

## Stacks Wallet 2.0 Security Assessment

### Blockstack

November 17, 2020 – Version 1.0

**Prepared for**

Kyran Burraston  
Mark Hendrickson

**Prepared by**

Shawn Fitzgerald  
Ava Howell

Feedback on this project?

<https://my.nccgroup.com/feedback/fd66a41b-2ce4-4514-a5e1-b546bca1876c>



## Synopsis

During the fall of 2020, Blockstack engaged the NCC Group Cryptography Services team to conduct a security assessment of the Stacks Wallet application. This application allows users to generate signing keys via Ledger or local storage and initiate transactions on the chain. Additionally users can view funds and related data. The review was delivered by 2 consultants over 7 person-days.

## Scope

NCC Group's evaluation included the following Stacks Wallet 2.0 repository:

- <https://github.com/blockstack/stacks-wallet/tree/release/stacking>

This was used to build the electron application for testing.

## Limitations

No significant limitations were encountered during testing and good coverage was achieved. Furthermore, the Blockstack team was attentive in providing support throughout the project.

## Key Findings

The assessment uncovered a number of medium and low security flaws. The most notable findings were:

- The Electron application configuration was missing a number of framework settings that are important for security.
- The Stacks Wallet contains external dependencies that were out of date and could lead to compromise of the application.
- Storage of the mnemonic lacked integrity protection, which could lead to unauthorized and undetected modification.

## Strategic Recommendations

**Electron application defense in depth** The electron development framework has historically had security issues and therefore it is important that care taken in the development and configuration of applications using the electron framework. This includes ensuring that all dependencies are up to date and do not contain known vulnerabilities. The application should follow common security configuration best practices, including setting web application protections such as correct output encoding and CORS policies. These should be periodically reviewed to ensure that the applications follows the latest recommendations.

## Target Metadata

|             |                   |
|-------------|-------------------|
| Name        | Stacks Wallet 2.0 |
| Type        | Desktop Client    |
| Platforms   | Electron          |
| Environment | Testing           |



## Engagement Data

|                 |                                     |
|-----------------|-------------------------------------|
| Type            | Cryptography and application review |
| Method          | Code-assisted                       |
| Dates           | 2020-11-09 to 2020-11-14            |
| Consultants     | 2                                   |
| Level of Effort | 7 person-days                       |

## Targets

|   |  |
|---|--|
| <a href="https://github.com/blockstack/stacks-wallet/tree/release/stacking">https://github.com/blockstack/stacks-wallet/tree/release/stacking</a> | 9d2a2835777a649a14bdcd1bc36884f1bcc128dd |
|---|--|






## Finding Breakdown

|                      |   |
|----------------------|---|
| Critical issues      | 0   |
| High issues          | 0   |
| Medium issues        | 2  |
| Low issues           | 5  |
| Informational issues | 0   |
| <b>Total issues</b>  | <b>7</b>  |

## Category Breakdown

|                 |   |
|-----------------|---|
| Configuration   | 3  |
| Cryptography    | 2  |
| Data Validation | 1  |
| Patching        | 1  |

### Key

|  |  |  |   |   |
|--|--|--|---|---|
|  Critical |  High |  Medium |  Low |  Informational |
|--|--|--|---|---|

# Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see [Appendix A on page 15](#).

| Title   | ID  | Risk   |
|---|-----|--------|
| Missing Electron Security Configuration                       | 005 | Medium |
| Out-of-Date Dependencies With Known Vulnerabilities           | 006 | Medium |
| Non-Constant-Time Elliptic Curve Operations                   | 001 | Low    |
| Overly Permissive Cross-Origin Resource Sharing (CORS) Policy | 002 | Low    |
| Inconsistent Output Encoding                                  | 003 | Low    |
| Lack of Integrity Verification in Mnemonic Encryption         | 004 | Low    |
| Recommendations on Argon2 KDF Parameters                      | 007 | Low    |

**Finding** **Missing Electron Security Configuration**

**Risk** **Medium** Impact: Medium, Exploitability: Medium

**Identifier** NCC-BLSK003-005

**Category** Configuration

**Impact** Overly permissive configuration of Electron may enable exploit chains that would have otherwise been blocked.

**Description** The Blockstack wallet uses Electron, a framework for developing cross-platform native applications using web technology. Electron code presents unique security challenges compared to traditional web code. Electron exposes a number of APIs which can break the traditional “sandboxed web” security model. Application vulnerabilities such as cross-site scripting can escalate to full remote code execution, depending on the configuration of Electron.

As part of the security review, NCC Group assessed the configuration of Electron in the Blockstack wallet. While critical security issues such as the use of `nodeIntegration` and disabling `webSecurity` were not present in the production configuration, a number of issues were noted:

- **Navigation is not disabled.** It was discovered that the configuration of Electron allowed the wallet to navigate to arbitrary origins. This is a violation of the principle of least privilege: navigation is a privilege that is not required by the Stacks Wallet, so it should not be enabled.
- **Sandboxing is not enabled.** It was discovered that the `sandbox` configuration option was not set in the production configuration. Ideally, `sandbox` should be enabled.
- **Remote module in use:** The electron `remote` module is considered deprecated for use in the renderer process, as its APIs are dangerous in the event of a compromised renderer. The remote module was found to be in use in `stacks-wallet`.
- **No permission request handler registered:** The Stacks Wallet doesn't register a permission request handler using `setPermissionRequestHandler`. Without registering a permission request handler, the application will silently approve all permission requests (such as requests to access the microphone and video).

**Recommendation** Since the Stacks Wallet is a single-page React application, it should be possible and safe to completely disable navigation using the `webContents.on('will-navigate')` event, and calling `event.preventDefault()`. Note that in addition to implementing an event handler that blocks navigation, it is also necessary to pass `"disableBlockFeatures": "Auxclick"` to the `webPreferences` of the main Electron window, to prevent a middle-click bypass of the navigation limitation.

The `sandbox` option should be used if it is determined that it does not break the Stacks Wallet application. Ideally, the application should be refactored such that it can run in an Electron window with `sandbox: true`.

Usage of the `remote` module in the renderer process should be removed and refactored out into the main process.

A permission request handler should be implemented, which returns `false` by default, applying the principle of least privilege.

Going forward, the `electronnegativity` tool can be used to audit the configuration of Electron in the Stacks Wallet.

**Finding** **Out-of-Date Dependencies With Known Vulnerabilities**

**Risk** **Medium** Impact: Medium, Exploitability: Undetermined

**Identifier** NCC-BLSK003-006

**Category** Patching

**Location**

- Stacks Wallet NPM dependencies

**Impact** Outdated components with publicly known vulnerabilities could lead to compromise of the Stacks Wallet application or assist in an exploit chain.

**Description** The Stacks Wallet relies on a number of external dependencies, which themselves have dependencies. This means that the total dependency tree is quite large, despite a relative few top-level dependencies of the Stacks Wallet application. Often, dependencies are updated to remove important vulnerabilities. As part of the assessment, NCC Group ran the `yarn audit` utility to check the status of every dependency in the tree. Two outdated dependencies were noted:

- node-fetch (Low severity, Denial of Service)
  - <https://www.npmjs.com/advisories/1556>
- node-forge (HIGH severity, prototype pollution)
  - <https://www.npmjs.com/advisories/1561>

These known vulnerabilities may lead to a disruption in the Stacks Wallet or be used in a more severe exploit chain.

**Recommendation** Update all dependencies using `yarn upgrade`, and run `yarn audit` again to verify that the known vulnerabilities have been removed.

**Finding** Non-Constant-Time Elliptic Curve OperationsRisk **Low** Impact: Low, Exploitability: Low

Identifier NCC-BLSK003-001

Category Cryptography

Location

- `stacks-transactions`.
- Underlying elliptic curve operations in the `elliptic` package.

Impact When using a software wallet, elliptic curve operations may leak information about the private keys associated with a Blockstack wallet through timing side-channels.

Description The Stacks Wallet implementation allows two ways for users to manage their wallet private keys and sign transactions. First, they can use the Ledger hardware wallet, which integrates with the Stacks Wallet application. Second, they can choose to use the Stacks application itself to manage their keys and sign transactions. In the latter case, the `stacks-transactions` library is used to sign transactions. The underlying signing function is as follows:

```
// stacks-transactions/src/keys.ts L147
import { ec as EC } from 'elliptic';
// ...snip...
export function signWithKey(privateKey: StacksPrivateKey, input: string):
  → MessageSignature {
  const ec = new EC('secp256k1');
  const key = ec.keyFromPrivate(privateKey.data.toString('hex').slice(0, 64),
    → 'hex');
  const signature = key.sign(input, 'hex', { canonical: true });
  const coordinateValueBytes = 32;
  const r = leftPadHexToLength(signature.r.toString('hex'), coordinateValueBytes
    → * 2);
  const s = leftPadHexToLength(signature.s.toString('hex'), coordinateValueBytes
    → * 2);
  if (signature.recoveryParam === undefined || signature.recoveryParam === null)
    → {
    throw new Error('"signature.recoveryParam" is not set');
  }
  const recoveryParam = intToHexString(signature.recoveryParam, 1);
  const recoverableSignatureString = recoveryParam + r + s;
  const recoverableSignature =
    → createMessageSignature(recoverableSignatureString);
  return recoverableSignature;
}
```

Note that for this software signing, the Javascript `elliptic` library is used for ECDSA over curve `secp256k1`. A common flaw in ECDSA implementations is *side-channel leaks*. A focus of cryptographic literature and study in recent years, side channel leaks occur when some computation over secret data leaks into the world through physical channels which are not defined by the abstract model of the software - one such example is timing leaks. Timing attacks have caused severe breaks in cutting-edge cryptographic implementations<sup>1,2</sup>. In general, a requirement for cryptographic implementations to avoid being broken by timing leaks is that their computations which involve secret data must not have data-dependent execution time, in other words they must be constant-time.

<sup>1</sup><https://cryptoservices.github.io/cryptography/attacks/2019/01/17/cat.html>

<sup>2</sup><https://www.nccgroup.trust/us/our-research/technical-advisory-return-of-the-hidden-number-problem/>

The `elliptic` library is *not* constant-time, and there are several immediately apparent timing leakages: first, for `secp256k1`, the implementation of scalar multiplication is in JavaScript, which tends to have innate issues with constant-time cryptography. Second, private scalars are represented using `bn.js`, which is inherently non-constant-time as it uses non-modular big integers, so operations with private scalars such as the private key itself or the per-signature nonce `k` will leak information about their values. Lastly, the implementation of elliptic curve scalar multiplication itself in `elliptic.js` for `secp256k1` uses the wNAF method, which is known to be leak information about the scalar through side channels.

The practical impact of this finding is limited, but it may be more impactful if the user who is using the Stacks wallet is doing so on a machine that has multiple trust domains, such as a shared server.

**Recommendation**

A simple solution is to limit the use of software wallets, or emphasize the security tradeoffs thereof. The timing leak can be removed from the software wallet by using an implementation of ECDSA which is known to be constant-time, such as the well-tested `libsecp256k1` which is used in the Bitcoin core project,<sup>3</sup> which can be compiled to WebAssembly to be run in the Stacks wallet.

<sup>3</sup><https://github.com/bitcoin-core/secp256k1>



**Finding** **Overly Permissive Cross-Origin Resource Sharing (CORS) Policy**

**Risk** **Low** Impact: Low, Exploitability: None

**Identifier** NCC-BLSK003-002

**Category** Configuration

**Location** `access-control-allow-origin` header in interactions with `stacks-node-api.krypton.blockstack.org`

**Impact** An attacking site could abuse the application's weak CORS policy to steal user data or perform dangerous actions in the context of the user's account.

**Description** [Cross-Origin Resource Sharing](#) (CORS) is a browser standard which allows for the resources or functionality of a web application to be accessed by other web pages originating from a different domain. Before CORS, browsers had a number of strict limitations on the ability of different web applications to communicate with each other. For example, an application could send a request to an application on a different domain, but could not read the response, making "web API" functionality impossible.

CORS makes safe cross-domain requests possible by allowing applications to opt-in to cross-domain communication. If an application wants to allow this type of communication, it can use the following CORS headers:

- `Access-Control-Allow-Origin`: a domain (or wildcard) which the application wants to allow communication from

By returning these headers, the responding application explicitly states that cross-domain communication is acceptable. The user's browser checks that the returned values are valid; if so, the browser permits the requesting application to send a cross-domain request and to read the response.

Caution must be taken when defining these header values. Normally, the [Same-Origin Policy](#) prevents most cross-domain requests. By returning CORS headers, that protection is severely weakened. An attacking site could abuse this weakness to steal user data or perform dangerous actions.

The server returns a permissive CORS policy, this could allow any site to make cross origin requests with the Electron application backend:

```
access-control-allow-origin: *
access-control-allow-headers: origin, content-type
access-control-allow-methods: POST, GET, OPTIONS
```

It should be noted that within the context of the Stacks Wallet there is little chance of exploitability because there are not sensitive values such as cookies or API tokens that could be returned from cross-origin requests. Nevertheless, good defense in depth would be to limit interactions from only whitelisted and trusted sources.

**Recommendation** Access-Control headers should be limited to the minimal set of permissions necessary to perform cross-domain functionality. First, Access-Control headers should only be returned for specific, necessary resources or routes, rather than applied globally. For each route, explicitly set the relevant Access-Control headers to allow only the specific methods, headers, or credentials which are necessary.

For the Access-Control-Allow-Origin header, domains should be specified using a whitelist. Only domains which are absolutely necessary for cross domain communication should be included.<sup>4</sup>

In the case of the Blockstack wallet, this will likely only be `stacks-node-api.krypton.blockstack.org` and related trusted sites.

---

<sup>4</sup><https://www.electronjs.org/docs/tutorial/security#6-define-a-content-security-policy>

**Finding** **Inconsistent Output Encoding**

**Risk** **Low** Impact: None, Exploitability: None

**Identifier** NCC-BLSK003-003

**Category** Data Validation

**Location** GET and POST requests to `stacks-node-api.krypton.blockstack.org`

**Impact** A lack of output encoding increases the risk of an exploitable cross-site scripting (XSS) vulnerability.

**Description** Cross-site scripting vulnerabilities allow a malicious agent to inject attacker controlled script into a user session. This can allow the attacker to carry out actions in the context of the user or steal passwords and other sensitive data. In the context of Electron applications XSS vulnerabilities are even more serious as they can execute low level Node.js API functions resulting in remote code execution vulnerabilities (RCE) in the context of the victim user machine.<sup>5</sup> Although no exploitable XSS vulnerabilities had been found during the assessment, there was a lack of output encoding in returned JSON objects. The following example interactions demonstrate this:

```
POST /v2/contracts/call-read/ST000000000000000000000000000000002AMW42H/pox/get-stacker-
→ infofagaf%3ca%3eerl fvxurij7 HTTP/1.1
→
Host: stacks-node-api.krypton.blockstack.org
...
Accept-Language: en-US

////////////////////////////////////

HTTP/1.1 200 OK
Date: Wed, 11 Nov 2020 06:58:09 GMT
Content-Type: application/json
...
Content-Length: 94

{"okay": false, "cause": "Unchecked(UndefinedFunction(\"get-stacker-infofagaf
→ <a>erl fvxurij7\"))"}
```

<sup>5</sup><https://medium.com/dealeron-dev/electron-and-xss-e9ecef1c7325>

```

GET /extended/v1/address/ST2B191A71QWSH39V2R28W7WMV8GCTNFTQAM5TEMYv1qph%3cimg%20s
→ rc%3da%20onerror%3dalert(1)%3eotlmd/transactions HTTP/1.1
Host: stacks-node-api.krypton.blockstack.org
...
Accept-Language: en-US

////////////////////////////////////

HTTP/1.1 400 Bad Request
Date: Wed, 11 Nov 2020 06:53:19 GMT
Content-Type: application/json; charset=utf-8
...
Strict-Transport-Security: max-age=15724800; includeSubDomains

{"error": "invalid STX address \"ST2B191A71QWSH39V2R28W7WMV8GCTNFTQAM5TEMYv1qph
→ <img src=a onerror=alert(1)>otlmd\""}

```

The above example, are not currently exploitable because the content type has been set to `application/json` and other options are not allowed. That being said, there is still a risk of type-0 or DOM based XSS vulnerabilities.

**Recommendation**

Except for alphanumeric characters, all values should be hex encoded using the `\uXXXX` Unicode escaping format. In addition to this a good defense in depth recommendation is to perform input validation as well as ensuring that all user-controlled parameters are JavaScript encoded when passed within an object back to the user.

**Finding** Lack of Integrity Verification in Mnemonic EncryptionRisk **Low** Impact: High, Exploitability: Medium

Identifier NCC-BLSK003-004

Category Cryptography

Location `/app/crypto/key-encryption.ts`

Impact A malicious entity can modify the encrypted mnemonic value without detection.

**Description** The `encryptMnemonic` and `decryptMnemonic` functions encrypt the user mnemonic and store it on disk using AES-CBC. This cryptographic mode of operation does not include integrity protection, and therefore the application cannot determine if this value has been modified on disk. This could allow an attacker to make modifications to the mnemonic such as reordering attacks or to fully swap mnemonics in an undetected manner. Although it is not likely that this could result in attacker controlled transactions, it could result in bogus transactions, since different signing keys would be derived.

**Recommendation** Transition to a cryptographic mode of operation that supports encryption and integrity in one operation. As an example, AES-GCM is an authentication encryption algorithm that is well vetted and widely supported.

**Finding** Recommendations on Argon2 KDF Parameters

**Risk** Low Impact: Low, Exploitability: Low

**Identifier** NCC-BLSK003-007

**Category** Configuration

**Location** `stacks-wallet/app/crypto/key-generation.ts`

**Impact** Not fully utilizing the Argon2 configuration parameters would allow for for efficient brute force attacks on the user's stored mnemonic.

**Description** The Argon2 key derivation function is used for deriving a wrapping key from the user supplied password. Argon2 can be used for both password storage and key derivation and is highly configurable, depending on the environment. The Stacks Wallet application uses the `argon2-browser` library that appears to be configured for password storage, which is not ideal for use case of non-interactive key derivation. This can result in less than optimal configuration parameters.

**Recommendation** Although parameters are not fixed as they depend on the environment, required execution time and target architecture; there are number of common parameter choices. The `libsodium` documentation,<sup>6</sup> can provide a good starting point from which execution times can be derived. The current recommendation is that highly sensitive data and non-interactive operations, key derivation will take about 3.5 seconds on a 2.8 Ghz Core i7 CPU and requires 1024 MiB of dedicated RAM.

<sup>6</sup>[https://libsodium.gitbook.io/doc/password\\_hashing/default\\_phf](https://libsodium.gitbook.io/doc/password_hashing/default_phf)

The following sections describe the risk rating and category assigned to issues NCC Group identified.

## Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

## Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

- Critical** Implies an immediate, easily accessible threat of total compromise.
- High** Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
- Medium** A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
- Low** Implies a relatively minor threat to the application.
- Informational** No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

## Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

- High** Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
- Medium** Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
- Low** Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

## Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

- High** Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.
- Medium** Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
- Low** Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

---

## Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

- Access Controls** Related to authorization of users, and assessment of rights.
- Auditing and Logging** Related to auditing of actions, or logging of problems.
- Authentication** Related to the identification of users.
- Configuration** Related to security configurations of servers, devices, or software.
- Cryptography** Related to mathematical protections for data.
- Data Exposure** Related to unintended exposure of sensitive information.
- Data Validation** Related to improper reliance on the structure or values of data.
- Denial of Service** Related to causing system failure.
- Error Reporting** Related to the reporting of error conditions in a secure fashion.
- Patching** Related to keeping software up to date.
- Session Management** Related to the identification of authenticated users.
- Timing** Related to race conditions, locking, or order of operations.



The team from NCC Group has the following primary members:

- Shawn Fitzgerald — Consultant  
[shawn.fitzgerald@nccgroup.com](mailto:shawn.fitzgerald@nccgroup.com)
- Ava Howell — Consultant  
[ava.howell@nccgroup.com](mailto:ava.howell@nccgroup.com)
- Javed Samuel — Practice Director, Cryptography Services  
[javed.samuel@nccgroup.com](mailto:javed.samuel@nccgroup.com)

The team from Blockstack has the following primary members:

- Kyran Burraston — Blockstack  
[kyran@blockstack.com](mailto:kyran@blockstack.com)
- Mark Hendrickson — Blockstack  
[mark@blockstack.com](mailto:mark@blockstack.com)