Scout Soroban

Precision and Recall Report

April, 2024

Scout Precision and Recall Report

Validation on Real Life Projects and Tool Improvements

Summary

We ran Scout on 71 smart contracts of 18 public Soroban projects.

After running Scout (cargo-scout-audit version 0.2.4) on the smart contracts, we identified a total of 847 triggered alarms, out of which 290 were determined to be false positives following a manual review of each finding. This results in a false positive rate of 34.24%. We further analyzed the false positives associated with each detector, focusing particularly on `'unsafe-unwrap'` and `'set-contract-storage'`, the two detectors with the highest number of false positives, to identify potential improvements to the tool's precision.

We subsequently refined the detectors and released an updated version of Scout (cargo-scout-audit version 0.2.6), which included enhancements. We then re-ran the tool, focusing on the revised detectors. Our modifications were not limited to the two detectors that produced false positives; we also adjusted other detectors that we believed could potentially lead to false positives in similar situations. As a result, our analysis led to improvements in the precision of five detectors.

In addition to analyzing Scout as a single source of triggers, we conducted two other analyses (refer to Appendices section below). Firstly, we examined the rates of false positives per smart contract/project, which reflects the perceived quality from the user's perspective (those who would run the tool in their project individually). Secondly, we assessed the rate of false positives per detector to determine the performance of each detector and identify areas needing improvement.

After working on some of the failing detectors we improved the precision in most cases for every project, but with some detectors the improvements in some projects were detrimental in other cases, producing a decrease in recall.

We have already begun the next iteration of Precision and Recall, focusing on further refining Scout's detectors. We will conduct new runs of the tool and analyze the results, including the latest detectors additions. This analysis will enable us to confirm the final rate of false positives after the improvements, completing Table 2: False Positives per Detector.

Improvements on Detectors

As we analyzed the different positives from running the tool, we identified that most of the false positives occur in the detectors `unsafe-unwrap` and `set-contract-storage`. We focused our work on improving the precision of these two detectors, as well as other detectors that could be enhanced from similar checks.

On `unsafe-unwrap`

For `unsafe-unwrap`, we noticed cases where previous checks in the analyzed code made the particular use of `unwrap()` not result in an error. We updated the detector to validate whether these checks are present in the code, decreasing the amount of false positive detections on a second run of the tool.

Example 1: False positive for `unsafe unwrap` with previous check.

```
    pub fn truncating_mul(self: &Self, x: i128) -> i128{
        let result = safe_mul(x, self.num, self.den);
        if result.is_err(){
            panic!("integer overflow")
        }
        result.unwrap()
    }
```

We also registered another class of false positives, which, due to the particular arithmetic and value assignment of the variables involved, would probably not result in a vulnerability, but found no way to discard them for this detector within the restrictions of our static analysis method.

Example 2: False positive due to arithmetic and value assignment in range. If the values assigned to variables do not exceed the range, `unwrap()` will not return an error.

```
pub fn some_function(e: &Env) {
    let mut total = get_total(e);
    total = total.checked_add(1).unwrap();
    total
}
```

Finally, we identified that the same checks could be applied to the detector unsafe-expect, and updated it accordingly.

On `set-contract-storage`

Upon analyzing false positives in the `set-contract-storage` detector, we identified use cases where the authorization to use `env.storage()` was done in a function outside of the analysis context of our detector, or the storage method being detected (i.e: `get`) did not represent a vulnerability.

We extended the analysis context of our detector to identify these authorizations in parent functions and added the capability for the detector to now differentiate between various storage types from the Soroban SDK.

Example 3: False positive for set contract storage. This example authorizes the call of storage and uses `get` which is non vulnerable.

```
pub fn some_function( env: Env) -> u32 {
    let storage = env.storage().persistent();
    if storage
        .get::<_, SomeState>(&DataKey::SomeState)
        .is_none() {
        panic_with_error!(&env, Error::NotInitialized);
    }

    let admin = storage.get::<_, Address>(&DataKey::Admin).unwrap();
    admin.require_auth();
            ...
}
```

On the other hand, we believe that some use cases using `DataKey` could now result in true positives, which are being discarded after the detector's update to differentiate between storage types. When the key used to modify the storage is not of type `soroban_sdk:Address`, but an enum `DataKey`, the detector overlooks the issue, without validating if a user address is being modified and if it represents a vulnerability. We are currently evaluating these cases to amend our detector.

Example 4: New False negative for `set contract storage`. This example is no longer detected after our update because the `DataKey` is not of type `soroban_sdk:Address`.

```
env.storage().instance().set(&DataKey::State, &State::new(storage, adder, subber));
```

The same extension of the analysis context was also applied on detectors `unprotected-mapping-operation` and `unprotected-update-contract-wasm`.

Improvements on Troubleshooting Documentation

As we used Scout over a variety of projects, we noticed some issues when running the tool on contracts performing crossed calls. For these cases we found that an easy troubleshoot was compiling the second contract first (`soroban contract build`) before running Scout on the first one.

On the other hand, as we tried Scout on different environments, we noticed some installation caveats. We wrote down a troubleshooting guide to aid the user on particular installation issues.

Responsible Disclosure

We have conducted an initial review of the Soroban projects and identified vulnerabilities in some of them that require attention. A more comprehensive security analysis is currently underway by one of our auditors. If vulnerabilities are confirmed, we will engage with each project and responsibly disclose our findings to facilitate their correction and contribute to the security of the ecosystem.

Therefore, this report does not contain explicit references to the analyzed projects.

Appendices

Appendix 1: False Positive Alarms per Project

Analyzing the number of false positives per project, we observe an average rate of false positives vs total positives of 21 %, and a median of 0%.

If we analyze only projects with detections, the average rate of false positives vs total positives per project increases to 48%, and the median to 51%.

We keep the identity of the analyzed projects anonymous as we confirm and responsibly disclose true positives found during our analysis of the tool's output.

The detectors run correspond to the ones available in the Scout version 0.2.4 at the commencement of this analysis.

Table 1: False positives per project

Project ID	Total Positives	False Positives	% False Positives
1	27	4	15%
2	20	0	0%
3	0	0	0%
4	0	0	0%
5	50	0	0%
6	55	0	0%
7	10	0	0%
8	48	0	0%
9	72	35	49%
10	25	0	0%
11	122	17	14%
12	44	17	39%
13	10	0	0%
14	70	63	90%
15	12	0	0%
16	15	10	67%
17	47	28	60%
18	220	116	53%

Appendix 2: False Positive Alarms per Detector

In the following table, we identify the total number of positives and false positives per detector across all analyzed smart contracts. The detectors run correspond to the ones available in the Scout version 0.2.4 at the beginning of this analysis. Notice that some detectors were never activated in the analyzed code. The false positives were analyzed in order to improve the detectors.

Table 2: False positives per detector

Detector	Total Positives	False Positives	% False Positives
Divide before multiply	0	0	0.00%
Unsafe unwrap	180	6	3.33%
Unsafe expect	65	0	0.00%
Overflow check	2	0	0.00%
Insufficiently random values	0	0	0.00%
Unprotected update current contract wasm	0	0	0.00%
Avoid core mem forget	0	0	0.00%
Set contract storage	478	284	59.41%
Avoid panic error	63	0	0.00%
Avoid unsafe block	0	0	0.00%
Dos unbounded operation	13	0	0.00%
Soroban version	46	0	0.00%
Total	**847**	**290**	**34.24%**

Some of the detectors named above are *highlighted*, meaning we focused our analysis on them and worked primarily to improve those.