LIGHTNING SECURITY

PENETRATION TEST REPORT SOLIDIFIED TECHNOLOGIES INC.

June 12, 2018

lightningsecurity.io
info@lightningsecurity.io

Table of Contents

Executive Summary

Lightning Security conducted a 20-hour penetration test of Solidified Technologies Inc., with the goal of discovering vulnerabilities in the following systems:

Web: `web.solidified.io` - Solidified platform (production)

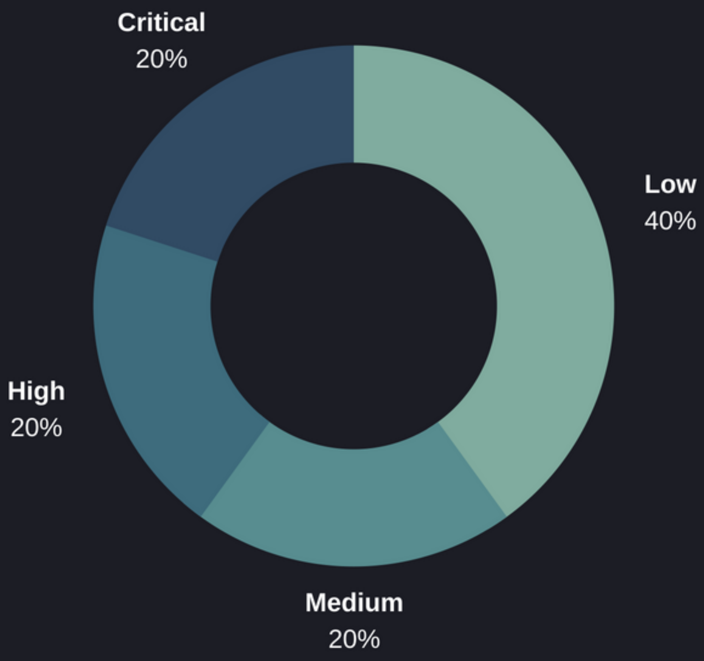
Web: `solidified.io` - Solidified landing page

As the aim of Solidified is to provide an open, secure environment for reporting vulnerabilities, the penetration test focused on assessing methods in which malicious attackers could cause either financial or reputational damage to the Solidified platform.

Lightning Security was given access only to the public-facing urls, in order to simulate a real attack scenario.

Findings Overview

Over the course of testing, Lightning Security identified 2 critical severity flaws, 2 high severity flaws, 2 medium severity flaws, and 4 low severity flaws:



The following issues were identified:

Description	Severity	Page
Race condition in pulling contract allows stealing ETH from escrow	Critical	4
Negative bounty amount allows for infinite account balance	Critical	5
Two-factor authentication bypass	High	6
Enumeration of private profile fields via API	High	7
Hashed password of current user exposed in API response	Medium	8
401 Response Injection via markdown images	Medium	9
Notifications of all users exposed in API response	Low	10
Contract can be created with lower bounty than permitted	Low	11
Arbitrary file extension allowed in contract	Low	12
Profile images stored in database as strings	Low	13

Race condition in pulling contract allows stealing ETH from escrow

Identified severity: Critical

Lightning Security identified a race condition in pulling contracts. By making multiple asynchronous requests to pull a contract, an attacker is able to be refunded multiple times the original amount from the escrow.

Reproduction Steps

1. Visit https://web.solidified.io/upload-contract and follow the steps to create a contract.
2. After the contract has been created, intercept (but do not execute) the request to pull the contract via Burp Suite.
3. Copy the request as a curl command in Burp Suite (right click -> copy as curl command)
4. Execute the command on the command line in the format `(command) & (command) & (command)`.
5. Observe that after executing, the ETH deposited into the escrow will be refunded multiple times, effectively stealing money from the escrow.
6. Alternatively, as the request to pull a contract takes multiple seconds to process, it may be possible to simply open the contract in multiple tabs and pull the contract at the same time in each tab.

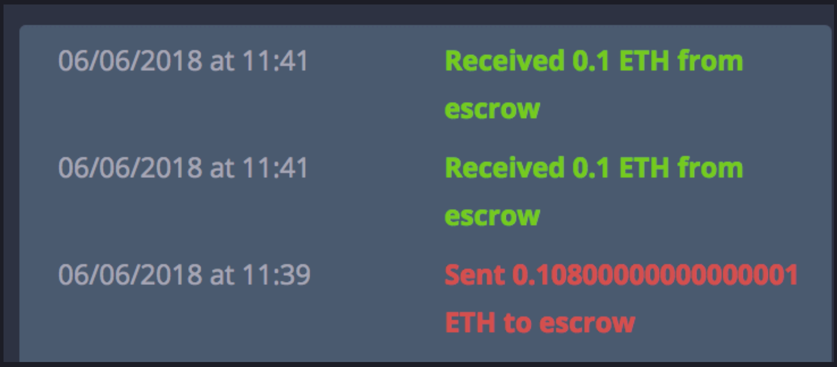
Impact

An attacker can exploit this to drain the balance of the escrow. If withdrawals are supported in the future, this would allow stealing all ETH stored in the escrow.

Suggested Remediation

Ensure that the server marks the contract as pulled before processing refunds.

Screenshot:



06/06/2018 at 11:41	Received 0.1 ETH from escrow
06/06/2018 at 11:41	Received 0.1 ETH from escrow
06/06/2018 at 11:39	Sent 0.10800000000000001 ETH to escrow

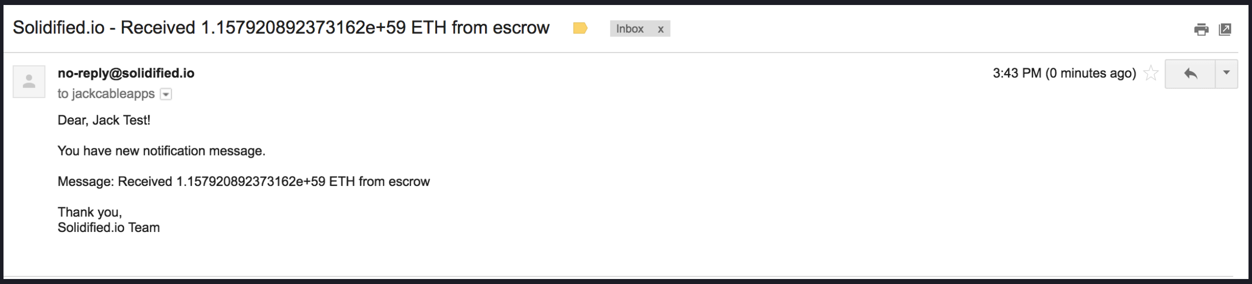
Negative bounty amount allows for infinite account balance

Identified severity: Critical

Lightning Security identified an underflow in awarding bounty amounts. By initializing a contract with a negative amount as a reward for a certain severity, a malicious contract creator is able to obtain an account with an infinite balance. This is due to an underflow in the account balance: when 1 ETH is removed from an account with a balance of 0.1 ETH, the balance of the account becomes 1.15 x 10^59 ETH.

Reproduction Steps

1. Visit https://web.solidified.io/upload-contract and follow the steps to create a contract.
2. Before submitting the final request, intercept the request in Burp Suite. Modify the `Minor` bounty amount to be `-1` .
3. Ensure the contract is ready to review.
4. As the account of another researcher, submit a `Major` severity bug. Note that a `Minor` severity bug cannot be submitted here, as the execution will fail.
5. As the contract owner, make a proposal to change the severity to `Minor` .
6. Observe that upon making this change, the Solidified platform will attempt to add `-1` ETH to the researcher's account. This will result in the account balance underflowing, becoming 1.15 x 10^59 ETH:


Solidified.io - Received 1.157920892373162e+59 ETH from escrow

no-reply@solidified.io
to jackcableapps
3:43 PM (0 minutes ago)

Dear, Jack Test!

You have new notification message.

Message: Received 1.157920892373162e+59 ETH from escrow

Thank you,
Solidified.io Team

Impact

This allows a malicious user to obtain an infinite account balance. When the Solidified platform enables withdrawals, this would allow the user to empty the entire balance of the escrow.

Suggested Remediation

Add validation to ensure bounty amounts are always greater than zero.

Two-factor authentication bypass

Identified severity: High

Lightning Security identified that it is possible to completely bypass two factor authentication for a Solidified account. This is due to the user's `TFASecret` being returned upon logging in. Additionally, all two-factor authentication validation is performed on the client-side, which can inherently be bypassed.

Note also that there are flaws present after having authenticated. The two-factor authentication secret is generated client-side. This should be generated server-side as a best practice. A two-factor authentication code should be required to disable two-factor authentication once logged in.

Reproduction Steps

1. Visit https://web.solidified.io/.
2. Follow the steps to enable two-factor authentication.
3. Observe that upon logging in, the `TFASecret` is returned in the JSON response.
4. From here, an attacker can generate a code which bypasses two-factor authentication.

Impact

This allows an attacker to bypass Solidified's two-factor authentication and log into any account unrestricted.

Suggested Remediation

- Never return the `TFASecret` in API responses.
- Generate the `TFASecret` server-side
- Validate two-factor authentication codes server-side
- Require a two-factor authenitcation code to disable two-factor authentication once logged in

Enumeration of private profile fields via API

Identified severity: High

All private profile fields can be enumerated via the `is-unique` endpoint of the Solidified API. This includes the hashed password, username, two factor authentication secret, balance, and addresses of users. Using this, an attacker can quickly determine if a given password is being used by any Solidified user.

Reproduction Steps

1. Visit `https://api.solidified.io/account/is-unique?field=_id&value=5afa0a8c2a5b5000110c6acd` .
2. Observe that `true` is returned as this is a valid `_id` of a user.
3. This can be repeated for all other user profile fields.

Impact

This allows an attacker to enumerate any private profile field of Solidified users' accounts. For instance, an attacker could use this to determine if a hashed password is being used by any user on Solidified.

Suggested Remediation

Restrict the `is-unique` endpoint to only allow querying account usernames.

Hashed password of current user exposed in API response

Identified severity: Medium

The hashed password of the current logged-in user is exposed via the API response from `https://api.solidified.io/account/` . This information should be private and never returned to the user.

Reproduction Steps

1. Log in at https://web.solidified.io/.
2. Observe that in the `GET` request to `https://api.solidified.io/account/` the hashed password of the user is returned in the `password` field.

Impact

This allows an attacker to view the hashed password of an account if they have access to the account, for instance via a physical device or a cross-site scripting flaw. The hashed password of a user should never be viewable.

Suggested Remediation

Blacklist the `password` field from all API responses.

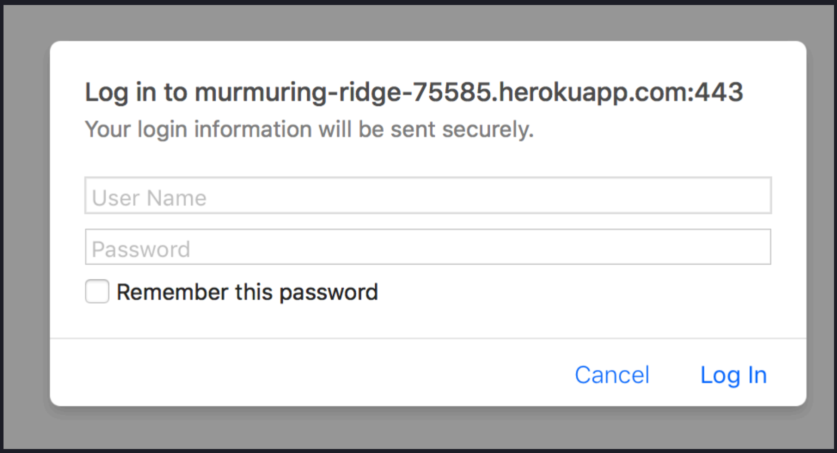
401 Response Injection via markdown images

Identified severity: Medium

Users can embed images in contract chats via markdown. As the response of the url is not restricted, this allows a user to embed a page that returns a 401 unauthenticated error, presenting an invasive alert and sending client data to an external server to users in Firefox, Sarari, Internet Explorer, and Edge.

Reproduction Steps

1. Create a contract.
2. Add the following as a comment on the contract: `![](https://murmuring-ridge-75585.herokuapp.com/alert)` .
3. Observe that in Safari, Firefox, Internet Explorer, and Edge an alert will display:



Log in to murmuring-ridge-75585.herokuapp.com:443
Your login information will be sent securely.

Impact

This allows an attacker to display an invasive alert to users, tricking them into sending credentials or taking other actions.

Suggested Remediation

Disable images in chat markdown.

Notifications of all users exposed in API response

Identified severity: Low

The notifications of all users are exposed in an API response from `https://api.solidified.io/notification/` . This allows an attacker to view activity of Solidified users in bulk. Note that this is mitigated due to user notificaitons already being displayed on their profile.

Reproduction Steps

1. Visit https://api.solidified.io/notification/.
2. Observe that the notifications of all users are returned.

Impact

This allows an attacker to view notifications of all Solidified users such as transaction history and account activity.

Suggested Remediation

Disable users from viewing the API response listing all notificaitons.

Contract can be created with lower bounty than permitted

Identified severity: Low

Although Solidifed enforces a minimum bounty and budget for a contract, it is possible to create a contract with a budget of any amount. This allows for creating spam contracts without actually having funds to support an audit.

Reproduction Steps

1. Visit https://web.solidified.io/upload-contract.
2. Follow the steps to create a contract, and intercept the response to finalize the creation.
3. Modify the `bountyPool` and reward parameters to any positive amount.
4. Observe that the contract is added despite the amounts being lower than required.

Impact

This allows a malicious user to spam creation of contracts with little funds.

Suggested Remediation

Enforce a minimum required amount for a contract's bounty pool.

Arbitrary file extension allowed in contract

Identified severity: Low

When uploading a contract, Solidified does not validate the extension of uploaded files. This allows an attacker to upload malicious files, which are then downloaded in the zipped folder when downloading a contract's files.

Reproduction Steps

1. Visit https://web.solidified.io/upload-contract.
2. Follow the steps to create a contract and intercept the final request.
3. Modify the extension of an uploaded `.sol` file to some other extension.
4. Observe that upon creation of the contract, the file can be downloaded.

Impact

This allows an attacker to distribute potentially malicious files to users, who may trust and execute files downloaded from Solidified.

Suggested Remediation

Restrict all uploaded files to have a `.sol` extension.

Profile images stored in database as strings

Identified severity: Low

The profile image of users is stored in Solidified's database as strings. This is inefficient and could lead to denial of service attacks, where users upload very large files to overload the database.

Reproduction Steps

1. Visit https://web.solidified.io/.
2. Observe that upon uploading a profile image, the image is returned as a string in the user's profile.

Impact

It is storage-ineffecient to store images as strings in the database, and allows for denial of service attacks with very large images.

Suggested Remediation

Upload images to an image hosting provider such as Amazon S3.