Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Audit-Report micro-btc-signer TS Library 01.2023

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“Create, sign & decode BTC transactions with minimum deps.”

From https://github.com/paulmillr/micro-btc-signer

This report details the scope, results, and conclusory summaries of a cryptography review and source code audit against the micro-btc-signer library.

The work was requested by Ryan Shea in December 2022 and initiated by Cure53 in January 2023, namely through CW02 and CW04. A total of eleven days were allocated to reach the coverage expected for this project. The testing conducted for this audit was divided into one distinct Work Package (WP) for execution efficiency, as follows:

- **WP1**: Cryptography review & code audit against micro-btc-signer library

Cure53 was granted access to the libraries and commits via GitHub, as well as any alternative means of access required to ensure a smooth review completion. For this purpose, the methodology chosen was white-box and a team comprising two skillmatched senior testers was assigned to the project’s preparation, execution, and finalization.

All preparatory actions were completed in January 2023, namely in CW01, to ensure the review could proceed without hindrance or delay. Communications were facilitated via a dedicated, shared Signal channel deployed between Ryan Shea, Paul Miller and Cure53, thereby creating an optimal collaborative working environment. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions.

In light of this, communications proceeded smoothly on the whole. The scope was well-prepared and transparent, no noteworthy roadblocks were encountered throughout testing, and cross-team queries remained minimal as a result.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was offered and subsequently conducted via the aforementioned Signal channel. Concerning the findings specifically, the Cure53 team achieved widespread coverage over the WP1 and WP2 scope items, detecting a total of four. One of the findings was categorized as a security vulnerability, whilst the remaining three were deemed general weaknesses with lower exploitation potential.

Even though the total yield of findings is relatively minimal, the overall impression gained of the micro-btc-signer TS library is rather negative, primarily owing to the fact that three closely-related programming paradigm issues were visible in the micro-btc-signer's code. These were considered inappropriate for the library's designated high-assurance deployment contexts and have been extrapolated in detail via the following *Cryptography Review* section.

Whilst this assessment was unable to detect any directly-exploitable issues within the micro-btc-signer library, Cure53 can only conclude that the library requires a rewrite in order to fully leverage TypeScript's enhanced type safety features. The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. This section will be followed by a chapter that details the performed cryptography review, which serves to provide in-depth analysis of the provided scope and findings, as well as stipulate the key goals of this audit and threat/attacker model.

Subsequently, the in-scope infrastructure coverage and advanced approaches instigated are detailed, in lieu of significant findings detected. Next, Cure53 highlights potential focus areas for future work and the considerations that should be adhered to for any micro-btc-signer TS library improvements moving forward.

The report will then list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the micro-btc-signer library, giving high-level hardening advice where applicable.

Scope

- **Cryptography reviews & code audit against Paul Miller's micro-btc-signer TS library**
    - **WP1**: Cryptography review & code audit against micro-btc-signer library
        - **Library in scope:**
            - https://github.com/paulmillr/micro-btc-signer
        - **Commit in scope:**
            - https://github.com/paulmillr/scure-btc-signer/commit/397ed56cd98e1908b3345572a123b953057531e9
    - **Test-supporting material was shared with Cure53**
    - **All relevant sources were shared with Cure53**

Cryptography Review

This section documents the testing methodology applied during this cryptography review and code audit, shedding light on the advanced approaches initiated to evaluate the micro-btc-signer TS library. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, particularly considering the absence of findings exhibiting significant security vulnerabilities on the scope examined by Cure53 for this audit.

Scope & Findings

micro-btc-signer implements a set of standard functionalities relevant to Bitcoin applications, such as Bitcoin wallets. During this deep-dive investigation, Cure53 verified the correct implementation of payment and transaction encoding functionality. *micro-btc-signer*'s API encompassed classic Bitcoin functionality plus relatively recent extensions, such as SegWit optimizations and Taproot-based *multisig*. All of these functionalities were reviewed, with the confirmation made that they match the expected behavior.

Given the complete lack of issues regarding the validity of the implemented functionality in *micro-btc-signer* - as well as the fact that the library exists purely as a protocol and application agnostic API and is not tied to specific use-case scenarios - this audit primarily focussed on best practices for programming a Bitcoin library intended for deployment in the significantly-sensitive context of cryptocurrency transactions within web-browser runtime environments.

Here, one must consider that *micro-btc-signer* cannot control the method by which it is called from the application layer that adopts it, since it constitutes a utility library for Bitcoin transaction functionality. In light of this, ensuring that the library remains resilient to misuse should be considered a critical requirement. Unfortunately, despite the usage of TypeScript, the library's stated focus on auditability, and the lack of outright vulnerabilities in the library's code, this assessment's conclusory outcomes neither confirm nor suggest that *micro-btc-signer* is designed with abuse-resistance as a paramount criteria. On the contrary, the library accommodates potential misuse in order to allow lax type inputs into critical transaction functionality.

Whilst one can argue that *micro-btc-signer*'s minimal code target renders security auditing easier to achieve and security bugs less likely to emerge, and even though this audit was unable to identify any directly exploitable cryptographic vulnerabilities in the library, Cure53 nevertheless observed the presence of three closely-related programming paradigm issues. These are visible in *micro-btc-signer*'s code and were deemed inappropriate for the library's designated high-assurance deployment context, as follows:

- **Extreme permissiveness for high-level API functionality input types:** An excessive volume of both private functions and exported API functionality for *micro-btc-signer* permit users to input surprisingly varied types for highly-sensitive data. For example, MBS-01-001 documents a scenario whereby an input value's data type can cause Bitcoin transactions to shift denominations in multiple orders of magnitude. This is particularly relevant given the extreme lack of type safety that JavaScript is known for.
- **Acute type overloading facilitates code that is tangibly more succinct, but technically and logically more challenging to audit in comparison with longer code:** Certain areas of *micro-btc-signer*'s code serve to illustrate that more succinct code does not always equal more auditable code. Specific core functionalities utilize extreme type overloading and other hacks in order to reduce effective lines of code, though this renders the library increasingly challenging to audit, as demonstrated in ticket MBS-01-002. Particularly in the context of a cryptographic library intended for deployment in high-assurance situations, this coding paradigm may be deemed inappropriate.
- **Inappropriately lacking use of TypeScript's enhanced type safety functionality:** Despite the fact that *micro-btc-signer* is written in TypeScript, testing confirmed that the majority of TypeScript's enhanced type safety features are leveraged inappropriately across the library's code and do not meet their full functionality potential. This behavior incurs a number of issues, including those detailed in tickets MBS-01-003 and MBS-01-004.

Key Goals

During the *micro-btc-signer* evaluation, the key goals from an audit perspective were to:

- Ensure that all Bitcoin transaction functionality - as well as Bitcoin script encoding and decoding - were implemented correctly, in adherence with the specification, and produced expected output when used over the wire by a high-level application adopting *micro-btc-signer* as its core Bitcoin functionality library.
- Ensure *micro-btc-signer* was misuse resistant, with due consideration of the excessively-lax type discipline for data inputs and outputs exhibited by the web runtime environment and JavaScript.

Threat & Attacker Model

Micro-btc-signer constitutes an application-agnostic library. As a result, typical cryptocurrency applications attack vectors - such as device compromise, network compromise, impersonation, and similar - are not immediately relevant in this context. In light of this, any would-be attacker may be attracted to any potential compromise opportunity offered by this library for a couple of reasons, as follows:

- **Inattentive application developers:** Scenarios whereby application developers using *micro-btc-signer* in their apps overly rely on the library itself for input validation, either for data types or data structures.
- **Uninformed subsequent *micro-btc-signer* maintainers:** Scenarios whereby future *micro-btc-signer* maintainers are not necessarily comprehensively aware of the specific tricks and programming styles adopted to minimize the library's code size or otherwise render it more "compact", which is one of the library's stipulated design objectives.

As demonstrated in this audit report - and considering *micro-btc-signer* from the perspective of both of the attack scenarios described above - a number of actual vulnerabilities and miscellaneous issues are incurred, despite the library's lack of outright implementation errors.

Test Methodology

Micro-btc-signer was evaluated through adoptance of two independent methodologies:

- **Implementation correctness verification:** Each of *micro-btc-signer*'s top-level functionalities were verified to be correctly implemented, whilst the test suite was verified to produce expected output in line with standard Bitcoin implementations.
- **Manual source code review:** A line-by-line manual reading of virtually the entire source code was initiated in order to determine the integration of best practices and potential for either underlying security issues or unhandled edge-case scenarios.

As alluded to above, the former methodology yielded no noteworthy findings, though the latter review raised a number of potential issues from the perspective of the attacker model considered for this audit.

Future Work & Considerations

Micro-btc-signer purports code line minimization as one of its primary goals in order to render the library more portable, more auditable, and easier to review. However, *micro-btc-signer* makes the fundamental mistake of confusing *less lines of code with less complexity*; despite achieving a minimal code footprint, *micro-btc-signer* does not fulfill the expectancy of code complexity minimization. On the contrary, code complexity is elevated in order to minimize code footprint. This elemental design error increases both the difficulty of auditing the library, as well as its susceptibility to application layer misuse and potential erroneous behaviors in the hands of subsequent maintainers.

Given the aforementioned design fault, Cure53 advises rewriting the library in order to focus on reducing *complexity*, rather than simply removing lines of code and expecting that to automatically translate into a reduction in code and logical complexity.

Whilst many pertinent instances were observed throughout the codebase, the issue described in ticket MBS-01-002 acts as the perfect standalone demonstration of how *micro-btc-signer* eschews TypeScript’s advanced type safety features, proper input validation, and a lessening of code complexity for the purpose of simply reducing effective lines of code. Ultimately, the library must be rewritten to avoid these erroneous behaviors, particularly in respect of its highly sensitive deployment use case and extremely malleable target runtime environment.

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *MBS-01-001*) to facilitate any future follow-up correspondence.

MBS-01-001 Crypto: *addOutputAddress* handles denomination via input type *(High)*

Testing confirmed that the *addOutputAddress* functionality in the *micro-btc-signer* library handles the Bitcoin input amount in entirely different currency denominations, depending on the input data type passed to the function. Notably, if the amount is passed as a string, it is interpreted as a Bitcoin; if the amount is passed as a number or BigInt, it will be handled as a satoshi (i.e. 0.00000001 BTC).

This is problematic for three reasons: firstly, the *micro-btc-signer* will be leveraged in sensitive contexts such as cryptocurrency wallets; secondly, the significantly malleable nature of data types in JavaScript; and finally, the *micro-btc-signer* authors lack control over how the library's exposed top-level API functionality (*addOutputAddress* and alternative functions*)* will be utilized in third-party application layer code. Generally speaking, permitting the denomination of a currency to shift radically depending on the data-type input represents an insufficiently secure programming paradigm, given the library's security context. Take, for example, the following JavaScript code:

```
> typeof(1 + "")
'string'
```

The code offered above serves to demonstrate that the language environment for *micro-btc-signer* exposes third-party application developers to an atypical degree of risk pertaining to the introduction of type confusion bugs. Rather than being accommodated by the underlying cryptographic library, these type confusion bugs should always throw errors and abort. This is particularly valid since *micro-btc-signer* is written in TypeScript, which allows developers to benefit from stronger compile-time type safety; in actuality, TypeScript's integrated safety features are not exploited to their full potential in many code areas.

Affected file:
index.ts

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected code:
```
private normalizeOutput(
             o: TransactionOutputUpdate,
             cur ? : TransactionOutput,
             allowedFields ? : (keyof typeof PSBTOutput)[]
       ): TransactionOutput {
             let {
                    amount,
                    script
             } = o;
             if (typeof amount === 'string') amount = Decimal.decode(amount);
             if (typeof amount === 'number') amount = BigInt(amount);

[...]

addOutputAddress(address: string, amount: string | bigint, network = NETWORK):
number {
        return this.addOutput({
              script: OutScript.encode(Address(network).decode(address)),
              amount: typeof amount === 'string' ? Decimal.decode(amount) :
amount,
        });
```

To mitigate this issue, Cure53 advises rewriting all exposed, high-level *micro-btc-signer* API functionality to adopt strict type safety measures, and ensuring it invariably fails in the event that input types do not meet expected requirements. This analysis applies to *addOutputAddress* above all, though the same recommendation can be made to many other code snippets in the library that equally allow lax type handling, which is inappropriate given the library's sensitive deployment context and as documented elsewhere in this report.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

MBS-01-002 Crypto: Overloaded compare function for numeric values *(Info)*

Testing confirmed that the *micro-btc-signer* library compresses the comparison function for numeric values represented as strings, numbers, BigInts, bytes, or even booleans into a single function. Whilst this behavior was most likely implemented in order for the library to fulfill its primary stipulation of offering the most Bitcoin functionality with the least amount of code, the resulting code ultimately renders the library increasingly challenging to audit, despite the achieved compositional compactness.

Furthermore, adopting such a high level of type malleability into a low-level logical primitive as fundamental as integer comparison risks rendering the library vulnerable to bugs introduced by future developers, who may not completely understand the subtle tricks employed in the functionality that allows a vast array of types to be compared against one another simultaneously.

Affected file:
index.ts

Affected code:
```
type CmpType = string | number | bigint | boolean | Bytes | undefined;
export function cmp(a: CmpType, b: CmpType): number {
       if (isBytes(a) && isBytes(b)) {
              // -1 -> a<b, 0 -> a==b, 1 -> a>b
              const len = Math.min(a.length, b.length);
              for (let i = 0; i < len; i++)
                     if (a[i] != b[i]) return Math.sign(a[i] - b[i]);
              return Math.sign(a.length - b.length);
       } else if (isBytes(a) || isBytes(b)) throw new Error(`cmp: wrong values
a=${a} b=${b}`);
       if (
              (typeof a === 'bigint' && typeof b === 'number') ||
              (typeof a === 'number' && typeof b === 'bigint')
       ) {
              a = BigInt(a);
              b = BigInt(b);
       }
```

```
        if (a === undefined || b === undefined) throw new Error(`cmp: wrong
values a=${a} b=${b}`);
        // Default js comparasion
        return Number(a > b) - Number(a < b);
}
```

To mitigate this issue, Cure53 advises dividing comparison functions into separate explicit functionality for each of the data types that must be compared against one another. Furthermore, one can recommend blocking the exportation and exposure of all comparison functions to the third-party application layer, since one cannot fully determine whether this functionality is intended for employment by the application layer.

MBS-01-003 Crypto: Unsafe exported functionality *(Medium)*

Testing confirmed that some of *micro-btc-signer's* exported functionality, including *p2sh* and *p2wsh*, access internal properties for input arguments without first validating their structure. This could lead to applications passing incorrectly structured data, which could be incorrectly interpreted by underlying cryptographic operations.

Affected file:
index.ts

Affected code:
```
export const p2sh = (child: P2Ret, network = NETWORK): P2Ret => {
        const hash = hash160(child.script);
        const script = OutScript.encode({
                type: 'sh',
                hash
        });
        checkScript(script, child.script, child.witnessScript);
        const res: P2Ret = {
                type: 'sh',
                redeemScript: child.script,
                script: OutScript.encode({
                        type: 'sh',
                        hash
                }),
                address: Address(network).encode({
                        type: 'sh',
                        hash
                }),
        };
        if (child.witnessScript) res.witnessScript = child.witnessScript;
        return res;
};
```

```
[...]

export const p2wsh = (child: P2Ret, network = NETWORK): P2Ret => {
	const hash = sha256(child.script);
	const script = OutScript.encode({
		type: 'wsh',
		hash
	});
	checkScript(script, undefined, child.script);
	return {
		type: 'wsh',
		witnessScript: child.script,
		script: OutScript.encode({
			type: 'wsh',
			hash
		}),
		address: Address(network).encode({
			type: 'wsh',
			hash
		}),
	};
};
```

To mitigate this issue, Cure53 advises rewriting any exposed, high-level API for *micro-btc-signer* to ensure that the structure of input arguments is thoroughly validated before being passed into any internal library logic.

MBS-01-004 Crypto: Byte array cast to boolean in comparison *(Info)*

Testing confirmed that certain areas of the *micro-btc-signer* library exploited JavaScript type conversion hacks in order to obtain cryptographically sensitive validations. For example, a double negation in *isValidPubkey* is exploited in order to convert a byte array into a boolean value.

Affected file:
index.ts

Affected code:
```
function validatePubkey(pub: Bytes, type: PubT): Bytes {
	const len = pub.length;
	if (type === PubT.ecdsa) {
		if (len === 32) throw new Error('Expected non-Schnorr key');
	} else if (type === PubT.schnorr) {
		if (len !== 32) throw new Error('Expected 32-byte Schnorr key');
	} else {
		throw new Error('Unknown key type');
	}
```

```
        secp.Point.fromHex(pub); // does assertValidity
        return pub;
}

function isValidPubkey(pub: Bytes, type: PubT): boolean {
        try {
                return !!validatePubkey(pub, type);
        } catch (e) {
                return false;
        }
}
```

Whilst this behavior does not appear to incur any security weakness in isolation at present, Cure53 nevertheless strongly advises adopting more coherent practices when writing sensitive cryptographic functionality. Generally speaking, cryptographic APIs remain an inappropriate area for potential type conversion hacks. Alternatively, code should be compositionally clear, maintainable, and idiomatic throughout. Hence, this coding style should not be adopted throughout the rest of the *micro-btc-signer* library.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW02 and CW04 testing against the micro-btc-signer library by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a rather mixed impression. In context, micro-btc-signer represents a JavaScript/TypeScript library purported to offer transaction creation, signing, decoding, and processing for Bitcoin ecosystem applications in a highly compact code target.

Cure53 was tasked with auditing micro-btc-signer's cryptographic implementations and determining whether the library is suitable for real-world deployment in sensitive application layer contexts, such as Bitcoin wallets. This review spanned the entirety of the ~2,700 LoC library codebase, covering the correctness of the implementation of various Bitcoin transaction processing functionality, in relation to extensions such as SegWit and Taproot in addition.

Whilst one could argue that micro-btc-signer's smaller code target renders auditing easier to achieve and security bugs less likely to emerge - and with due consideration to the fact that this assessment could not detect any directly exploitable cryptographic vulnerabilities in the library - the testing team still noted three closely-related programming paradigm issues visible in micro-btc-signer's code. These were considered inappropriate for the library's chosen high-assurance deployment contexts, as covered in the introduction:

- Issues such as MBS-01-001 underline the risk created by scenarios whereby micro-btc-signer alters the currency denomination of input amounts by orders of magnitude, depending on their input-data type.
- Issues such as MBS-01-002, MBS-01-003, and MBS-01-004 demonstrate insufficiently lax coding paradigms, given the sensitive contexts in which micro-btc-signer is likely to be deployed.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

As such, whilst this audit was unable to detect directly exploitable issues in the micro-btc-signer library, one can conclude by reiterating the recommendations from the introduction. Namely, Cure53 advises rewriting the library in order to comprehensively integrate TypeScript's enhanced type safety features, which can be achieved by:

- Specifying full custom data types for all input data structures.
- Fully validating all input types to all high-level API functionality, and rendering private all functionality that does not require exposure to the application layer.
- Avoiding overloading functions with type conversion hacks, and enforcing that each function only accepts one data type for each input. As argued previously, in the eventuality this behavior facilitates repeated or more verbose code, this will not necessarily guarantee that micro-btc-signer will be more challenging to audit.

Cure53 would like to thank Ryan Shea and Paul Miller for their excellent project coordination, support, and assistance, both before and during this assignment.