VULNERABILITY ASSESSMENT

Raven | P2SH

SENSITIVE INFORMATION – DISTRIBUTE WITH CAUTION

Executive Summary

Raven engaged Independent Security Evaluators (ISE) to evaluate the security posture of Raven's P2SH, a transaction method built off Bitcoin's pay to script hash (P2SH).[1] Raven's P2SH focuses on transferring tokenized assets. ISE performed an assessment to discover vulnerabilities within the system that could lead to the compromise of Raven's proprietary data, user assets, or the availability of the service.

ISE considers attack surfaces, permissions, and application logic specific to P2SH that an advanced attacker may exploit, and manually tests against such exploits. ISE uses automated tools to gain an understanding of the system and identify common issues, but the focus of the assessment is discovering vulnerabilities that scanners will miss. ISE reviews all reported findings for accuracy and assigns severity based on exploit complexity, impact, and attack chaining.

ISE has assessed the following components:

- Source code changes, GitHub pull request #873 ("Add P2SH support")

A summary of discovered issues is shown in the table below.

	Critical	High	Medium	Low	Info	Total
Discovered	0	0	0	0	4	4
Resolved	0	0	0	0	0	0
Closed	0	0	0	0	0	0
Remaining	**0**	**0**	**0**	**0**	**4**	**4**

A history of ISE's assessments is shown in the table below.

Revision	Date	Description
1	June 2021	Initial assessment of source code changes in GitHub pull request #873 ("Add P2SH support")

This report expires on June 30, 2022. Expiration facilitates ongoing communication and assessment efforts to address changes in technology, the product, and its supporting environment. ISE recommends that Raven takes action to address current issues and continue security assessments to identify additional issues. A list of recommendations for further security review can be found in the Additional Recommendations section.

[1] https://en.bitcoin.it/wiki/BIP_0016

Summary of Findings

The following tables contain the vulnerabilities ISE has identified in P2SH, along with their severity and status. Each issue's severity is derived from the issue's impact to P2SH's assets and its exploitability. More information can be found in the Severity Ratings, Statuses, and Assessment Results sections.

Source Code

Vulnerability	Identifier	Severity	Status
Missing Bounds Check on Vector Index	ISE-RAVEN-P2SH-R1-01	Info	Unresolved
Use of Uninitialized Variables	ISE-RAVEN-P2SH-R1-02	Info	Unresolved
Variable Naming Inconsistencies	ISE-RAVEN-P2SH-R1-03	Info	Unresolved
Missing Validation of scriptType	ISE-RAVEN-P2SH-R1-04	Info	Unresolved

Table of Contents

Introduction

Raven contracted Independent Security Evaluators (ISE) to evaluate the security of the P2SH system. ISE performed an assessment to discover vulnerabilities within P2SH that could lead to unwanted results, such as compromise of assets or user information, disruption of service, or leveraging P2SH's systems or functionality for other attacks.

System Overview

Raven is a peer-to-peer, open-source blockchain project with a focus on asset creation, transfer, and governance. The Raven codebase is built off a fork from Bitcoin, consisting mostly of C and C++ code. Raven assets are meant to serve as a representation of a physical or digital item of value. Assets are represented by a quantity of Ravencoins (RVN). This enables assets to be divided into shares. Assets can also be transferred from one Raven address to another using the same types of scripts as those available to Bitcoin. Raven also implements custom scripts, such as a pay to script hash (P2SH) protocol for asset transfers.

Assets

Before accurately assessing a system's security posture, it is necessary to identify assets and their value. Assets include tangible elements such as information or equipment and extend to more abstract elements such as reputation. The impact of the loss of these assets should be quantified to the degree possible; however, this can be a difficult and subjective process and is outside the scope of ISE's insight into the client's operations.

- User information
    - Private keys
    - Redeem scripts
- Financial assets
    - Ravencoin cryptocurrency (RVN)
    - Ravencoin assets
- Access and availability
- Business reputation

Scope

ISE's evaluation covered the following components of P2SH:

- Changes to Raven codebase submitted in GitHub pull request #873

The following were not included in ISE's assessment and should be included in future assessments:

- iOS Wallet
- Android Wallet
- Mining algorithms

- Messaging
- Voting and governance
- Issuing and transfer of assets

Methodology

ISE specializes in hands-on assessments that consider assets, attack surfaces, permissions, and logic specific to the target system. The goal is to discover security issues that a variety of attackers may exploit, and manually test against such exploits using access to the platform, e.g., test accounts, server access, source code, documentation, etc. In general, ISE encourages sharing as much relevant access as possible because a deeper knowledge of the system facilitates more efficient testing and more valuable results.

ISE reviews all reported findings for accuracy and assigns severity based on exploit complexity, impact, and attack chaining.

Application Testing

ISE assessed Raven's asset P2SH by manually testing its functionality for vulnerabilities and developing custom exploits where applicable. ISE used automated tools to gain an understanding of the application and identify common issues, but the focus of the assessment was identifying issues that a scanner would miss. The following is a list of core areas of testing:

- Manual source code review includes:
    - Reviewing all source code changes in the pull request to spot inconsistencies, logical flaws, and unsafe usage of the C++ language, such as undefined behavior and potential memory corruption vulnerabilities.
    - Reviewing the functions called by new code to trace logical flow through the program. The goal is to ensure that return values and arguments are properly used and validated. Validation includes verifying that variable assignments fall under an expected value and confirming that buffers are bounds checked before use.
    - Reviewing how new code fits into existing functions. This includes ensuring that logical flow is consistent, there are no dead branches, data structures are used properly, and that new code does not break the original design of the function.
    - Reviewing code quality to ensure that code complexity is not increased more than necessary. Keeping code complexity low makes the codebase easier to manage and debug and reduces the chances of programming errors. Part of this review includes ensuring that naming conventions across variables are consistent, that code duplication is kept to a minimum, and that large code chunks are broken up into functions.
    - Reviewing the use of design patterns to ensure that new code is consistent with existing patterns.
- Automated code scanning: ISE makes use of several code scanners to spot potential flaws. Scanning results are manually verified to eliminate false positives.
- Using regtest and Raven's test framework for dynamic testing, which includes:
    - Setting up a multi-node network to simulate transactions between two parties. Additionally, other nodes were created to simulate an external party that may execute malicious actions to undermine a P2SH transaction.

- Using the multi-node network, ISE confirmed that the entire flow of a P2SH transaction functions as expected.
- Testing the assumptions of the P2SH function, which includes:
  - Fuzzing all inputs to test input validation.
  - Using unauthorized cryptographic keys in signing requests to validate authentication controls.
  - Passing malformed scripts with unexpected values.
  - Testing variable type validation.
  - Testing for the possibility of denial-of-service attacks.
  - Testing management of private keys to ensure that the new P2SH functionality does not disclose key values to unauthorized nodes.

Access

ISE performed the assessment with access to the following resources:

- Source code for pull request located at https://github.com/RavenProject/Ravencoin/pull/873
- Opensource documentation of Ravencoin found on https://ravencoin.org/

Timeline

This section includes a summary of the history of ISE's assessments of P2SH.

Revision 1: JUNE 2021 – Initial Assessment

Components tested:

- Source code changes, GitHub pull request #873 ("Add P2SH support")

The following issues were discovered:

- ISE-RAVEN-P2SH-R1-01
- ISE-RAVEN-P2SH-R1-02
- ISE-RAVEN-P2SH-R1-03
- ISE-RAVEN-P2SH-R1-04

Severity Ratings

The severity ratings given in this report include critical, high, medium, low, informational, and unknown, with critical indicating the most severe, low the least, and unknown expressing that insufficient information was available to make a proper assessment. In determining severity, ISE takes into consideration public vulnerability ranking systems; however, in practice, the severity of vulnerabilities varies widely with actual deployment, configuration, and implementation of systems, as well as the value of assets protected and the perceived and anticipated threats to those assets. Thus, the severity ratings chosen here are custom to the system or infrastructure evaluated, and not copied from these sources verbatim.

The two metrics that most affect severity are exposure and impact of a successful attack. Exposure is a combination of elements including how accessible a vulnerable system is and the ease in which an attack can be performed. Impact is determined by factors such as asset value and damage to those assets. The following chart illustrates how severity is assigned:

CRITICAL	These are issues that are either readily exploitable or of substantial exposure paired with significant damage should an attack be successful. In some cases, the risk of discovery may be lower, but the potential damage warrants immediate attention.
HIGH	These issues affect assets heavily but may be less exploitable or require additional attack steps to exploit successfully.
MEDIUM	Medium severity issues present a notable risk to the system but may be limited to a subset of assets or require the attacker to have increased access or privileges to exploit the issue.
LOW	Low severity issues do not pose an immediate threat to the most valuable assets but may affect those assets under certain circumstances. Alternatively, these issues may readily affect low-sensitivity assets.
UNKNOWN	Issues of unknown severity typically could not be fully assessed due to missing resources or information. They are a security concern that require additional investigation to either assign a severity or resolve the issue.
INFO	Informational issues are unlikely to be a threat to the system but provide important information that stakeholders should be aware of, especially if they could be affected by future changes to the system. Informational issues may also signify hardening steps that can be used enhance the system's security.

Statuses

The following are descriptions of the various statuses with which ISE marks reported issues. Unresolved issues are unmarked, while other statuses reflect potential changes in a reported issue throughout the remediation process.

RESOLVED Resolved issues have been remediated. Occasionally, the implemented mitigations may no longer be effective, or ISE identifies additional instances of the issue. In these cases, the issue may become unresolved again.

PARTIAL Partially resolved issues have been mitigated to an extent, but not fully resolved. The meaning may depend on the context of the issue. For example, an issue with several different instances may be partially resolved if only a portion of the instances are resolved.

DEFERRED Deferred issues are unresolved; however, the client acknowledges the issue and a remediation plan is in place. This status is used to reflect the client's intention to fix the issue in the near future.

CLOSED Closed issues are unresolved or partially resolved, but the client is aware of their impact and accepts them as a risk. ISE's policy is to only close issues that do not have a direct impact to the security of a system and their remediation costs outweigh the security benefits.

Assessment Results

ISE discovered the following vulnerabilities within P2SH and its supporting environment. ISE provides recommendations for addressing identified issues in the form of resolutions and mitigations. Resolutions fully remediate the issue, while mitigations reduce the risk of, but may not completely remediate, the issue.

IMPORTANT NOTE: ISE's goal is to discover as many issues as possible within the boundaries of an assessment's budget and scope. However, undiscovered issues or additional instances of reported issues may always exist within a system. Raven should work with ISE to implement remediations for issues in this report and conduct ongoing assessments to address code and infrastructure changes.

Source Code

Missing Bounds Check on Vector Index

ISE-RAVEN-P2SH-R1-01

INFO

Attack Requirements	Insecure usage of SearchForRVN() function by developer error.
Affected Assets	All.
Impact	Potential undefined behavior with security implications.

On line 628 of `src/script/script.cpp`, there is a variable called `startingValue`, used as an index for the `script` vector. This index is not bounds checked before use. Referencing a vector out of bounds in C++ could result in undefined behavior with security implications. Although the function `SearchForRVN()` is currently only called using a starting value that is within bounds, future uses of the function cannot assume that `startingValue` will be bounds checked before use.

To avoid a potential buffer over-read, it is recommended to ensure that the supplied index be validated before using it to access vector elements. A suggestion to implement bounds checking was previously made on the GitHub pull request for P2SH.[2] However, it has not yet been implemented. The following code snippet highlights the locations in the function `SearchForRVN()` where `startingValue` is used.

```
int SearchForRVN(const CScript& script, const int startingValue) {

    // Initialize the start value
    int index = -1;

    // Search for RVN at the two places in the script it can be depending on the size of
the script
    if (script[startingValue] == RVN_R) { // Check to see if RVN starts at the starting
value ( this->size() < 105)
        if (script[startingValue + 1] == RVN_V)
            if (script[startingValue + 2] == RVN_N)
                index = startingValue + 3;
    } else {
        if (script[startingValue + 1] == RVN_R) // Check to see if RVN starts at starting
value + 1 ( this->size() >= 105)
```

[2] https://github.com/RavenProject/Ravencoin/pull/873/files?file-filters%5B%5D=.cpp&file-filters%5B%5D=.h&file-filters%5B%5D=.py#diff-6da39bed8a76fcaae51f021f0cdd906748aa99fdbf2d41981947f6a1b18fbe7bR631

```
                if (script[startingValue + 2] == RVN_V)
                    if (script[startingValue + 3] == RVN_N)
                        index = startingValue + 4;
        }

        return index;

    }
```

Figure 1. Function SearchForRvn() is missing bounds checking on startingValue.

Resolution: Bounds Check Index

Raven should add logic which ensures that the integer value of `startingValue` is within the range of the vector's size before it is used to reference vector values. If `startingValue` is not within bounds, then the function should not attempt to access `script`, instead returning a value indicating failure.

Resolution Status

This issue is unresolved.

Use of Uninitialized Variables

ISE-RAVEN-P2SH-R1-02

INFO

Attack Requirements	Accidental usage of uninitialized variables by developer error.
Affected Assets	All.
Impact	Potential undefined behavior with security implications.

ISE found several instances in P2SH's source code which use uninitialized variables. There also appears to be an inconsistency in initialization, as these variables (`nType` and `nScriptType`) are initialized to zero in some areas but not in others.

Leaving variables uninitialized in C++ is not recommended because uninitialized variables can be initialized with random values at runtime. This can lead to undefined behavior under certain conditions.[3] The names and locations of these variables are listed in Table 1.

File	Line Numbers	Variable Name	Type	Status
src/consensus/tx_verify.cpp	837	nType	Integer	Unresolved
src/consensus/tx_verify.cpp	838	nScriptType	Integer	Unresolved
src/validation.cpp	1885, 2224, 2718	nScriptType	Integer	Unresolved
src/txmempool.cpp	455, 499	nScrptType	Integer	Unresolved
src/wallet/wallet.cpp	2426	nType	Integer	Unresolved
src/wallet/wallet.cpp	2427	nScriptType	Integer	Unresolved
src/assets/assets.cpp	4455	nType	Integer	Unresolved

Table 1. Locations of uninitialized variables.

[3] https://en.cppreference.com/w/cpp/language/ub

For example, the Enum `TX_NONSTANDARD` indicates a non-standard transaction type. In all cases, both variables (`nType` and `nScriptType`) are passed by reference to `IsAssetScript()`. This function performs a series of checks on the `CScript` object to determine its transaction script type. If the checks succeed in determining the transaction type, the variables are assigned an Enum value corresponding to a transaction type, and the function returns `true`. If all checks fail, then the function returns `false` and one or both variables remain unassigned. In this case, the value contained in the variable is the one it held prior to entering the function, which may be the contents of uninitialized memory from the stack.

Resolution: Initialize Uninitialized Variables

ISE recommends that Raven set all uninitialized variables to appropriate default values. In the case of `nType` and `nScriptType`, the initial value used in other parts of the code is zero. Implementing this change will maintain consistency throughout the codebase and prevent undefined behavior.

Resolution Status

This issue is unresolved.

Variable Naming Inconsistencies

ISE-RAVEN-P2SH-R1-04

INFO

Attack Requirements	Misunderstanding and/or misuse of variables by developer error.
Affected Assets	All.
Impact	Complicates code management.

ISE found several instances of variable naming inconsistencies in the `/src/script.script.cpp` file. For example, the variable, `fIsOwner`, is used interchangeably with the variable, `isOwner` in arguments to the function `IsAssetScript()`, as highlighted in Figure 2 and Figure 3.

Using inconsistent naming could cause developers to incorrectly assume that `isOwner` and `fIsOwner` are processed differently by the `IsAssestScript()` function. Another consequence is that management of the codebase is further complicated, since those making edits need to keep track of two variable names instead of one.

```
bool CScript::IsAssetScript() const
{
    int nType = 0;
    int nScriptType = 0;
    bool isOwner = false;
    int start = 0;
    return IsAssetScript(nType, nScriptType, isOwner, start);
}

bool CScript::IsAssetScript(int& nType, bool& fIsOwner) const
{
    int start = 0;
    int nScriptType = 0;
    return IsAssetScript(nType, nScriptType, fIsOwner, start);
}

bool CScript::IsAssetScript(int& nType, int& nScriptType, bool& isOwner) const
{
    int start = 0;
```

```
    return IsAssetScript(nType, nScriptType, isOwner, start);
}

bool CScript::IsAssetScript(int& nType, int& nScriptType, bool& fIsOwner, int&
nStartingIndex) const
{
```

Figure 2. Code lines 231-254. The variable isOwner is used interchangeably with fIsOwner.

```
    bool isOwner = false;
    IsAssetScript(nType, nScriptType, isOwner);
```

Figure 3. Code lines 307 and 308. The variable isOwner is used interchangebly with fIsOwner.

Resolution: Use Consistent Naming Convention

ISE recommends that Raven use a consistent naming convention for function parameter names across all overloaded instances of the function `IsAssetScript()`. This involves updating `isOwner` and `fIsOwner` to the same name. Doing so will improve Raven's ability to manage P2SH's codebase, as well as reduce confusion for developers using these variables in function arguments.

Resolution Status

This issue is unresolved.

Missing Validation of scriptType

ISE-RAVEN-P2SH-R1-05

INFO

Attack Requirements	Access to P2SH.
Affected Assets	All.
Impact	May assign a destination address on an invalid scriptType value.

ISE identified a conditional statement where the variable `scriptType` is not checked against an expected Enum value.

In Figure 4, the highlighted `if` statement checks whether `scriptType` is equal to `TX_SCRIPTHASH`. If it is not equal, then the code automatically branches into the subsequent `else` statement. This means that `scriptType` could have a value which does not map onto a valid transaction type, and the code branch would still be evaluated. This could result in unexpected behavior that is difficult to debug. This issue was found in the file `src/script/standard.cpp`, lines 250-254.

```
    } else if (whichType == TX_NEW_ASSET || whichType == TX_REISSUE_ASSET || whichType ==
TX_TRANSFER_ASSET) {
        if (scriptType == TX_SCRIPTHASH) {
            addressRet = CScriptID(uint160(vSolutions[0]));
        } else {
            addressRet = CKeyID(uint160(vSolutions[0]));
        }
        return true;
    } else if (whichType == TX_RESTRICTED_ASSET_DATA) {
        if (vSolutions.size()) {
            addressRet = CKeyID(uint160(vSolutions[0]));
            return true;
        }
```

```
        }
```

Figure 4. scriptType is not validated beyond TX_SCRIPTHASH.

Resolution: Check scriptType Against Expected Values

ISE recommends that Raven modify their code so that the value of `scriptType` is checked against expected values before continuing to branch. This can be done by replacing `else` with an `else if` or `switch` statement. Additionally, the statement `return true;` needs to be placed within the `if` and `else if` blocks so that the function only succeeds if `scriptType` matches an expected Enum value.

Resolution Status

This issue is unresolved.

Additional Recommendations

ISE recommends the following to further improve Raven's security posture:

- Recurring reassessments to further enhance P2SH's security and verify mitigations to previously discovered issues
- In-depth source code analysis
- Review of mobile applications
- Review of third-party application integrations
- Review of development workflows
- Review of third-party libraries and frameworks

About ISE

ISE is an independent security firm headquartered in Baltimore, Maryland. We are dedicated to providing clients proven scientific strategies for defense. On every assessment our team of analysts and developers use adversary-centric approaches to protect digital assets, harden existing technologies, secure infrastructures, and work with development teams to improve our clients' overall security.

Assessing the client through the eyes of potential attackers allows us to understand possible threats and how to protect against those attacks. Our security analysts are experienced in information technology and product development which allows them to understand the security problems the client faces at every level. In addition, we conduct independent security research which allows us to stay at the forefront of the ever-changing world of information security.

Attacks on information systems cannot be stopped, however with robust security services provided by an experienced team, the effects of these attacks can often be mitigated or even prevented. We appreciate the confidence placed in us as a trusted security advisor. Please don't hesitate to get in touch for additional assistance with your security needs.

Independent Security Evaluators, LLC

4901 Springarden Drive
Suite 200
Baltimore, MD 21209
(443) 270-2296

contact@ise.io

https://www.ise.io/