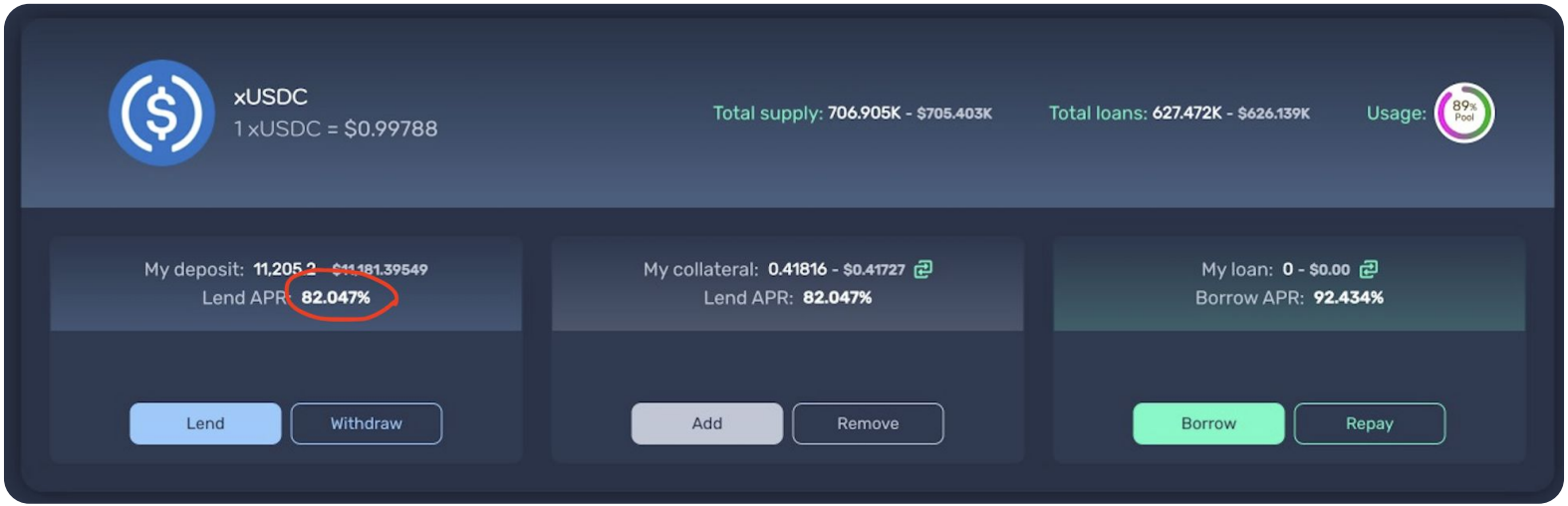
HACKEN

How to steal $1.1M in 15m

Bartosz Barwikowski
L1 Researcher & Auditor at Hacken

In May 2024 during my free time I discovered a lending protocol offering 82% APR in USDC.



A crazy high value! It's a great deal or a scam so I decided to verify that.

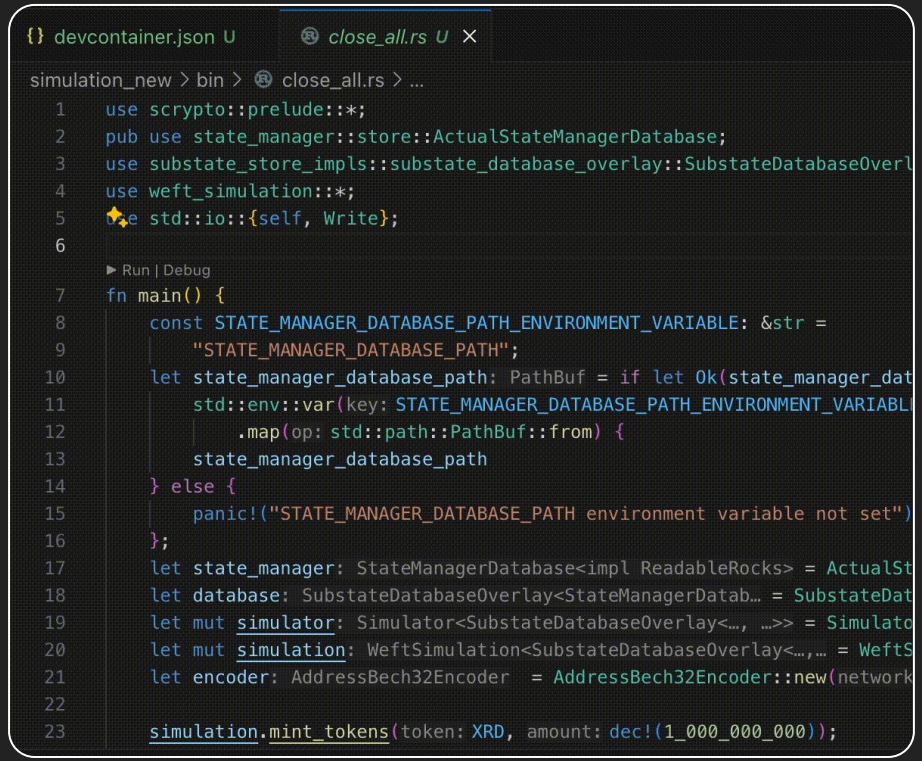
```
#include <hacken.io>


Int main() {
  printf(‘Let’s dig into the source code’);
  return 0;
}
```

HACKEN

Checking the source code and verifying assets and loans

After spending a few hours on reviewing the 1500 lines of source code, everything seemed to work as intended.

To be sure, I quickly wrote a program to verify that all loans and collateral really exist on chain and are healthy. But what does it mean that loan is healthy?



Healthy and unhealthy loan example

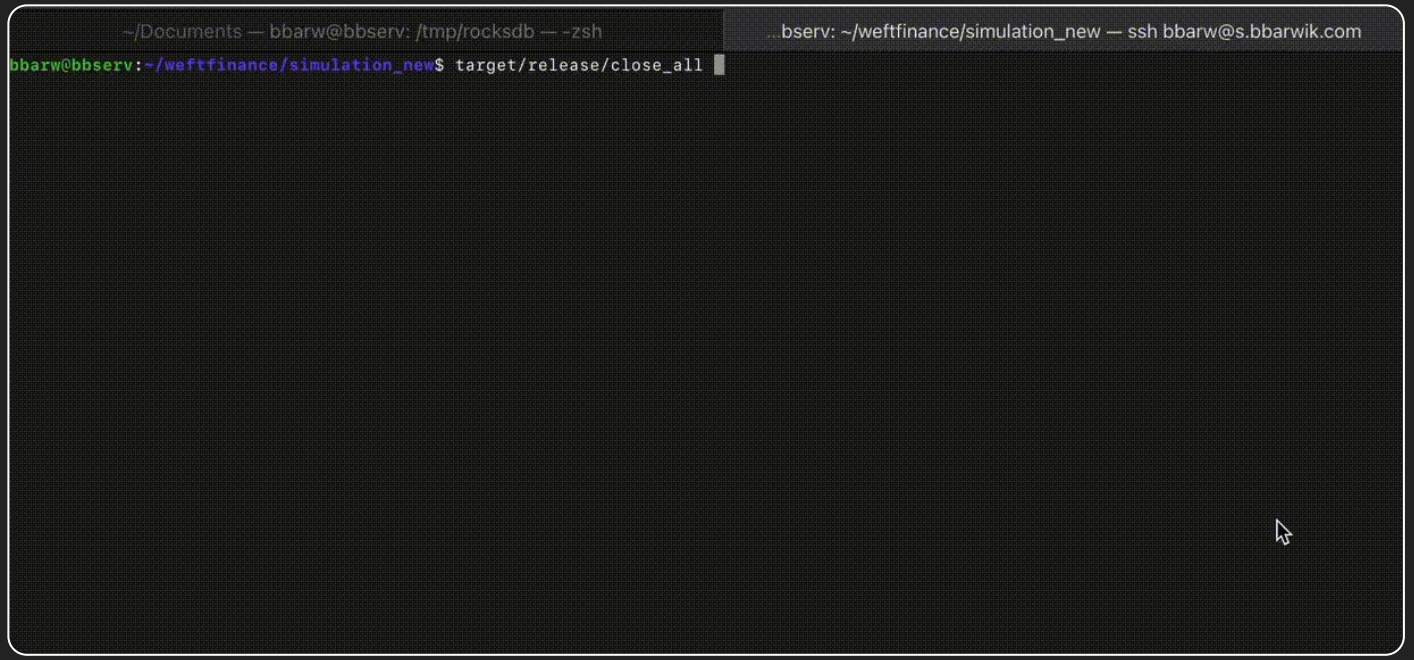
Healthy loan	
Collateral	$100
Discounted collateral	$50
Debt	$40
Debt to collateral	80%

Unhealthy loan	
Collateral	$100
Discounted collateral	$50
Debt	$60
Debt to collateral	120%

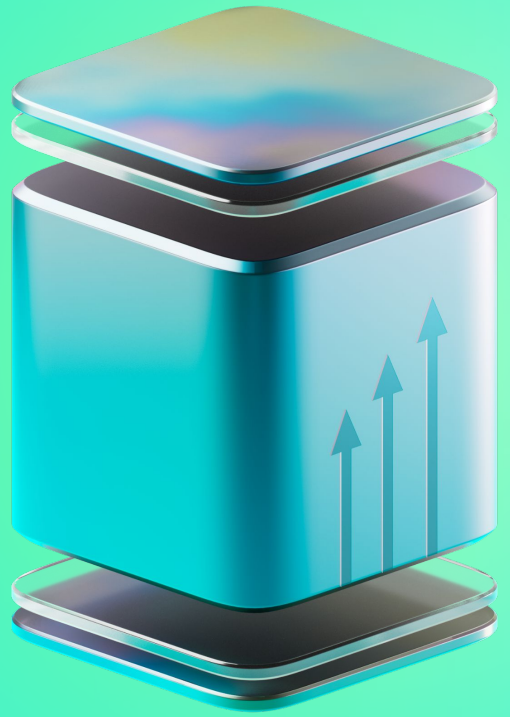
A healthy loan has just higher discounted collateral than debt.
To mitigate sudden token price change only 50% of collateral is counted as discounted collateral.

HACKEN

Verifying assets and loans - running the program



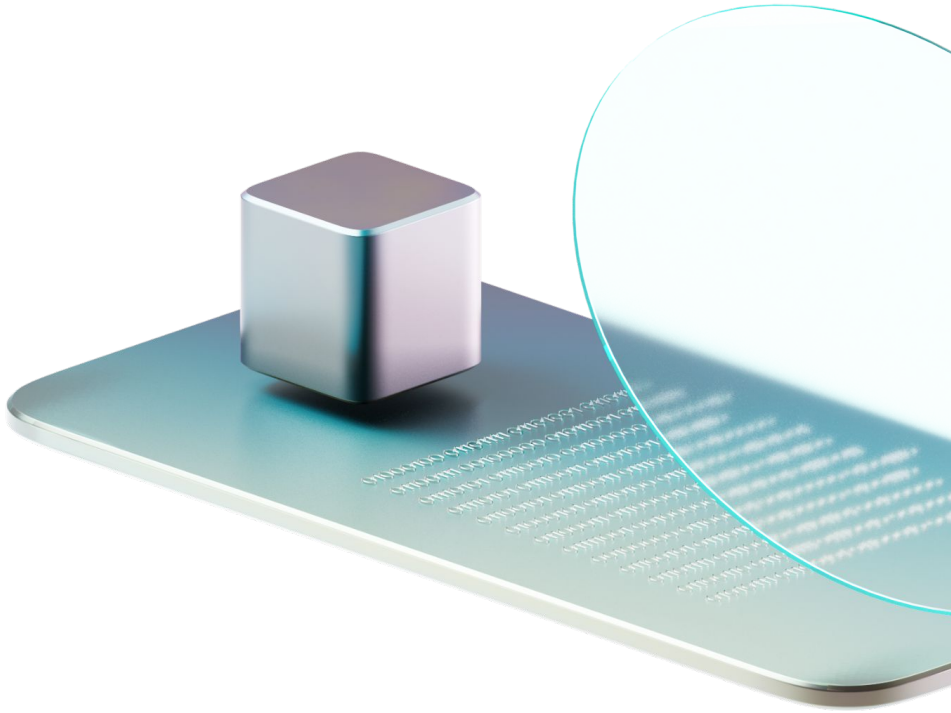
Digging deeper



The high complexity of a smart contract and small severity issues

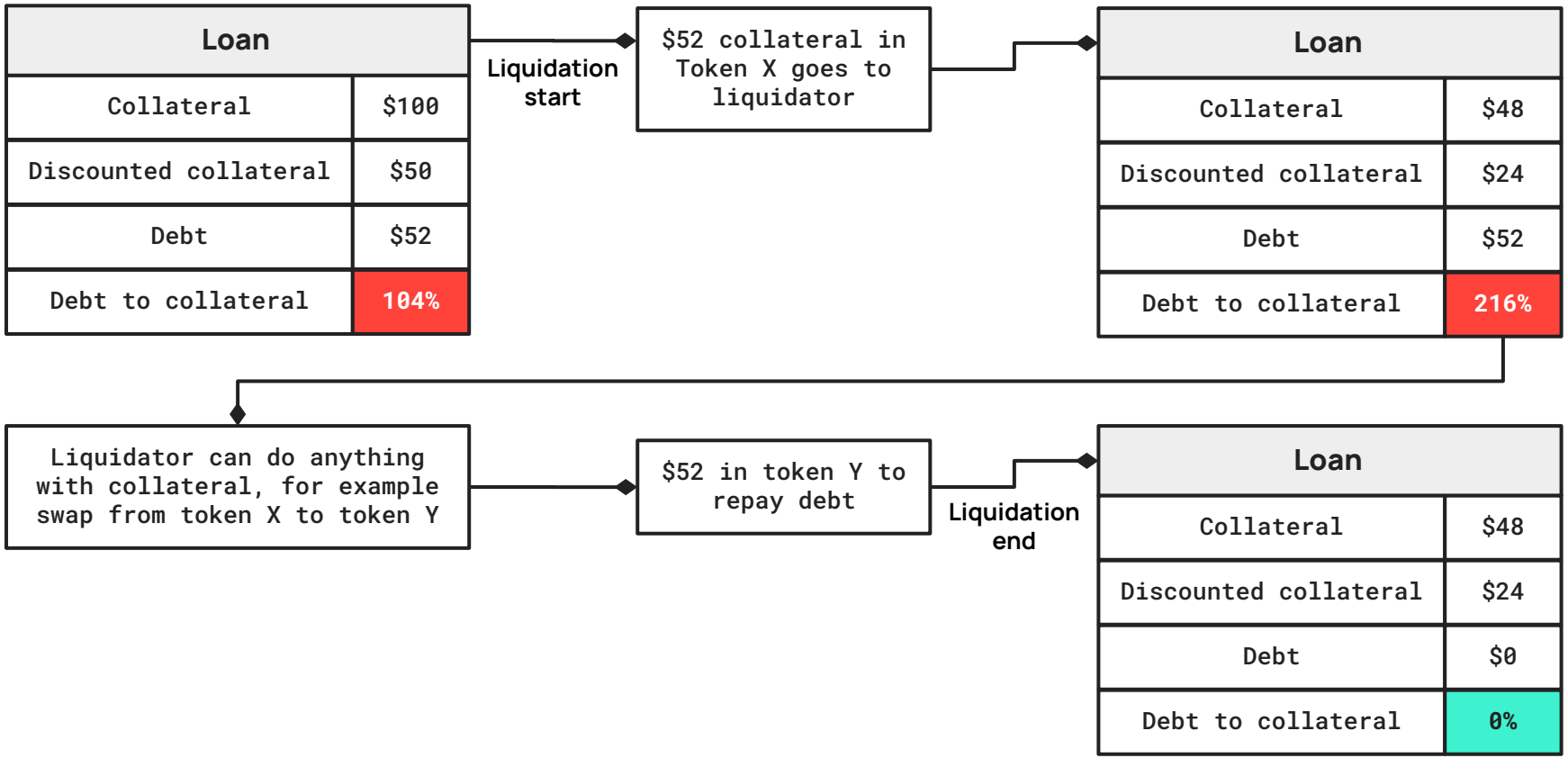
During verification of source code I quickly notice that the project is very complex. It had an advanced liquidation process which worked like a flash loan and very complex features like possibility to have a single collateral for multiple loans. From my experience as an auditor, I knew how easy is to make a mistake in such complex workflows so I started to analyze them in detail.

I quickly noticed the issue in liquidation process and in multiple loans for the same collateral.



Liquidation of unhealthy loan

HACKEN



Linked loans

The smart contract allowed to create a multiple loans for the same collateral, a behavior which could be used to lend tokens for separate cases.

A loan using collateral of different loan is called **linked loan**. When debt on linked loan changes it also changes in a parent loan which keeps information about total debt of all linked loans.

Mother loan 1	
Collateral	$100
Discounted collateral	$50
Total Debt	$40
Debt to collateral	80%

Linked loan 1.1	
Debt	$5

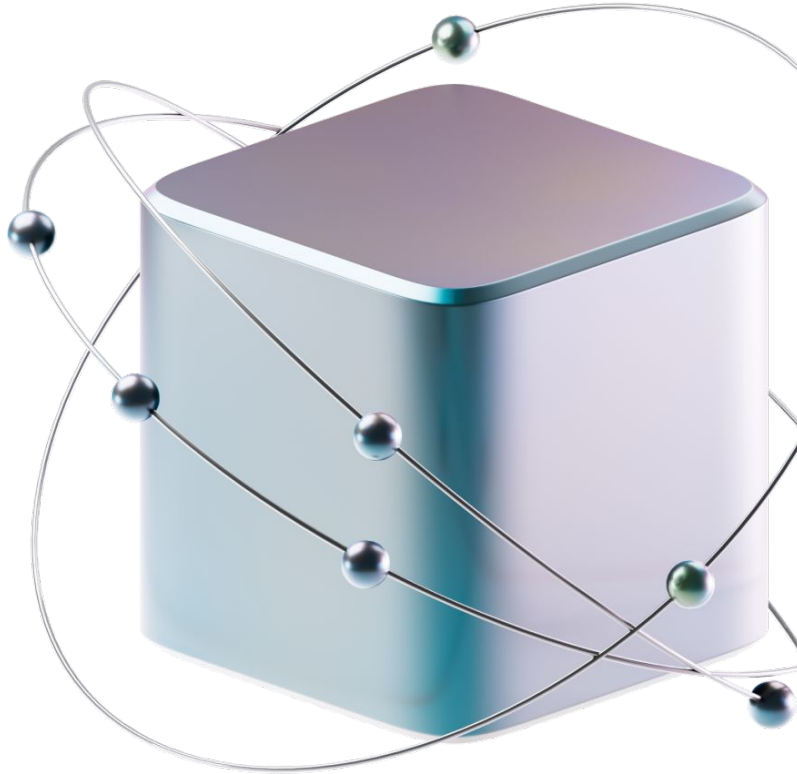
Linked loan 1.2	
Debt	$10

Linked loan 1.3	
Debt	$25

Creation of linked loan

To create a linked loan first the user creates a new standard loan and then uses a function **link_cdp** to link it to other loan. To link a loan it must be unused which in this case mean is **cannot have any collateral**.

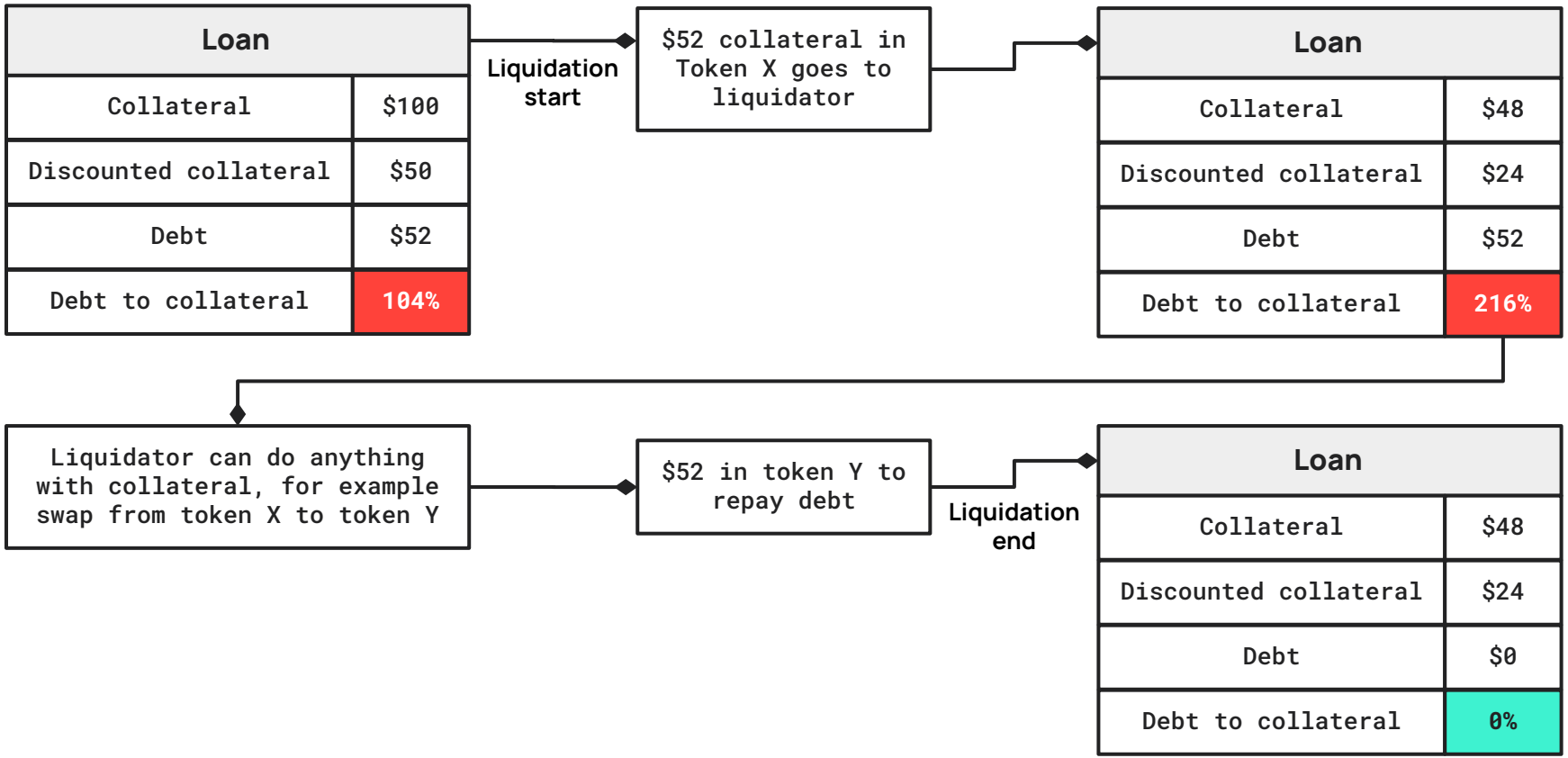
However, **there is no validation if a loan has any debt**, the validation checks only collateral because in theory it should not be possible for a loan with debt and without collateral to exist.



Is it really impossible to have a loan with only debt and no collateral?

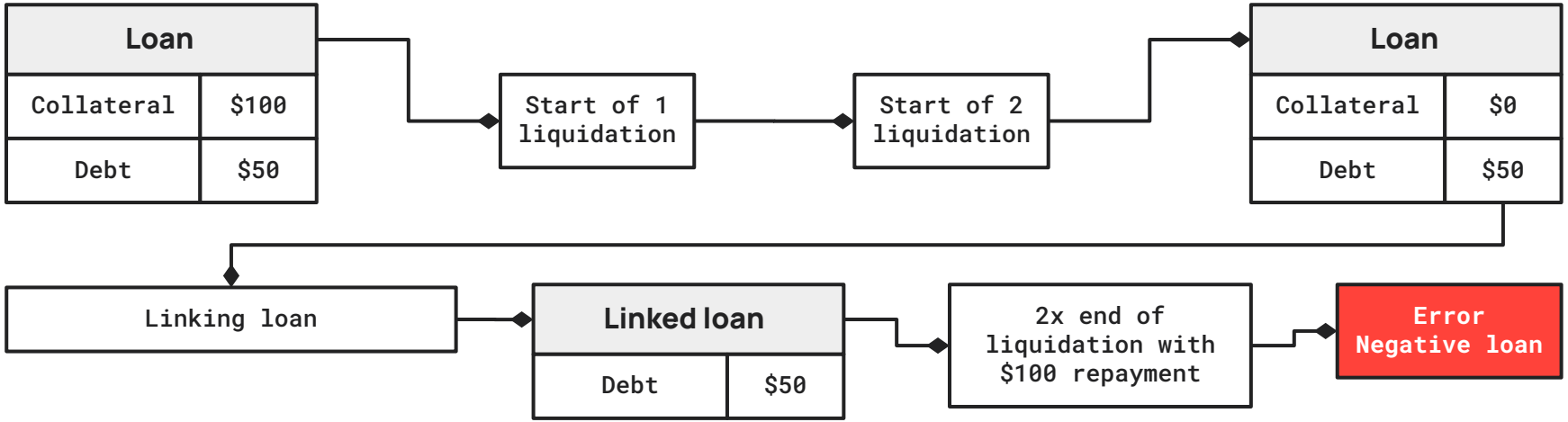
Liquidation of unhealthy loan

HACKEN

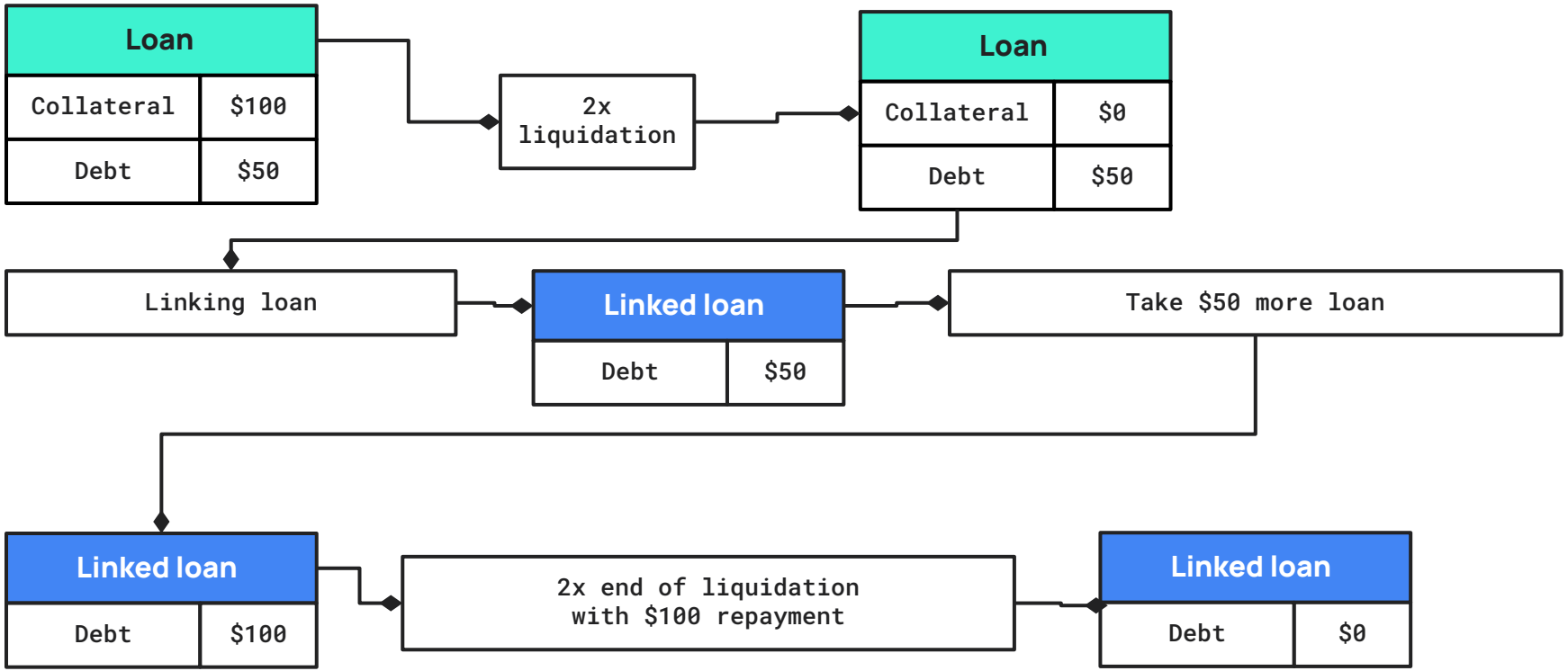


How to abuse it to create a loan with debt and no collateral

By starting liquidation two times it is possible to temporarily remove all collateral and create a loan with only debt. Then it is possible to link to loan to another loan. However, the transaction will fail during end of liquidation because it is not possible to repay $50 debt with $100 repayment.



How I made it work? I just need to get an extra debt during liquidation!



What does it accomplish? 1/2

So how this behavior can be abused?

It's simple, there's another bug during linking loan. When a loan is being linked to another one it is assumed it has no collateral and no debt. Because of that, the total debt of parent loan is not updated.

Master loan	
Collateral	$100
Discounted collateral	$50
Debt	$10
Debt to collateral	20%

Loan with no collateral	
Collateral	$0
Discounted collateral	$0
Debt	$10
Debt to collateral	inf%

Linking →

Master loan	
Collateral	$100
Discounted collateral	$50
Debt	$10
Debt to collateral	20%

No debt changes!

What does it accomplish? 2/2

However, when during the same transaction the debt for linked loan with no collateral is repaid then it is being removed from debt of a master loan!

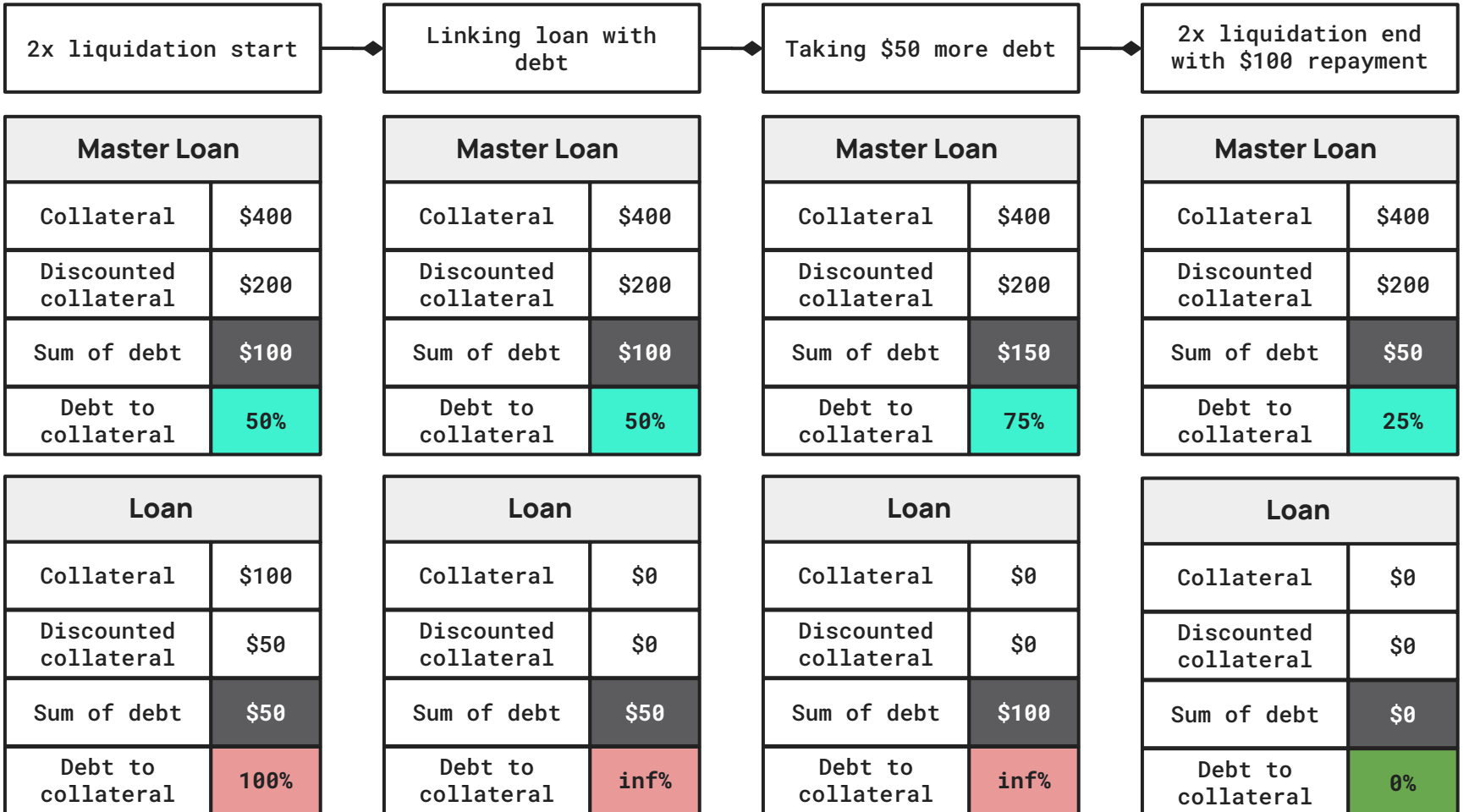
Master loan	
Collateral	$100
Discounted collateral	$50
Debt	$10
Debt to collateral	20%

Linked loan	
Collateral	$0
Discounted collateral	$0
Debt	$10
Debt to collateral	inf%

Repayment of linked loan →

Master loan	
Collateral	$100
Discounted collateral	$50
Debt	$0
Debt to collateral	0%

Debt is removed!


2x liquidation start
Linking loan with debt
Taking $50 more debt
2x liquidation end with $100 repayment
Master Loan
Collateral $400
Discounted collateral $200
Sum of debt $100
Debt to collateral 50%
Loan
Collateral $100
Discounted collateral $50
Sum of debt $50
Debt to collateral 100%
Master Loan
Collateral $400
Discounted collateral $200
Sum of debt $100
Debt to collateral 50%
Loan
Collateral $0
Discounted collateral $0
Sum of debt $50
Debt to collateral inf%
Master Loan
Collateral $400
Discounted collateral $200
Sum of debt $150
Debt to collateral 75%
Loan
Collateral $0
Discounted collateral $0
Sum of debt $100
Debt to collateral inf%
Master Loan
Collateral $400
Discounted collateral $200
Sum of debt $50
Debt to collateral 25%
Loan
Collateral $0
Discounted collateral $0
Sum of debt $0
Debt to collateral 0%

Time for exploit!

Coding the exploit

I spent 30 hours coding the 1,000+ line exploit, addressing issues like rounding errors, forced liquidation, transaction costs, and multi-asset support. The exploit ran on a mainnet ledger copy, simulating real transactions.

bin > simulation.rs > main

```rust
use scrypto::prelude::*;
pub use state_manager::store::ActualStateManagerDatabase;
use substate_store_impls::substate_database_overlay::SubstateDatabaseOverlay;
use weft_simulation::*;

▶ Run | Debug
fn main() {
    const STATE_MANAGER_DATABASE_PATH_ENVIRONMENT_VARIABLE: &str =
        "STATE_MANAGER_DATABASE_PATH";
    let state_manager_database_path: PathBuf = if let Ok(state_manager_database_path: PathBuf) =
```

Result:

$1.1M

Extracted
(in test environment)

15

minutes to steal everything

~1000

transactions

Then I contacted the projects and we fixed the issue =)

My recommendations

HACKEN

Recommendations

- Keep your projects as simple as you can
- Have a developer with experience in cybersecurity
- Take an audit
- Have a bug bounty program
- Write as many tests as you can, cover every use case
- Use monitoring platforms (eg. Hacken Extractor) to pause smart contract when attack is detected

@bbarwik

Bartosz Barwikowski

L1 Researcher & Auditor at Hacken