Quantstamp Security Assessment Certificate

January 11th 2021 — Quantstamp Verified

MakerDAO Dai Peg Stability Module

This security review was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Maker Protocol Module
Reviewers	Joseph Xu, Technical R&D Advisor Jan Gorzny, Blockchain Researcher Fayçal Lalidji, Security Auditor
Timeline	2020-12-08 through 2021-01-11
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	MIP29 - Peg Stability Module Maker Docs
Documentation Quality	High
Test Quality	Medium
Source Code	

Repository	Commit
dss-psm	8ee442f (initial report)
dss-psm	bcb1066 (updated report)

Total Issues	**5** (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**1** (0 Resolved)
Informational Risk Issues	**1** (0 Resolved)
Undetermined Risk Issues	**3** (3 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp conducted a security review on the Dai Peg Stability Module (PSM) and identified 5 issues, along with a number of best practices suggestions. Two issues are marked as having Low or Informational severity, while three issues are marked as having Undetermined severity due to the interactions with rest of the Maker Protocol. While we have not identified clear security threats that can harm the system, we recommend fully addressing the findings to better secure the smart contracts.

The scope of this security review was limited to smart contract issues and economic issues were not considered as part of this review.

Update: The three Undetermined severity findings have been fixed or mitigated as of commit `bcb1066`. The dev acknowledges the remaining findings but prefers to keep the code as-is. The PSM contract was deployed on the Mainnet on 2020-12-18.

ID	Description	Severity	Status
QSP-1	Insufficient Input Validation	Low	Acknowledged
QSP-2	Unlocked Pragma	Informational	Acknowledged
QSP-3	Unused Administrative Functions	? Undetermined	Fixed
QSP-4	Inconsistent Visibility and Authorization of Functions	? Undetermined	Mitigated
QSP-5	`sellGem()` and `buyGem()` Process Fees Differently	? Undetermined	Fixed

Quantstamp Review Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp reviewing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this security review.

Setup

Tool Setup:

- Slither v0.7.0
- Mythril v0.22.16

Steps taken to run the tools:

1. Installed the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`
3. Installed the Mythril tool: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth analyze path/to/contract`

Findings

QSP-1 Insufficient Input Validation

Severity: Low Risk

Status: Acknowledged

File(s) affected: `All contracts`

Description: The contracts are missing input validation for several important function arguments.

- In `join-5-auth.sol`, the functions `join()` and `exit()` do not check for trivial inputs such as `wad = 0`.
- In `lerp.sol`, the arguments `start_` and `end_` are not validated in the constructor. Extremely large values of `start_` and `end_` may cause overflow as indicated on L64 (though highly unlikely).
- In `psm.sol`, the argument `vow_` is not validated in the constructor.
- In `psm.sol`, the argument `data` is not validated in function `file()`.

We note that arguments that can affect `tin` and `tout` parameters that govern transaction fees within the Peg Stability Module should be validated as much as possible using reasonable maximum and minimum values. These include `start` and `end` variables in `lerp.sol`, as well as the `data` argument for function `file()` in `psm.sol`. Lack of input validation can leave the contract vulnerable to manual errors should transaction fees need to be updated in the future.

Recommendation: Introduce `require()` statements to validate key input parameters, especially ones related to the transaction fees.

Update: The dev acknowledges the comment but prefers to keep the code as-is.

QSP-2 Unlocked Pragma

Severity: Informational

Status: Acknowledged

File(s) affected: `All contracts`

Related Issue(s): SWC-103,

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.7`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

Update: The dev acknowledges the comment but prefers to keep the code as-is.

QSP-3 Unused Administrative Functions

Severity: Undetermined

Status: Fixed

File(s) affected: `psm.sol`

Description: Functions `hope()` and `nope()` are not used within the scope of the current repository. While we do not see immediate threat, these functions are quite important as they can be used to authorize `move()` and `frob()` functions within the `Vat` contract.

Recommendation: Clarify the purpose of having `hope()` and `nope()` functions in `psm.sol`.

Update: The dev has clarified the purpose of these functions as a comment within the `psm.sol` contract. These functions can be used to transfer control of the PSM vault to another contract; an example usage scenario would be contract upgrades.

QSP-4 Inconsistent Visibility and Authorization of Functions

Severity: Undetermined

Status: Mitigated

File(s) affected: `join-5-auth.sol, lerp.sol`

Related Issue(s): SWC-100

Description: We have noted functions with inconsistent visibility setting in relation to the rest of the Maker Protocol.

- **[Fixed]** In `join-5-auth.sol`, the function `join()` is currently set to `public` visibility. This is inconsistent with existing token adapter contracts in the Maker Protocol that set `join()` to `external` visibility.
- **[Fixed]** In `join-5-auth.sol`, the function `exit()` is currently set to `public` visibility. This is inconsistent with existing token adapter contracts in the Maker Protocol that set `exit()` to `external` visibility.

We have also noted functions that do not have the modifier `auth`, which could lead to undesirable system behaviors under certain assumptions.

- **[Acknowledged]** In `join-5-auth.sol`, the function `exit()` does not have the modifier `auth`. This opens up the possibility of an attacker receiving Dai for free if the attacker can either (i) call `DssPsm.hope()` or (ii) modify the PSM's gem balance using `Vat.frob()` as described in the exploit scenario below. Either of these assumptions are extremely difficult to satisfy so there is no immediate threat, but a proper authorization so that only the PSM can call `join()` and `exit()` would eliminate such risk.
- **[Mitigated]** In `lerp.sol`, the function `tick()` does not have the modifier `auth`. This allows anyone to update the transaction fee, which may be undesirable.

Exploit Scenario: Consider the following procedure for an attacker to receive Dai for free:

1. The attacker calls `DssPsm.sellGem()` to exchange a collateral (gem) with Dai on a 1:1 rate.
2. The attacker uses `DssPsm.hope()` to enable manipulation of the PSM contract's gem balance in the `Vat` contract.
3. The attacker calls `Vat.frob()` to manipulate the gem balance in the `Vat` contract to move the collateral amount in Step 1 to the attacker's own address.
4. The attacker calls the public `AuthGemJoin5.exit()` to get back the collateral that has been used in Step 1, completing the transaction sequence.

Note that this attack is only possible if the PSM or the `Vat` contract are somehow vulnerable (e.g., compromised admin key) so that the attacker can call either `DssPsm.hope()` or

`Vat.frob()`.

Recommendation: Set the appropriate visibility and authorization for the affected functions.

Update: Contract `join-5-auth.sol` has been updated so that functions `join()` and `exit()` now have `external` visibility. The dev acknowledges the comment regarding modifier `auth` for the function `exit()` but prefers to keep the code as-is. Contract `lerp.sol` has been modified to de-auth itself by calling `deny()` upon completion. The PSM contract was deployed on the Mainnet on 2020-12-18 and operations involving the `lerp.sol` contract have been completed without any issue.

QSP-5 `sellGem()` and `buyGem()` Process Fees Differently

Severity: Undetermined

Status: Fixed

File(s) affected: `psm.sol`

Description: The `sellGem()` function and `buyGem()` function handle fees differently from an operational perspective. A user calling `sellGem()` can simply transfer a collateral token (gem) and receive back Dai less fees. On the other hand, a user calling `buyGem()` would need to transfer Dai plus fees to receive back a collateral token. This difference may lead to usability-related issues.

Recommendation: Clarify if this is the intended design.

Update: The dev has indicated that this approach to handling fees is a conscious decision to avoid dealing with decimal division and rounding leftovers.

Automated Analyses

Slither

Slither found 23 results using 72 detectors across 11 contracts (including inherited library contracts). We have eliminated false positives and report only relevant issues.

For `lerp.sol`:

```
INFO:Detectors:
Reentrancy in Lerp.init() (src/lerp.sol#52-58):
	External calls:
	- target.file(what,start) (src/lerp.sol#55)
	State variables written after the call(s):
	- started = true (src/lerp.sol#57)
Reentrancy in Lerp.tick() (src/lerp.sol#60-76):
	External calls:
	- target.file(what,end) (src/lerp.sol#72)
	- target.deny(address(this)) (src/lerp.sol#73)
	State variables written after the call(s):
	- done = true (src/lerp.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Reentrancy in Lerp.init() (src/lerp.sol#52-58):
	External calls:
	- target.file(what,start) (src/lerp.sol#55)
	State variables written after the call(s):
	- startTime = block.timestamp (src/lerp.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Lerp.add(uint256,uint256) (src/lerp.sol#19-21) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool)((z = x + y) >= x) (src/lerp.sol#20)
Lerp.sub(uint256,uint256) (src/lerp.sol#22-24) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool)((z = x - y) <= x) (src/lerp.sol#23)
Lerp.tick() (src/lerp.sol#60-76) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp > startTime,Lerp/no-time-elasped) (src/lerp.sol#62)
	- block.timestamp < add(startTime,duration) (src/lerp.sol#64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

For `psm.sol`:

```
INFO:Detectors:
DssPsm.constructor(address,address,address) (src/psm.sol#47-59) ignores return value by dai__.approve(daiJoin_,uint256(- 1)) (src/psm.sol#57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DssPsm.constructor(address,address,address).vow_ (src/psm.sol#47) lacks a zero-check on :
		- vow = vow_ (src/psm.sol#55)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in DssPsm.buyGem(address,uint256) (src/psm.sol#105-116):
	External calls:
	- require(bool,string)(dai.transferFrom(msg.sender,address(this),daiAmt),DssPsm/failed-transfer) (src/psm.sol#109)
	- daiJoin.join(address(this),daiAmt) (src/psm.sol#110)
	- vat.frob(ilk,address(this),address(this),address(this),- int256(gemAmt18),- int256(gemAmt18)) (src/psm.sol#111)
	- gemJoin.exit(usr,gemAmt) (src/psm.sol#112)
	- vat.move(address(this),vow,mul(fee,RAY)) (src/psm.sol#113)
	Event emitted after the call(s):
	- BuyGem(usr,gemAmt,fee) (src/psm.sol#115)
Reentrancy in DssPsm.sellGem(address,uint256) (src/psm.sol#93-103):
	External calls:
	- gemJoin.join(address(this),gemAmt,msg.sender) (src/psm.sol#97)
	- vat.frob(ilk,address(this),address(this),address(this),int256(gemAmt18),int256(gemAmt18)) (src/psm.sol#98)
	- vat.move(address(this),vow,mul(fee,RAY)) (src/psm.sol#99)
	- daiJoin.exit(usr,daiAmt) (src/psm.sol#100)
	Event emitted after the call(s):
	- SellGem(usr,gemAmt,fee) (src/psm.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

Mythril

Mythril reports the following results for `join-5-auth.sol`.

- External call to user supplied address (SWC-107) in function `join(address,uint256,address)`.
- External call to user supplied address (SWC-107) in function `exit(address,uint256,address)`.
- Multiple calls in a single transaction (SWC-113) in function `join(address,uint256,address)`.
- Multiple calls in a single transaction (SWC-113) in function `exit(address,uint256,address)`.

Mythril did not detect any issue for `lerp.sol`. Mythril failed to analyze `psm.sol`.

Adherence to Best Practices

- The contracts are sparsely commented. Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec) - see Solidity's official documentation. It is recommended to fully annotate all external and public functions in the Solidity contracts using the NatSpec format.

- State variables are updated after external calls in `lerp.sol`. As a best practice, the checks-effects-interactions pattern should be used so that state variables are updated before an external call.

- Events are emitted after external calls in `psm.sol`. As a best practice, the checks-effects-interactions pattern should be used so that events are emitted before external call.

- Certain failure messages are repeated within the same contract, which can lead to the messages being uninformative upon transaction failure. Specifically, we note repeated failure messages in `join-5-auth.sol` (L65, 71, 78; L73, 80) and `lerp.sol` (L53, 62).

Test Results

Test Suite Results

All tests pass.

```
Running 13 tests for src/psm.t.sol:DssPsmTest
[PASS] testFail_sellGem_over_line() (gas: 44383)
[PASS] test_sellGem_fee() (gas: 423917)
[PASS] test_swap_both_other_small_fee() (gas: 1094161)
[PASS] test_lerp_tin() (gas: 910000)
[PASS] test_swap_both_other() (gas: 1067000)
[PASS] test_swap_both_fees() (gas: 645471)
[PASS] testFail_swap_both_small_fee_insufficient_dai() (gas: 868113)
[PASS] testFail_sellGem_insufficient_gem() (gas: 502893)
[PASS] testFail_direct_deposit() (gas: 32319)
[PASS] test_swap_both_no_fee() (gas: 563651)
[PASS] testFail_two_users_insufficient_dai() (gas: 1278463)
[PASS] test_swap_both_zero() (gas: 279585)
[PASS] test_sellGem_no_fee() (gas: 390794)
```

Code Coverage

Coverage analysis can be obtained using `hevm dapp-test --coverage`. This command outputs the source code with annotations. The annotated main contract source code can be found below. Summary statistics of code coverage is unavailable.

```
***** hevm coverage for src/join-5-  auth.sol

;;;;; // SPDX-License-Identifier: AGPL-3.0-or-later
.....
;;;;; /// join-5-  auth.sol -- Non-standard token adapters
.....
;;;;; // Copyright (C) 2018 Rain <rainbreak@riseup.net>
;;;;; // Copyright (C) 2018-2020 Maker Ecosystem Growth Holdings, INC.
;;;;; //
;;;;; // This program is free software: you can redistribute it and/or modify
;;;;; // it under the terms of the GNU Affero General Public License as published by
;;;;; // the Free Software Foundation, either version 3 of the License, or
;;;;; // (at your option) any later version.
;;;;; //
;;;;; // This program is distributed in the hope that it will be useful,
;;;;; // but WITHOUT ANY WARRANTY; without even the implied warranty of
;;;;; // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;;;;; // GNU Affero General Public License for more details.
;;;;; //
;;;;; // You should have received a copy of the GNU Affero General Public License
;;;;; // along with this program.  If not, see <https://www.gnu.org/licenses/>.
.....
;;;;; pragma solidity ^0.6.7;
.....
;;;;; import "dss/  lib.sol";
.....
;;;;; interface VatLike {
;;;;;     function slip(bytes32, address, int256) external;
..... }
.....
;;;;; interface GemLike {
;;;;;     function decimals() external view returns (uint8);
;;;;;     function transfer(address, uint256) external returns (bool);
;;;;;     function transferFrom(address, address, uint256) external returns (bool);
..... }
.....
;;;;; // Authed GemJoin for a token that has a lower precision than 18 and it has decimals (like USDC)
.....
;;;;; contract AuthGemJoin5 is LibNote {
;;;;;     // --- Auth ---
#####     mapping (address => uint256) public wards;
          function rely(address usr) external note auth { wards[usr] = 1; }
          function deny(address usr) external note auth { wards[usr] = 0; }
          modifier auth { require(wards[msg.sender] == 1); _; }
.....
          VatLike public vat;
          bytes32 public ilk;
          GemLike public gem;
          uint256 public dec;
#####     uint256 public live;  // Access Flag
.....
          constructor(address vat_, bytes32 ilk_, address gem_) public {
              gem = GemLike(gem_);
              dec = gem.decimals();
              require(dec < 18, "GemJoin5/decimals-18-or-higher");
              wards[msg.sender] = 1;
              live = 1;
              vat = VatLike(vat_);
              ilk = ilk_;
.....     }
.....
#####     function cage() external note auth {
#####         live = 0;
.....     }
.....
          function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
              require(y == 0 || (z = x * y) / y == x, "GemJoin5/overflow");
.....     }
.....
          function join(address urn, uint256 wad, address _msgSender) external note auth {
              require(live == 1, "GemJoin5/not-live");
              uint256 wad18 = mul(wad, 10 ** (18 - dec));
              require(int256(wad18) >= 0, "GemJoin5/overflow");
              vat.slip(ilk, urn, int256(wad18));
              require(gem.transferFrom(_msgSender, address(this), wad), "GemJoin5/failed-transfer");
.....     }
.....
          function exit(address guy, uint256 wad) external note {
              uint256 wad18 = mul(wad, 10 ** (18 - dec));
              require(int256(wad18) >= 0, "GemJoin5/overflow");
              vat.slip(ilk, msg.sender, -int256(wad18));
              require(gem.transfer(guy, wad), "GemJoin5/failed-transfer");
.....     }
..... }

***** hevm coverage for src/  lerp.sol

;;;;; pragma solidity ^0.6.7;
.....
;;;;; interface FileLike {
;;;;;     function deny(address) external;
;;;;;     function file(bytes32, uint256) external;
..... }
.....
;;;;; // Perform linear interpolation on a dss administrative value over time
.....
;;;;; contract Lerp {
.....
;;;;;     // --- Auth ---
#####     mapping (address => uint256) public wards;
#####     function rely(address usr) external auth { wards[usr] = 1; }
#####     function deny(address usr) external auth { wards[usr] = 0; }
#####     modifier auth { require(wards[msg.sender] == 1); _; }
.....
#####     uint256 constant WAD = 10 ** 18;
#####     function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require((z = x + y) >= x);
.....     }
#####     function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require((z = x - y) <= x);
.....     }
#####     function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require(y == 0 || (z = x * y) / y == x);
.....     }
.....
#####     FileLike immutable public target;
#####     bytes32 immutable public what;
#####     uint256 immutable public start;
#####     uint256 immutable public end;
#####     uint256 immutable public duration;
.....
#####     bool public started;
#####     bool public done;
#####     uint256 public startTime;
.....
          constructor(address target_, bytes32 what_, uint256 start_, uint256 end_, uint256 duration_) public {
              require(duration_ != 0, "Lerp/no-zero-duration");
              require(start_ != end_, "Lerp/start-end-equal");
              target = FileLike(target_);
              what = what_;
              start = start_;
              end = end_;
              duration = duration_;
              started = false;
              done = false;
              wards[msg.sender] = 1;
.....     }
.....
#####     function init() external auth {
#####         require(!started, "Lerp/already-started");
#####         require(!done, "Lerp/finished");
#####         target.file(what, start);
#####         startTime = block.timestamp;
#####         started = true;
.....     }
.....
#####     function tick() external {
#####         require(started, "Lerp/not-started");
#####         require(block.timestamp > startTime, "Lerp/no-time-elasped");
#####         require(!done, "Lerp/finished");
#####         if (block.timestamp < add(startTime, duration)) {
;;;;;             // 0 <= t < WAD
#####             uint256 t = mul(WAD, sub(block.timestamp, startTime)) / duration;
;;;;;             // y = (end - start) * t + start [Linear Interpolation]
;;;;;             //   = end * t + start - start * t [Avoids overflow by moving the subtraction to the end]
```

```
#####             target.file(what, sub(add(mul(end, t) / WAD, start), mul(start, t) / WAD));
;;;;;         } else {
;;;;;             // Set the end value and de-auth yourself
#####             target.file(what, end);
#####             target.deny(address(this));
#####             done = true;
.....         }
.....     }
.....
..... }

***** hevm coverage for src/  psm.sol

;;;;; pragma solidity ^0.6.7;
.....
;;;;; import { DaiJoinAbstract } from "dss-interfaces/dss/  DaiJoinAbstract.sol";
;;;;; import { DaiAbstract } from "dss-interfaces/dss/  DaiAbstract.sol";
;;;;; import { VatAbstract } from "dss-interfaces/dss/  VatAbstract.sol";
.....
;;;;; interface AuthGemJoinAbstract {
;;;;;     function dec() external view returns (uint256);
;;;;;     function vat() external view returns (address);
;;;;;     function ilk() external view returns (bytes32);
;;;;;     function join(address, uint256, address) external;
;;;;;     function exit(address, uint256) external;
..... }
.....
;;;;; // Peg Stability Module
;;;;; // Allows anyone to go between Dai and the Gem by pooling the liquidity
;;;;; // An optional fee is charged for incoming and outgoing transfers
.....
;;;;; contract DssPsm {
.....
;;;;;     // --- Auth ---
#####     mapping (address => uint256) public wards;
#####     function rely(address usr) external auth { wards[usr] = 1; emit Rely(usr); }
#####     function deny(address usr) external auth { wards[usr] = 0; emit Deny(usr); }
#####     modifier auth { require(wards[msg.sender] == 1); _; }
.....
#####     VatAbstract immutable public vat;
#####     AuthGemJoinAbstract immutable public gemJoin;
#####     DaiAbstract immutable public dai;
#####     DaiJoinAbstract immutable public daiJoin;
#####     bytes32 immutable public ilk;
#####     address immutable public vow;
.....
;;;;;     uint256 immutable internal to18ConversionFactor;
.....
#####     uint256 public tin;         // toll in [wad]
#####     uint256 public tout;        // toll out [wad]
.....
;;;;;     // --- Events ---
;;;;;     event Rely(address indexed usr);
;;;;;     event Deny(address indexed usr);
;;;;;     event File(bytes32 indexed what, uint256 data);
;;;;;     event SellGem(address indexed owner, uint256 value, uint256 fee);
;;;;;     event BuyGem(address indexed owner, uint256 value, uint256 fee);
.....
;;;;;     // --- Init ---
          constructor(address gemJoin_, address daiJoin_, address vow_) public {
              wards[msg.sender] = 1;
              emit Rely(msg.sender);
              AuthGemJoinAbstract gemJoin__ = gemJoin = AuthGemJoinAbstract(gemJoin_);
              DaiJoinAbstract daiJoin__ = daiJoin = DaiJoinAbstract(daiJoin_);
              VatAbstract vat__ = vat = VatAbstract(address(gemJoin__.vat()));
              DaiAbstract dai__ = dai = DaiAbstract(address(daiJoin__.dai()));
              ilk = gemJoin__.ilk();
              vow = vow_;
              to18ConversionFactor = 10 ** (18 - gemJoin__.dec());
              dai__.approve(daiJoin_, uint256(-1));
              vat__.hope(daiJoin_);
.....     }
.....
;;;;;     // --- Math ---
#####     uint256 constant WAD = 10 ** 18;
#####     uint256 constant RAY = 10 ** 27;
#####     function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require((z = x + y) >= x);
.....     }
#####     function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require((z = x - y) <= x);
.....     }
#####     function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
#####         require(y == 0 || (z = x * y) / y == x);
.....     }
.....
;;;;;     // --- Administration ---
#####     function file(bytes32 what, uint256 data) external auth {
#####         if (what == "tin") tin = data;
#####         else if (what == "tout") tout = data;
#####         else revert("DssPsm/file-unrecognized-param");
.....
#####         emit File(what, data);
.....     }
.....
;;;;;     // hope can be used to transfer control of the PSM vault to another contract
;;;;;     // This can be used to upgrade the contract
#####     function hope(address usr) external auth {
#####         vat.hope(usr);
.....     }
#####     function nope(address usr) external auth {
#####         vat.nope(usr);
.....     }
.....
;;;;;     // --- Primary Functions ---
#####     function sellGem(address usr, uint256 gemAmt) external {
#####         uint256 gemAmt18 = mul(gemAmt, to18ConversionFactor);
#####         uint256 fee = mul(gemAmt18, tin) / WAD;
#####         uint256 daiAmt = sub(gemAmt18, fee);
#####         gemJoin.join(address(this), gemAmt, msg.sender);
#####         vat.frob(ilk, address(this), address(this), address(this), int256(gemAmt18), int256(gemAmt18));
#####         vat.move(address(this), vow, mul(fee, RAY));
#####         daiJoin.exit(usr, daiAmt);
.....
#####         emit SellGem(usr, gemAmt, fee);
.....     }
.....
#####     function buyGem(address usr, uint256 gemAmt) external {
#####         uint256 gemAmt18 = mul(gemAmt, to18ConversionFactor);
#####         uint256 fee = mul(gemAmt18, tout) / WAD;
#####         uint256 daiAmt = add(gemAmt18, fee);
#####         require(dai.transferFrom(msg.sender, address(this), daiAmt), "DssPsm/failed-transfer");
#####         daiJoin.join(address(this), daiAmt);
#####         vat.frob(ilk, address(this), address(this), address(this), -int256(gemAmt18), -int256(gemAmt18));
#####         gemJoin.exit(usr, gemAmt);
#####         vat.move(address(this), vow, mul(fee, RAY));
.....
#####         emit BuyGem(usr, gemAmt, fee);
.....     }
.....
..... }
```

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

`c10a60551d589934fd6b84f7e44b60b1c6024731f79966a986576f6e0ae260c9 ./src/psm.sol`

`3e9db1571ec7b08fb7d8a7fc783f7476d31ea346f6864b0c56452a3295e290de ./src/lerp.sol`

`b20dc072f70eeb40d1eb3ac9ff9288049ab7d397149775f7997445ca7f847cab ./src/join-5-auth.sol`

Tests

`b5a8f1d4966ddf6e44767f5bace5872097b80bf879dcdb6517011a2af4b32e92 ./src/psm.t.sol`

Changelog

- 2020-12-17 - Initial report
- 2021-01-11 - Updated report based on commit `bcb1066`

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.