// HALBORN

PACMAN TOKEN

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: **12.10-12.2020**
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/10/2020	Gabi Urrutia
1.0	Document Final	12/10/2020	Steven Walbroehl

CONTACTS

CONTACT	COMPANY	EMAIL
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

1.1 INTRODUCTION

PACMAN engaged Halborn to conduct a security assessment on their Token smart contracts beginning on December 10th, 2020 and ending December 12th, 2020. The security assessment was scoped to Token contracts and an audit of the security risk and implications regarding the changes introduced by the development team at PACMAN prior to its production release shortly following the assessments deadline.

Contracts scoped in this assessment are Token Contracts: PAC.sol. PACMAN Token is an upgradeable token through a proxy contract when the token code becomes stale. Thus, the Token is upgraded in other the referenced contract PAC.sol, creating a new token. However, in this audit, the Upgraded token contract is out of the scope.

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties related to the PACMAN Token Contracts was performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter- contract functions and calls should be validated for expected logic and state

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of PACMAN Token as an upgradable token.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Truffle, Ganache)

1.3 SCOPE

IN-SCOPE:
Code related to PACMAN smart contracts. Specific commit of contracts:
0xAE2c8C4CB3CD69D0Dfa4490c024b6b1aEC64223f

OUT-OF-SCOPE:
External contracts, External Oracles, other smart contracts in the repository or imported by PACMAN, economic attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	2

SECURITY ANALYSIS	RISK LEVEL
DEPRECATED PRAGMA VERSION OF SOLC	Low
EXPERIMENTAL FEATURES ENABLED	Low
STATIC ANALYSIS REPORT	Informational
AUTOMATED SECURITY SCAN REPORT	Informational

FINDINGS & TECH DETAILS

3.1 DEPRECATED PRAGMA VERSION OF SOLC – LOW

Description

The current version in use for PACMAN token contracts is pragma 0.6.11. While this version is still functional, and most security issues safely implemented by mitigating the PACMAN contracts with other utility contracts such as SafeMath.sol and ReentrancyGuard.sol, the risk to the long-term sustainability and integrity of the solidity code increases.

Code Location:

PAC.sol Line #2

```
1   // SPDX-License-Identifier: UNLICENSED
2   pragma solidity 0.6.11;
```

Deprecated or Upgraded Items

The follow list identifies areas of code improvements, and deprecated functionality that will need to be refactored into the existing version in order to come to the latest pragma level.

UPDATED OR DEPRECATED FEATURE DETAIL	VERSION RELEASED	IMPACTED CODE
Functions		
The state mutability of functions can now be restricted during inheritance: Functions with default state mutability can be overridden by pure and view functions while view functions can be overridden by pure functions	v0.7.0	None Located in Manual Audit
Disallow virtual for library functions. This is a new type checker.	v0.7.0	None Located in Manual Audit
Multiple events with the same name and	v0.7.0	None Located in Manual

parameter types in the same inheritance hierarchy are disallowed.		Audit

Conversions

Exponentiation and shifts of literals by non-literals will always use either the type uint256 (for non-negative literals) or int256 (for negative literals) to perform the operation	v0.7.0	None Located in Manual Audit

Inline Assembly

disallowed. (a period) in user-defined function and variable names in inline assembly. It is still valid if you use Solidity in Yul-only mode.	v0.7.0	None Located in Manual Audit
Slot and offset of storage pointer variable x are accessed via x.slot and x.offset instead of x_slot and x_offset.	v0.7.0	None Located in Manual Audit

Variables

Removal of unsafe features and methods. If a struct or array contains a mapping, it can only be used in storage. Previously, mapping members were silently skipped in memory, which is confusing and error-prone.	v0.7.0	None Located in Manual Audit
Assignments to structs or arrays in storage does not work if they contain mappings. Previously, mappings were silently skipped during the copy operation, which is misleading and error-prone.	v0.7.0	None Located in Manual Audit

Recommendation:
At the time of this audit, the current version is already at 0.7. When possible, use the updated pragma versions to take advantage of new features that provide checks and accounting, as well as prevent insecure use of code.

3.2 EXPERIMENTAL FEATURES ENABLED - LOW

Description:

ABIEncoderV2 is enabled to be able to pass struct type into a function both web3 and another contract. The use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(...) as arguments in external function calls or in event data without prior assignment to a local variable. Using return does not trigger the bug. The types bytesNN and bool will result in corrupted data while enum might lead to an invalid revert.

Furthermore, arrays with elements shorter than 32 bytes may not be handled correctly even if the base type is an integer type. Encoding such arrays in the way described above can lead to other data in the encoding being overwritten if the number of elements encoded is not a multiple of the number of elements that fit a single slot. If nothing follows the array in the encoding (note that dynamically-sized arrays are always encoded after statically- sized arrays with statically-sized content), or if only a single array is encoded, no other data is overwritten. There are known bugs that are publicly released while using this feature. However, the bug only manifests itself when all of the following conditions are met:

- Storage data involving arrays or structs is sent directly to an external function call, to abi.encode or to event data without prior assignment to a local (memory) variable
- There is an array that contains elements with size less than 32 bytes or a struct that has elements that share a storage slot or members of type bytesNN shorter than 32 bytes.

In addition to that, in the following situations, your code is NOT affected:

- All the structs or arrays only use uint256 or int256 types
- If you only use integer types (that may be shorter) and only encode at most one array at a time
- If you only return such data and do not use it in abi.encode, external calls or event data.

Reference:
https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2-bug/

Code Location:
PAC.sol Line #3

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity 0.6.11;
3  pragma experimental ABIEncoderV2;
```

ABIEncoderV2 is enabled to be able to pass struct type into a function both web3 and another contract. Naturally, any bug can have wildly varying consequences depending on the program control flow, but we expect that this is more likely to lead to malfunction than exploitability. The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

Recommendation:
When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256)

3.3 STATIC ANALYSIS REPORT- INFORMATIONAL

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the Token contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:

Static analysis results show some warnings. This is due to the fact that there are external calls where the state variables get written afterwards. However, these findings are false positives due to the fact that Open Zeppelin library of mitigation contracts are in pace. In particular “ReentrancyGuard.sol” protects the PACMAN contracts and prevents exploitation.

```
INFO:Detectors:
Parameter APYPoolToken.initialize(address,IERC20,AggregatorV3Interface)._underlyer (APYPoolToken.sol#41) is not in mixedCase
Parameter APYPoolToken.initialize(address,IERC20,AggregatorV3Interface)._priceAgg (APYPoolToken.sol#42) is not in mixedCase
Parameter APYPoolToken.setPriceAggregator(AggregatorV3Interface)._priceAgg (APYPoolToken.sol#72) is not in mixedCase
Function ContextUpgradeSafe.__Context_init() (Context.sol#18-20) is not in mixedCase
Function ContextUpgradeSafe.__Context_init_unchained() (Context.sol#22-25) is not in mixedCase
Variable ContextUpgradeSafe.__gap (Context.sol#37) is not in mixedCase
Function ERC20UpgradeSafe.__ERC20_init(string,string) (ERC20.sol#57-60) is not in mixedCase
Function ERC20UpgradeSafe.__ERC20_init_unchained(string,string) (ERC20.sol#62-69) is not in mixedCase
Variable ERC20UpgradeSafe.__gap (ERC20.sol#317) is not in mixedCase
Variable Initializable._____gap (Initializable.sol#61) is not in mixedCase
Function OwnableUpgradeSafe.__Ownable_init() (Ownable.sol#26-29) is not in mixedCase
Function OwnableUpgradeSafe.__Ownable_init_unchained() (Ownable.sol#31-38) is not in mixedCase
Variable OwnableUpgradeSafe.__gap (Ownable.sol#78) is not in mixedCase
Function PausableUpgradeSafe.__Pausable_init() (Pausable.sol#32-35) is not in mixedCase
Function PausableUpgradeSafe.__Pausable_init_unchained() (Pausable.sol#37-42) is not in mixedCase
Variable PausableUpgradeSafe.__gap (Pausable.sol#84) is not in mixedCase
Function ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init() (ReentrancyGuard.sol#24-26) is not in mixedCase
Function ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init_unchained() (ReentrancyGuard.sol#28-39) is not in mixedCase
Variable ReentrancyGuardUpgradeSafe.__gap (ReentrancyGuard.sol#63) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

Code syntax detections. Not security related.

The other finding is involved with using dangerous strict equalities, which can be manipulated by an attacker in some circumstances. In this case, we can see that this equality function is being used to calculate amounts and total amounts.

Code Location:

PAC.sol Line #151

```
146     function getEthValueFromTokenAmount(uint256 amount↑)
147         public
148         view
149         returns (uint256)
150     {
151         if (amount↑ == 0) {
152             return 0;
153         }
```

PAC.sol Line #251

```
242     function _calculateMintAmount(
243         uint256 depositEthAmount↑,
244         uint256 totalEthAmount↑
245     ) internal view returns (uint256) {
246         // NOTE: When totalSupply > 0 && totalEthAmount == 0
247         // others can lay claim to other users deposits
248
249         uint256 totalSupply = totalSupply();
250
251         if (totalEthAmount↑ == 0 || totalSupply == 0) {
252             return depositEthAmount↑.mul(DEFAULT_APT_TO_UNDERLYER_FACTOR);
253         }
```

Recommendation:

Consider use of >= or <= rather than ==, to prevent an attacker from trapping the contract due to strict equalities.

3.4 AUTOMATED SECURITY SCAN – INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with PAC.sol.

Results:

MythX detected 0 High findings, 0 Medium, and 3 Low.

0 High		0 Medium		3 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-131	Low	Unused function parameter "from".	ERC20.sol	L: 315 C: 34
SWC-131	Low	Unused function parameter "to".	ERC20.sol	L: 315 C: 48
SWC-131	Low	Unused function parameter "amount".	ERC20.sol	L: 315 C: 60

MythX detected 0 High findings, 1 Medium, and 0 Low.

0 High		1 Medium		0 Low
ID	SEVERITY	NAME	FILE	LOCATION
SWC-110	Medium	An assertion violation was triggered.	UpgradeableProxy.sol	L: 48 C: 2

THANK YOU FOR CHOOSING

// HALBORN