStartTime + daPriceCurve
284:         uint256 dropPerStep = (startPrice - lowestPrice)
285:             (daPriceCurveLength / daDropInterval); //@auc
287:         uint256 elapsed = block.timestamp - daStartTime;/
288:         uint256 steps = elapsed / daDropInterval; //@audi
292:         if (stepDeduction > startPrice) { //@audit gas: s
293:             return lowestPrice; //@audit gas: lowestPrice
295:         uint256 currentPrice = startPrice - stepDeductior
296:         return currentPrice > lowestPrice ? currentPrice
401:             IWETH(weth).deposit{value: amount}(); //@audi
402:             IERC20(weth).transfer(to, amount); //@audit c
609:         require(address(vault) != address(0), 'no vault')
610:         require(payable(vault).send(_amount)); //@audit c
617:         require(address(vault) != address(0), 'no vault')
618:         require(payable(vault).send(address(this).balance
```

[G-02] Unchecking arithmetics operations that can't underflow/overflow

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an `unchecked` block:

https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic

I suggest wrapping L295 with an `unchecked` block (see `@audit`):

```
File: ForgottenRunesWarriorsMinter.sol
291:         // don't go negative in the next step
292:         if (stepDeduction > startPrice) {
293:             return lowestPrice;
```

```
294:         }
295:         uint256 currentPrice = startPrice - stepDeducti
```

[G-03] Unnecessary `initialize()` function

The `initialize()` function isn't an initializer. It just calls `setMinter()`, which has the same visibility and authorization level as `initialize()`:

```
File: ForgottenRunesWarriorsGuild.sol
52:     function initialize(address newMinter) public onlyOwner
53:         setMinter(newMinter);
54:     }
...
137:     function setMinter(address newMinter) public onlyOwner
138:         minter = newMinter;
139:     }
```

It could even be called repeatedly.

As the `initialize()` function is not needed, I suggest deleting it and directly calling `setMinter()` to "Conveniently initialize the contract".

[G-04] `ForgottenRunesWarriorsGuild.forwardERC20s()` and `ForgottenRunesWarriorsMinter.forwardERC20s()`: Unnecessary require statements

Here, as the `onlyOwner` modifier is applied, the `address(0)` checks are not needed here:

```
contracts/ForgottenRunesWarriorsGuild.sol:
  173     function forwardERC20s(IERC20 token, uint256 amount)
  174:        require(address(msg.sender) != address(0)); //@au
  175         token.transfer(msg.sender, amount);
  176     }

contracts/ForgottenRunesWarriorsMinter.sol:
  627     function forwardERC20s(IERC20 token, uint256 amount)
  628:        require(address(msg.sender) != address(0)); //@au
```

```
629             token.transfer(msg.sender, amount);
630         }
```

I suggest removing these checks.

[G-05] `ForgottenRunesWarriorsMinter`: `bidSummon()` and `publicSummon()`: Unnecessary require statement

The code is as such:

```
File: ForgottenRunesWarriorsMinter.sol
130:     function bidSummon(uint256 numWarriors)
131:         external
132:         payable
133:         nonReentrant
134:         whenNotPaused
135:     {
136:         require(numSold < maxDaSupply, 'Auction sold out');
137:         require(numSold + numWarriors <= maxDaSupply, 'Not
138:         require(daStarted(), 'Auction not started');
139:         require(!mintlistStarted(), 'Auction phase over');
140:         require(
141:             numWarriors > 0 && numWarriors <= 20,
142:             'You can summon no more than 20 Warriors at a t
143:         );
...
201:     function publicSummon(uint256 numWarriors)
202:         external
203:         payable
204:         nonReentrant
205:         whenNotPaused
206:     {
207:         require(numSold < maxForSale, 'Sold out');
208:         require(numSold + numWarriors <= maxForSale, 'Not e
209:         require(publicStarted(), 'Public sale not started')
210:         require(
211:             numWarriors > 0 && numWarriors <= 20,
212:             'You can summon no more than 20 Warriors at a t
213:         );
```

Logically speaking, `numSold + numWarriors <= maxForSale` could only reach the edge-case if `numWarriors == 0`, but that's prevented with the condition that follows in both functions: `numWarriors > 0 && numWarriors <= 20`. Meaning that, with `numSold + numWarriors <= maxForSale` and `numWarriors > 0`, we don't need to check if `numSold < maxForSale` as it just can't happen.

I suggest removing the 2 `require(numSold < maxDaSupply)` checks L136 and L207.

Furthermore, notice that `'Not enough remaining'` and `'Sold out'` kinda mean the same thing, so the additionnal require statement might not be justified.

[G-06] Boolean comparisons

Comparing to a constant (`true` or `false`) is a bit more expensive than directly checking the returned boolean value. I suggest using `if(!directValue)` instead of `if(directValue == false)` here:

```
ForgottenRunesWarriorsMinter.sol:182:        require(mintlistMir
ForgottenRunesWarriorsMinter.sol:238:        require(claimlistMi
```

[G-07] `> 0` is less efficient than `!= 0` for unsigned integers (with proof)

`!= 0` costs less gas compared to `> 0` for unsigned integers in `require` statements with the optimizer enabled (6 gas)

Proof: While it may seem that `> 0` is cheaper than `!=`, this is only true without the optimizer enabled and outside a require statement. If you enable the optimizer at 10k AND you're in a `require` statement, this will save gas. You can see this tweet for more proofs: https://twitter.com/gzeon/status/1485428085885640706

I suggest changing `> 0` with `!= 0` here:

```
ForgottenRunesWarriorsMinter.sol:141:            numWarriors > (
```

```
ForgottenRunesWarriorsMinter.sol:211:            numWarriors > 0
```

Also, please enable the Optimizer.

[G-08] `ForgottenRunesWarriorsMinter.currentDaPrice() : >` should be `>=`

The return statement is as follows:

```
ForgottenRunesWarriorsMinter.sol:296:        return currentPrice
```

Strict inequalities (`>`) are more expensive than non-strict ones (`>=`). This is due to some supplementary checks (ISZERO, 3 gas)
Furthermore, `lowestPrice` is read from storage while `currentPrice` is read from memory.

Therefore, it's possible to always save 3 gas and sometimes further save 1 SLOAD (when `currentPrice == lowestPrice`) by replacing the code to:

```
ForgottenRunesWarriorsMinter.sol:296:        return currentPrice
```

[G-09] Splitting `require()` statements that use `&&` saves gas

If you're using the Optimizer at 200, instead of using the `&&` operator in a single require statement to check multiple conditions, I suggest using multiple require statements with 1 condition per require statement:

```
contracts/ForgottenRunesWarriorsMinter.sol:
  140          require(
  141:             numWarriors > 0 && numWarriors <= 20,
  142              'You can summon no more than 20 Warriors at a
  143          );
```

```
210         require(
211:             numWarriors > 0 && numWarriors <= 20,
212              'You can summon no more than 20 Warriors at a
213         );
```

[G-10] `++i` costs less gas compared to `i++` or `i += 1`

`++i` costs less gas compared to `i++` or `i += 1` for unsigned integer, as pre-increment is cheaper (about 5 gas per iteration). This statement is true even with the optimizer enabled.

`i++` increments `i` and returns the initial value of `i`. Which means:

```
uint i = 1;
i++; // == 1 but i == 2
```

But `++i` returns the actual incremented value:

```
uint i = 1;
++i; // == 2 and i == 2 too, so no need for a temporary variable
```

In the first case, the compiler has to create a temporary variable (when used) for returning `1` instead of `2`

Instances include:

```
ForgottenRunesWarriorsGuild.sol:104:         numMinted += 1;
ForgottenRunesWarriorsMinter.sol:162:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:193:         numSold += 1;
ForgottenRunesWarriorsMinter.sol:220:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:248:         numClaimed += 1;
ForgottenRunesWarriorsMinter.sol:259:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:355:         for (uint256 i = st
```

I suggest using `++i` instead of `i++` to increment the value of an uint variable.

[G-11] Increments can be unchecked

In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.

ethereum/solidity#10695

Instances include:

```
ForgottenRunesWarriorsMinter.sol:162:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:220:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:259:         for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:355:         for (uint256 i = st
```

The code would go from:

```
for (uint256 i; i < numIterations; i++) {
 // ...
}
```

to:

```
for (uint256 i; i < numIterations;) {
 // ...
 unchecked { ++i; }
}
```

The risk of overflow is inexistant for a `uint256` here.

[G-12] Public functions to external

The following functions could be set external to save gas and improve code quality. External call cost is less expensive than of public functions.

```
initialize(address) should be declared external:
```

- ForgottenRunesWarriorsGuild.initialize(address) (contracts/Fo
exists(uint256) should be declared external:
- ForgottenRunesWarriorsGuild.exists(uint256) (contracts/Forgot
setProvenanceHash(string) should be declared external:
- ForgottenRunesWarriorsGuild.setProvenanceHash(string) (contra
withdrawAll() should be declared external:
- ForgottenRunesWarriorsGuild.withdrawAll() (contracts/Forgotte
forwardERC20s(IERC20,uint256) should be declared external:
- ForgottenRunesWarriorsGuild.forwardERC20s(IERC20,uint256) (co
numDaMinters() should be declared external:
- ForgottenRunesWarriorsMinter.numDaMinters() (contracts/Forgot
issueRefunds(uint256,uint256) should be declared external:
- ForgottenRunesWarriorsMinter.issueRefunds(uint256,uint256) (c
refundAddress(address) should be declared external:
- ForgottenRunesWarriorsMinter.refundAddress(address) (contract
selfRefund() should be declared external:
- ForgottenRunesWarriorsMinter.selfRefund() (contracts/Forgotte
pause() should be declared external:
- ForgottenRunesWarriorsMinter.pause() (contracts/ForgottenRune
unpause() should be declared external:
- ForgottenRunesWarriorsMinter.unpause() (contracts/ForgottenRu
setSelfRefundsStartTime(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setSelfRefundsStartTime(uint256)
setPhaseTimes(uint256,uint256,uint256,uint256) should be declare
- ForgottenRunesWarriorsMinter.setPhaseTimes(uint256,uint256,ui
setMintlist1MerkleRoot(bytes32) should be declared external:
- ForgottenRunesWarriorsMinter.setMintlist1MerkleRoot(bytes32)
setMintlist2MerkleRoot(bytes32) should be declared external:
- ForgottenRunesWarriorsMinter.setMintlist2MerkleRoot(bytes32)
setClaimlistMerkleRoot(bytes32) should be declared external:
- ForgottenRunesWarriorsMinter.setClaimlistMerkleRoot(bytes32)
setStartPrice(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setStartPrice(uint256) (contract
setLowestPrice(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setLowestPrice(uint256) (contrac
setDaPriceCurveLength(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setDaPriceCurveLength(uint256) (
setDaDropInterval(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setDaDropInterval(uint256) (cont
setFinalPrice(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setFinalPrice(uint256) (contract
setMaxDaSupply(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setMaxDaSupply(uint256) (contrac
setMaxForSale(uint256) should be declared external:
- ForgottenRunesWarriorsMinter.setMaxForSale(uint256) (contract
setMaxForClaim(uint256) should be declared external:

```
 - ForgottenRunesWarriorsMinter.setMaxForClaim(uint256) (contrac
withdraw(uint256) should be declared external:
 - ForgottenRunesWarriorsMinter.withdraw(uint256) (contracts/For
withdrawAll() should be declared external:
 - ForgottenRunesWarriorsMinter.withdrawAll() (contracts/Forgott
forwardERC20s(IERC20,uint256) should be declared external:
 - ForgottenRunesWarriorsMinter.forwardERC20s(IERC20,uint256) (c
```

[G-13] No need to explicitly initialize variables with default values

If a variable is not set/initialized, it is assumed to have the default value (`0` for `uint` , `false` for `bool` , `address(0)` for address...). Explicitly initializing it with its default value is an anti-pattern and wastes gas.

As an example: `for (uint256 i = 0; i < numIterations; ++i) {` should be replaced with `for (uint256 i; i < numIterations; ++i) {`

Instances include:

```
ForgottenRunesWarriorsGuild.sol:24:    uint256 public numMinted
ForgottenRunesWarriorsMinter.sol:162:        for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:220:        for (uint256 i = 0;
ForgottenRunesWarriorsMinter.sol:259:        for (uint256 i = 0;
```

I suggest removing explicit initializations for default values.

[G-14] Upgrade pragma to at least 0.8.4

Using newer compiler versions and the optimizer give gas optimizations. Also, additional safety checks are available for free.

The advantages here are:

- **Low level inliner** (>= 0.8.2): Cheaper runtime gas (especially relevant when the contract has small functions).
- **Optimizer improvements in packed structs** (>= 0.8.3)

- **Custom errors** (>= 0.8.4): cheaper deployment cost and runtime cost. *Note*: the runtime cost is only relevant when the revert condition is met. In short, replace revert strings by custom errors.

Consider upgrading pragma to at least 0.8.4:

```
ForgottenRunesWarriorsGuild.sol:1:pragma solidity ^0.8.0;
ForgottenRunesWarriorsMinter.sol:1:pragma solidity ^0.8.0;
```

[G-15] Use `msg.sender` instead of OpenZeppelin's `_msgSender()` when meta-transactions capabilities aren't used

`msg.sender` costs 2 gas (CALLER opcode). `_msgSender()` represents the following:

```
function _msgSender() internal view virtual returns (address pay
  return msg.sender;
}
```

When no meta-transactions capabilities are used: `msg.sender` is enough.

See https://docs.openzeppelin.com/contracts/2.x/gsn for more information about GSN capabilities.

Consider replacing `_msgSender()` with `msg.sender` here:

```
ForgottenRunesWarriorsGuild.sol:101:        require(_msgSender()
ForgottenRunesWarriorsGuild.sol:115:            _isApprovedOrOwr
```

In the solution, `msg.sender` is used everywhere else:

```
ForgottenRunesWarriorsGuild.sol:164:        require(payable(msg.
ForgottenRunesWarriorsGuild.sol:174:        require(address(msg.
ForgottenRunesWarriorsGuild.sol:175:        token.transfer(msg.s
```

```
ForgottenRunesWarriorsMinter.sol:113:         setVaultAddress(msg
ForgottenRunesWarriorsMinter.sol:151:         daMinters.push(msg.
ForgottenRunesWarriorsMinter.sol:152:         daAmountPaid[msg.se
ForgottenRunesWarriorsMinter.sol:153:         daNumMinted[msg.ser
ForgottenRunesWarriorsMinter.sol:163:             _mint(msg.sende
ForgottenRunesWarriorsMinter.sol:182:         require(mintlistMir
ForgottenRunesWarriorsMinter.sol:183:         mintlistMinted[msg.
ForgottenRunesWarriorsMinter.sol:186:         bytes32 node = kecc
ForgottenRunesWarriorsMinter.sol:194:         _mint(msg.sender);
ForgottenRunesWarriorsMinter.sol:221:             _mint(msg.sende
ForgottenRunesWarriorsMinter.sol:238:         require(claimlistMi
ForgottenRunesWarriorsMinter.sol:239:         claimlistMinted[msg
ForgottenRunesWarriorsMinter.sol:242:         bytes32 node = kecc
ForgottenRunesWarriorsMinter.sol:249:         _mint(msg.sender);
ForgottenRunesWarriorsMinter.sol:373:         _refundAddress(msg.
ForgottenRunesWarriorsMinter.sol:628:         require(address(msg
ForgottenRunesWarriorsMinter.sol:629:         token.transfer(msg.
```

[G-16] Use Custom Errors instead of Revert Strings to save Gas

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met)

Source: https://blog.soliditylang.org/2021/04/21/custom-errors/:

> Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., `revert("Insufficient funds.");`), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Custom errors are defined using the `error` statement, which can be used inside and outside of contracts (including interfaces and libraries).

Instances include:

```
ForgottenRunesWarriorsGuild.sol:68:         require(
ForgottenRunesWarriorsGuild.sol<img class="emoji-icon" alt="emoj
```

```
ForgottenRunesWarriorsGuild.sol:101:        require(_msgSender()
ForgottenRunesWarriorsGuild.sol:114:        require(
ForgottenRunesWarriorsGuild.sol:164:        require(payable(msg.
ForgottenRunesWarriorsGuild.sol:174:        require(address(msg.
ForgottenRunesWarriorsMinter.sol:136:        require(numSold < m
ForgottenRunesWarriorsMinter.sol:137:        require(numSold + r
ForgottenRunesWarriorsMinter.sol:138:        require(daStarted()
ForgottenRunesWarriorsMinter.sol:139:        require(!mintlistSt
ForgottenRunesWarriorsMinter.sol:140:        require(
ForgottenRunesWarriorsMinter.sol:146:        require(
ForgottenRunesWarriorsMinter.sol:177:        require(numSold < m
ForgottenRunesWarriorsMinter.sol:178:        require(mintlistSta
ForgottenRunesWarriorsMinter.sol:179:        require(msg.value =
ForgottenRunesWarriorsMinter.sol:182:        require(mintlistMir
ForgottenRunesWarriorsMinter.sol:187:        require(
ForgottenRunesWarriorsMinter.sol:207:        require(numSold < m
ForgottenRunesWarriorsMinter.sol:208:        require(numSold + r
ForgottenRunesWarriorsMinter.sol:209:        require(publicStart
ForgottenRunesWarriorsMinter.sol:210:        require(
ForgottenRunesWarriorsMinter.sol:214:        require(
ForgottenRunesWarriorsMinter.sol:234:        require(numClaimed
ForgottenRunesWarriorsMinter.sol:235:        require(claimsStart
ForgottenRunesWarriorsMinter.sol:238:        require(claimlistMi
ForgottenRunesWarriorsMinter.sol:243:        require(
ForgottenRunesWarriorsMinter.sol:258:        require(address(rec
ForgottenRunesWarriorsMinter.sol:372:        require(selfRefunds
ForgottenRunesWarriorsMinter.sol:488:        require(
ForgottenRunesWarriorsMinter.sol:492:        require(
ForgottenRunesWarriorsMinter.sol:609:        require(address(vau
ForgottenRunesWarriorsMinter.sol:610:        require(payable(vau
ForgottenRunesWarriorsMinter.sol:617:        require(address(vau
ForgottenRunesWarriorsMinter.sol:618:        require(payable(vau
ForgottenRunesWarriorsMinter.sol:628:        require(address(msg
```

I suggest replacing revert strings with custom errors.

gzeon (judge) commented:

> Most are valid, except:
>
> **ForgottenRunesWarriorsMinter.currentDaPrice(): > should be >=**
>
> Strict is cheaper since there is no opcode for non-strict comparison in evm.

> **No need to explicitly initialize variables with default values**

> Yes, but I don't think it saves gas in for loop with optimizer enabled.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top