SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	Optimism
Prepared by:	Sherlock
Lead Security Expert:	Trust
Dates Audited:	March 27 - April 4, 2024
Prepared on:	June 7, 2024

Introduction

The first open source, permissionless, feature-complete fault proof system in the Ethereum ecosystem.

Scope

Repository: ethereum-optimism/optimism

Branch: develop

Commit: 5137f3b74c6ebcac4f0f5a118b0f4909df03aec6

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
4	0

Security experts who found valid issues

Trust
GalloDaSballo
MiloTruck
obront
guhu95
Stiglitz
lemonmon
fibonacci
0xdeadbeef
zigtur
haxatron
nirohgo
ctf_sec
tallo
bin2chen

Issue M-1: Incorrect game type can be proven and finalized due to unsafe cast

Source: https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84

Found by

Stiglitz, guhu95, lemonmon, obront

Summary

The `gameProxy.gameType().raw()` conversions used by OptimismPortal2 in the proving and finalization steps incorrectly casts the `gameType` to a `uint8` instead of a `uint32`, which causes mismatched game types to be considered equivalent. In the event that a game is exploitable, this can be used to skirt around the off-chain monitoring to finalize an invalid withdrawal.

Vulnerability Detail

Each game can be queried for its `gameType`, which is compared to the current `respectedGameType` in the Portal to confirm the game is valid.

GameType is represented as a `uint32`, allowing numbers up to `2 ** 32 - 1`.

```
type GameType is uint32;
```

However, when converting the GameType to an integer type in order to perform comparisons in the proving and finalization process, we unsafely downcase to a `uint8`:

```
function raw(GameType _gametype) internal pure returns (uint8 gametype_) {
    assembly {
        gametype_ := _gametype
    }
}
```

This means that for any `oldGameType % 256 == X`, any `newGameType % 256 == X` will be considered the same game type.

This has the potential to shortcut the safeguards to allow malicious games to be finalized.

As is explained in the comments, only games of the current `respectedGameType` will be watched by the off-chain challenger. This is why we do not allow games that

pre-date the last update to be finalized:

> // The game must have been created after `respectedGameTypeUpdatedAt`. This is to prevent users from creating // invalid disputes against a deployed game type while the off-chain challenge agents are not watching.

However, the watcher will not be watching games where `gameType % 256 == respectedGameType % 256`.

Let's imagine a situation where game type `0` has been deemed unsafe. It is well known that a user can force a `DEFENDER_WINS` state, even when it is not correct.

At a future date, when the current game type is `256`, a user creates a game with `gameType = 0`. It is not watched by the off chain challenger. This game can be used to prove an invalid state, and then finalize their withdrawal, all while not being watched by the off chain monitoring system.

Proof of Concept

The following proof of concept can be added to its own file in `test/L1` to demonstrate the vulnerability:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import { Test } from "forge-std/Test.sol";
import "./OptimismPortal2.t.sol";

contract UnsafeDowncastTest is CommonTest {
    // Reusable default values for a test withdrawal
    Types.WithdrawalTransaction _defaultTx;
    bytes32 _stateRoot;
    bytes32 _storageRoot;
    bytes32 _outputRoot;
    bytes32 _withdrawalHash;
    bytes[] _withdrawalProof;
    Types.OutputRootProof internal _outputRootProof;

    // Use a constructor to set the storage vars above, so as to minimize the
↪   number of ffi calls.
    function setUp() public override {
        super.enableFaultProofs();
        super.setUp();

        _defaultTx = Types.WithdrawalTransaction({
            nonce: 0,
```

```
            sender: alice,
            target: bob,
            value: 100,
            gasLimit: 100_000,
            data: hex""
        });
        // Get withdrawal proof data we can use for testing.
        (_stateRoot, _storageRoot, _outputRoot, _withdrawalHash,
↪   _withdrawalProof) =
            ffi.getProveWithdrawalTransactionInputs(_defaultTx);

        // Setup a dummy output root proof for reuse.
        _outputRootProof = Types.OutputRootProof({
            version: bytes32(uint256(0)),
            stateRoot: _stateRoot,
            messagePasserStorageRoot: _storageRoot,
            latestBlockhash: bytes32(uint256(0))
        });

        // Fund the portal so that we can withdraw ETH.
        vm.deal(address(optimismPortal2), 0xFFFFFFFF);
    }

    function testWrongGameTypeSucceeds() external {
        // we start with respected gameType == 256
        vm.prank(superchainConfig.guardian());
        optimismPortal2.setRespectedGameType(GameType.wrap(256));

        // create a game with gameType == 0, which we know is exploitable
        FaultDisputeGame game = FaultDisputeGame(
            payable(
                address(
                    disputeGameFactory.create(
                        GameType.wrap(0), Claim.wrap(_outputRoot),
↪   abi.encode(uint(0xFF))
                    )
                )
            )
        );

        // proving works, even though gameType is incorrect
        vm.warp(block.timestamp + 1);
        optimismPortal2.proveWithdrawalTransaction({
            _tx: _defaultTx,
            _disputeGameIndex: disputeGameFactory.gameCount() - 1,
            _outputRootProof: _outputRootProof,
            _withdrawalProof: _withdrawalProof
```


```
        });

        // warp beyond the game duration and resolve the game
        vm.warp(block.timestamp + 4 days);
        game.resolveClaim(0);
        game.resolve();

        // warp another 4 days so withdrawal can be finalized
        vm.warp(block.timestamp + 4 days);

        // finalizing works, even though gameType is incorrect
        uint beforeBal = bob.balance;
        optimismPortal2.finalizeWithdrawalTransaction(_defaultTx);
        assertEq(bob.balance, beforeBal + 100);
    }
}
```

Impact

The user is able to prove and finalize their withdrawal against a game that is not being watched and is known to be invalid. This would allow them to prove arbitrary withdrawals and steal all funds in the Portal.

Code Snippet

https://github.com/sherlock-audit/2024-02-optimism-2024/blob/main/optimism/packages/contracts-bedrock/src/dispute/lib/LibUDT.sol#L117-L126

https://github.com/sherlock-audit/2024-02-optimism-2024/blob/main/optimism/packages/contracts-bedrock/src/L1/OptimismPortal2.sol#L260-L261

https://github.com/sherlock-audit/2024-02-optimism-2024/blob/main/optimism/packages/contracts-bedrock/src/L1/OptimismPortal2.sol#L497-L500

Tool used

Manual Review

Recommendation

```
- function raw(GameType _gametype) internal pure returns (uint8 gametype_) {
+ function raw(GameType _gametype) internal pure returns (uint32 gametype_) {
      assembly {
          gametype_ := _gametype
```

```
        }
}
```

Discussion

smartcontracts

We see this as a valid medium severity issue

sherlock-admin4

The protocol team fixed this issue in the following PRs/commits:
https://github.com/ethereum-optimism/optimism/pull/10152

nevillehuang

Based on scope highlighted below (issue exists and affects portal contract, which is a non-game contract) and sherlock scoping rules

https://docs.google.com/document/d/1xjvPwAzD2Zxtx8-P6UE69TuoBwtZPbpwf5zBHAvBJBw/edit

> 2. In case the vulnerability exists in a library and an in-scope contract uses it and is affected by this bug this is a valid issue.

I believe this is a medium severity issue given the following constraints:

- At least 256 games must exist in a single game type
- This issue doesn't bypass the airgap/Delayed WETH safety net, so can still be monitored off-chain to trigger a fallback mechanism to pause the system and update the respected game type if a game resolves incorrectly.

zobront

Escalate

I believe this issue should be judged as High Severity.

The purpose of this contest was to examine the safeguards that could lead to the catastrophic consequences of having an invalid fault proof accepted. We were given the constraints of assuming the game is buggy. This means that (a) none of those issues were accepted, but also that (b) issues that would arise IF the system were very buggy are valid.

This is the only issue in the contest that poses this extreme risk.

While it has the condition that 255 other games are created, based on the assumption that the game is buggy, it doesn't seem out of the question that a large number of additional game types would need to be deployed. This is the only requirement for this issue to be exploitable (counter to what the judge mentioned

above), because Optimism's off chain watcher only watches the currently active game).

More importantly, in the event that this happens, the consequences are catastrophic. A game that is (a) not being watched and (b) known to be buggy, is accepted as valid (both in the proving step of withdrawal and the finalization step of withdrawal).

This leads to a very real, very extreme risk of a fraudulent withdrawal getting through the system.

With the constraints of the contest in mind (assuming the game is buggy), as well as the potential billions of dollars of lost funds that could occur, I believe this is the exact kind of issue that was crucial to find, and clearly fits the criteria for High Severity.

sherlock-admin2

> Escalate
>
> I believe this issue should be judged as High Severity.
>
> The purpose of this contest was to examine the safeguards that could lead to the catastrophic consequences of having an invalid fault proof accepted. We were given the constraints of assuming the game is buggy. This means that (a) none of those issues were accepted, but also that (b) issues that would arise IF the system were very buggy are valid.
>
> This is the only issue in the contest that poses this extreme risk.
>
> While it has the condition that 255 other games are created, based on the assumption that the game is buggy, it doesn't seem out of the question that a large number of additional game types would need to be deployed. This is the only requirement for this issue to be exploitable (counter to what the judge mentioned above), because Optimism's off chain watcher only watches the currently active game).
>
> More importantly, in the event that this happens, the consequences are catastrophic. A game that is (a) not being watched and (b) known to be buggy, is accepted as valid (both in the proving step of withdrawal and the finalization step of withdrawal).
>
> This leads to a very real, very extreme risk of a fraudulent withdrawal getting through the system.
>
> With the constraints of the contest in mind (assuming the game is buggy), as well as the potential billions of dollars of lost funds that could occur, I believe this is the exact kind of issue that was crucial to find, and clearly fits the criteria for High Severity.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

guhu95

Three additional points to support the escalation in favor of high:

1. There isn't a constraint of 256 prior games due to use of non-sequential game types.
2. DoS impact that is caused by the mitigation actions qualifies for high severity.
3. Off-chain monitoring for this issue is not plausible without prior knowledge of the issue.

1. Games types are not sequential

> "- At least 256 games must exist in a single game type"
>
> "While it has the condition that 255 other games are created"

This appears to be neither a constraint nor a condition:

1. `setImplementation` does not require sequential game type values.
2. The 3 already defined GameTypes are not sequential: 0, 1, and 255, and all are configured in the same factory during deployment.
3. The further use of non-sequential game types is highly likely due to "namespacing" via higher order bits, as is already done with predeploy addresses (`0x42...01`), and with their implementation addresses (`0xc0de...01`) etc. This kind of namespacing will result in many exploitable collisions.

2. The DoS impact of mitigation qualifies as high

> ...to pause the system and update the respected game type if a game resolves incorrectly.

Switching the respected game type pauses the bridge for a significant amount of time qualifying as a DoS issue for the valid withdrawals delayed by the mitigation.

The DoS impact for a valid withdrawal that would otherwise be finalizable is well over one week:

1. Off-chain monitoring needs to detect the suspicious `WithdrawalProven` that was not expected. The issue needs to be validated to require pausing (SLA of 24 hours).

2. A new implementation of the dispute game, with a new game type value, and t he new anchor registry (which are immutable) will need to be deployed.
3. The factory will need to be updated by the owner (SLA of 72 hours) to include the new implementation.
4. The respected game type for the portal would need to be updated by guardian (SLA of 24 hours).
5. New dispute games will need to be created by proposers for the withdrawals backlog caused by the delays.
6. Only after all these steps the re-proving for previously valid withdrawals for previously valid games can be restarted, and would require waiting at least 7 days from the point of unpausing.

Because this blocks all cross-chain interactions on the bridge for a prolonged period of time, and delays message passing, it blocks all cross-chain protocols operating across this bridge (including their time-sensitive operations) and not only locks up funds.

3. Off-chain monitoring is conditional on knowing of this issue

- This issue doesn't bypass the airgap/Delayed WETH safety net, so can still be monitored off-chain to trigger a fallback mechanism

While it is theoretically possible to monitor for this off-chain, it is unlikely to result in this action without knowledge of this vulnerability. This is because a creation of new instance of an old game, that is no longer "respected" by the portal, should not raise cause for concern (if the issue is unknown at that point).

trust1995

Escalate

Firstly, the finding is brilliant and extremely well noticed by the participants. In my mind, the finding falls under Low severity, with the reasoning below:

- As far as devs are concerned, there are a maximum of 256 game types. The bug is an unsynchronized view between the underlying structure and the definition of GameType as uint32. All evidence points to the fact Optimism *did not* plan to make use of over 256 game types.
- From a practical standpoint, even if over 256 game types were planned to be supported, to get to such a high amount of different game types is extremely unlikely (as of now, three are planned). The odds of the architecture not getting refactored, closing the issue, by the time 256 game types are needed, I estimate to be under a thousandth of a percent.
- For there to be an impact, the following must hold:

- A new, **vulnerable** game type must be defined (highly hypothetical) after 256 game types.
- It's encoding suffix must line up with the `respectedGameId` set by the admin
- All honest challengers must not look at the vulnerable game type, despite the fact that challenging it is +EV (they are guaranteed to pick up the attacker's bond if the claim is invalid)
- The airgap is **not** bypassed - At any the guardian is able to blacklist the game and make it unfinalizable. This reason caused for dozens of issues in this contest to be invalidated, and not applying it for this bug is inconsistent and shows unsound logic.

sherlock-admin2

Escalate

Firstly, the finding is brilliant and extremely well noticed by the participants. In my mind, the finding falls under Low severity, with the reasoning below:

- As far as devs are concerned, there are a maximum of 256 game types. The bug is an unsynchronized view between the underlying structure and the definition of GameType as uint32. All evidence points to the fact Optimism *did not* plan to make use of over 256 game types.
- From a practical standpoint, even if over 256 game types were planned to be supported, to get to such a high amount of different game types is extremely unlikely (as of now, three are planned). The odds of the architecture not getting refactored, closing the issue, by the time 256 game types are needed, I estimate to be under a thousandth of a percent.
- For there to be an impact, the following must hold:
- A new, **vulnerable** game type must be defined (highly hypothetical) after 256 game types.
- It's encoding suffix must line up with the `respectedGameId` set by the admin
- All honest challengers must not look at the vulnerable game type, despite the fact that challenging it is +EV (they are guaranteed to pick up the attacker's bond if the claim is invalid)
- The airgap is **not** bypassed - At any the guardian is able to blacklist the game and make it unfinalizable. This reason caused for dozens

of issues in this contest to be invalidated, and not applying it for this bug is inconsistent and shows unsound logic.

The escalation could not be created because you are not exceeding the escalation threshold.

You can view the required number of additional valid issues/judging contest payouts in your Profile page, in the Sherlock webapp.

guhu95

@trust1995 the main argument you present (sequential game types constraint on likelihood) is refuted by the evidence in my message above yours (see *"1. Games types are not sequential"*)

Would you mind specifying what part of the reasoning or evidence you agree and disagree with?

There are more details and links above, but for your convenience these are: **1.** Setter doesn't require sequential numbers. **2.** The three existing games are non-sequential (1, 2, **255**) and are all added to the factory on deployment. **3.** Namespacing via higher bits is already prevalent in the codebase and makes this a highly probable scenario.

trust1995

The game types defined below follow a common pattern where the upper value is set as a placeholder for a safe non-production value. It's clearly not meant to assume they do skipping as a policy, and any experienced developer can confirm the intention is to keep running from 0,1, up to 255.

```
library GameTypes {
    /// @dev A dispute game type the uses the cannon vm.
    GameType internal constant CANNON = GameType.wrap(0);
    /// @dev A permissioned dispute game type the uses the cannon vm.
    GameType internal constant PERMISSIONED_CANNON = GameType.wrap(1);
    /// @notice A dispute game type that uses an alphabet vm.
    ///         Not intended for production use.
    GameType internal constant ALPHABET = GameType.wrap(255);
}
```

This is further confirmed by their docs which outline the intended structure of the GameID:

```
/// @notice A `GameId` represents a packed 1 byte game ID, an 11 byte timestamp,
 ↪  and a 20 byte address.
/// @dev The packed layout of this type is as follows:
///
///    Bits       Value
```

```
///
///  [0, 8)     Game Type
///  [8, 96)    Timestamp
///  [96, 256)  Address
///
```

It's very hard to look at these points of evidence and think there is any intention to have more than 256 game types to be played. I realize the issue will be heavily debated since a lot of money is on the line, so throwing this quote which summarizes escalations in a nutshell:

> "It is difficult to get a man to understand something, when his salary depends on his not understanding it." - Upton Sinclair

guhu95

> the fact Optimism *did not* plan to make use of over 256 game types
>
> any intention to have more than 256 game types to be played
>
> the intention is to keep running from 0,1, up to 255

The project clearly decided (before this contest) that game types values higher than `256` are needed. This is easy to see in these facts:

1. They've previously (in Jan) refactored GameType from uint8 to uint32, leaving no room for doubt on this aspect.
2. They've fixed the vulnerability as recommended instead of switching back to `uint8`.
3. They've accepted the finding as valid.

The team's intention (and explicit previous switch) to use `uint32` over `uint8` clearly shows the likelihood of using game types with values > 255. This removes this incorrectly considered constraint.

This finding justifies **high severity** for both the unconditionally broken key safety mechanism of `respectedGameType` allowing forged withdrawals, and the prolonged bridge DoS which would result from its mitigation.

MightyFox3

Issues predicted to arise from future integrations or updates, which aren't documented in the current documentation or README, are not considered valid. For instance, although the audit currently includes only three game types, even if the number were to exceed 255 in future implementations, such scenarios are categorized under future integrations.

> Future issues: Issues that result out of a future integration/implementation that was not mentioned in the docs/README

> or because of a future change in the code (as a fix to another issue) are not valid issues.

Referencing the Optimism official dispute game documents, the game type is clearly defined as a uint8. This definition does not suggest any future expansion beyond 255 game types, thereby rendering any inconsistencies between the code and documents as minor and of low severity.

bemic

The previous comment by @guhu95 seems to be a sufficient counterargument. Nevertheless, the fact that a 2-day-old github profile is part of the discussion is interesting.

trust1995

> The previous comment by @guhu95 seems to be a sufficient counterargument. Nevertheless, the fact that a 2-day-old github profile is part of the discussion is interesting.

You really will do anything to get the last answer in a thread, even with 0 content to add except cringeworthy ad-hominem.

bemic

Pardon, let me clarify.

I do not find the argument "*future integration/implementation/code change*" to be related. The problem stems from the current state of the codebase, where no changes are necessary.

As mentioned, few months ago the team made a very specific change to the code using a PR called "Bump GameType size to 32 bits", where they changed the type from `uint8` to `uint32`. This clearly indicates that a number > 255 is expected.

It is important to note again, that this does not necessarily mean more than 255 games. Larger type can be used to encode different game types more categorically.

You correctly pointed out that the documentation contains `uint8`. However, the documentation cannot be taken as a source of truth in cases like this one. Otherwise, projects can describe the correct and expected behavior in their documentation and use the argument "*inconsistencies between code and documentation*" as a reason to mark every problem as Low.

guhu95

Regarding:

> Issues predicted to arise from future integrations or updates, which aren't documented in the current documentation or README, are not considered valid.

First, the game type is an argument of the both the game and the factory, so can have any value depending on usage - so all `uint32`'s possible **4294967296** values are fully in scope, and not only the specific 3 values. It's `uint32`, not an enum.

Second, even if it was an enum, in this case the README explicitly allows "future integrations issues" for `OptimismPortal2`:

> **Should potential issues, like broken assumptions about function behavior, be reported if they could pose risks in future integrations, even if they might not be an issue in the context of the scope? If yes, can you elaborate on properties/invariants that should hold?**
> Yes, but this should be limited to the OptimismPortal2 contract. Contracts other than the OptimismPortal2 contract are not intended for external integrations and risks for future integrations into these contracts will likely not be considered valid.

trust1995

> You correctly pointed out that the documentation contains `uint8`. However, the documentation cannot be taken as a source of truth in cases like this one. Otherwise, projects can describe the correct and expected behavior in their documentation and use the argument "*inconsistencies between code and documentation*" as a reason to mark every problem as Low.

We've seen two strong points of evidence for source of truth - the in-code documentation of GameType and the docs page. On the other hand we see a commit bumping GameType to uint32, without adding any game types. It seems speculative to infer they plan to use larger values, contest rules state we need to give project the assumption of competence in cases like these. For impact to occur, the following has to occur:

- Optimism must intend to creative game types of uint8
- Future audits of the codebase with the new game type **must miss** a bug that is **directly in scope**
- The issue must be missed by the extremely detailed test suite ran by Optimism
- The mismatched game type (the phantom game) must not be tracked, or must have a **second unrelated vulnerability** allowing to use it for proofs
- Finally the air-gap protections must be bypassed, a fact which reduced to Low many other submissions.

  > First, the game type is an argument of the both the game and the factory, so can have any value depending on usage - so all `uint32`'s possible **4294967296** values are fully in scope, and not only the specific 3 values. It's `uint32`, not an enum.

Nope, not anything that can be misconfigured by an admin can be viewed as in-scope. That's an indefensible statement which, if correct, would inflate any contest by dozens of useless findings.

nevillehuang

1. It is not impossible to ever reach over 255 gametypes, given any possible incorrect resolution logic will also force a game type upgrade, however I believe the likelihood is low. Since the root cause is in a non-game contract, based on agreed upon scope and low likelihood, I believe medium severity is appropriate, as no safety mechanism is bypassed.
2. I don't think we can assume the behavior of off-chain mechanisms here that act as a safety mechanism, since it is explicitly mentioned as out of scope and that such scenarios will always be monitored comprehensively.

   Off-chain mechanisms exist as part of the system but are not in scope for this competition. Assume that comprehensive monitoring exists that will detect most obviously detectable malicious activity.

zobront

@nevillehuang Making sure you've seen this comment from OptimismPortal2:

> // The game must have been created after `respectedGameTypeUpdatedAt`. This is to prevent users from creating // invalid disputes against a deployed game type while the off-chain challenge agents are not watching.

You should check with the Optimism team about this if you're unclear. This situation is explicitly not being watched, and therefore is the exact kind of bypass this whole contest was designed to detect.

If they agree that this bypasses the safety mechanism, I can't see how this could be anything except High Severity.

guhu95

@nevillehuang in addition to the above consideration of off-chain watchers, please also consider the DoS impact of mitigation described above in https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84#issuecomment-2073880067.

I am no expert on Sherlock rules, but to me the DoS impact appears to also qualify for high severity.

MightyFox3

1. It is not impossible to ever reach over 255 gametypes, given any possible incorrect resolution logic will also force a game type upgrade, however I believe the likelihood is low. Since the root

cause is in a non-game contract, based on agreed upon scope and low likelihood, I believe medium severity is appropriate, as no safety mechanism is bypassed.

2. I don't think we can assume the behavior of off-chain mechanisms here that act as a safety mechanism, since it is explicitly mentioned as out of scope and that such scenarios will always be monitored comprehensively.

   Off-chain mechanisms exist as part of the system but are not in scope for this competition. Assume that comprehensive monitoring exists that will detect most obviously detectable malicious activity.

Only three games are currently implemented, even though there are over 255 game types planned for the future. This does not apply to the existing codebase. Thank you.

guhu95

1. It is not impossible to ever reach over 255 gametypes, given any possible incorrect resolution logic will also force a game type upgrade, however **I believe the likelihood is low**. Since the root cause is in a non-game contract, based on agreed upon scope and **low likelihood**

@nevillehuang since it also might have been lost in the long discussion, I'd like to point out again the fact the Optimism explicitly decided to switch from uint8 to uint32. Would you not agree that this directly establishes the likelihood as likely? Why would they switch from `uint8` to `uint32` if they didn't consider it necessarily needed and therefore likely?

Furthermore, as any value above max `uint8` may trigger the bug, any "next" game can cause this, without having to go through 255 game types before that.

Please reconsider your view of the likelihood of a game type with value > 255, especially given Optimism's explicit switch away from `uint8`.

To sum up, I see three independent arguments for high being presented:

1. Possible bypass of safeguards as documented by the team, and pointed out in https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84#issuecomment-2081733431
2. The likelihood argument discussed throughout the issue, but mostly summed up in https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84#issuecomment-2078702993 (and in the current comment)

3. Severe and prolonged DoS due to mitigation as presented in https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84#issuecomment-2073880067

nevillehuang

@guhu95

This depends on **255 distinct unique gametype**, NOT 255 FDG games of the same type. My understanding is that to reach this point, an additional 250+ game types must have been introduced from new game types or game type switches (such as due to resolution bug logic or any other game bug). The assumptions of sequential/non-sequential gaming Ids can go both ways.

I think the severity here comes down to whether or not the off-chain watching mechanism is bypassed, which seems to be so as indicated by code comments here implying so. There is conflicting statements per contest details stated here, that states off-chain mechanisms are out of scope and is assumed to be comprehensive enough. If the off-chain mechanism is confirmed to be bypassed, then I agree with high severity.

guhu95

@nevillehuang

> This depends on 255 distinct unique gametype
>
> My understanding is that to reach this point, an additional 250+ game types must have been introduced

Please help me understand why all of `2..254` must be assumed to be used before using any of the `256..4294967295` values.

1. There's no requirement in the code for sequential game types.
2. The existing code deploys the factory with 3 types that are already non sequential: 0, 1, 255.
3. A value like 0x4200, 0x1000 or 0x42000001, can be the very next game type to be used. Such semantic "versioning" or "namespacing" is both highly practical (reduces chances of errors) and already common (OP predeploys, chainIds, opcodes).
4. If "using up" all first 256 games types would be the anticipated approach, there would be little need to deliberately switch from `uint8` to `uint32`.

Using all of `0..255` before ever touching the `256..4294967295` range seems like the least likely scenario. It's like having a huge fridge, but insisting to keep cramming everything into it's tiniest compartment (of just 0.0000059% of available space).

nevillehuang

@guhu95 I can see your point, I just believe it has no relevance to considering the issues severity, and that the focus should be on whether the safety mechanisms are bypassed or not.

trust1995

> I think the severity here comes down to whether or not the off-chain watching mechanism is bypassed, which seems to be so as indicated by code comments here implying so. There is conflicting statements per contest details stated here, that states off-chain mechanisms are out of scope and is assumed to be comprehensive enough. If the off-chain mechanism is confirmed to be bypassed, then I agree with high severity.

The statements are not conflicting. The rules state very clearly that off-chain monitoring is OOS and assumed trustable. Airgaps must therefore come from the code itself. The comment linked to explains an added validation step in the code, which is not bypassed. I would appreciate answers to the detailed arguments raised here.

0xjuaan

The following comments (referred to by @trust1995 previously) clearly state that the `gameId` will only be represented by 1 byte (the first 8 bits of the uint32). From that, it can be concluded that the protocol does not intend to have more than 256 different game types.

Based on this documentation provided, casting from uint32 to uint8 is a safe and correct way to obtain the `gameId`.

Doesn't this clearly make the submission invalid? Please let me know if I am missing something.

Me and a lot of other people would have submitted this issue if it wasn't for the following documentation in DisputeTypes.sol:

```
48    /// @notice A `GameId` represents a packed 1 byte game ID, an 11 byte timestamp, and a 20 byte address.
49    /// @dev The packed layout of this type is as follows:
50    /// ┌────────────┬────────────┐
51    /// |    Bits    |   Value    |
52    /// ├────────────┼────────────┤
53    /// | [0, 8)     | Game Type  |
54    /// | [8, 96)    | Timestamp  |
55    /// | [96, 256)  | Address    |
56    /// └────────────┴────────────┘
```

bemic

> .. gameId will only be represented by 1 byte (the first 8 bits of the uint32). .. casting from uint32 to uint8 is a safe and correct way to obtain the gameId.

I see one fact wrong in your comment @0xjuaan. `GameId` is `bytes32` (256 bits) not `uint32` (32 bits). The casting is performed on `GameType` which is `uint32`.

We can see in the PR with the fix that casting and this part of in-code documentation was updated:

```
[0, 32) Game Type
[32, 96) Timestamp
```

0xjuaan

Oh ok yeah that makes sense (I got GameId and GameType mixed up). This is a great finding in that case!

thanks @bemic for clarifying

guhu95

@nevillehuang the last part of that README sentence is important for this discussion:

> Assume that comprehensive monitoring exists that will detect `most obviously detectable` malicious activity.

I understand ***"most obviously detectable"*** to mean that monitoring should be assumed thorough and reasonably scoped, but NOT all-seeing and all-validating.

This ***"most obviously detectable"*** also resolves the conflict with the ***"while the off-chain challenge agents are not watching"*** comment. It presumes "blindspots", like monitoring "all games all the time", that require on-chain logic, which was the focus of the contest, and which this bug thoroughly breaks.

To me this bug's impact is highly non-obvious and so has an unacceptable risk of bypassing the safety measures.

Evert0x

Let me state some of the facts that this discussion highlighted

IF the game types are sequential, 250+ new game types must be created before the bug gets triggered. ELSE, the game types are not sequential; the bug could trigger when the next game type is created.

In both scenarios, the bug is only activated by a specific external condition, introducing new game type(s). This trigger condition is why I believe **Medium severity** should be assigned.

Also, for the record

At the time of the audit, the following information was NOT KNOWN:

- If the team intends to deploy 250+ new game types
- If the new game types are going to be deployed in a sequential way

The following information was KNOWN

- The three defined game types are 0, 1, and 255

zobront

@Evert0x I'm not sure how you think about likelihood vs severity, but for what it's worth, I see this as:

1) Agree that it's not extremely likely. I agree with @guhu95 that it is a clear possibility based on the Optimism team's actions, but clearly isn't something that would immediately be vulnerable.
2) But the purpose of this contest is to make sure the safeguards are solid against all possible risks with the games, and all the external conditions required for this to happen would come from games having issues. If this contest said "assume games can be exploited" (which is what disqualified so many other issues), that is the only assumption needed for this to be vulnerable.
3) The outcome is not just "bad" but catastrophic: all could be stolen from the OptimismPortal (not including ERC20s in the Optimism bridge, plus all assets bridged to Base, Blast, etc if they follow this upgrade).

trust1995

> 2. If this contest said "assume games can be exploited" (which is what disqualified so many other issues), that is the only assumption needed for this to be vulnerable.

Yet at the same breath, you don't bypass the airgap, which disqualified so many other issues. Yes you can make the argument that off-chain setups would not necessarily detect it, but I think that's a diversion tactic because since the dawn of security contests the scope was strictly on-chain security, which remains intact.

zobront

The airgap would be bypassed and all the funds from the bridge would be vulnerable.

I'm not going to play definition games here. I'm talking about what would happen in reality with users funds.

On Thu, May 2, 2024 at 10:52 AM Trust **@**.***> wrote:

> 2. If this contest said "assume games can be exploited" (which is what disqualified so many other issues), that is the only assumption needed for this to be vulnerable.

> Yet at the same breath, you don't bypass the airgap, which disqualified so many other issues. Yes you can make the argument that off-chain setups would not necessarily detect it, but I think that's a diversion tactic because since the dawn of security contests the scope was strictly on-chain security, which remains intact.
>
> — Reply to this email directly, view it on GitHub https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/84#issuecomment-2090865347, or unsubscribe https://github.com/notifications/unsubscribe-auth/ABL3ULEGKDC4LXLA4GJZCFLZAJOKBAVCNFSM6AAAAABFXP4F72VHI2DSMVQWIX3LMV43OSLTON2WKQ3PNVWWK3TUHMZDAOJQHA3DKMZUG4 . You are receiving this because you were mentioned.Message ID: **@**.*** com>

guhu95

> At the time of the audit, the following information was NOT KNOWN:
>
> - If the new game types are going to be deployed in a sequential way

In my understanding, there is overwhelming evidence for non-sequential, so it was "known" at the time of the audit (and was detailed in my duplicate). Forgive me for reiterating that evidence:

1. No requirement in the code.
2. Exiting games already non sequential: 0, 1, 255.
3. Values like 0x4200, 0x1000 or 0x42000001, are frequently used in practice. This semantic "versioning" or "namespacing" is both highly practical (reduces chances of errors) and already common (OP predeploys, chainlds, opcodes).
4. If "sequential" would be the anticipated approach, there would be no need to switch from uint8 to uint32.

By analogy, "sequential", is like after upgrading to a huge new fridge, insisting to keep cramming everything into it's tiniest compartment (of just 0.0000059% of available space). It is overwhelmingly implausible.

Given this evidence, non-sequential must be assumed, therefore is not an external condition, but the default assumption. Any new game type with last byte 0x00, 0x01, or 0xff will collide with the existing ones.

@Evert0x

I propose to amend the "known facts" like so:

1. New game types are certain to be used. The current game implementation is not final - it's assumed WIP at the time of the audit (to the point of being

OOS), and the only way to use a new game implementation is via game types.

2. The game types are overwhelmingly likely to be non-sequential.

Evert0x

> IV. How to identify a high issue:
>
> 1. Definite loss of funds without (extensive) limitations of external conditions.

This is the requirement for high. It's clear that this issue requires external conditions to materialize.

I don't disagree with the points you listed, but I don't see it as an argument to assign High Severity.

zobront

@Evert0x Just to make sure I understand your point: In a contest where the explicit instructions were to assume that games could be broken, where any time a game is broken it must be incremented by at least 1, you think it's a "extensive" limitation that 256 games are reached?

I'm not valuing the fact that the hack is in the billions of dollars at all (which obviously should be weighted), but just on the definition above, I'm not positive I understand your disagreement?

guhu95

@Evert0x

> I don't disagree with the points you listed, but I don't see it as an argument to assign High Severity.

Thanks for recognizing my arguments. However, if you do agree with that reasoning, it directly leads to these conclusions: 1) game types will definitely be updated; 2) they definitely be updated such that the issue will happen after only a handful of updates (between 1 and "a few"):

- Even if we assume that game type will be updated initially once a month.
- And the update increment is either `+1` with 50%, or `bump to new version` with 50%.
- It means that the probability of the bug is 50% after one month, 75% after two months, ... **99.975% after one year**. Such probabilities cannot be "extensive limitations", and instead are "nearly certain".

One may choose a different `P(jump)` and different `bumps-in-first-year` , but with `1 - [1 - P(jump)]^(bumps-in-first-year)` it's very difficult to justify numbers that will result in anything corresponding to "extensive limitation".

This not even considering that this can affect the OP stack (and not just Optimism), so affects up to already 19 (!?) rollups in its first year (with ~19B TVL). This multiplies the probability by the number of OP Stack rollups using this system.

nevillehuang

This is information from the op handbook

> Off-chain monitoring can observe FaultDisputeGame contract resolutions and trigger a fallback mechanism to pause the system and update the respected game type if a game resolves incorrectly.

The issue here is highlighting a possible bypass in the off-chain monitoring mentioned above because of a code comment highlighted of how it is presumably supposed to work. But the contest details stated it is OOS.

> Off-chain mechanisms exist as part of the system but are not in scope for this competition.

My opinion is since there is still an issue arising from an inscope root cause that results in incorrect resolution and thus finalization of withdrawals but off-chain monitoring mechanism is assumed to be comprehensive and not be bypassed, medium severity is appropriate since no airgap/safety mechanism is assumed to be breached

zobront

@nevillehuang I would agree with this if the issue discovered was in the off chain mechanisms (ie if the issue highlighted a fix that should be made to the offchain mechanisms).

But that is not the case. The off chain piece behaves perfectly appropriately and exactly as documented. I am pointing out no fault in that part of the system.

What I am pointing out is that as a result of this CORRECT behavior, the on chain contract is highly vulnerable (airgap bypassed).

> off-chain monitoring mechanism is assumed to be comprehensive

I don't think this is right. When a part of the system is marked as out of scope, it means it is assumed to act correctly and according to its specifications. It doesn't mean it is assumed to magically act in ways that it actually doesn't to save the day when the in scope system has an error.

nevillehuang

@zobront How would the airgap be bypassed when the off-chain monitoring is presumed to have caught the incorrect resolution, where like you mentioned it

means the off-chain monitoring mechanism is assumed to have acted correctly and according to its specifications?

guhu95

The full scope quote is this (emphasis mine):

> Assume that comprehensive monitoring exists that will detect **MOST OBVIOUSLY** detectable malicious activity.

It's "specification" is not that it will catch **anything** and **everything**.

I this case, it will likely **NOT be monitoring** the games that are NO LONGER being used, since this is **obviously** not needed. Proving a withdrawal using a game that is **no longer being used** being the root cause of the issue here.

This is further supported by this code comment that assumes games that are not **currently** being used are not being monitored.

> // The game must have been created after respectedGameTypeUpdatedAt. This is to prevent users from creating // invalid disputes against a deployed game type while the off-chain challenge agents are not watching.

zobront

Its specification is that it accurately monitors the currently set game type and catches all exploits in that game type.

It does that action correctly, so there is no issue with the off chain mechanism.

But because of the on chain issue discovered, the airgap will be bypassed (because the off chain mechanism is not watching the other game type).

My point is that since the off chain mechanism is out of scope, we should NOT reward issues in this mechanism. But this doesn't mean we can assume it has different behavior that magically solves all on chain issues.

To summarize: If we assume the off chain mechanism works exactly as specified (which is reasonable since it's out of scope), then the on chain issue will bypass the airgap, so that's how it would be most fair to judge.

On Tue, May 7, 2024 at 3:58 PM 0xnevi **@**.***> wrote:

> @zobront https://github.com/zobront How would the airgap be bypassed when the off-chain monitoring is presumed to have caught the incorrect resolution, where like you mentioned it means the off-chain monitoring mechanism is assumed to have acted correctly and according to its specifications?
>
> — Reply to this email directly, view it on GitHub https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issu

> es/84#issuecomment-2099299989, or unsubscribe https://github.com/notifications/unsubscribe-auth/ABL3ULAFT2MO5GJTVSFCBFLZBE6ATAVCNFSM6AAAAABFXP4F72VHI2DSMVQWIX3LMV43OSLTON2WKQ3PNVWWK3TUHMZDAOJZGI4TSOJYHE . You are receiving this because you were mentioned.Message ID: **@**.*** com>

trust1995

Discussing an airgap in an off-chain context is useless once those components were defined as OOS. The only source of truth for airgap / not an airgap is the on-chain state.

This discussion is orthogonal to the 256 game requirement, which by itself is an extensive limitation.

guhu95

@nevillehuang

> How would the airgap be bypassed when the off-chain monitoring is presumed to have caught the incorrect resolution

The resolution of the OTHER game **may be fully correct** according to that game's implementation. It is another game entirely, so:

- it could be an older version of the game that is vulnerable - which is why it's no longer used. And this is also why this game won't be monitored - it makes no sense to challange if it's buggy.
- it could be the permissioned version (as shown, the permissioned one is added to the factory on deployment).
- it could be the alphabet testing game (which is added to the factory on deployment). no reason to monitor this one
- it could be a game used by a different OP stack rollup, proving withdrawals for it, that would allow fully correct resolutions of the game to be used to replay withdrawal from that other rollup in Optimism.
- ...

The assumption that the OTHER game is possible to challange because it's resolved incorrectly is not needed here. A just as likely scenario is that the OTHER game is resolved correctly, but the bridge MUST NOT be using it.

Evert0x

Although for a different reason, I believe it's right to assign Medium as well.

High is reserved for unrestricted losses. Watsons were to assume that the game's resolution logic was broken, not that game types were added regularly.

I still believe it's an extensive limitation for a new game type to get (created, audited, and) deployed with a specific ID, as that's the trigger that can potentially cause this catastrophic bug.

guhu95

> Watsons were to assume that the game's resolution logic was broken, not that game types were added regularly.

@Evert0x But a broken game is resolved by updating the game type. Doesn't the assumption **"the game's resolution logic is broken"** unavoidably and directly includes the assumption of updating the game type?

How can there be an extensive limitation if one thing directly causes the other?

Evert0x

Result: Medium Has Duplicates

@guhu95 assuming "the game's resolution logic is broken" and assuming "the team will continuously deploy new game types" are two different things.

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- zobront: rejected

guhu95

@Evert0x your response:

> @guhu95 assuming "the game's resolution logic is broken" and assuming "the team will continuously deploy new game types" are two different things.

Did not answer either of the specific questions I've asked.

> Doesn't the assumption "the game's resolution logic is broken" unavoidably and directly includes the assumption of updating the game type?
>
> How can there be an extensive limitation if one thing directly causes the other?

I can see the escalation shows as resolved now, but https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/201 was re-opened 3 days after escalation resolution, so not sure how to interpret the resolution update.

Issue M-2: Fault game factory can be manipulated to DOS game type using malicious `l2BlockNumber`

Source: https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90

Found by

0xdeadbeef, GalloDaSballo, Trust, bin2chen, ctf_sec, fibonacci, haxatron, nirohgo, obront, tallo, zigtur

Summary

All new games are proven against the most recent L2 block number in the `ANCHOR_STATE_REGISTRY`. This includes requiring that the block number we are intending to prove is greater than the latest proven block number in the registry. Due to insufficient validations of the passed L2 block number, it is possible for a user to set the latest block to `type(uint256).max`, blocking all possible future games from being initialized.

Vulnerability Detail

New games are created for a given root claim and L2 block number using the factory, by cloning the implementation of the specified game type and passing these values as immutable args (where `_extraData` is the L2 block number).

```
proxy_ = IDisputeGame(address(impl)).clone(abi.encodePacked(_rootClaim,
↪  parentHash, _extraData));
proxy_.initialize{ value: msg.value }();
```

As a part of the initialize function, we pull the latest confirmed `root` and `rootBlockNumber` from the `ANCHOR_STATE_REGISTRY`. These will be used as the "starting points" for our proof. In order to confirm they are valid starting points, we require that the L2 block number we passed is greater than the last proven root block number.

```
(Hash root, uint256 rootBlockNumber) = ANCHOR_STATE_REGISTRY.anchors(GAME_TYPE);

// Should only happen if this is a new game type that hasn't been set up yet.
if (root.raw() == bytes32(0)) revert AnchorRootNotFound();

// Set the starting output root.
startingOutputRoot = OutputRoot({ l2BlockNumber: rootBlockNumber, root: root });
```

```
// Do not allow the game to be initialized if the root claim corresponds to a
↪  block at or before the
// configured starting block number.
if (l2BlockNumber() <= rootBlockNumber) revert UnexpectedRootClaim(rootClaim());
```

However, the L2 block number we pass does not appear to be sufficiently validated. If we look at the Fault Dispute Game, we can see that disputed L2 block number passed to the oracle is calculated using the `_execLeafIdx` and does not make any reference to the L2 block number passed via `extraData`:

```
uint256 l2Number = startingOutputRoot.l2BlockNumber +
↪  disputedPos.traceIndex(SPLIT_DEPTH) + 1;

oracle.loadLocalData(_ident, uuid.raw(), bytes32(l2Number << 0xC0), 8,
↪  _partOffset);
```

This allows us to pass an L2 block number that is disconnected from the proof being provided.

After the claim is resolved, we update the `ANCHOR_STATE_REGISTRY` to include our new root by calling `tryUpdateAnchorState()`.

```
function tryUpdateAnchorState() external {
    // Grab the game and game data.
    IFaultDisputeGame game = IFaultDisputeGame(msg.sender);
    (GameType gameType, Claim rootClaim, bytes memory extraData) =
↪  game.gameData();

    // Grab the verified address of the game based on the game data.
    // slither-disable-next-line unused-return
    (IDisputeGame factoryRegisteredGame,) =
        DISPUTE_GAME_FACTORY.games({ _gameType: gameType, _rootClaim: rootClaim,
↪  _extraData: extraData });

    // Must be a valid game.
    require(
        address(factoryRegisteredGame) == address(game),
        "AnchorStateRegistry: fault dispute game not registered with factory"
    );

    // No need to update anything if the anchor state is already newer.
    if (game.l2BlockNumber() <= anchors[gameType].l2BlockNumber) {
        return;
    }

    // Must be a game that resolved in favor of the state.
```

```
    if (game.status() != GameStatus.DEFENDER_WINS) {
        return;
    }

    // Actually update the anchor state.
    anchors[gameType] = OutputRoot({ l2BlockNumber: game.l2BlockNumber(), root:
↪   Hash.wrap(game.rootClaim().raw()) });
}
```

As long as the L2 block number we passed is greater than the last proven one, we update it with our new root. This allows us to set the `ANCHOR_STATE_REGISTRY` to contain an arbitrarily high `blockRootNumber`.

If we were to pass `type(uint256).max` as this value, it would be set in the `anchors` mapping, and would cause all other games to fail to initialize, because there is no value they could pass for the L2 block number that would be greater, and would therefore fail the check described above.

Proof of Concept

The following test can be dropped into `DisputeGameFactory.t.sol` to demonstrate the vulnerability:

```
function testZach_DOSWithMaxBlockNumber(uint256 newBlockNum) public {
    // propose a root with a block number of max uint256
    bytes memory maxBlock = abi.encode(type(uint256).max);
    IDisputeGame game = disputeGameFactory.create(GameType.wrap(0),
↪   Claim.wrap(bytes32(uint(1))), maxBlock);
    assertEq(game.l2BlockNumber(), type(uint256).max);

    // when the game passes, it's saved in the anchor registry
    vm.warp(block.timestamp + 4 days);
    game.resolveClaim(0);
    game.resolve();

    // now we can fuzz newly proposed block numbers, and all fail for the same
↪   reason
    bytes memory maxBlock2 = abi.encode(newBlockNum);
    vm.expectRevert(abi.encodeWithSelector(UnexpectedRootClaim.selector, 1));
    disputeGameFactory.create(GameType.wrap(0), Claim.wrap(bytes32(uint(1))),
↪   maxBlock2);
}
```


Impact

For no cost, the factory can be DOS'd from creating new games of a given type.

Code Snippet

https://github.com/sherlock-audit/2024-02-optimism-2024/blob/main/optimism/packages/contracts-bedrock/src/dispute/FaultDisputeGame.sol#L528-L539

https://github.com/sherlock-audit/2024-02-optimism-2024/blob/main/optimism/packages/contracts-bedrock/src/dispute/AnchorStateRegistry.sol#L59-L87

Tool used

Manual Review

Recommendation

In order to ensure that ordering does not need to be preserved, `ANCHOR_STATE_REGISTRY` should store a mapping of claims to booleans. This would allow users to prove against any proven state, instead of being restricted to proving against the latest state, which could be manipulated.

Discussion

smartcontracts

Factually valid although the impact here isn't different than having any game resolve incorrectly which would poison the `AnchorStateRegistry` and require the game type to be changed.

nevillehuang

Based on scoping details below, I believe this issue is valid and in-scope of the contest, as the root cause stems from the lack of a sanity check within the dispute game factory allowing large `l2BlockNumber` to be appended

https://docs.google.com/document/d/1xjvPwAzD2Zxtx8-P6UE69TuoBwtZPbpwf5zBHAvBJBw/edit

The potential to block the entire fault proofs system entirely by preventing further creation of new games is significant, so I believe it warrants high severity given the potential to block withdrawals from an OP bridge. Although the admin can temporarily resolve this by switching game type, I believe it is not a valid solution given the attack can be easily repeated.

0xjuaan

just a thought @nevillehuang:

> I believe it is not a valid solution given the attack can be easily repeated

Is this actually true? If they change the game type, the new FaultDisputeGame implementation will be fixed and won't have this vulnerability so the attack can't be repeated. Because of that, the sponsor's comment seems to make the most sense and calling this a high severity issue is quite sus.

Evert0x

Forwarding a comment from the protocol team

--

This issue isn't valid because the decoupling of the L2 block number that's determined during output bisection and the one on the root claim is intentional. They claim that you can propose an output belonging to block n (so, right hash), but for the wrong block number in the future (i.e. n + 1). The challenger would be able to challenge this, as they would see that the output at the claimed block is wrong (or that the block just doesn't exist) The program, once ran, can either show:

- The output at the given block number isn't correct (i.e. the proposed block number is part of the safe chain captured by the data available on L1 at the L1 head hash persisted when the game starts)
- The given block number cannot be derived with the data available on L1 (i.e. the block number is super far in the future, and doesn't even exist)

Essentially a proposal of this form would be invalidated by the current fault proof system so the bug itself wouldn't be possible

Haxatron

This bug operates under assumption that the FP system can cause a invalid game to be resolved as valid, and there were multiple ways to do this in the contest (see #8). If this can occur then no more dispute games can be created for the same game type which will lead to DoS. Only possible solution is to update game type as pointed out by comments above.

JeffCX

In this case, one single invalid game resolution with very large block number DOS the whole game type,

update game type does not seems to be a long term solution, there are not many game type to update.

The fix is still add proper validation for block number or the fix in this report can be used as well

> In order to ensure that ordering does not need to be preserved, ANCHOR_STATE_REGISTRY should store a mapping of claims to booleans. This would allow users to prove against any proven state, instead of being restricted to proving against the latest state, which could be manipulated.

lemonmon1984

> For no cost, the factory can be DOS'd from creating new games of a given type.

I want to note that the attacker is risking the bonds. They will likely to lose it if any honest party challenges them.

JeffCX

Whether the attacker get challenged or not is not in-scope,

the audit and report is under the assumption that the game can be resolved incorrectly

> FaultDisputeGame resolution logic is not included in the scope of this contest. Participants should assume that the FaultDisputeGame can resolve incorrectly (i.e.g, can resolve to DEFENDER_WINS when it should resolve to CHALLENGER_WINS or vice versa).

and in case the game resolved incorrectly, massive DOS for game type occurs as outlined in the report.

zobront

To share my perspective here:

TLDR: This is a difficult case, because the issue should be in scope, but the outcome that it causes is no worse than the manual fixes that would happen when the safeguards work properly.

Severity: As much as I'd like to, I can't see a justification for High. The outcome does not seem bad enough.

Scope: This does seem to meet the definitions laid out in the scope document. The issue is in the in scope contracts, and the outcome (DOS of game type) should be sufficient for a Medium. However, it is a weird dynamic because when safeguards are used, it also causes a DOS of game type, so it seems strange that the same outcome could be a valid issue.

Conclusion: My assessment is that this should remain as a valid Medium, because the contest rules didn't rule out all game type DOS, only those caused by game contract logic. That being said, I recognize this is difficult to judge and respect whatever decision the judge makes.

JeffCX

as the original well-written report highlights

> As long as the L2 block number we passed is greater than the last proven one, we update it with our new root. This allows us to set the ANCHOR_STATE_REGISTRY to contain an arbitrarily high blockRootNumber.
>
> If we were to pass type(uint256).max as this value, it would be set in the anchors mapping, and would cause all other games to fail to initialize, because there is no value they could pass for the L2 block number that would be greater, and would therefore fail the check described above.

the impact of DOS game creation and game type means no user can finalize their withdraw transaction / execution l2 -> l1 message, which is a leads to clearly loss of fund and lock of fund as multiple duplicates highlight such as #206

nevillehuang

This issue seems invalid per sponsor comments here.

Any reasons addressing sponsor comments why this should be a valid issue?

@zobront @JeffCX @Haxatron

Haxatron

Hi,

Already given my reasoning above.

I will defer to @zobront and @JeffCX for any additional comments

Acknowledge this one is quite tricky to judge.

zobront

I explained my assessment based on that sponsor comment here: https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90#issuecomment-2090639437

On Fri, May 3, 2024 at 4:02 PM Haxatron @.***> wrote:

> Hi,
>
> Already given my reasoning above.
>
> I will defer to @zobront https://github.com/zobront and @JeffCX https://github.com/JeffCX for any additional comments
>
> Acknowledge this one is quite tricky to judge.

> — Reply to this email directly, view it on GitHub https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90#issuecomment-2093758255, or unsubscribe https://github.com/notifications/unsubscribe-auth/ABL3ULAOEIBLV2L6ENQ5SL3ZAP3OPAVCNFSM6AAAAABFXP5JESVHI2DSMVQWIX3LMV43OSLTON2WKQ3PNVWWK3TUHMZDAOJTG42TQMRVGU . You are receiving this because you were mentioned.Message ID: **@**.*** com>

JeffCX

> Forwarding a comment from the protocol team
>
> --
>
> This issue isn't valid because the decoupling of the L2 block number that's determined during output bisection and the one on the root claim is intentional. They claim that you can propose an output belonging to block n (so, right hash), but for the wrong block number in the future (i.e. n + 1). The challenger would be able to challenge this, as they would see that the output at the claimed block is wrong (or that the block just doesn't exist) The program, once ran, can either show:
>
> - The output at the given block number isn't correct (i.e. the proposed block number is part of the safe chain captured by the data available on L1 at the L1 head hash persisted when the game starts)
> - The given block number cannot be derived with the data available on L1 (i.e. the block number is super far in the future, and doesn't even exist)
>
> Essentially a proposal of this form would be invalidated by the current fault proof system so the bug itself wouldn't be possible

Emm seems like this is saying that the game cannot be resolved incorrectly....

but during judging, we mark the game resolution logic out of scope and use the argument to invalid many issue

It is contradictory to use the argument "incorrect game resolution out of scope" to invalid many other issue.

while use the argument "game cannot be resolved incorrectly" to invalid this issue.

the comments strongly contradicts the readme as well:

from read me

> Participants should assume that the FaultDisputeGame can resolve incorrectly (i.e.g, can resolve to DEFENDER_WINS when it should resolve to CHALLENGER_WINS or vice versa).

the report is perfect derivation from the statement above without worrying about the game dispute logic...

> Participants should assume that the FaultDisputeGame can resolve incorrectly (i.e.g, can resolve to DEFENDER_WINS when it should resolve to CHALLENGER_WINS or vice versa).

then I think the original judging decision still stands.

guhu95

I actually don't understand why the sponsor's claim is correct. If it does make sense to anyone else, can someone please explain?

What I understand they're saying is that the game can't resolve correctly in this way. But I don't understand how that is possible if the `extraData`'s `l2Blocknumber` is never actually used by the proof system? The only user of that value is `AnchorStateRegistry` and it doesn't validate it.

There is no check I can see that checks that `extraData`'s `l2Blocknumber` is actually in the `rootClaim` in any way.

From the point of view of the proof system, the block number it is using is unrelated to the one later being passed to `AnchorStateRegistry`.

If so, isn't it the case that anyone can always frontrun the legitimate proposals and use the legitimate data, but pass in `type(uint).max` as l2BlockNumber? Isn't that a perpetual DoS of the system?

What am I missing here?

Haxatron

If I am not wrong, it will be used in VM.step() after adding the block number into the preimage oracle via addLocalData()

On Sun, 5 May 2024, 10:04 Guhu, @.***> wrote:

> I actually don't understand why the sponsor's claim https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90#issuecomment-2076645104 is correct. If it does make sense to anyone else, can someone please explain?
>
> What I understand they're saying is that the game can't resolve correctly in this way. But I don't understand how that is possible if the extraData's l2Blocknumber is never actually used by the proof system? The only user of that value is AnchorStateRegistry and it doesn't validate it.
>
> There is no check I can see that checks that extraData's l2Blocknumber is actually in the rootClaim in any way.

> From the point of view of the proof system, the block number it is using is unrelated to the one later being passed to AnchorStateRegistry.
>
> If so, isn't it the case that anyone can always frontrun the legitimate proposals and use the legitimate data, but pass in type(uint).max as l2BlockNumber? Isn't that a perpetual DoS of the system?
>
> What am I missing here?
>
> — Reply to this email directly, view it on GitHub https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90#issuecomment-2094551960, or unsubscribe https://github.com/notifications/unsubscribe-auth/ASHOYPIERRIR4LDWW7ZS5YTZAWHSBAVCNFSM6AAAAABFXP5JESVHI2DSMVQWIX3LMV43OSLTON2WKQ3PNVWWK3TUHMZDAOJUGU2TCOJWGA . You are receiving this because you were mentioned.Message ID: **@**.*** com>

guhu95

@Haxatron here?. It doesn't seem to be using the l2number value from `extraData` here (or anywhere else in the game itself).

It does use the `l1head()` from `extraData` above but not the L2 number.

Haxatron

Apologies, you are correct, the L2 block number referenced on that line is the anchor root block number rather than the l2 block number passed via the extraData. Perhaps, this requires more clarification from protocol team...

guhu95

@Evert0x @nevillehuang it looks like the sponsor's argument for this being invalid is not well understood. The finding is also marked "won't fix", so if it is valid, it is not mitigated. It would appear that a mitigation would be needed at both game level and at the registry level, as fixing one without the other would leave the other vulnerable.

Can @smartcontracts maybe have another look at this, and possibly address the questions in https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/90#issuecomment-2094551960?

Evert0x

After a discussion with the protocol it's clear that this issue should be a valid Medium

trust1995

So permissionless shutdown of withdrawals/messaging until redeploy (Freeze of Funds of 2 weeks) is considered a Medium on Sherlock? Was the magnitude of

effect this would have on the Optimism ecosystem considered?

A 1-hour shutdown of Blast was the most talked about incident for months, how would a 2-week FoF be interpreted?

guhu95

In addition to the prolonged DoS, the DoS appears to be **repeatable**, and require changes to the registry and portal, and not just updating the game.

In order for the **DoS to NOT be repeatable**, the fix must be possible at the **game level**, such that updating the game type is sufficient. But it doesn't appear to be the case:

Please see these (new) fix PRs in OP dealing with this issue: https://github.com/ethereum-optimism/optimism/pull/10431/files, https://github.com/ethereum-optimism/optimism/pull/10434/files.

They add extensive changes to both the ASR and the Portal to deal with the l2BlockNumber issue. The changes to the game are minimal, and it appears that this issue is NOT fixable by just updating the game implementation.

[I suspect this is because in the game, the l2blocknumber is PART of disputed L2 output state, so is not a mutually agreed on external input, unlike the L1 number, which comes directly from the L1 `block.number` itself. The dispute is about the rootClaim output state, not about the input l2blocknumber and something about the proving setup prevents it from being used this way (or from the game being able to distinguish which one of these inputs is incorrect). But this is just my limited and possibly wrong understanding]

Summary: because the issue is not fixable by updating ONLY the game, and an upgrade of the ASR and Portal are needed, the safety measures are inadequate and the DoS WITHOUT the full fix is repeatable.

spearfish5609

> most talked about incident for months

this incident has not even happened 2 months ago and people cared about it for a few days at most

nirohgo

> After a discussion with the protocol it's clear that this issue should be a valid Medium

@Evert0x can you elaborate why? (for those of us who weren't on that discussion)

Evert0x

The justification for Medium severity is as follows

> The Proxy Admin Owner is a TRUSTED role that can:
>
> - Upgrade all smart contracts that sit behind a Proxy contract.
> - Set the implementation contract for any dispute game type within the DisputeGameFactory.
> - Modify the initial bond cost for any dispute game type within the DisputeGameFactory.
> - Remove ETH from the DelayedWETH contract.
>
> The Proxy Admin Owner is assumed to be honest and responsive with an SLA of 72 hours.

As stated in the README, part of the security model is a honest and responsive admin that can recover from a DoS within 72 hours.

> **Are there any off-chain mechanisms or off-chain procedures for the protocol (keeper bots, arbitrage bots, etc.)?**
>
> Off-chain mechanisms exist as part of the system but are not in scope for this competition. Assume that comprehensive monitoring exists **that will detect most obviously detectable malicious activity.**

The supplied block being higher than the actual block number in the EVM is obviously detectable malicious activity.

In conclusion, once the proposal is detected, the Proxy Admin Owner is trusted to be responsive within 72 hours and is able to switch the game type to a permissioned implementation within a new AnchorStateRegistry to mitigate the DoS.

Note: It's important to note that there's a difference between switching the game type (which invalidates all withdrawals with that game type) and switching the implementation (which does not invalidate withdrawals).

guhu95

@Evert0x but can switching to a permissioned implementation be considered final "recovery"?

If assumed permanent - it permanently breaks core functionality (no fraud proofs from that point).

If assumed temporary - it only postpones the switch of the game type and the withdrawals DoS.

trust1995

This downplayed take is plagued with intellectual dishonesty.

> The supplied block being higher than the actual block number in the EVM is obviously detectable malicious activity.

If it was obvious as something to look for, Opt would have validated the l2BlockNumber is the same as the VM block number. Detection is highly unprobable. Please provide the defender off-chain code to show awareness of this vulnerability. Once again the benefit of the doubt is given to an opaque statement by the sponsor and against honest Watsons. For fairness of discussion, it must be assumed Opt is aware only at the moment games cannot be created.

> able to switch the game type to a permissioned implementation within a new AnchorStateRegistry to mitigate the DoS.

For the past month where Optimism had access to the repo, their suggested fix was moving to a new game type, confirming the 2 week DOS. Only couple of days ago came the idea of overriding the same game type to avoid the DOS. Using this to reduce severity is unacceptable. It essentially extended their 3 day SLA to 1 month, letting them theorize over best response over a tremendously long time and then argue the optimized response would be what they would be rolling with on day 1. Clear intellectual dishonesty.

Additionally, from an air-gap perspective (up to High according to the README), the resolution and updating of the anchor state registry is instantaneous, making new withdrawals impossible from day 0 and bypassing intended airgaps. A combination of FoF impact (High impact) + airgap bypass (High focus of contest) + permissionless attacker (High likelihood) makes it verry clear high severity is in order.

I will also state that over the past week Optimism has catapulted a variety of arguments against the submission which were technically proven wrong, showing they have no problem misrepresenting an issue or its characteristics in order to reduce its severity.

spearfish5609

I dont know why the optimism team is trying to find some weird loopholes to argue for downgrade if they could just use official sherlock docs to justify it:

according to https://docs.sherlock.xyz/audits/judging/judging#v.-how-to-identify-a-medium-issue : `Breaks core contract functionality, rendering the contract useless or leading to loss of funds.` is a medium

about DOS: https://docs.sherlock.xyz/audits/judging/judging#iii.-sherlocks-standards requires both

1. `The issue causes locking of funds for users for more than a week`
2. `The issue impacts the availability of time-sensitive functions`

to be high severity

1 is true and 2 is questionable if we assume that admin deploys a new game type in time so funds can be recovered and users can just make new game instance, where these functions are available again.

I dont see any air-gap bypass unless we use different definitions. My understanding is that the air-gap is the delay before withdraw of funds can happen and its not possible for users to withdraw early

Evert0x

@guhu95 It's not a final recovery, but safety mechanisms are put in place first to mitigate the DoS and, secondly, to remove the DoS factor.

Evert0x

@trust1995 Forwarding from the protocol team the detection code for this case.

So in a nutshell the monitoring service is here: https://github.com/ethereum-optimism/optimism/blob/5137f3b74c6ebcac4f0f5a118b0f4909df03aec6/op-dispute-mon/mon/monitor.go#L87

This service calls out to a forecasting function which checks the L2 block number and the claim provided against the real output root for that block number: https://github.com/ethereum-optimism/optimism/blob/5137f3b74c6ebcac4f0f5a118b0f4909df03aec6/op-dispute-mon/mon/forecast.go#L69

Claimed L2 block number and output are pulled from the game's metadata: https://github.com/ethereum-optimism/optimism/blob/5137f3b74c6ebcac4f0f5a118b0f4909df03aec6/op-dispute-mon/mon/extract/extractor.go#L54

So in the case of that bug, the service would try to get the block number for the future block that doesn't exist yet, get the following error, disagree, and raise an alert: https://github.com/ethereum-optimism/optimism/blob/5137f3b74c6ebcac4f0f5a118b0f4909df03aec6/op-dispute-mon/mon/validator.go#L40

Evert0x

@spearfish5609 I don't think I'm using weird loopholes to decide on the severity of this issue.

It's not always clear if the DoS should be judged as indefinite just because the admin can recover from it. However, in this case, the language in the README makes it clear.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/ethereum-optimism/optimism/pull/10438

Issue M-3: Theft of initial bonds from proposers who are using smart wallets

Source: https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/194

Found by

GalloDaSballo, Trust

Summary

Proposal of output roots through the DisputeGameFactory from Smart Wallets is vulnerable to frontrunning attacks which will steal the initial bond of the proposer.

Vulnerability Detail

A fault dispute game is built from the factory, which initializes the first claim in the array below:

```
claimData.push(
    ClaimData({
        parentIndex: type(uint32).max,
        counteredBy: address(0),
        claimant: tx.origin,
        bond: uint128(msg.value),
        claim: rootClaim(),
        position: ROOT_POSITION,
        clock: LibClock.wrap(Duration.wrap(0),
↪   Timestamp.wrap(uint64(block.timestamp)))
    })
);
```

The sender passes a `msg.value` which equals the required bond amount, and the registered `claimant` is `tx.origin`. At the end of the game, if the claim is honest, the funds will be returned to the `claimant`.

Smart Wallets are extremely popular ways of holding funds and are used by all types of entities for additional security properties and/or flexibility. A typical smart wallet will receive some `execute()` call with parameters, verify it's authenticity via signature / multiple signatures, and perform the requested external call. That is how the highly popular Gnosis Safe operates among many others. Smart Wallets are agnostic to whoever actually called the `execute()` function, as long as the data is authenticated.

These properties as well as the use of `tx.origin` in the FaultDisputeGame make it easy to steal the bonds of honest proposals:

- Scan the mempool for calls to Gnosis `execTransaction()` or any other variants.
- Copy the TX content and call it from the attacker's EOA.
- The Smart Wallet will accept the call and send the `msg.value` to the DisputeGameFactory.
- The `claimant` will now be the attacker.
- Upon resolution of the root claim, the attacker will receive the initial bond.

Impact

Theft of funds from an honest victim who did not interact with the system in any wrong way.

Code Snippet

```
claimData.push(
    ClaimData({
        parentIndex: type(uint32).max,
        counteredBy: address(0),
        claimant: tx.origin,
        bond: uint128(msg.value),
        claim: rootClaim(),
        position: ROOT_POSITION,
        clock: LibClock.wrap(Duration.wrap(0),
↪   Timestamp.wrap(uint64(block.timestamp)))
    })
);
```

Tool used

Manual Review

Recommendation

The Factory needs to pass down the real msg.sender to the FaultDisputeGame.

Discussion

smartcontracts

This report is not entirely correct. It is not possible to "steal" funds from a wallet. Instead, it is the case that the user creating the `FaultDisputeGame` would not receive their bonds at the end of the game. Although this behavior was intentional as the contracts are meant to be used by EOAs directly and not smart contract wallets, we believe this is a valid low-severity issue and we will fix it.

trust1995

> This report is not entirely correct. It is not possible to "steal" funds from a wallet. Instead, it is the case that the user creating the FaultDisputeGame would not receive their bonds at the end of the game.

What you described is essentially stealing - an honest user's bond will be claimed by the attacker.

> Although this behavior was intentional as the contracts are meant to be used by EOAs directly and not smart contract wallets, we believe this is a valid low-severity issue and we will fix it.

- If the behavior is intentional, then why fix it?
- Also, no mention anywhere of the assumption that disputers (which are permissionless) should be EOAs, therefore we can't view that as reducing severity or OOS in any capacity.

smartcontracts

I think the implied contract is relatively clear, the user who creates the game is the `tx.origin` and not the `msg.sender`. Smart contract wallets weren't an intended user of the contracts. Either way impact is relatively limited (smallest bond size is at the initialization level). I think it's a pretty clear footgun though and should be fixed to prevent issues down the line.

smartcontracts

So actual stance here is that this is valid but low-likelihood and low-impact in practice.

sherlock-admin4

The protocol team fixed this issue in the following PRs/commits:
https://github.com/ethereum-optimism/optimism/pull/10149

nevillehuang

Based on scope details below, any issue with root cause of the issue stemming from FDG contract will be considered OOS of this contest if airgap and/or delayed WETH mechanism implemented for off-chain review of game results and bond distribution is not shown to be bypassed

https://docs.google.com/document/d/1xjvPwAzD2Zxtx8-P6UE69TuoBwtZPbpwf5zBHAvBJBw/edit

trust1995

@nevillehuang The root cause is clearly in the factory contract using an unusafe tx.origin parameter, as demonstrated in the submission. The finding is in scope.

trust1995

Escalate

The issue is in scope, because:

- The bug's origin is certainly not in the FDG's *initialize()* function - without any changes to the factory there is NO actual way to determine who the correct claimant should be. The FDG does not have the neceesary context, and the root cause is lack of sending the msg.sender of the factory as a parameter. This is further evidenced by the fact the fix changed the Factory's call
- The impact is clearly high
- Based on the following ruling, the submission must be treated as in-scope: `Issues with a root cause in the non-game contracts are IN SCOPE`

sherlock-admin2

> Escalate
>
> The issue is in scope, because:
>
> - The bug's origin is certainly not in the FDG's *initialize()* function - without any changes to the factory there is NO actual way to determine who the correct claimant should be. The FDG does not have the neceesary context, and the root cause is lack of sending the msg.sender of the factory as a parameter. This is further evidenced by the fact the fix changed the Factory's call
> - The impact is clearly high
> - Based on the following ruling, the submission must be treated as in-scope: `Issues with a root cause in the non-game contracts are IN SCOPE`

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

Agree that this issue is valid, given the root cause can be seen as stemming from the factory contract. Additionally, there is no mention whether only an EOA is allowed to interact with the contracts. Based on agreed upon scope and line drawn, I believe medium severity to be appropriate since no safety mechanisms (DelyayedWETH) is bypassed.

MightyFox3

Firstly, it's important to clarify that funds cannot be "stolen" from a wallet in the manner described. The scenario involves the user who initiates the FaultDisputeGame; they would not receive their bonds back at the conclusion of the game, which differs significantly from the notion of funds being stolen.

Regarding the vulnerabilities outlined in the original report, it seems there are misconceptions about the ease of exploiting these issues. The steps provided suggest that an attacker can simply scan the mempool, copy transaction content, and execute it from their own externally owned account (EOA). However, this overlooks critical security measures inherent in the system:

- The Gnosis smart contract requires signatures from its owners, Alice and Bob, to authorize any execution of the `execTransaction()` function. This means copying the transaction content and executing it from an attacker's EOA is not feasible unless there is a flaw in how signatures are validated, which is not typical for smart contract wallets, including the Gnosis Safe Wallet.
- Furthermore, even if Alice and Bob authorize a transaction, the claim that funds are lost is incorrect. If Alice is the transaction originator and her EOA executes the transaction, she (tx.origin) retains the ability to claim the bond. Therefore, there is no actual loss of funds.

Given these clarifications, it would be more accurate to assess the severity of the issue as low, rather than medium.

trust1995

@MightyFox3 you seem to have completely missed the meat of the exploit, so allow me to re-iterate:

> Firstly, it's important to clarify that funds cannot be "stolen" from a wallet in the manner described. The scenario involves the user who initiates the FaultDisputeGame; they would not receive their bonds back at the conclusion of the game, which differs significantly from the notion of funds being stolen.

Respectfully, when an attacker can receive a bond deposited by a victim's account without proofing their claim was invalid, it is considered a theft of funds.

- The Gnosis smart contract requires signatures from its owners, Alice and Bob, to authorize any execution of the `execTransaction()`

> function. This means copying the transaction content and executing it from an attacker's EOA is not feasible unless there is a flaw in how signatures are validated, which is not typical for smart contract wallets, including the Gnosis Safe Wallet.

Of course it requires signatures, this is the part of the original submission: `Copy the TX content and call it from the attacker's EOA.` The attacker requires a proposer who is using smart wallet, like the title says. That is not a side exploit or any actual blocking limitation, since we assume the functionality is a valid way of interaction (not otherwise noted).

> Furthermore, even if Alice and Bob authorize a transaction, the claim that funds are lost is incorrect. If Alice is the transaction originator and her EOA executes the transaction, she (tx.origin) retains the ability to claim the bond. Therefore, there is no actual loss of funds.

Honestly don't understand the argument - is this saying that if the exploit is botched (doesn't frontrun like it should), it fails? It is shown in the submission that a malicious frontrunner will be registered as the claimant, and receive the bond at the end of the dispute.

MightyFox3

Thank you for clarifying the situation further.

Alice and Bob, who own the smart contract wallet, need to agree and sign off on any transactions that initiate the claim. If Alice submits the transaction and there's no frontrunning interference, she should be the one to claim the bond.

However, if Bob were to submit his own claim before Alice's is processed—a practice known as frontrunning—he would then be eligible to claim the bond. This could be unfair to Alice, especially if Bob does this deliberately. But such a situation is quite rare since both parties need to agree to initiate the transaction.

This makes the potential problem less severe, as it relies heavily on one party acting against the agreed-upon terms.

54710adk341

You can't just 'front-run' any given smart wallet.

Safe.sol#L141-L161

```
{
    if (guard != address(0)) {
        Guard(guard).checkTransaction(
            // Transaction info
            to,
            value,
            data,
```

```
            operation,
            safeTxGas,
            // Payment info
            baseGas,
            gasPrice,
            gasToken,
            refundReceiver,
            // Signature info
            signatures,
            msg.sender
        );
    }
}
```

Smart wallets have guards in place, they check against the `msg.sender`. Front-running would make `execTransaction()` fail since the `msg.sender` would be different.

trust1995

> Thank you for clarifying the situation further.
>
> Alice and Bob, who own the smart contract wallet, need to agree and sign off on any transactions that initiate the claim. If Alice submits the transaction and there's no frontrunning interference, she should be the one to claim the bond.
>
> However, if Bob were to submit his own claim before Alice's is processed—a practice known as frontrunning—he would then be eligible to claim the bond. This could be unfair to Alice, especially if Bob does this deliberately. But such a situation is quite rare since both parties need to agree to initiate the transaction.
>
> This makes the potential problem less severe, as it relies heavily on one party acting against the agreed-upon terms.

No, that's the point. Charlie, an **unprivileged attacker** who observes the TX Alice sent to the mempool, copies the contents and sends it from his EOA. They frontrun the origin TX and steal the bond.

trust1995

> You can't just 'front-run' any given smart wallet.
>
> Safe.sol#L141-L161

```
{
    if (guard != address(0)) {
        Guard(guard).checkTransaction(
```


```
            // Transaction info
            to,
            value,
            data,
            operation,
            safeTxGas,
            // Payment info
            baseGas,
            gasPrice,
            gasToken,
            refundReceiver,
            // Signature info
            signatures,
            msg.sender
        );
    }
}
```

> Smart wallets have guards in place, they check against the `msg.sender`. Front-running would make `execTransaction()` fail since the `msg.sender` would be different.

That's a wildly incorrect statement. The design of smart wallets is exactly with account abstraction in mind - The TX contents, gas , calldata etc are all signed by the multisig and then *anyone* can transmit the TX to the blockchain. A TX should be perfectly secure regardless of who is initiating the smart wallet execution call.

The contestant is referring to the optional guard feature, which can perform any type of filtering at the discretion of the multisig. The only two multisigs I've checked, the Chainlink MS and the Optimism MS, don't use any guards. It is, broadly speaking, a mostly unused feature used to perform arbitrary custom validation, and has no relevance to the submission.

lemonmon1984

But at the end, the optimism team can utilize DelayedWETH to address the situation. There is no airgap bypass, and based on the security measures such as DelayedWETH, the funds are secure.

bemic

Signatures of Safe owners for a specific transaction are crafted off-chain and passed into the function as input parameters. Once there are enough signatures to pass the threshold, the Safe transaction will be executed. Who is the one who calls the Execute function? It does not matter.

This is the known problem of Safe multisig wallet. There are Guards made specifically to avoid this situation, and they let only one of the owners call the

actual execution. However, they are **not** set up by default. I see this as a known and real problem of Safe. But other protocols like optimism are not forced to be compatible (although they probably should).

Evert0x

This issue is either invalid as it flags a design recommendation to mitigate an attack vector. From the perspective of the smart contract it's functioning as normal, it's just that the user didn't take the necessary measures to profit from this.

Or it's valid and Medium as the loss requires specific conditions (smart wallet) and it's constrained as it only applies to the initial bond.

Will revisit this issue

trust1995

> From the perspective of the smart contract it's functioning as normal, it's just that the user didn't take the necessary measures to profit from this.

The rationale above can be said about any smart contract exploit, from the perspective of the smart contract, everything is functioning as normal. It's not a design recommendation, because Optimism did not limit interaction with the contracts to only EOAs, and any usage without using a private mempool (extremely likely) is vulnerable.

> Or it's valid and Medium as the loss requires specific conditions (smart wallet)

As a C4 Supreme Court judge, that's not the type of conditions that merit lowering a severity. Consider as a thought experiment, a bug that results in loss of funds, only if the first byte of an address is 0xFF. Would that condition reduce severity to Med? Absolutely not, because we realize that over time and considering enough users, it is extremely likely there will be affected users. It is incorrect to look at the single-victim level when the bug is affecting all potential victims.

> it's constrained as it only applies to the initial bond.

This argument could also be used if the initial bond is $100000000. Would that make such billion dollar exploits Med? Just to show that saying it is constrained does not cap severities, what matters it the potential concrete impact. There is no respectable judge on the planet that would rule impact of loss of 0.08 ETH = $240 as lower than high.

Evert0x

> The rationale above can be said about any smart contract exploit, from the perspective of the smart contract, everything is functioning as normal. It's not a design recommendation, because Optimism did not

> limit interaction with the contracts to only EOAs, and any usage without using a private mempool (extremely likely) is vulnerable.

From the perspective of the protocol's mechanisms it doesn't matter if Alice or Bob executes this transaction. The functionality works as intended as the person executing the transaction will receive the bond. Of course this can't be said about every exploit.

nevillehuang

Bonds should belong to the person(s) creating the games that is signing the transaction to create a claim. However, given the permisionless nature of transaction execution for smart wallets as seen here, someone can fron-run and copy the transaction, bypass the transaction checks and act as the `tx.origin` of that initial proposal of the FDG, receiving that initial bond after resolution. I don't think it should be high severity given the DelayedWETH safety mechanism is not bypassed, so I believe medium severity is appropriate here.

darkbit0

Hey @nevillehuang, Clearly the issue exists in the FaultDisputeGame contract. `tx.origin` has been used in FaultDisputeGame which its issues were out of scope (unless it bypasses the air-gap). Just because one fix is involving changing the Factory's code doesn't mean the issue exists in out of FaultDisputeGame's code.

Also if any smart wallet allows attackers to front-run its txs, then those smart wallets have vulnerability and the real root cause of this issue is in those smart wallet's code which weren't in the scope of this contest. Users who uses those smart wallets accepted their risk and they also have options to protect their txs and avoid front-runners (by using private mempools or using Guard feature of smart wallet or using smart wallet without front-run issue). There were a lot of similar situations in that past contests that 3rd party systems bugs could effect the protocol and there were fixes for those issues in in-scope Contracts (adding more checks or ...) but those issues were considered as OOS.

trust1995

@darkbit0 It is considered very poor contest etiquette to repeat arguments already discussed. It is showing lack of respect for Watsons and judge's time, and in my opinion should even be punished.

> Hey @nevillehuang, Clearly the issue exists in the FaultDisputeGame contract. tx.origin has been used in FaultDisputeGame which its issues were out of scope (unless it bypasses the air-gap). Just because one fix is involving changing the Factory's code doesn't mean the issue exists in out of FaultDisputeGame's code.

Was argued above, and neville's response was:

> Agree that this issue is valid, given the root cause can be seen as stemming from the factory contract. Additionally, there is no mention whether only an EOA is allowed to interact with the contracts. Based on agreed upon scope and line drawn, I believe medium severity to be appropriate since no safety mechanisms (DelyayedWETH) is bypassed.

Then:

> Also if any smart wallet allows attackers to front-run its txs, then those smart wallets have vulnerability and the real root cause of this issue is in those smart wallet's code which weren't in the scope of this contest. Users who uses those smart wallets accepted their risk and they also have options to protect their txs and avoid front-runners (by using private mempools or using Guard feature of smart wallet or using smart wallet without front-run issue)

This was already explored in depth before your attempt to re-open the discussion.

> > That's a wildly incorrect statement. The design of smart wallets is exactly with account abstraction in mind - The TX contents, gas , calldata etc are all signed by the multisig and then anyone can transmit the TX to the blockchain. A TX should be perfectly secure regardless of who is initiating the smart wallet execution call.
>
> The contestant is referring to the optional guard feature, which can perform any type of filtering at the discretion of the multisig. The only two multisigs I've checked, the Chainlink MS and the Optimism MS, don't use any guards. It is, broadly speaking, a mostly unused feature used to perform arbitrary custom validation, and has no relevance to the submission.

54710adk341

> I think the implied contract is relatively clear, the user who creates the game is the `tx.origin` and not the `msg.sender`. Smart contract wallets weren't an intended user of the contracts. Either way impact is relatively limited (smallest bond size is at the initialization level). I think it's a pretty clear footgun though and should be fixed to prevent issues down the line.

This sums it up pretty well, this is a clear footgun, hence this is a valid Low. User errors are not Medium under Sherlock rules.

Evert0x

This comment reflects my current stance on this issue https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/194#issuecomment-2094911840

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- trust1995: accepted

MightyFox3

https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/15

@Evert0x @nevillehuang

trust1995

> #15
>
> @Evert0x @nevillehuang

??? Has nothing to do with this submission.

Issue M-4: Loss of bond amounts on re-org attacks

Source: https://github.com/sherlock-audit/2024-02-optimism-2024-judging/issues/201

Found by

MiloTruck, Trust

Summary

The `move()` function lacks proper identification of the target of the move, leading to successful re-org attacks which can take the honest participant's funds.

Vulnerability Detail

Participants in the game can call `attack()`, `defend()` or `move()`, each accepting a `parentIndex` which corresponds to the claim being challenged, and a `_claim` commitment.

When participants claim, they have a particular claim in mind which they wish to challenge, and then pass on that claim's index. However, between the moment they sent the TX and the moment that TX is executed, a block reorg can take place. When it occurs, the challenge corresponding to that ID may change to another challenge, which may be valid or invalid in a different way. Regardless, the participant's commitment to that `move()` will be wrong, and they stand to lose their bond amount.

Chain reorgs are very prevalent in Ethereum mainnet, where the contract is deployed. You can check this index of reorged blocks on etherscan. It is **incorrect** to assume the attacker will wait until it achieved finality, because there's no warnings or documentation available for them to identify this as a threat. Therefore, it remains a very valid concern with reasonable hypotheticals.

Note that in high depths, the bond amount is very large, leading to a large loss of funds.

Possible flow:

- Attacker submits invalid claim hash
- Honest defenders rush to prove the claim wrong (note that only first defender gets the bond, so they will rush to submit the TX. They would not be concerned about waiting against reorgs without warning)
- A block re-org occurs
- The attacker replaces the invalid claim with a valid claim hash

- Defender's TXs are applied on top of the valid claim.
- Attacker can scoop up all the defenders' bonds

Impact

Loss of bond value for honest participants of the dispute game.

Code Snippet

Tool used

Manual Review

Recommendation

Every move needs to include the key parameters which it wishes to attack/defend - the claim hash and the Position in the game tree.

Discussion

smartcontracts

This is the intended behavior of the contract so we have confirmed the factuality of the report and marked as "won't fix". Challenger software can handle this case offchain.

nevillehuang

I believe this is out of scope, given there is no network admins in mainnet and thus doesn't satisfy the the requirements for the exception

> Chain re-org and network liveness related issues are not considered valid. Exception: If an issue concerns any kind of a network admin (e.g. a sequencer), can be remedied by a smart contract modification, the protocol team considers external admins restricted and the considered network was explicitly mentioned in the contest README, it may be a valid medium. It should be assumed that any such network issues will be resolved within 7 days, if that may be possible.

trust1995

Escalate

The finding is in-scope as Medium severity for the following reasons:

- The impact is direct loss of bonds of an honest challenger, who did not make any mistakes

- There are no other preconditions except a re-org on the blockchain
- As shown in the report, there is more than sufficient likelihood for re-orgs on ETH to render this a valuable concern that needs to be protected from
- The issue is CLEARLY a smart contract issue, it is a lack of sufficient identification of a claim and is fixed by adding one line of code. It does not belong to the usual category of re-org issues which can be treated as unavoidable risks of blockchain architecture. The impact and circumstances are concrete and likely.
- The challenge game is presented as a race where the first challenger picks up the bond - it is only natural that challengers will pop up as quickly as possible to challenge a honeypot claim. There are zero warnings or ways where a challenger can foresee such an attack is possible - unless we assume challengers are coding gurus which would audit the code and identify this re-org attack could steal their bonds. To be clear, a simple warning saying challengers should wait until the claim block is finalized would be sufficient to close the issue as a user-error, but that's not the case.
-

sherlock-admin2

Escalate

The finding is in-scope as Medium severity for the following reasons:

- The impact is direct loss of bonds of an honest challenger, who did not make any mistakes
- There are no other preconditions except a re-org on the blockchain
- As shown in the report, there is more than sufficient likelihood for re-orgs on ETH to render this a valuable concern that needs to be protected from
- The issue is CLEARLY a smart contract issue, it is a lack of sufficient identification of a claim and is fixed by adding one line of code. It does not belong to the usual category of re-org issues which can be treated as unavoidable risks of blockchain architecture. The impact and circumstances are concrete and likely.
- The challenge game is presented as a race where the first challenger picks up the bond - it is only natural that challengers will pop up as quickly as possible to challenge a honeypot claim. There are zero warnings or ways where a challenger can foresee such an attack is possible - unless we assume challengers are coding gurus which would audit the code and identify this re-org attack could steal their bonds. To be clear, a simple warning saying challengers

> should wait until the claim block is finalized would be sufficient to close the issue as a user-error, but that's not the case.

-

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

Based on sherlock rules, I believe this is still invalid based on comments here.

Evert0x

The following rule applies

> Chain re-org and network liveness related issues are not considered valid.

Planning to reject escalation and keep issue state as is

trust1995

@nevillehuang The rationale for the scoping rules on Sherlock excluding re-org attacks is the assumption that a TX sender is responsible for waiting for finality (stated by Judge on discord). However, due to Optimism-specific circumstances detailed in depth by the dup submission by MiloTruck, it is proven that an honest party cannot afford to wait for finality (the so called chess-clock mechanism). For this reason, and the fact that re-orgs on L1 are proven to be highly likely, it is only common sense to see that the issue is a valid risk of loss of funds for Medium severity.

nevillehuang

Hi @Evert0x although I believe this issue could still be possibly out of scope due to it being related to resolution logic, I think @trust1995 has a point here. Re-orgs are often times out of the control of a user, and in optimisms case, this directly leads to a serious inconsistency where user would dispute a claim incorrectly (although there are safeguards).

I believe that re-org issues exception could be re-considered for sherlock's scope in the future, possibly a proposal could be put up to address this per discussed, especially so when a fund loss impact is involved.

Evert0x

I understand that re-org attacks are possibly more interesting to L1's or L2's than the average protocol. However, using the same judging rules as every Watson used

during the contest, it would only be fair to invalidate it. As the language is clear

> Chain re-org and network liveness related issues are not considered valid.

trust1995

@Evert0x The language is clear, but guidelines always have exceptions. It is up to the judge to apply common sense and the contest-specific context to every verdict. Blindly following every rule will lead to injustice and counterexamples where an impact is clearly real and valuable, but is not rewarded. That is well understood in the Sherlock rulebook:

> Note: Despite these rules, **we must understand that because of the complexity & subjective nature of smart contract security, there may be issues that are judged beyond the purview of this guide**. However, for the vast majority of cases, this guide should suffice. **Sherlock's internal judges continue to have the last word on considering any issue as valid or not**.

The chain re-org and liveness rule already has an exception.

> Exception: If an issue concerns any kind of a network admin (e.g. a sequencer), can be remedied by a smart contract modification, the protocol team considers external admins restricted and the considered network was explicitly mentioned in the contest README, it may be a valid medium. It should be assumed that any such network issues will be resolved within 7 days, if that may be possible.

The submission abides by all the criteria for that exception except the network (mainnet) does not have an admin. The intention around that criteria is that because there's no admin, we can assume actors can wait for finality before submitting a transaction, and they could not get attacked. However in the Optimism codebase, we have shown the game clock forces honest parties to respond before blocks are finalized, re-opening the vector.

It is very clear that the combination of the impact, simple code fix, execution before finality, and ease of exploit make an extremely sound case for Medium severity.

Judging is not clerk work, it requires making nuanced decisions and not continuously falling back on previous decisions, which were made with different contexts. Apply common sense, and determine if the submission is worthy of H/M.

Evert0x

Result: Invalid Has Duplicates

The judges have the last of opinion but objectivity is held to a high regard. As the language is so clear, I believe it's the correct judgment

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- trust1995: rejected

Evert0x

This was initially deemed invalid by a strict interpretation of our judging guidelines ("Chain re-org and network liveness related issues are not considered valid."). This rule exists as the "blockchain is trusted" from the perspective of app builders. However, a different trust level applies when building an L1/L2.

After a discussion with the lead judge and the protocol team I'm assigning Medium severity.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/ethereum-optimism/optimism/pull/10520/files

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.