SpruceID

Security Assessment

February 18, 2022

Prepared For:
Marco Schurtenberger | *Tezos Foundation*
marco.schurtenberger@tezos.foundation

Prepared By:
Artur Cygan | *Trail of Bits*
artur.cygan@trailofbits.com

Fredrik Dahlgren | *Trail of Bits*
fredrik.dahlgren@trailofbits.com

Changelog:

October 26, 2021:	Initial report draft
February 18, 2022:	Added Appendix D: Fix Log

Executive Summary

From September 13 to October 22, 2021, the Tezos Foundation engaged Trail of Bits to review the security of the SpruceID system. Trail of Bits conducted this assessment over 12 person-weeks, with 2 engineers working from the following GitHub repositories and commits:

- `spruceid/credible`: `3faef5c`
- `spruceid/did-tezos`: `fc1d078`
- `spruceid/didkit`: `436f53b` `eb8a4d0`
- `spruceid/kepler`: `16379fe`
- `spruceid/ssi`: `6d82e3e` `400effa`
- `spruceid/tzprofiles`: `c768043`

We used the first week of the assessment to familiarize ourselves with the documentation provided by W3C and Spruce on verifiable credentials and distributed identifiers, drawing security properties from the documentation to verify within the codebase. From there, we began reviewing the `SSI` library, focusing on the high-level verification flow for verifiable credentials and potential edge cases and type confusion issues related to JSON Web Tokens (JWTs). We also ran static analysis tools such as Semgrep, Dylint, `cargo-geiger`, and `cargo-audit` on the codebase.

During the second week of the assessment, we continued our review of the `SSI` library. We focused on verifiable credential proof verification, Resource Description Framework (RDF) dataset canonicalization, credential revocation, and Tezos DID resolution.

During the third week, we began reviewing DIDKit. We ran static analysis tools and manually reviewed the DIDKit language bindings for C, Swift, Java, Wasm (node and web implementations), Python, and Flutter. We also reviewed the DIDKit HTTP server implementation.

During the fourth week, we focused on the Credible iOS and Android applications. We ran the applications through Data Theorem's Mobile Secure analysis toolchain. We also manually reviewed the Flutter implementation, focusing on key generation, key recovery, and secure storage.

During the fifth week, we began reviewing the Tezos Profiles web application. We focused on the web application (implemented under `dapp`) and the Cloudflare worker (implemented under `worker`).

During the sixth week, we completed our review of Tezos Profiles, focusing on the API, indexer, smart contract client, and connection to Kepler.

This audit resulted in 41 findings. Most of the issues that we identified concern opportunities for code hardening, many of which are related to user input validation. Insufficient data validation resulted in a number of high-severity issues that allow an attacker to crash `SSI`-based applications (TOB-SPRUCE-001, TOB-SPRUCE-002) or to manipulate the Tezos Profiles worker's internal requests (TOB-SPRUCE-031). Some of the more severe issues in Credible are related to the application's protection of sensitive user data like key material and network traffic. Two of these issues allow a malicious user to steal the recovery phrase (and, hence, the private key) from an unlocked device (TOB-SPRUCE-018, TOB-SPRUCE-019); one issue allows a malicious user to intercept and tamper with Transport Layer Security (TLS)-protected traffic to and from the application (TOB-SPRUCE-023). Finally, we identified two issues that allow a malicious Tezos Profiles user to associate his or her Tezos address with identities belonging to other users (TOB-SPRUCE-033, TOB-SPRUCE-037).

The `SSI` library is the foundation of both the Credible iOS and Android applications as well as the Tezos Profiles web application. The `SSI` library is written in Rust, which provides strong type and memory safety guarantees that benefit the entire system. We also found that the library is well structured and uses Rust traits to decouple the components from each other. This makes the codebase easier to understand and evaluate, which in turn benefits the overall security of the system. On the other hand, both Credible and Tezos Profiles are still under active development, and there is functionality missing that would make the applications more secure for users. In particular, the Credible iOS and Android applications lack security features like biometric authentication and local data encryption, which are required to protect sensitive user data if a device is stolen or lost. The Tezos Profiles web application does not sanitize user input. It also implicitly trusts claims returned by the smart contract and by Kepler. This allows a malicious user to impersonate other users of the system.

Going forward, we recommend that the Spruce team focus on improving input validation throughout the `SSI` library, Credible, and Tezos Profiles to ensure that malicious or invalid user input does not enter the system. We also recommend that the Spruce team strengthen the local authentication controls performed by the Credible iOS and Android applications to mitigate potential data loss if users lose control of their devices.

Update: After the initial assessment, Trail of Bits reviewed the fixes implemented for the issues presented in this report. Detailed information on the results from the fix review can be found in Appendix D.

Project Dashboard

Application Summary

Name	SpruceID

Versions	`spruceid/credible: 3faef5c`
	`spruceid/did-tezos: fc1d078`
	`spruceid/didkit: 436f53b`
	`eb8a4d0`
	`spruceid/kepler: 16379fe`
	`spruceid/ssi: 6d82e3e`
	`400effa`
	`spruceid/tzprofiles: c768043`

Types	Rust, Dart, JavaScript, TypeScript
Platforms	Android, iOS, Web

Engagement Summary

Dates	September 13–October 22, 2021
Method	Full knowledge
Consultants Engaged	2
Level of Effort	12 person-weeks

Vulnerability Summary

Total High-Severity Issues	10	■■■■■■■■■■
Total Medium-Severity Issues	6	■■■■■■
Total Low-Severity Issues	4	■■■■
Total Informational-Severity Issues	15	■■■■■■■■■■■■■■■
Total Undetermined-Severity Issues	6	■■■■■■
Total	41	

Category Breakdown

Access Controls	3	■■■
Configuration	4	■■■■
Cryptography	4	■■■■

Data Exposure	3	■ ■ ■
Data Validation	18	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Denial of Service	2	■ ■
Error Reporting	1	■
Patching	4	■ ■ ■ ■
Undefined Behavior	2	■ ■
Total	41	

Code Maturity Recommendations

The following section highlights areas where process or code improvements would increase the maturity of the codebase and reduce the likelihood of associated risks.

Category Name	Description
Cryptography	**Recommended.** The `SSI` library outsources cryptographic operations like signing and signature verification to third-party crates. This significantly reduces the attack surface of the library. Higher-level components like the Credible and Tezos Profiles applications delegate cryptographic operations to either the `SSI` library (using DIDKit) or to third-party wallets. However, both the `SSI` library and Credible handle sensitive data like private keys, which should be zeroed out after use.
Data Exposure	**Strongly recommended.** The Credible iOS and Android applications do not implement local authentication. This should be required to mitigate the risk of a lost or stolen device. We also noted that Kepler logs potentially sensitive data as part of the authentication flow. We strongly recommend that the Spruce team perform an internal audit of the sensitive data handled by the different components of the system to reduce the risk of unintended exposure of sensitive data in the future.
Data Validation	**Required.** The `SSI` library performs ample structural verification of received credentials. However, we identified a number of low-level validation issues related to both Unicode string validation and verification of incoming network traffic from untrusted remote endpoints. Many of the issues identified in the Credible iOS and Android applications and the Tezos Profiles web application are related to missing or insufficient input validation. This is especially true for the Tezos Profiles web application. We strongly recommend that the Spruce team identify trust boundaries throughout the system and, for each trust boundary, ensure that data crossing the boundary is properly validated by the implementation.
Dependency Management	**Strongly recommended.** The dependency management for all the components is fairly standard for the respective language ecosystems. In many system components, a number of dependencies are outdated and have known vulnerabilities. The dependencies can be updated to resolve these issues, but this could become a recurring issue. To mitigate future risk, we recommend

	integrating automated tools for dependency vulnerability scanning and introducing security policies for patching deployed services once vulnerabilities are discovered.
Error Handling	**Recommended.** The Rust components (the `SSI` library and the Cloudflare worker) typically propagate errors back to the API consumer using the `Result` type. This design conforms with best practices. However, in some cases, the TypeScript applications also report errors back to the user. For web applications, this should typically be avoided since it can leak sensitive details about the internal state of the application. We recommend that the Spruce team implement a centralized logging system for internal errors and ensure that public-facing APIs report only high-level error messages to the end user.
Function Composition	**Strongly recommended.** The Tezos Profiles web application consists of a number of components: a TypeScript application based on the `SSI` library, a Cloudflare worker responsible for parsing new witnesses, a Tezos smart contract tracking user claims, a blockchain indexer that caches the state of user contracts, and Kepler, which is used as a storage back end. The interactions and trust relationships between these components are completely undocumented and rather involved; we recommend that the Spruce team identify the different actors and establish clear trust boundaries within the system.
Specification	**Strongly recommended.** We recommend that the Spruce team improve the documentation for Tezos Profiles. The documentation should include descriptions of the components that make up Tezos Profiles and the interactions and relationships between these components.
Testing and Verification	**Strongly recommended.** The `SSI` library contains a comprehensive test suite, but it could be extended to cover more edge cases and invalid inputs. We also recommend implementing fuzzing harnesses using `cargo-fuzz` against parsing functions and property tests using the `quickcheck` crate for better test coverage. The Tezos Profiles web application contains no unit tests or property tests. These should be added to ensure that user input is properly sanitized and that high-level Svelte components are implemented correctly. We also recommend extending the test suites for the Credible iOS and Android applications.

Engagement Goals

The engagement was scoped to provide a security assessment of the `SSI` library, DIDKit, the Credible iOS and Android applications, and the Tezos Profiles web application.

Specifically, we sought to answer the following questions:

- Is it possible to trick the `SSI` library into accepting forged or invalid credentials?
- Is the `SSI` library's cryptography resilient to timing attacks?
- Does the `SSI` library adhere to relevant W3C specifications?
- Is it possible to mount a denial-of-service attack against `SSI`-based systems by crashing the application or exhausting the resources of the system?
- Are cryptographic keys generated correctly and stored securely?
- Is authentication and authorization implemented correctly?
- Is untrusted data properly validated throughout the system?
- Can secret or confidential data be exposed?
- Are well-defined trust boundaries upheld by the Tezos Profiles web application?
- Are dependencies patched and up to date?

Coverage

`SSI`. We ran static analysis tools like Clippy, Semgrep, Dylint, `cargo-geiger`, and `cargo-audit` on the codebase to discover instances of unidiomatic Rust, code quality issues, and known vulnerable dependencies. We also identified and wrote fuzzing harnesses for potential fuzz targets such as parsing functions. We then performed a manual review of the `SSI` library, focusing on the credential verification flow, DID resolution, RDF dataset canonicalization, and type confusion issues common to other JWT implementations.

DIDKit. We ran a number of static analysis tools on the codebase to identify code quality issues and common vulnerable code patterns. We then performed a manual review of the DIDKit language bindings for C, Swift, Java, Wasm, Python, and Flutter. We also manually reviewed the DIDKit HTTP server, focusing on security hardening and untrusted data validation.

Credible. We used Data Theorem's Mobile Secure service for the initial analysis of the iOS and Android applications and triaged its results. We also performed an in-depth manual code review of the Flutter application and ran dynamic tests on the debug builds of the application.

Kepler. We ran a number of static analysis tools on and performed a non-exhaustive manual code review of Kepler. This was performed on a best-effort basis. We transferred our effort to Tezos Profiles, as agreed with the Tezos Foundation.

Tezos Profiles. We started by running static analysis tools like Semgrep, `eslint`, and `jshint` on the codebase to gauge the overall code quality of each component. We then performed an in-depth manual review of the web application, focusing on the user flows, such as user onboarding, adding a new claim, deleting a claim, and viewing a user profile. We also performed a manual review of the Cloudflare worker and Tezos Profiles API. Finally, we attempted to exploit the trust relationships between the system's components.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

❑ **Use the `starts_with` method, which is safe when working with multibyte characters, instead of slicing the input.** TOB-SPRUCE-001

❑ **Revise the process for parsing EIP-712 types so that it checks the input for the substring `[` before splitting the variable `string` on `[`, and that it returns an error if `[` is missing.** TOB-SPRUCE-002

❑ **Replace the call to `unwrap` in `bls_generate_keypair` with a check to verify that the `G::deserialize` function does not return an error.** TOB-SPRUCE-004

❑ **Fix the `did-tezos` code to call the correct URL corresponding to the network associated with the DID.** TOB-SPRUCE-005

❑ **Revise the `PassthoughDigest::update` function so that it panics if the input is not exactly 32 bytes.** TOB-SPRUCE-006

❑ **Ensure that the revocation list is loaded over a secure channel using HTTPS.** TOB-SPRUCE-007

❑ **Use `reqwest::Response::take` when reading the response body from the network to limit the size of the input to the application.** TOB-SPRUCE-008

❑ **Revise the encoding procedure so that one-element `credentialSubject` arrays are unpacked to a single value or produce an error.** TOB-SPRUCE-009

❑ **Hide the `https://example.edu/issuers/14` issuer behind a `#[cfg(test)]` attribute to ensure that it is not exposed in release builds of `SSI`.** TOB-SPRUCE-010

❑ **Document the possibility that `tzkt_url` can be configured through a CLI switch.** TOB-SPRUCE-011

❑ **Implement the smart contract address–based Tezos DIDs or update the specification if they are no longer part of the system.** TOB-SPRUCE-012

❑ **Revise the `CredentialStatus::check` function so that it rejects credentials with an invalid revocation list index.** TOB-SPRUCE-014

❑ **Implement timeouts (using Hyper, for example) to terminate slow connections to the DIDKit HTTP server, and implement rate limiting for individual clients.** For example, consider implementing rate limiting using the `tower` crate. TOB-SPRUCE-015

❑ **Rewrite the request handler logic to limit the number of bytes read from the network.** (The implementation of `hyper::body::aggregate` can be used as an inspiration.) TOB-SPRUCE-016

❑ **Implement `Zeroize` and `Drop` on either the `Params` or the `Base64urlUInt` type.** Alternatively, wrap both types using the `Zeroizing` type from the `zeroize` crate. TOB-SPRUCE-017

❑ **Use the `local_auth` plugin to integrate local authentication using Face ID or Touch ID (on iOS) and fingerprint APIs (on Android).** TOB-SPRUCE-018

❑ **Protect sensitive pages from being displayed in the application switcher.** TOB-SPRUCE-019

❑ **Revise the `OnBoardingRecoveryPage` class so that it checks that the recovered entropy string is the same size that is generated by the `OnBoardingGenPhrasePage`.** TOB-SPRUCE-021

❑ **Pass a `receiveTimeout` to the `Dio` constructor to limit the time the connection is kept alive.** Pass a custom `ProgressCallback` as `onReceiveProgress` to `Dio.get` to check the size of the response. TOB-SPRUCE-022

❑ **Hide the code that modifies the HTTP client behavior behind a debug compilation flag.** Alternatively, remove the code and add a self-signed certificate to the certificate store used by Credible during development. TOB-SPRUCE-023

❑ **Add a page to Credible that displays after users create new mnemonics to check that they saved the mnemonic.** TOB-SPRUCE-024

❑ **Fork the `bip39` library, fix this issue, and create a pull request on the main repository.** Until the pull request has been merged, use your own fork of the library in the application. TOB-SPRUCE-025

❑ **Disable third-party keyboards within the Credible iOS application.** TOB-SPRUCE-026

❑ **Consider disabling backups by setting `android:allowBackup` to `false` in the application manifest.** TOB-SPRUCE-027

❑ **Revise `checkIsWebsiteLive` so that it falls back to HTTP if HTTPS is not available rather than returning an error.** TOB-SPRUCE-028

❑ **Rewrite the regular expression used to validate domain names to conform with the format specified in RFC 1034.** TOB-SPRUCE-029

❑ **Update the Tezos Profiles dependencies to their newest versions.** Monitor the referenced GitHub thread regarding the `chrono` crate segfault issue. TOB-SPRUCE-030

❑ **Avoid constructing URLs directly based on user input.** Always validate all data that comes from untrusted sources. TOB-SPRUCE-031

❑ **Remove the Instagram attestation or fix the bugs in the code to restore the witness functionality.** TOB-SPRUCE-032

❑ **Sanitize all user input by normalizing it and ensuring that it contains only valid characters.** (In this case, this would require checking that the network name contains only alphanumeric characters.) TOB-SPRUCE-033, TOB-SPRUCE-034

❑ **Handle errors internally and do not report error messages from third-party frameworks to users.** TOB-SPRUCE-035

❑ **Consider the effect of raising the number of confirmations required for newly created contracts on the system.** TOB-SPRUCE-036

❑ **Fix the Tezos Profiles application so that, when displaying identity claims on `tzprofiles.com`, it verifies that the credential subject is equal to the user account associated with the page.** Additionally, ensure that the application checks that the credential issuer is given by `did:web:tzprofiles.com.` TOB-SPRUCE-037

❑ **Do not check the Michelson code into the repository.** Produce the Michelson version from the ReLIGO version at build time and use it as a build artifact in TypeScript. Lock the LIGO compiler version and use a hash of Michelson code to detect any differences should they arise after compilation. TOB-SPRUCE-038

❑ **Fix the Tezos Profiles worker so that it validates the data fetched from the Twitter and Cloudflare APIs.** This will make the Tezos Profiles service more resilient. TOB-SPRUCE-039

❑ **Remove the logging mechanism or change it to a debug level and ensure that it is not executed in production builds.** TOB-SPRUCE-040

❑ **Update the Kepler dependencies to their newest versions.** Monitor the referenced GitHub thread regarding the `chrono` crate segfault issue. TOB-SPRUCE-041

Long term

❑ **Integrate property testing and fuzzing into the development lifecycle to uncover other cases in which user-controlled input is not properly validated.** TOB-SPRUCE-001

❑ **Avoid calls to panicking functions like `expect` and `unwrap` in functions returning a `Result`.** Integrate property testing and fuzzing into the development lifecycle to uncover other cases in which user-controlled input is not properly validated. TOB-SPRUCE-002

❑ **Consider the impact of replacing the implementation of `From<SecretKey>` for `BlsSecretKey` with a safe implementation.** TOB-SPRUCE-003

❑ **Review the use of panicking functions like `unwrap` throughout the codebase.** Document each use and explain why it is safe. TOB-SPRUCE-004

❑ **Add tests to ensure that the fixed behavior is correct and to prevent future regressions.** TOB-SPRUCE-005

❑ **Consider updating the "Revocation List 2020" specification to indicate that the revocation list should be provided over a secure channel.** TOB-SPRUCE-007

❑ **Expand the test suite to include more examples of incorrect verifiable credentials.** TOB-SPRUCE-009

❑ **Implement and document data source customizability to enhance SpruceID decentralization.** TOB-SPRUCE-011

❑ **Investigate the possibility of using `TZIP-16` metadata to resolve the DID manager contract.** TOB-SPRUCE-013

❑ **Review the codebase for locations in which private key material is stored on the stack and rewrite the code to prevent this, if possible.** Alternatively, consider using the `clear_stack_on_return` function defined by the `clear_on_drop` crate to clear the stack when the function returns. TOB-SPRUCE-017

❑ **Consider using the more restrictive `IOSAccessibility.passcode` access level, which additionally requires the user to have a passcode set on the device.** (If the passcode is removed, stored items are deleted from the keychain.) TOB-SPRUCE-020

❑ **Do not use development code as a default, as developers may forget to remove it before the release of the application.** TOB-SPRUCE-023

❑ **Stay up to date with changes to iOS that might permit data exfiltration from the client.** TOB-SPRUCE-026

❑ **Consider using a HEAD request and limiting the response time and size to ensure that the `checkIsWebsiteLive` test is fast.** TOB-SPRUCE-028

❑ **Run `cargo-audit` and `npm audit` as part of the CI/CD pipeline and ensure that the team is alerted to any vulnerable dependencies that are detected.** TOB-SPRUCE-030

❑ **Pay attention to compiler warnings, as the catch-all pattern issue could have been detected before the application's release.** Configure a CI/CD pipeline to alert the team when compiler warnings are raised. TOB-SPRUCE-032

❑ **Provide links to user attestations in the user profile on `tzprofiles.com` to allow anyone to verify that the attestations are correct.** This would also allow users to check *when* a claim was made, which is often useful since identity claims may not be valid forever. (For example, a user may change his or her username, and another user may set up a new account with the original username.) TOB-SPRUCE-033

❑ **Consider raising the number of confirmations required for newly created contracts to a number that better aligns with the Tezos recommendation.** TOB-SPRUCE-036

❑ **Consider allowing `tzprofiles.com` users to view or save the actual verifiable credentials from Kepler for extra security.** TOB-SPRUCE-037

❑ **Avoid duplicating code, as it is harder to patch.** TOB-SPRUCE-038

❑ **Ensure that data coming from third-party services is always validated.** TOB-SPRUCE-039

❑ **Do not log any sensitive data in production.** The logs might be shipped to a third-party service for analysis, making them susceptible to a data leak. TOB-SPRUCE-040

❑ **Run `cargo-audit` as part of the CI/CD pipeline and ensure that the team is alerted to any vulnerable dependencies that are detected.** TOB-SPRUCE-041

Findings Summary

#	Title	Type	Severity
1	The Tezos DID resolver accepts invalid input that can crash the program	Data Validation	High
2	The program can crash when parsing EIP-712 types	Data Validation	High
3	Potentially unsafe dependency on the internal representation of types from the bbs crate	Undefined Behavior	Informational
4	Potential panic when creating a new BLS key pair	Data Validation	Informational
5	Tezos DID resolver does not take the network into account	Data Validation	Undetermined
6	PassthroughDigest reduces the entropy of the output for digests that are not 32 bytes	Cryptography	Informational
7	HTTPS is not enforced when loading a revocation list	Cryptography	Medium
8	Potential resource exhaustion when loading a revocation list	Data Validation	Medium
9	JWT encoding may produce an invalid credential	Data Validation	Undetermined
10	Issuer that is used for testing is exposed in release builds	Configuration	Informational
11	DIDKit CLI option to change tzkt_url is not documented	Configuration	Informational
12	Smart contract address–based Tezos DIDs are not implemented	Undefined Behavior	Informational
13	The DID manager resolution process supports only the default TZIP-19 contract	Patching	Informational
14	Verifiable credentials with invalid revocation list indices are accepted by default	Data Validation	Informational
15	DIDKit HTTP server is vulnerable to slowloris attacks	Denial of Service	Undetermined

16	DIDKit HTTP server is vulnerable to memory resource exhaustion	Denial of Service	Undetermined
17	Private key material is not cleared from memory when no longer needed	Cryptography	Medium
18	Credible lacks protection against unauthorized user access	Access Controls	Medium
19	Credible does not protect sensitive data when switching applications	Access Controls	Medium
20	Consider using IOSAccessibility.passcode to protect keychain items on iOS	Access Controls	Medium
21	Credible does not validate the length of the recovery phrase	Data Validation	Informational
22	The QR code handler is vulnerable to denial-of-service attacks	Data Validation	Low
23	Credible disables TLS certificate verification	Data Validation	High
24	Credible does not prompt users to write down generated mnemonics	Configuration	Informational
25	The bip39.generateMnemonic function generates non-uniform entropy	Cryptography	Low
26	The Credible iOS client allows third-party keyboards	Data Exposure	High
27	The Credible Android client allows backups of stored credentials	Data Exposure	Undetermined
28	The checkIsWebsiteLive function checks only for pages served over HTTPS	Data Validation	Informational
29	The isValidUrl function does not conform to RFC 1034	Data Validation	Low
30	Tezos Profiles uses vulnerable dependencies	Patching	High
31	The Tezos Profiles worker is vulnerable to URL injection attacks	Data Validation	High
32	The Tezos Profiles Instagram attestation is broken	Data Validation	Informational

33	The Tezos Profiles web app is vulnerable to URL injection attacks	Data Validation	High
34	The Tezos Profiles API is vulnerable to URL parameter injection attacks	Data Validation	Undetermined
35	The Tezos Profiles API reports internal errors to users	Error Reporting	Low
36	Too few confirmations required when deploying new smart contracts	Configuration	Informational
37	Tezos Profiles does not validate the credential subject or issuer	Data Validation	High
38	The Tezos Profiles contract code is duplicated	Patching	Informational
39	The Tezos Profiles worker insufficiently validates data coming from third-party APIs	Data Validation	High
40	Kepler logs authentication data	Data Exposure	Informational
41	Kepler uses vulnerable Rust dependencies	Patching	High

1. The Tezos DID resolver accepts invalid input that can crash the program

Severity: High
Difficulty: Low
Type: Data Validation
Finding ID: TOB-SPRUCE-001
Target: `SSI`, `did-tezos resolver`

Description

The Tezos DID resolver, implemented in the `SSI` library, does not sufficiently validate data and accepts input that will cause a panic. Figure 1.1 shows an example of input that crashes the program. The string slicing process, shown in figure 1.2, causes this problem. In Rust, slicing strings with unknown content is unsafe because strings are UTF-8 encoded and their slices are guaranteed to be valid UTF-8 sequences. Slicing in the middle of a multibyte character invalidates this property and causes a panic.

Every other service using the `SSI` library, such as DIDKit and Kepler, is also affected by the vulnerability.

```
$ didkit did-resolve did:tz:🤯aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
thread 'main' panicked at 'byte index 3 is not a char boundary; it is inside '🤯' (bytes
0..4) of `🤯aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`', /.../ssi/did-tezos/src/lib.rs:84:23
```

Figure 1.1: Input that crashes the program while resolving a Tezos DID

```
impl DIDResolver for DIDTz {
    async fn resolve(/* <redacted> */) {
        let (network, address) = match did.split(':').collect::<Vec<&str>>().as_slice() {
            ["did", "tz", address] if address.len() == 36 => {
                ("mainnet", address.to_string())
            },
            ["did", "tz", network, address] if address.len() == 36 => {
                (*network, address.to_string())
            }
            _ => {
                return (
                    ResolutionMetadata::from_error(&ERROR_INVALID_DID),
                    None,
                    None,
                )
            }
        };
        // ... <redacted>
        let prefix = &address[0..3];
        // ... <redacted>
    }
}
```

Figure 1.2: Relevant code with the vulnerability marked in red
(did-tezos/src/lib.rs#L45-L84)

A similar issue exists in the `decode_tzsig` function (figure 1.3). The panic cannot be triggered in this case because the input (the `sig_bs58` variable) is first passed to the `bs58::decode` function, which returns an error if the input contains non-ASCII characters.

```
pub fn decode_tzsig(sig_bs58: &str)
        -> Result<(Algorithm, Vec<u8>), DecodeTezosSignatureError> {
    let tzsig = bs58::decode(&sig_bs58).with_check(None).into_vec()?;
    if tzsig.len() < 5 {
        return Err(DecodeTezosSignatureError::SignaturePrefix(
            sig_bs58.to_string(),
        ));
    }
    // sig_bs58 has been checked as base58. But use the non-panicking get function anyway,
    // for good measure.
    let (algorithm, sig) = match sig_bs58.get(0..5) {
        Some("edsig") => (Algorithm::EdBlake2b, tzsig[5..].to_vec()),
        Some("spsig") => (Algorithm::ESBlake2bK, tzsig[5..].to_vec()),
        Some("p2sig") => (Algorithm::ESBlake2b, tzsig[4..].to_vec()),
        _ => // ... <redacted>
    };
    // ... <redacted>
}
```

Figure 1.3: [src/tzkey.rs#L196-L219]

Exploit Scenario
An attacker prepares a malicious DID, which is resolved in a production deployment of Kepler. The system crashes and becomes unavailable to other users.

Recommendations
Short term, use the [starts_with method], which is safe when working with multibyte characters, instead of slicing the input.

Long term, integrate property testing and fuzzing into the development lifecycle to uncover other cases in which user-controlled input is not properly validated.

References

- [Storing UTF-8 Encoded Text with Strings: The Rust Programming Language]

2. The program can crash when parsing EIP-712 types

Severity: High
Type: Data Validation
Target: `ssi/src/eip712.rs`

Difficulty: Low
Finding ID: TOB-SPRUCE-002

Description

The implementation of the `TryFrom` trait for the `EIP712Type` calls `unwrap` multiple times, which could lead to a panic on maliciously crafted input. (Calling `unwrap` should generally be avoided in functions returning a `Result`.)

```
fn try_from(string: String) -> Result<Self, Self::Error> {
    // ... <redacted>
    if string.ends_with("]") {
        let mut parts = string.rsplitn(2, "[");
        let amount_str = parts.next().unwrap().split("]").next().unwrap();
        let base = EIP712Type::try_from(parts.next().unwrap().to_string())?;
        if amount_str.len() == 0 {
            return Ok(EIP712Type::Array(Box::new(base)));
        } else {
            return Ok(EIP712Type::ArrayN(
                Box::new(base),
                usize::from_str(amount_str)?,
            ));
        }
    }
    // ... <redacted>
}
```

Figure 2.1: `EIP712Type::try_from::<String>` panics on an input string that does not contain `[` ([`ssi/src/eip712.rs`]()).

The implementation splits the string on the substring `[`, but if the string does not contain `[`, the result, `parts`, will contain only a single element. This means that the second call to `next` will return `None`, and the subsequent call to `unwrap` will panic.

The following is a panicking test case that can be added to `src/eip712.rs`:

```
use std::convert::TryFrom;

#[test]
#[should_panic]
fn test_eip712_type_from_string() {
    EIP712Type::try_from("AAAAAAAAARGH!]".to_string());
}
```

Figure 2.2: A panicking test case for TOB-SPRUCE-002

Other services that rely on `SSI`, like DIDKit and Kepler, are also affected.

Exploit Scenario
An attacker prepares a malicious `EthereumEip712Signature2021` verifiable credential that is deserialized in a production deployment of Kepler. The system crashes and becomes unavailable to other users.

Recommendations
Short term, revise the process for parsing EIP-712 types so that it checks the input for the substring `[` before splitting the variable `string` on `[`, and that it returns an error if `[` is missing.

Long term, avoid calls to panicking functions like `expect` and `unwrap` in functions returning a `Result`. Integrate property testing and fuzzing into the development lifecycle to uncover other cases in which user-controlled input is not properly validated.

3. Potentially unsafe dependency on the internal representation of types from the bbs crate

Severity: Informational
Difficulty: N/A
Type: Undefined Behavior
Finding ID: TOB-SPRUCE-003
Target: `ssi/src/bbs.rs`

Description

The implementation of `From<SecretKey>` for `BlsSecretKey` uses `std::mem::transmute` to convert the internal representation of a `bbs` crate `SecretKey` to a `BlsSecretKey`.

```
impl From<SecretKey> for BlsSecretKey {
    fn from(x: SecretKey) -> Self {
        unsafe { std::mem::transmute(x) }
    }
}
```

Figure 3.1: The implementation of `From<SecretKey>` introduces a potentially unsafe dependency on the internal representation of a type from a third-party crate ([`ssi/src/bbs.rs`]).

This behavior is currently safe because the internal representation of a `SecretKey` is the same as that of a `BlsSecretKey`. However, because the `bbs` crate is not developed by Spruce, the internal representation used by this crate may change at any time. This would silently break the implementation of the `From<SecretKey>` trait.

Recommendations

Long term, consider the impact of replacing the implementation of `From<SecretKey>` for `BlsSecretKey` with a safe implementation.

4. Potential panic when creating a new BLS key pair

Severity: Informational
Type: Data Validation
Target: `ssi/src/bbs.rs`

Difficulty: Low
Finding ID: TOB-SPRUCE-004

Description

Passing an invalid blinding value to `BlsKeyPair::new` causes the implementation to panic. The `BlsKeyPair::new` function calls `bls_generate_keypair` to generate a new key pair. If a blinding value is supplied to the function, it is passed to `G::deserialize`, in which `G` is a generic `CurveProjective` implementation.

```
fn bls_generate_keypair<G: CurveProjective<Engine = Bls12, Scalar = Fr> + SerDes>(
    seed: Option<&[u8]>,
    blinder: Option<&[u8]>,
) -> BlsKeyPair<G> {
    // ... <redacted>

    let r = match blinder {
        Some(g) => {
            let mut data = g.to_vec();
            let mut gg = g.clone();

            // ... <redacted>
            let mut blinding_g = G::deserialize(&mut gg, true).unwrap();
            let r = gen_sk(data.as_slice());
            blinding_g.mul_assign(r);
            pk.add_assign(&blinding_g);
            Some(r)
        }
        None => None,
    };

    BlsKeyPair {
        secret_key: BlsSecretKey(sk),
        public_key: BlsPublicKey(pk),
        blinder: r,
    }
}
```

Figure 4.1: The blinding value is passed to `G::deserialize` without prior validation ([`ssi/src/bbs.rs`](ssi/src/bbs.rs)).

If the call to `deserialize` fails, the subsequent call to `unwrap` will panic.

The following is a panicking test case that can be added to `ssi/src/bbs.rs`:

```rust
    use super::*;
    use pairing_plus::bls12_381;

    #[test]
    #[should_panic]
    fn test_curve_pair() {
        let _ = BlsKeyPair::<bls12_381::G1>::new(None, Some(&[13]));
    }
```

Figure 4.2: A panicking test case for TOB-SPRUCE-004

This issue is only of informational severity because the code is currently not used by the `SSI` library.

Recommendations

Short term, replace the call to `unwrap` in `bls_generate_keypair` with a check to verify that the `G::deserialize` function does not return an error.

Long term, review the use of panicking functions like `unwrap` throughout the codebase. Document each use and explain why it is safe.

5. Tezos DID resolver does not take the network into account

Severity: Undetermined
Type: Data Validation
Target: ssi/`did-tezos/src/lib.rs`

Difficulty: Low
Finding ID: TOB-SPRUCE-005

Description

The Tezos DID resolver uses the TzKT API to fetch contracts created by a Tezos account. The process always calls the mainnet API rather than considering the Tezos network associated with the DID. For example, the correct URL for the Florencenet API is https://api.florencenet.tzkt.io/, but the process calls https://api.tzkt.io/ (figure 5.1). The mainnet URL is hard-coded (figure 5.2).

```
$ didkit did-resolve did:tz:florencenet:tz1UR7LHjsPo7x3ArSPnt2WjFRxhKuWt1Qas
error sending request for url
(https://api.tzkt.io/v1/contracts?creator=tz1UR7LHjsPo7x3ArSPnt2WjFRxhKuWt1Qas&sort=lastActi
vity&select=address&codeHash=1222545108): ...
$ didkit did-resolve did:tz:tz1UR7LHjsPo7x3ArSPnt2WjFRxhKuWt1Qas
error sending request for url
(https://api.tzkt.io/v1/contracts?creator=tz1UR7LHjsPo7x3ArSPnt2WjFRxhKuWt1Qas&sort=lastActi
vity&select=address&codeHash=1222545108): ...
```

Figure 5.1: Incorrect URL called for Florencenet observed without access to the internet

```
impl Default for DIDTz {
    fn default() -> Self {
        Self {
            tzkt_url: "https://api.tzkt.io/",
        }
    }
}
```

Figure 5.2: `did-tezos/src/lib.rs#L35-L41`

Exploit Scenario

A developer uses the Tezos DID resolver, which does not work correctly and leads to system malfunction.

Recommendations

Short term, fix the `did-tezos` code to call the correct URL corresponding to the network associated with the DID.

Long term, add tests to ensure that the fixed behavior is correct and to prevent future regressions.

6. PassthroughDigest reduces the entropy of the output for digests that are not 32 bytes

Severity: Informational
Type: Cryptography
Target: `ssi/src/passthrough_digest.rs`

Difficulty: N/A
Finding ID: TOB-SPRUCE-006

Description

When signing documents using ECDSA over either secp256r1 or secp256k1, the document hash is wrapped as follows:

```
let digest = Digest::chain(<PassthroughDigest as Digest>::new(), &hash);
```

Figure 6.1: A number of signature implementations use the `PassthoughDigest` type to wrap a hash with an object that implements the `Digest` trait.

The result is then passed to `DigestSigner::try_sign_digest` to sign the digest. The right-hand side of this statement creates a new `PassthoughDigest` instance, calls `update` on it with the given `hash` as input, and then returns the newly created `PassthoughDigest` instance.

The `PassthoughDigest::update` function stores the input in the value field if the input size is exactly 32 bytes. If the input size is not exactly 32 bytes, `PassthoughDigest::update` will store either the first byte of the input repeated 32 times, or 0 repeated 32 times.

```
impl Update for PassthroughDigest {
    fn update(&mut self, data: impl AsRef<[u8]>) {
        let d = data.as_ref();
        if d.len() == 32 {
            self.value = d.try_into().unwrap();
        } else if d.len() > 0 {
            self.value = [d[0]; 32];
        } else {
            self.value = [0; 32];
        }
    }
}
```

Figure 6.2: If the input size is different than 32, the stored value will always take one of 256 different values ([`ssi/src/passthrough_digest.rs`](ssi/src/passthrough_digest.rs)).

The actual digest, which is signed by `DigestSigner::try_sign_digest`, is provided by the output from `PassthroughDigest::finalize_into`, which just copies the stored value. If this construction is ever used with a hash function with a different digest size, this output will always be one of 256 different values `[b, b, b, …, b]` for b < 256.

Exploit Scenario

A new proof suite using a hash function with a digest size that is not 32 bytes is added to the `SSI` library. The hash function is used with `PassthoughDigest` to implement the `Digest` trait. Since the output of `PassthroughDigest::finalize_into` is now always one of 256 different values, an attacker is able to forge signatures by simply lifting a valid signature from one document to another with a matching `PassthroughDigest` output.

Recommendations
Short term, revise the `PassthoughDigest::update` function so that it panics if the input is not exactly 32 bytes.

7. HTTPS is not enforced when loading a revocation list

Severity: Medium
Difficulty: Medium
Type: Cryptography
Finding ID: TOB-SPRUCE-007
Target: `ssi/src/revocation.rs`

Description
The `load_credential` function is used to load a revocation list from the URL given by the `revocationListCredential` field. The function does not check that the credential is loaded using HTTPS. (This requirement is not enforced by the "Revocation List 2020" specification either.)

Exploit Scenario 1
Bob, an issuer of verifiable credentials, configures his system to use plaintext HTTP to share his revocation list. Eve, an attacker with a privileged network position, presents a revoked credential for verification to Alice. When Alice requests the revocation list from Bob, Eve replaces the response in-flight with an earlier version of Bob's revocation list. The earlier version of the revocation list still verifies correctly, and according to the earlier version of the list, Eve's credential is valid. Thus, Eve's credential will be verified as valid by Alice, even though it has been revoked by Bob.

Exploit Scenario 2
An update to the `reqwest` library implements the file URL scheme. This allows an attacker to point the `revocationListCredential` URL to the local file system. By exploiting timing differences when attempting to load the revocation list from a file, the attacker leaks information about the layout of the local file system.

Recommendations
Short term, ensure that the revocation list is loaded over a secure channel using HTTPS.

Long term, consider updating the "Revocation List 2020" specification to indicate that the revocation list should be provided over a secure channel.

8. Potential resource exhaustion when loading a revocation list

Severity: Medium
Difficulty: Medium
Type: Data Validation
Finding ID: TOB-SPRUCE-008
Target: `ssi/src/revocation.rs`

Description

The `load_resource` function is used to load the revocation list from a remote URL. The function uses `reqwest::Response::bytes` to obtain the full response body as bytes.

```
async fn load_resource(url: &str) -> Result<Vec<u8>, LoadCredentialError> {
    // ... <redacted>
    let resp = client
        .get(url)
        .header("Accept", accept)
        .send()
        .await
        .map_err(|e| LoadResourceError::Request(e))?;
    // ... <redacted>
    let bytes = resp
        .bytes()
        .await
        .map_err(|e| LoadResourceError::Response(e.to_string()))?
        .to_vec();
    Ok(bytes)
}
```

Figure 8.1: Using `reqwest::Response::bytes` to read the response body may lead to an out-of-memory error if the response is large ([`ssi/src/revocation.rs`]).

This method does not impose any limits on the size of the response, and using it may leave the client open to memory exhaustion attacks from the server or an attacker in a privileged network position.

Exploit Scenario

An `SSI`-based application requests a revocation list from an issuer. An attacker with a privileged network position replaces the revocation list in the response with a very large payload. When the application reads the data over the network, it runs out of memory and crashes, causing a denial of service.

Recommendations

Short term, use `reqwest::Response::take` when reading the response body from the network to limit the size of the input to the application.

9. JWT encoding may produce an invalid credential

Severity: Undetermined
Type: Data Validation
Target: `ssi/src/vc.rs`

Difficulty: High
Finding ID: TOB-SPRUCE-009

Description

The `Credential::generate_jwt` and `Presentation::generate_jwt` functions implement JSON Web Token (JWT) encoding of verifiable credentials and presentations. The encoding has the following constraint (from [Verifiable Credentials Data Model 1.0: JWT Encoding]):

"Implementers are warned that JWTs are not capable of encoding multiple subjects and are thus not capable of encoding a verifiable credential with more than one subject. JWTs might support multiple subjects in the future and implementers are advised to refer to the JSON Web Token Claim Registry for multi-subject JWT claim names or the Nested JSON Web Token specification."

A verifiable credential with a one-element `credentialSubject` array (figure 9.1) will be successfully encoded with the `generate_jwt` function. However, the resulting JWT (figure 9.2) cannot be successfully converted back with `decode_verify_jwt`. This issue occurs because one-element arrays are treated as single elements during the subject encoding (figures 9.3, 9.4).

```
{
    "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "id": "http://example.org/credentials/192783",
    "type": "VerifiableCredential",
    "issuer": "did:example:foo",
    "issuanceDate": "2020-08-25T11:26:53Z",
    "credentialSubject": [{
        "id": "did:example:a6c78986cc36418b95a22d7f736",
        "spouse": "Example Person"
    }]
}
```

Figure 9.1: A verifiable credential with a one-element `credentialSubject` array

```
{
  "iss": "did:example:foo",
  "nbf": 1598354813,
  "jti": "http://example.org/credentials/192783",
  "sub": "did:example:a6c78986cc36418b95a22d7f736",
  "aud": "did:example:90336644520443d28ba78beb949",
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": "VerifiableCredential",
    "credentialSubject": [
      {
        "id": "did:example:a6c78986cc36418b95a22d7f736",
        "spouse": "Example Person"
      }
    ],
    "issuanceDate": "2020-08-25T11:26:53Z"
  }
}
```

Figure 9.2: The result of the JWT generation from the verifiable credential presented in figure 9.1

```
pub fn to_jwt_claims(&self) -> Result<JWTClaims, Error> {
    let subject = match self.credential_subject.to_single() { … }
}
```

Figure 9.3: ssi/src/vc.rs#L780-L781

```
pub fn to_single(&self) -> Option<&T> {
    match self {
        Self::One(value) => Some(&value),
        Self::Many(values) => {
            if values.len() == 1 {
                Some(&values[0])
            } else {
                None
            }
        }
    }
}
```

Figure 9.4: ssi/src/one_or_many.rs#L61-L63

Exploit Scenario
A developer uses the `SSI` library with the assumption that all JWTs encoded with the `generate_jwt` function will successfully decode. The assumption is not met, leading to a system malfunction.

Recommendations
Short term, revise the encoding procedure so that one-element `credentialSubject` arrays are unpacked to a single value or produce an error.

Long term, expand the test suite to include more examples of incorrect verifiable credentials.

10. Issuer that is used for testing is exposed in release builds

Severity: Informational
Difficulty: N/A
Type: Configuration
Finding ID: TOB-SPRUCE-010
Target: `ssi/src/ldp.rs`

Description

The `ensure_or_pick_verification_relationship` function supports only DID issuer fields, according to a comment in the code, and it returns an `UnsupportedNonDIDIssuer` error if the issuer is not a DID.

```
pub(crate) async fn ensure_or_pick_verification_relationship(
    options: &mut LinkedDataProofOptions,
    document: &(dyn LinkedDataDocument + Sync),
    key: &JWK,
    resolver: &dyn DIDResolver,
) -> Result<(), Error> {
    let issuer = match document.get_issuer() {
        None => {
            // No issuer: no check is done.
            // TODO: require issuer - or invokers set for ZCap
            return Ok(());
        }
        Some(issuer) => issuer,
    };
    // ... <redacted>
    if !issuer.starts_with("did:") {
        // Allow some for testing purposes only.
        match &issuer[..] {
            "https://example.edu/issuers/14" => {
                // We don't have a way to actually resolve this to anything.
                // Just allow it for vc-test-suite for now.
                return Ok(());
            }
            _ => {
                return Err(Error::UnsupportedNonDIDIssuer(issuer.to_string()));
            }
        }
    }
    // ... <redacted>
    Ok(())
}
```

Figure 10.1: The issuer `https://example.edu/issuers/14` is exposed in release builds (`ssi/src/ldp.rs`).

However, the issuer `https://example.edu/issuers/14` is also allowed for testing purposes. Because this issuer is used only for testing, it should be hidden behind a `#[cfg(test)]` attribute.

Recommendations

Short term, hide the `https://example.edu/issuers/14` issuer behind a `#[cfg(test)]` attribute to ensure that it is not exposed in release builds of `SSI`.

11. DIDKit CLI option to change tzkt_url is not documented

Severity: Informational
Difficulty: High
Type: Configuration
Finding ID: TOB-SPRUCE-011
Target: `DIDKit CLI`

Description

The Tezos DID resolver has a configurable `tzkt_url` that can be passed from the command-line interface (CLI) (figure 11.1); however, the option is not documented anywhere.

```
$ didkit did-resolve did:tz:delphinet:tz1TzrmTBSuiVHV2VfMnGRMYvTEPCP42oSM8 -i
'tzkt_url=http://localhost:8888'
ResolutionInputMetadata { accept: None, version_id: None, version_time: None, no_cache:
None, property_set: Some({"tzkt_url": String("http://localhost:8888")}) }
error sending request for url
(http://localhost:8888/v1/contracts?creator=tz1TzrmTBSuiVHV2VfMnGRMYvTEPCP42oSM8&sort=lastAc
tivity&select=address&codeHash=1222545108): error trying to connect: tcp connect error:
Connection refused (os error 61)
```

Figure 11.1: The hidden option in `didkit did-resolve`

Exploit Scenario

A DIDKit CLI user does not trust https://tzkt.io/ and wants to use another trusted Tezos node as a data source for DID resolving. The option is not documented, and the user takes an unnecessary risk.

Recommendations

Short term, document the possibility that `tzkt_url` can be configured through a CLI switch.

Long term, implement and document data source customizability to enhance SpruceID decentralization.

12. Smart contract address–based Tezos DIDs are not implemented

Severity: Informational
Difficulty: High
Type: Undefined Behavior
Finding ID: TOB-SPRUCE-012
Target: `ssi/did-tezos/src.rs`

Description
The Tezos DID resolver does not implement smart contract address–based DIDs. While smart contract address–based DIDs are part of the Tezos DID method specification, there is no mention of them in the code.

Recommendations
Short term, implement the smart contract address–based Tezos DIDs or update the specification if they are no longer part of the system.

13. The DID manager resolution process supports only the default TZIP-19 contract

Severity: Informational
Difficulty: N/A
Type: Patching
Finding ID: TOB-SPRUCE-013
Target: `did-tezos/src/explorer.rs`

Description

The `retrieve_did_manager` function is used to retrieve the DID manager for an account-based Tezos DID. In the specification, the DID manager is defined as follows:

"In the case of an account address, the DID Manager smart contract is defined as the first smart contract that (1) is deployed by the account and (2) implements TZIP-19."

However, `retrieve_did_manager` simply queries the `TzKT` API for a list of contracts matching a fixed code hash (1222545108) and then returns the first contract from that list (or `None` if the list is empty).

```
pub async fn retrieve_did_manager(bcd_url: &str, address: &str)
        -> Result<Option<String>> {
    let client = reqwest::Client::builder().build()?;
    let url = Url::parse(bcd_url)?;
    let contracts: Vec<String> = client
        .get(url.join("/v1/contracts")?)
        .query(&[
            ("creator", address),
            ("sort", "lastActivity"),
            ("select", "address"),
            // TODO using codeHash while all contracts have the same code and
            // until tezedge-client provide a way to fetch TZIP-016 metadata.
            ("codeHash", "1222545108"),
        ])
        .send()
        .await?
        .json()
        .await?;

    if contracts.len() > 0 {
        Ok(Some(contracts[0].clone()))
    } else {
        Ok(None)
    }
}
```

Figure 13.1: The Tezos DID manager resolution process currently supports only the default TZIP-19 contract (`did-tezos/src/explorer.rs`).

Recommendations

Long term, investigate the possibility of using `TZIP-16` metadata to resolve the DID manager contract.

14. Verifiable credentials with invalid revocation list indices are accepted by default

Severity: Informational
Type: Data Validation
Target: `ssi/src/revocation.rs`

Difficulty: N/A
Finding ID: TOB-SPRUCE-014

Description

The implementation of `CredentialStatus::check` for `RevocationList2020Status` checks the credential input against the revocation list by examining the bit at index `revocationListIndex` in the decoded revocation list. If the bit is set, the credential has been revoked. However, if the `revocationListIndex` in the credential is invalid (e.g., if it is larger than the size of the decoded list), the function returns an empty `VerificationResult`, which indicates that the check was passed.

```
impl CredentialStatus for RevocationList2020Status {
    async fn check(
        &self,
        credential: &Credential,
        resolver: &dyn DIDResolver,
    ) -> VerificationResult {
        let mut result = VerificationResult::new();

        // ... <redacted>
        let revoked = match bitstring.get(credential_index) {
            Some(bitref) => *bitref,
            None => false,
        };
        if revoked {
            return result.with_error("Credential is revoked.".to_string());
        }
        result
    }
}
```

Figure 14.1: If the `credential_index` is invalid, `bitstring.get(credential_index)` returns None, and the check passes ([`ssi/src/revocation.rs`]).

Since the credential contains invalid data, we recommend as a defense-in-depth measure to reject the credential in this case.

Recommendations

Short term, revise the `CredentialStatus::check` function so that it rejects credentials with an invalid revocation list index.

15. DIDKit HTTP server is vulnerable to slowloris attacks

Severity: Undetermined
Difficulty: Low
Type: Denial of Service
Finding ID: TOB-SPRUCE-015
Target: `didkit/http`

Description
The HTTP component of DIDKit is vulnerable to slowloris attacks. Opening 253 connections to the server and keeping them alive exhausts the thread pool and causes further connection attempts to fail.

```
❯ target/release/didkit-http
Listening on http://127.0.0.1:50235/

❯ slowloris -s 256 -p 50235 127.0.0.1
[29-09-2021 09:50:01] Attacking 127.0.0.1 with 256 sockets.
[29-09-2021 09:50:01] Creating sockets...
[29-09-2021 09:50:01] Sending keep-alive headers... Socket count: 253

❯ curl --head 127.0.0.1:50235
curl: (56) Recv failure: Connection reset by peer
```

Figure 15.1: Opening a large number of connections to the DIDKit HTTP server will exhaust the number of available connections.

Exploit Scenario
An attacker runs a slowloris attack against the DIDKit HTTP server. This exhausts the resources available to the server and causes all further connection attempts from other users to fail.

Recommendations
Short term, implement timeouts (using Hyper, for example) to terminate slow connections to the DIDKit HTTP server, and implement rate limiting for individual clients. For example, consider implementing rate limiting using the `tower` crate.

Alternatively, add documentation to the project README file recommending the use of a hardened reverse proxy to protect the server against hostile behavior.

16. DIDKit HTTP server is vulnerable to memory resource exhaustion

Severity: Undetermined
Type: Denial of Service
Target: `didkit/http`

Difficulty: Low
Finding ID: TOB-SPRUCE-016

Description
The DIDKit HTTP server is vulnerable to memory exhaustion attacks. Since the server does not limit the size of incoming requests, it is possible to exhaust the memory available to the server and crash the application.

In `DIDKitHTTPSvc::verify_credentials`, the application reads the request body using the `hyper::body::aggregate` function, which simply aggregates the data read from the network. The Hyper documentation includes the following warning about this function:

"Care needs to be taken if the remote is untrusted. The function doesn't implement any length checks and an [sic] malicious peer might make it consume arbitrary amounts of memory. Checking the `Content-Length` is a possibility, but it is not strictly mandated to be present."

By executing the following script, which simply sends data from `/dev/zero` to the `verify/credentials` API endpoint, it is possible to exhaust the memory available to the server and ultimately crash the application.

```
HOST = $1
PORT = $2
while true
do
  dd if=/dev/zero bs=1024 count=1024
done |
  curl --trace-ascii - \
    -H "Transfer-Encoding: chunked" \
    -H "Content-Type: application/json" \
    -X POST -T - "http://$HOST:$PORT/verify/credentials"
```

Figure 16.1: This bash script uses HTTP chunked transfer encoding to transfer large amounts of data to the `verify/credentials` endpoint.

The following POST request handlers are vulnerable to the same attack:

- `DIDKitHTTPSvc::verify_credentials`
- `DIDKitHTTPSvc::issue_credentials`
- `DIDKitHTTPSvc::verify_presentations`
- `DIDKitHTTPSvc::prove_presentations`

Exploit Scenario
An attacker uses the script in figure 16.1 to transfer large amounts of data to the DIDKit HTTP server. This exhausts the memory available to the server and causes the application to crash, thus making it unavailable to other users.

Recommendations
Short term, rewrite the request handler logic to limit the number of bytes read from the network. (The implementation of `hyper::body::aggregate` can be used as an inspiration.)

Alternatively, add documentation to the project README file recommending the use of a hardened reverse proxy to protect the server against hostile behavior.

17. Private key material is not cleared from memory when no longer needed

Severity: Medium
Type: Cryptography
Target: `ssi/src/jwk.rs` and `ssi/src/jws.rs`

Difficulty: High
Finding ID: TOB-SPRUCE-017

Description

The [zeroize](#) crate provides a simple `no_std` trait interface ([Zeroize](#)) and wrapper type ([`Zeroizing<Z: Zeroize>`](#)) that can automatically zeroize memory when variables are dropped. The zeroizing of memory is a safety measure taken by many cryptographic libraries to prevent secret leakage if an attacker has already compromised the system. The `SSI` library does not zeroize private key material or other sensitive data in memory.

```
#[derive(Debug, Serialize, Deserialize, Clone, PartialEq, Hash, Eq)]
#[serde(tag = "kty")]
pub enum Params {
    EC(ECParams),
    RSA(RSAParams),
    #[serde(rename = "oct")]
    Symmetric(SymmetricParams),
    OKP(OctetParams),
}

#[derive(Debug, Serialize, Deserialize, Clone, PartialEq, Hash, Eq)]
pub struct ECParams {
    // Parameters for Elliptic Curve Public Keys
    #[serde(rename = "crv")]
    pub curve: Option<String>,
    #[serde(rename = "x")]
    pub x_coordinate: Option<Base64urlUInt>,
    #[serde(rename = "y")]
    pub y_coordinate: Option<Base64urlUInt>,

    // Parameters for Elliptic Curve Private Keys
    #[serde(rename = "d")]
    #[serde(skip_serializing_if = "Option::is_none")]
    pub ecc_private_key: Option<Base64urlUInt>,
}
```

Figure 17.1: Private key material is stored as an instance of the `Params` type on a JSON Web Key (JWK) ([`ssi/src/jwk.rs`](#)).

The JWK implementation stores private key material using the `Params` type. Depending on the key type, the private key is stored using one or more `Base64urlUInt` fields. These are not zeroized when the corresponding key is dropped.

Moreover, when new keys are generated by the library, private key material is sometimes leaked to the program stack. This data is not zeroized when the function returns.

```rust
#[cfg(feature = "ring")]
pub fn generate_ed25519() -> Result<JWK, Error> {
    use ring::signature::KeyPair;
    let rng = ring::rand::SystemRandom::new();
    let doc = ring::signature::Ed25519KeyPair::generate_pkcs8(&rng)?;
    let key_pkcs8 = doc.as_ref();
    let keypair = ring::signature::Ed25519KeyPair::from_pkcs8(key_pkcs8)?;
    let public_key = keypair.public_key().as_ref();
    // reference: ring/src/ec/curve25519/ed25519/signing.rs
    let private_key = &key_pkcs8[0x10..0x30];
    Ok(JWK {
        params: Params::OKP(OctetParams {
            curve: "Ed25519".to_string(),
            public_key: Base64urlUInt(public_key.to_vec()),
            private_key: Some(Base64urlUInt(private_key.to_vec())),
        }),
        public_key_use: None,
        key_operations: None,
        algorithm: None,
        key_id: None,
        x509_url: None,
        x509_certificate_chain: None,
        x509_thumbprint_sha1: None,
        x509_thumbprint_sha256: None,
    })
}
```

Figure 17.2: Private key material is leaked to the stack when generating new Ed25519 keys ([ssi/src/jwk.rs]).

```rust
#[cfg(feature = "k256")]
pub fn generate_secp256k1() -> Result<JWK, Error> {
    let mut rng = rand::rngs::OsRng {};
    let secret_key = k256::SecretKey::random(&mut rng);
    let sk_bytes = secret_key.to_bytes();
    let public_key = secret_key.public_key();
    Ok(JWK {
        params: Params::EC(ECParams {
            ecc_private_key: Some(Base64urlUInt(sk_bytes.to_vec())),
            ..ECParams::try_from(&public_key)?
        }),
        public_key_use: None,
        key_operations: None,
        algorithm: None,
        key_id: None,
        x509_url: None,
        x509_certificate_chain: None,
        x509_thumbprint_sha1: None,
```

```
            x509_thumbprint_sha256: None,
        })
    }
```

Figure 17.3: Private key material is leaked to the stack when generating new secp256k1 keys (ssi/src/jwk.rs).

```
    #[cfg(feature = "p256")]
    pub fn generate_p256() -> Result<JWK, Error> {
        let mut rng = rand::rngs::OsRng {};
        let secret_key = p256::SecretKey::random(&mut rng);
        use p256::elliptic_curve::ff::PrimeField;
        let sk_bytes = secret_key.secret_scalar().to_repr();
        let public_key: p256::PublicKey = secret_key.public_key();
        Ok(JWK {
            params: Params::EC(ECParams {
                ecc_private_key: Some(Base64urlUInt(sk_bytes.to_vec())),
                ..ECParams::try_from(&public_key)?
            }),
            public_key_use: None,
            key_operations: None,
            algorithm: None,
            key_id: None,
            x509_url: None,
            x509_certificate_chain: None,
            x509_thumbprint_sha1: None,
            x509_thumbprint_sha256: None,
        })
    }
```

Figure 17.4: Private key material is leaked to the stack when generating new secp256r1 keys (ssi/src/jwk.rs).

The implementation of `JWK::generate_ed25519` based on the `ed25519-dalek` crate is not vulnerable in this way. However, the implementation of the `sign_bytes` function in `ssi/src/jws.rs` is.

Exploit Scenario

An attacker gains elevated privileges that enable her to execute arbitrary code on the system and dump the memory of a running `SSI`-based application. As a result, the attacker can read private key material from application memory.

Recommendations

Short term, implement `Zeroize` and `Drop` on either the `Params` or the `Base64urlUInt` type. Alternatively, wrap both types using the `Zeroizing` type from the `zeroize` crate.

Long term, review the codebase for locations in which private key material is stored on the stack and rewrite the code to prevent this, if possible. Alternatively, consider using the `clear_stack_on_return` function defined by the clear_on_drop crate to clear the stack when the function returns.

18. Credible lacks protection against unauthorized user access

Severity: Medium
Difficulty: High
Type: Access Controls
Finding ID: TOB-SPRUCE-018
Target: `credible/lib/app`

Description
When loaded, the `OnBoardingGenPhrasePage` class and the `RecoveryPage` class both display the mnemonic used to derive the private key stored by the application. If an unauthorized user gains access to a device, she would be able to open the application and read the user's mnemonic, gaining access to the private key.

Exploit Scenario
Eve, a malicious user, steals Bob's device. Since the Credible application does not require any form of user authentication, Eve can open the application and steal Bob's private key.

Recommendations
Short term, use the `local_auth` plugin to integrate local authentication using Face ID or Touch ID (on iOS) and fingerprint APIs (on Android).

19. Credible does not protect sensitive data when switching applications

Severity: Medium
Difficulty: High
Type: Access Controls
Finding ID: TOB-SPRUCE-019
Target: `credible/lib/app`

Description
Credible does not protect sensitive pages from being displayed when switching applications. If the user closes the application with sensitive information (like the recovery phrase) on the screen, this information will be displayed in the application switcher.

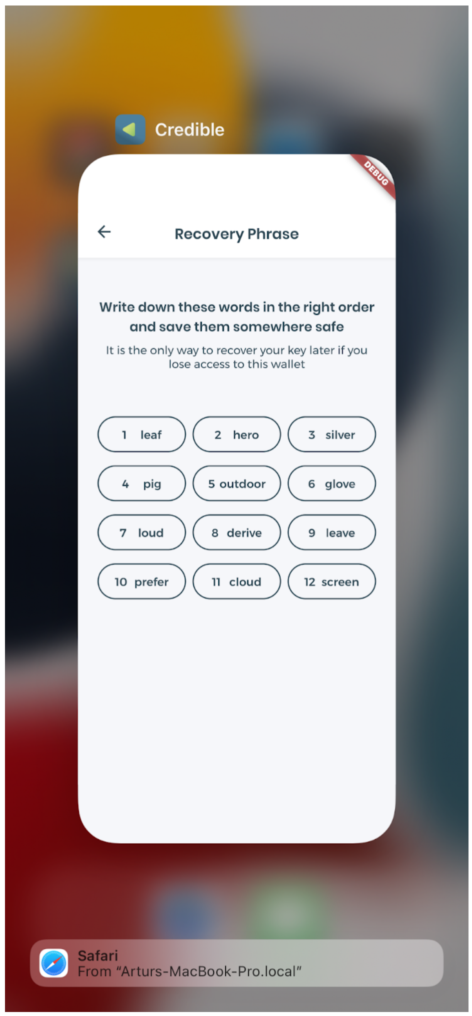

Figure 19.1: A malicious user may be able to read the recovery phrase from the application switcher on the device.

Exploit Scenario

Bob installs Credible and generates a new private key during the onboarding process. He then closes the application with the recovery phrase still on the screen. Malory gains access to Bob’s device and reads the recovery phrase in the application switcher, gaining access to Bob’s private key.

Recommendations

Short term, prevent sensitive pages from being displayed in the application switcher.

20. Consider using IOSAccessibility.passcode to protect keychain items on iOS

Severity: Medium
Difficulty: High
Type: Access Controls
Finding ID: TOB-SPRUCE-020
Target: `credible/lib/app/interop/secure_storage/secure_storage_io.dart`

Description
The `SecureStorageProvider` implementation used on iOS uses the `IOSAccessibility.unlocked_this_device` access level, which means that stored items can be accessed only when the device is unlocked by the user.

```
class SecureStorageIO extends SecureStorageProvider {
  FlutterSecureStorage get _storage => FlutterSecureStorage();

  IOSOptions get _defaultIOSOptions => IOSOptions(
        accessibility: IOSAccessibility.unlocked_this_device,
  );

  // ... <redacted>
}
```

Figure 20.1: Using `IOSAccessibility.unlocked_this_device` means that application data is unprotected if the user has not enabled a passcode on the device.

If the user does not have a passcode set on the device, the key material would be completely unprotected if an attacker gains access to the device.

Exploit Scenario
Malory gains access to Bob's iOS device, which has Credible installed. Since Bob does not use a passcode, Malory can dump the keychain and recover Bob's private key.

Recommendations
Long term, consider using the more restrictive `IOSAccessibility.passcode` access level, which additionally requires the user to have a passcode set on the device. (If the passcode is removed, stored items are deleted from the keychain.)

21. Credible does not validate the length of the recovery phrase

Severity: Informational
Difficulty: N/A
Type: Data Validation
Finding ID: TOB-SPRUCE-021
Target: `credible/lib/app/pages/on_boarding/recovery.dart`

Description

`OnBoardingRecoveryPage` can be used to input a mnemonic BIP39 phrase to recover the corresponding Ed25519 key pair. The given mnemonic is validated by calling `bip39.validateMnemonic`.

This function checks that the input is a valid BIP39 mnemonic seed phrase over the correct dictionary, but it verifies only that the corresponding entropy byte array size is between 16 and 32 bytes, which means that the input may include extra words.

```
  @override
  void initState() {
    super.initState();


    mnemonicController = TextEditingController();
    mnemonicController.addListener(() {
      setState(() {
        edited = mnemonicController.text.isNotEmpty;
        buttonEnabled = bip39.validateMnemonic(mnemonicController.text);
      });
    });

    edited = false;
    buttonEnabled = false;
  }
```

Figure 21.1: The recovery page does not verify that the length of the input is the same length as the mnemonic generated by the application (credible/lib/app/pages/on_boarding/recovery.dart).

Recommendations

Short term, revise the `OnBoardingRecoveryPage` class so that it checks that the recovered entropy string is the same size that is generated by `OnBoardingGenPhrasePage`.

22. The QR code handler is vulnerable to denial-of-service attacks

Severity: Low
Difficulty: Medium
Type: Data Validation
Finding ID: TOB-SPRUCE-022
Target: `credible/lib/app/pages/qr_code/bloc/qrcode.dart`

Description

The `QRCodeBlock._accept` method uses the `Dio get` method to load data from a URL obtained by decoding a QR code. However, no options are passed to `Dio`, so the server could either simply keep the connection alive without sending any data or exhaust the memory resources of the client by returning a very large response.

```
class QRCodeBloc extends Bloc<QRCodeEvent, QRCodeState> {
  final Dio client;
  final ScanBloc scanBloc;

  Stream<QRCodeState> _accept(
    QRCodeEventAccept event,
  ) async* {
    final log = Logger('credible/qrcode/accept');

    late final data;

    try {
      final url = event.uri.toString();
      final response = await client.get(url);
      data =
          response.data is String ? jsonDecode(response.data) : response.data;
    } on DioError catch (e) {
      log.severe('An error occurred while connecting to the server.', e);

      yield QRCodeStateMessage(StateMessage.error(
          'An error occurred while connecting to the server. '
          'Check the logs for more information.'));
    }
    // ... <redacted>
  }
}
```

Figure 22.1: The client does not limit the response time or the size of the response, which leaves the application open to denial-of-service or resource-exhaustion attacks (`credible/lib/app/pages/qr_code/bloc/qrcode.dart`).

Exploit Scenario
Malory convinces Alice to scan a specially crafted QR code that contains the URL for a server controlled by Malory. The server returns a very large response, which causes the Credible application to run out of memory and crash. Each time Alice scans the QR code, the same behavior occurs, causing Alice to lose trust in the application.

Recommendations
Short term, pass a `receiveTimeout` to the `Dio` constructor to limit the time the connection is kept alive. Pass a custom `ProgressCallback` as `onReceiveProgress` to `Dio.get` to check the size of the response.

23. Credible disables TLS certificate verification

Severity: High
Type: Data Validation
Target: `credible/lib/app/app_module.dart`

Difficulty: Low
Finding ID: TOB-SPRUCE-023

Description

Credible disables Transport Layer Security (TLS) certificate verification for the HTTP client used by the application. According to the comment in figure 23.1, this modification most likely exists for development purposes. However, this code is risky, as developers may forget to update it before the application is released.

```
// TODO: Remove this after testing is done
// This allows self-signed certificates on the servers.
final dio = Dio();
if (dio.httpClientAdapter is DefaultHttpClientAdapter) {
  (dio.httpClientAdapter as DefaultHttpClientAdapter)
      .onHttpClientCreate = (HttpClient client) {
    client.badCertificateCallback =
        (X509Certificate cert, String host, int port) => true;
    return client;
  };
}
return dio;
```

Figure 23.1: The Credible application accepts all TLS certificates, which could allow an attacker to tamper with data sent over the network (`credible/lib/app/app_module.dart`).

Exploit Scenario

A Credible user performs an action that involves Tezos DID resolution. An attacker can spoof the TzKT API and present a response that allows him to find an identity that would otherwise be invalid.

Recommendations

Short term, hide the code that modifies the HTTP client behavior behind a debug compilation flag. Alternatively, remove the code and add a self-signed certificate to the certificate store used by Credible during development.

Long term, do not use development code as a default, as developers may forget to remove it before the release of the application.

24. Credible does not prompt users to write down generated mnemonics

Severity: Informational
Difficulty: N/A
Type: Configuration
Finding ID: TOB-SPRUCE-024
Target: `credible/lib/app/pages/on_boarding`

Description
After generating a new mnemonic in Credible, users are not prompted to write it down. Users that do not save their mnemonics will lose them if they lose their phones.

Recommendations
Short term, add a page to Credible that displays after users create new mnemonics to check that they saved the mnemonic.

25. The bip39.generateMnemonic function generates non-uniform entropy

Severity: Low
Type: Cryptography
Target: `bip39/lib/src/bip39_base.dart`

Difficulty: N/A
Finding ID: TOB-SPRUCE-025

Description

The `OnBoardingGenPhrasePage` page uses the `bip39` library to generate BIP39-compatible mnemonics, which are then used by the `OnBoardingGenPage` class to generate Ed25519 key pairs.

Internally, the `bip39` library uses the cryptographically secure pseudorandom number generator (CSPRNG) `Random.secure()` from the `dart:math` library to generate entropy for the mnemonic in `_randomBytes`.

```
Uint8List _randomBytes(int size) {
  final rng = Random.secure();
  final bytes = Uint8List(size);
   for (var i = 0; i < size; i++) {
     bytes[i] = rng.nextInt(_SIZE_BYTE);
   }
  return bytes;
}
```

Figure 25.1: The `_randomBytes` function generates byte values in the range [0, 255), which means that it will never generate the value 255 ([`bip39/lib/src/bip39_base.dart`]).

Here, `_SIZE_BYTE` is defined as 255. Since the [`nextInt`] method generates integers in the half-open interval [0, max) for a given upper bound max, the value 255 will never be generated.

Recommendations

Short term, fork the `bip39` library, fix this issue, and create a pull request on the main repository. Until the pull request has been merged, use your own fork of the library in the application.

26. The Credible iOS client allows third-party keyboards

Severity: High
Difficulty: High
Type: Data Exposure
Finding ID: TOB-SPRUCE-026
Target: `ios/Runner/AppDelegate.swift`

Description

The Credible iOS application does not disable custom keyboards. Third-party keyboards were introduced in iOS 8, allowing users to install custom keyboards that replace the system's default keyboard and can be used in any app.

While custom keyboards are not used when users type into secure text fields (such as password fields), they can log all user keystrokes for regular text fields. Since the `OnBoardingRecoveryPage` recovery page is a regular text field, a custom keyboard could steal the recovery phrase entered by the user.

To disable third-party keyboards on iOS, add the following to the `AppDelegate` implementation in `ios/Runner/AppDelegate.swift`:

```
func application(
    application: UIApplication,
    shouldAllowExtensionPointIdentifier extensionPointIdentifier: String
) -> Bool {
    return extensionPointIdentifier != UIApplicationKeyboardExtensionPointIdentifier
}
```

Figure 26.1: Add this method to `AppDelegate` to disable third-party keyboards (`ios/Runner/AppDelegate.swift`).

Exploit Scenario

An attacker creates and installs a custom keyboard on the device belonging to a Credible user. The keyboard silently exfiltrates the recovery phrase entered by the user.

Recommendations

Short term, disable third-party keyboards within the Credible iOS application.

Long term, stay up to date with changes to iOS that might permit data exfiltration from the client.

27. The Credible Android client allows backups of stored credentials

Severity: Undetermined
Type: Data Exposure
Target: `AndroidManifest.xml`

Difficulty: Low
Finding ID: TOB-SPRUCE-027

Description

Credentials received by the Credible application are stored in the `sembast` database `wallet.db`. This file is copied from the device if a user initiates a backup. Android auto backup also preserves application data by uploading it to the user's Google Drive for apps that target and run on Android 6.0 (API level 23) or higher.

If the credential store contains sensitive data, this data could be leaked when the application's data is backed up.

Exploit Scenario

A malicious user gains control of a Credible user's device and backs up the device using `adb`, gaining control of sensitive credentials stored by the application.

Recommendations

Short term, consider disabling backups by setting `android:allowBackup` to `false` in the application manifest.

28. The checkIsWebsiteLive function checks only for pages served over HTTPS

Severity: Informational
Difficulty: Low
Type: Data Validation
Finding ID: TOB-SPRUCE-028
Target: `tzprofiles/dapp/src/helpers/claims.ts`

Description
As part of the Domain Name System (DNS) verification flow, the `checkIsWebsiteLive` function performs a GET request to check whether a user-provided domain is live. The function attempts to load the corresponding page, but is successful only if the page is served over HTTPS. An error is returned if the page is served only over HTTP.

```
export const checkIsWebsiteLive = async (url: string): Promise<boolean> => {
  try {
    await fetch(`https://${url}`);
    return true;
  } catch (err) {
    console.log(err);
    return false;
  }
};
```

Figure 28.1: The domain name is passed to `checkIsWebsiteLive`. If the site does not implement TLS, an error is returned (`tzprofiles/dapp/src/helpers/claims.ts`).

Recommendations
Short term, revise `checkIsWebsiteLive` so that it falls back to HTTP if HTTPS is not available rather than returning an error.

Long term, consider using a HEAD request and limiting the response time and size to ensure that the `checkIsWebsiteLive` test is fast.

29. The isValidUrl function does not conform to RFC 1034

Severity: Low
Type: Data Validation
Target: `tzprofiles/dapp/src/helpers/dns.ts`

Difficulty: Low
Finding ID: TOB-SPRUCE-029

Description

When a user claims ownership of a domain name, the `isValidUrl` function uses a regular expression to check whether the domain name is valid.

```
export const isValidUrl = (str: string): boolean => {
  var pattern = new RegExp('^(([a-z\\d]([a-z\\d-]*[a-z\\d])*)\\.)+[a-z]{2,}$');
  return !!pattern.test(str);
};
```

Figure 29.1: The `isValidUrl` function uses a regular expression to validate domain names (`tzprofiles/dapp/src/helpers/dns.ts`).

However, the regular expression does not match the domain name format specified by section 3.5 in RFC 1034. In particular, the RFC does not accept sub-domain labels beginning with a digit, which are accepted by `isValidUrl`. On the other hand, `isValidUrl` does not accept uppercase characters, which are accepted by the RFC.

```
> var pattern = new RegExp("^(([a-z\\d]([a-z\\d-]*[a-z\\d])*)\\.)+[a-z]{2,}$");
undefined
> pattern.test("123.456.com")  // Invalid according to RFC 1034.
true
> pattern.test("GOOGLE.COM")   // Valid according to RFC 1034.
false
```

Figure 29.2: The regular expression used in `isValidUrl` does not conform to the format specified in RFC 1034.

Recommendations

Short term, rewrite the regular expression used to validate domain names to conform with the format specified in RFC 1034.

30. Tezos Profiles uses vulnerable dependencies

Severity: High
Difficulty: High
Type: Patching
Finding ID: TOB-SPRUCE-030
Target: `tzprofiles/worker/Cargo.lock`

Description
The Tezos Profiles worker uses the following vulnerable Rust dependencies:

Dependency	Version	ID	Description
`chrono`	0.4.19	RUSTSEC-2020-0159	Potential segfault in `localtime_r` invocations
`hyper`	0.14.8	RUSTSEC-2021-0079	Integer overflow in `hyper`'s parsing of the `Transfer-Encoding` header leads to data loss
		RUSTSEC-2021-0078	Lenient `hyper` header parsing of `Content-Length` could allow request smuggling
`time`	0.1.43	RUSTSEC-2020-0071	Potential segfault in the `time` crate
`tokio`	1.6.1	RUSTSEC-2021-0072	Task dropped in wrong thread when aborting `LocalSet` task
`zerioze_derive`	1.1.0	RUSTSEC-2021-0115	`#[zeroize(drop)]` doesn't implement `Drop` for enums

Other than `chrono`, all the dependencies can simply be updated to their newer versions to fix the vulnerabilities. The `chrono` crate issue has not been mitigated and remains problematic. A specific sequence of calls must occur to trigger the vulnerability, which is discussed in this GitHub thread in the `chrono` repository.

A number of `npm` dependencies are vulnerable under `dapp`, `api/service`, and `contracts`. Run `npm audit` for further guidance on how to fix these vulnerabilities.

Exploit Scenario
An attacker exploits a known vulnerability in Tezos Profiles and performs a denial-of-service attack by taking down the worker.

Recommendations
Short term, update the Tezos Profiles dependencies to their newest versions. Monitor the referenced GitHub thread regarding the `chrono` crate segfault issue.

Long term, run `cargo-audit` and `npm audit` as part of the CI/CD pipeline and ensure that the team is alerted to any vulnerable dependencies that are detected.

31. The Tezos Profiles worker is vulnerable to URL injection attacks

Severity: High
Type: Data Validation
Target: `tzprofiles/worker/src`

Difficulty: Medium
Finding ID: TOB-SPRUCE-031

Description

The Tezos Profiles worker calls third-party APIs to witness attestations. Calls to GitHub, Cloudflare, and Discord can be partially controlled by user input due to unsafe URL construction (figures 31.1–31.3). There is no data validation for the incoming URL parameters to the worker's HTTP API (figure 31.4). It is possible to perform path traversal attacks on calls to GitHub and Discord and URL parameter injection attacks on calls to GitHub, Discord, and Cloudflare.

Changing the host is not possible, so attacks must be conducted within the original host.

```
pub async fn retrieve_gist_message(gist_id: String) -> Result<GitHubResponse> {
    let client = reqwest::Client::new();
    let request_url = format!("https://api.github.com/gists/{}", gist_id);
    …
}
```

Figure 31.1: tzprofiles/worker/src/github.rs#L33-L49

```
pub async fn retrieve_txt_records(domain: String) -> Result<DnsResponse> {
    let client = reqwest::Client::new();
    let request_url = format!(
        "https://cloudflare-dns.com/dns-query?name={}&type=txt&ct=application/dns-json",
        domain
    );
    …
}
```

Figure 31.2: tzprofiles/worker/src/dns.rs#L48-L63

```
pub async fn retrieve_discord_message(
    discord_authorization_key: String,
    channel_id: String,
    message_id: String,
) -> Result<DiscordResponse> {
    …
    let client = reqwest::Client::new();
    let request_url = format!(
        "https://discord.com/api/channels/{}/messages/{}",
        channel_id, message_id
    );
    …
}
```

Figure 31.3: tzprofiles/worker/src/discord.rs#L27-L52

```
async function handle_github_lookup(request) {
  try {
    const { gist_lookup } = wasm_bindgen;
    const { searchParams } = new URL(request.url);

    const pk = decodeURIComponent(searchParams.get("pk"));
    const gistId = decodeURIComponent(searchParams.get("gistId"));
    const githubUsername = decodeURIComponent(searchParams.get("handle"));

    await wasm_bindgen(wasm);
    const vc = await gist_lookup(TZPROFILES_ME_PRIVATE_KEY, pk, gistId, githubUsername);

    return new Response(vc, {
      status: 200,
      headers: headers,
    });
  } catch (error) {
    return new Response(error, { status: 500, headers: headers });
  }
}
```

Figure 31.4: tzprofiles/worker/worker/worker.js#L227-L246

Exploit Scenario
An attacker performs an attack on a GitHub attestation by pointing the GitHub API call to a different URL, which returns an attacker-controlled payload with the same structure as the gist call. The attacker can now pose as any GitHub user.

Recommendations
Short term, avoid constructing URLs directly based on user input. Always validate all data that comes from untrusted sources.

32. The Tezos Profiles Instagram attestation is broken

Severity: Informational
Difficulty: High
Type: Data Validation
Finding ID: TOB-SPRUCE-032
Target: `tzprofiles/worker/src`

Description

The Tezos Profiles worker contains code for attesting Instagram profiles. However, the code is broken in two places. The first issue is a typo in the function name (figure 32.1): the function should be named `handle_tzp_instagram_login` rather than `handle_instagram_tzp_login`, so the endpoint crashes with a 500 error. The second issue is a bug in the code. The `handle_tzp_instagram_login` function calls `handle_login_flow` with `instagram::LoginFlow::TZP`; however, the `TZP` branch is unreachable, so the `handle_login_flow` function will work as if `LoginFlow::DEMO` was passed. The code will always match the first pattern because `DEMO` is a catch-all pattern. The patterns have to be `LoginFlow::DEMO` and `LoginFlow::TZP` to match the enum variants (figure 32.2). The Rust compiler warns about this issue.

As noted by the Spruce team, the Instagram attestation is incomplete and will likely be removed in the future.

```
async function handler_instagram_login(request) {
  try {
    const { searchParams } = new URL(request.url);

    const code = searchParams.get("code");

    const { handle_instagram_tzp_login } = wasm_bindgen;
    …
}
```

Figure 32.1: tzprofiles/worker/worker/worker.js#L285-L326

```
pub enum LoginFlow {
    DEMO,
    TZP,
}

pub async fn handle_login_flow(
    …
    flow: LoginFlow,
) -> Result<KVWrapper> {
        …
        let (sig, permalink) = match flow {
         // DEMO is a catch-all pattern
         DEMO => retrieve_demo_post(&user, &access_token).await?,
         TZP => retrieve_tzp_post(&user, &access_token).await? // Unreachable
        };
        …
}
```

Figure 32.2: tzprofiles/worker/src/instagram.rs#L100-L128

Recommendations
Short term, remove the Instagram attestation or fix the bugs in the code to restore the witness functionality.

Long term, pay attention to compiler warnings, as the catch-all pattern issue could have been detected before the application's release. Configure a CI/CD pipeline to alert the team when compiler warnings are raised.

33. The Tezos Profiles web app is vulnerable to URL injection attacks

Severity: High
Type: Data Validation
Target: `tzprofiles/dapp`

Difficulty: Low
Finding ID: TOB-SPRUCE-033

Description

The Tezos Profiles `view` endpoint does not sanitize the `network` or `address` path components. This means that a malicious user could control the `TzKT` URL used to obtain a user's `tzprofiles` contract. An attacker could craft a malicious `tzprofiles.com` URL to present a `tzprofiles.com` page containing her address with the credentials of a different user.

When a user requests the `tzprofiles.com/view/${network}/${address}` page, this updates the `network` and triggers `network.subscribe`, which updates the `TzKT` base URL, `tzktBase`.

```
network.subscribe((network) => {
  if (network === NetworkType.CUSTOM) {
    // ... <redacted>
  } else {
    networkStr.set(network);
    // TODO can't read from writeable, but then I don't understand why others work.
    networkStrTemp = network;
    strNetwork = network;

    urlNode = `https://${network}.smartpy.io/`;
    nodeUrl.set(urlNode);

    tzktBaseTemp = `https://api.${networkStrTemp}.tzkt.io`;
    tzktBase.set(tzktBaseTemp);
  }
});
```

Figure 33.1: The `tzktBase` variable is set to `https://api.${network}.tzkt.io` in `network.subscribe` (`tzprofiles/dapp/src/store.ts`).

The `tzktBase` variable is then passed to a `ContractClient` instance and is used to fetch the `tzprofiles` contract for the user. This is done in `ContractClient.retrieveAllContracts`. (The call to `this.tsktPrefix()` below simply returns the string `${tzktBase}/v1/`.)

```
private async retrieveAllContracts(offset: number, walletAddress: string):
        Promise<Array<string>> {
    let pageLimit = 100;
    let prefix = this.tzktPrefix();
    let searchRes = await axios.get(`${prefix}contracts?
        creator=${walletAddress}&
```

```
            offset=${offset}&
            limit=${pageLimit}&
            sort=firstActivity&
            select=address`);
        if (searchRes.status !== 200) {
            throw new Error(`Failed in explorer request: ${searchRes.statusText}`);
        }
        if (!searchRes.data || searchRes.data.length === 0) {
            return [];
        }
        let {data} = searchRes;
        let pageCount = data.length;
        if (pageCount == pageLimit) {
            return data.concat(
                await this.retrieveAllContracts(offset + pageCount, walletAddress));
        }
        return data;
    }
```

Figure 33.2: The user-controlled `network` variable is used to construct the URL for the `TzKT` API (`tzprofiles/contract/lib/contractClient.ts`).

When `axios.get` is called, the Unicode input is normalized before the domain name is resolved. This means that a `network` parameter like `evil.pizz℀` results in a request to `https://api.evil.pizza/c.tzkt.io` in `ContractClient.retrieveAllContracts`, since the Unicode character "℀" is normalized to "a/c."

Exploit Scenario

Mallory, a malicious Tezos Profiles user, controls the domain `api.evil.pizza`. To impersonate Bob, another Tezos Profiles user, on `tzprofiles.com`, she gives the following link to Alice:

```
tzprofiles.com/evil.pizz℀/<Malory's address>
```

When Alice clicks this link in her browser, the web application attempts to obtain Malory's `tzprofiles` contract address by making a request to the following:

```
https://api.evil.pizza/c.tzkt.io/v1/contracts?
       creator=<Malory's address>&
       offset=${offset}&
       limit=${pageLimit}&
       sort=firstActivity&
       select=address
```

Since Malory controls `api.evil.pizza`, she can return Bob's `tzprofiles` contract address. When `ContractClient.retrieveClaims` is called with Bob's contract address, Bob's claims are returned and displayed to Alice with Malory's address.

We note that the Tezos node endpoint is also potentially controlled by Malory since it is constructed from the network parameter as `https://${network}.smartpy.io/`.

Recommendations
Short term, sanitize all user input by normalizing it and ensuring that it contains only valid characters. (In this case, this would require checking that the network name contains only alphanumeric characters.)

Long term, provide links to user attestations in the user profile on `tzprofiles.com` to allow anyone to verify that the attestations are correct. This would also allow users to check *when* a claim was made, which is often useful since identity claims may not be valid forever. (For example, a user may change his or her username, and another user may set up a new account with the original username.)

34. The Tezos Profiles API is vulnerable to URL parameter injection attacks

Severity: Undetermined
Type: Data Validation
Target: `tzprofiles/api/service`

Difficulty: Low
Finding ID: TOB-SPRUCE-034

Description

The `tzprofiles` API does not sanitize the request path or parameters. As a result, attackers can redirect requests from the API back end to the `TzKT` API to an arbitrary host.

```
❯ curl https://api.tzprofiles.com/address/evil.pizza%2fpath | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   627  100   627    0     0   1953      0 --:--:-- --:--:-- --:--:--  1953
{
  "message": "getaddrinfo ENOTFOUND api.evil.pizza",
  "name": "Error",
  "stack": "Error: getaddrinfo ENOTFOUND api.evil.pizza
    at GetAddrInfoReqWrap.onlookup [as oncomplete] (node:dns:69:26)",
  "config": {
    "url": "https://api.evil.pizza/path.tzkt.io/v1/contracts?
      creator=address&offset=0&limit=100&sort=firstActivity&select=address",
    "method": "get",
    "headers": {
      "Accept": "application/json, text/plain, */*",
      "User-Agent": "axios/0.21.1"
    },
    "transformRequest": [
      null
    ],
    "transformResponse": [
      null
    ],
    "timeout": 0,
    "xsrfCookieName": "XSRF-TOKEN",
    "xsrfHeaderName": "X-XSRF-TOKEN",
    "maxContentLength": -1,
    "maxBodyLength": -1
  },
  "code": "ENOTFOUND"
}
```

Figure 34.1: Since the Tezos Profiles API does not sanitize the input parameters, a malicious user can redirect requests from the back end to arbitrary hosts (`tzprofiles/api/service/index.js`).

Since the indexer is used only for mainnet requests, it is not clear whether this issue is currently exploitable.

Recommendations
Short term, sanitize all user input by normalizing it and ensuring that it contains only valid characters. (In this case, this would require checking that the network name contains only alphanumeric characters.)

35. The Tezos Profiles API reports internal errors to users

Severity: Low
Difficulty: Low
Type: Error Reporting
Finding ID: TOB-SPRUCE-035
Target: `tzprofiles/api/service`

Description

The Tezos Profiles API returns internal error messages to users.

```
❯ curl https://api.tzprofiles.com/foo | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   956  100   956    0     0  1975      0 --:--:-- --:--:-- --:--:--  1971
{
  "message": "Request failed with status code 400",
  "name": "Error",
  "stack": "Error: Request failed with status code 400
    at createError (/axios/lib/core/createError.js:16:15)
    at settle (/axios/lib/core/settle.js:17:12)
    at IncomingMessage.handleStreamEnd (/axios/lib/adapters/http.js:260:11)
    at IncomingMessage.emit (node:events:381:22)
    at endReadableNT (node:internal/streams/readable:1307:12)
    at processTicksAndRejections (node:internal/process/task_queues:81:21)",
  "config": {
    "url": "https://api.mainnet.tzkt.io/v1/contracts?
      creator=foo&offset=0&limit=100&sort=firstActivity&select=address",
    "method": "get",
    "headers": {
      "Accept": "application/json, text/plain, */*",
      "User-Agent": "axios/0.21.1"
    },
    "transformRequest": [
      null
    ],
    "transformResponse": [
      null
    ],
    "timeout": 0,
    "xsrfCookieName": "XSRF-TOKEN",
    "xsrfHeaderName": "X-XSRF-TOKEN",
    "maxContentLength": -1,
    "maxBodyLength": -1
  }
}
```

Figure 35.1: Leaking internal error messages to users may make it easier to compromise an application (`tzprofiles/api/service`).

Reporting internal messages to users could leak sensitive data about the internal server state, which could be used to compromise the application.

Exploit Scenario
A malicious user learns of a vulnerability in the Tezos Profiles API from a verbose error message. She uses the vulnerability to compromise the application.

Recommendations
Short term, handle errors internally and do not report error messages from third-party frameworks to users.

36. Too few confirmations required when deploying new smart contracts

Severity: Informational
Difficulty: N/A
Type: Configuration
Finding ID: TOB-SPRUCE-036
Target: `tzprofiles/contract/lib/contractClient.ts`

Description

When the `tzprofiles` smart contract is created, the `ContractClient` waits for only three `CONFIRMATION_CHECKS` confirmations before returning the contract address to the caller. Tezos recommends waiting for at least seven confirmations to ensure finality. Since a healthy chain bakes a new block every minute, the wait time would be seven minutes.

```
// Magic Number controlling how long to wait before confirming success.
// Seems to be an art more than a science, 3 was suggested by a help thread.
const CONFIRMATION_CHECKS = 3;

// ... <redacted>

if (this.signer.type === "wallet") {
    let opSender = yield this.tezos.wallet.originate(args);
    originationOp = yield opSender.send();
    let c = yield originationOp.contract();
    contractAddress = c.address;
}
else {
    originationOp = yield this.tezos.contract.originate(args);
    yield originationOp.confirmation(CONFIRMATION_CHECKS);
    contractAddress = originationOp.contractAddress;
}
return contractAddress;
```

Figure 36.1: The contract client waits for only three confirmations before returning the address of the newly created contract (`tzprofiles/contract/lib/contractClient.ts`).

(The Tezos documentation actually recommends waiting for six confirmations, but this number is based on Nomadic Labs' analysis of Emmy+, which has since been updated; Nomadic Labs now recommends seven confirmations for a scenario with a 20% attacker controller stake.)

Recommendations

Short term, consider the effect of raising the number of confirmations required for newly created contracts on the system.

Long term, consider raising the number of confirmations required for newly created contracts to a number that better aligns with the Tezos recommendation.

37. Tezos Profiles does not validate the credential subject or issuer

Severity: High
Difficulty: Medium
Type: Data Validation
Finding ID: TOB-SPRUCE-037
Target: `tzprofiles/contract/lib/contractClient.ts`

Description

When displaying a user's profile on `tzprofiles.com`, the application does not validate the credential subject against the user's address or the credential issuer against the DID `did:web:tzprofiles.com`.

When viewing a user's credentials on `tzprofiles.com`, the `ContractClient` first retrieves the `tzprofiles` smart contract and obtains the list of claims stored by the contract. For each claim, the corresponding verifiable credential is retrieved from Kepler. The credential is verified using DIDKit, and the credential hash is checked against the hash stored on-chain by the contract. If all checks pass, the credential is displayed as valid to the user.

```
  private async processContentList(contentList: ContentList<ContentType, Hash, Reference>):
      Promise<
          [
              Array<InvalidContent<ContentType, Hash, Reference>>,
              Array<ValidContent<Content, ContentType, Reference>>
          ]> {
      let invalid: Array<InvalidContent<ContentType, Hash, Reference>> = [];
      let valid: Array<ValidContent<Content, ContentType, Reference>> = [];
      for (let i = 0, n = contentList.length; i < n; i++) {
          let [reference, hash, contentType] = contentList[i];
          let content: Content;

          try {
              // Obtains the verifiable credential from Kepler.
              content = await this.dereferenceContent(reference);
          } catch (err) {
              invalid.push([reference, hash, contentType, null, err]);
              continue;
          }

          try {
              // Verifies the credential and credential type.
              await this.validateType(content, contentType);
          } catch (err) {
              invalid.push([reference, hash, contentType, content, err]);
              continue;
          }

          try {
              // Checks the hash against the hash stored on-chain.
              let h = await this.hashContent(content);
```

```
            if (h !== hash) throw new Error("Hashes do not match");
        } catch (err) {
            invalid.push([reference, hash, contentType, content, err]);
            continue;
        }
        // If all three checks pass, the credential is considered valid.
        valid.push([reference, content, contentType]);
    }

    return [invalid, valid];
}
```

Figure 37.1: Nothing stops malicious users from updating their smart contracts with references to other users' attestations (tzprofiles/contract/lib/contractClient.ts).

Since the credential subject is not checked against the account associated with the displayed page, a malicious user could impersonate other users by updating her `tzprofiles` contract with Kepler references to credentials belonging to other users. Since the credential issuer is not checked against the fixed DID `did:web:tzprofiles.com`, a malicious user could upload her own self-signed credentials to Kepler and add the corresponding references to her `tzprofiles` contract, allowing her to make arbitrary identity claims.

Exploit Scenario 1
Malory updates her `tzprofiles` smart contract with Kepler references belonging to another `tzprofiles.com` user, Bob. The verifiable credential is still tied to Bob, but since the page does not verify the credential subject or display the credential, Malory could trick unsuspecting user Alice into thinking that the identity attested to by the credential belongs to Malory.

Exploit Scenario 2
Malory signs a verifiable credential attesting that she controls Bob's Twitter handle and uploads it to Kepler. She updates her `tzprofiles` smart contract with the corresponding Kepler references. The claim made by her self-signed credential is included on her `tzprofiles.com` profile. Alice navigates to Malory's profile and now believes that Malory owns Bob's Twitter handle.

Recommendations
Short term, fix the Tezos Profiles application so that, when displaying identity claims on `tzprofiles.com`, it verifies that the credential subject is equal to the user account associated with the page. Additionally, ensure that the application checks that the credential issuer is given by `did:web:tzprofiles.com`.

Long term, consider allowing `tzprofiles.com` users to view or save the actual verifiable credentials from Kepler for extra security.

38. The Tezos Profiles contract code is duplicated

Severity: Informational
Difficulty: High
Type: Patching
Finding ID: TOB-SPRUCE-038
Target: `tzprofiles/contract/lib/contract.ts`

Description
The Tezos Profiles contract is written in the ReLIGO language, which is compiled to the Michelson language. The Michelson version is duplicated in two places: in a file in the same directory as the ReLIGO version and as a string in the TypeScript code under `tzprofiles/contract/lib/contract.ts`.

Exploit Scenario
A Spruce developer updates the ReLIGO code but forgets to update the Michelson code.

Recommendations
Short term, do not check the Michelson code into the repository. Produce the Michelson version from the ReLIGO version at build time and use it as a build artifact in TypeScript. Lock the LIGO compiler version and use a hash of Michelson code to detect any differences should they arise after compilation.

Long term, avoid duplicating code, as it is harder to patch.

39. The Tezos Profiles worker insufficiently validates data coming from third-party APIs

Severity: High
Difficulty: High
Type: Data Validation
Finding ID: TOB-SPRUCE-039
Target: `tzprofiles/worker/src`

Description

The Tezos Profiles worker pulls data from third-party APIs. In some places, the worker does not sufficiently validate the data, which can result in a crash. The DNS witness uses the Cloudflare API, which returns TXT records for a requested domain. If the TXT data contains a single `"` sign, the code in figure 39.1 will assume that `trimmed_signature` has at least one `"` sign at the beginning and at the end, causing the code to panic with the error `begin <= end (1 <= 0) when slicing`. At the moment, this assumption is correct: the Cloudflare API wraps the raw user-controlled TXT data with a pair of `"` signs, so the crash never occurs. This Cloudflare API behavior seems to be undocumented; furthermore, other providers might not behave in the same way.

```
pub fn find_signature_to_resolve(dns_result: DnsResponse) -> Result<String> {
    for answer in dns_result.answer {
        let mut trimmed_signature: &str = &answer.data;
        if trimmed_signature.starts_with('"') && trimmed_signature.ends_with('"') {
            trimmed_signature = &answer.data[1..answer.data.len() - 1];
        }
        if trimmed_signature.starts_with("tzprofiles-verification") {
            return Ok(trimmed_signature.to_string());
        }
    }

    return Err(anyhow!("Signature not found"));
}
```

Figure 39.1: [tzprofiles/worker/src/dns.rs#L22-L34]()

A similar case occurs when the worker processes data returned by the Twitter API; however, this time, the payload is not user controlled. The code does not check the `users` and `data` arrays' sizes and assumes that there will always be at least one element (figure 39.2). While this assumption is correct when the Twitter API functions normally, the worker may panic when a malicious or buggy payload is returned.

```
if twitter_handle.to_lowercase() != twitter_res.includes.users[0].username.to_lowercase() {
    jserr!(Err(anyhow!(format!(
        "Different twitter handle {} v. {}",
        twitter_handle.to_lowercase(),
        twitter_res.includes.users[0].username.to_lowercase()
    ))));
}

let (sig_target, sig) = jserr!(extract_signature(twitter_res.data[0].text.clone()));
```

Figure 39.2: [tzprofiles/worker/src/lib.rs#L197-L205]()

Exploit Scenario
The Cloudflare API changes its definition in such a way that it does not wrap the TXT data with additional quotes. An attacker sets up a TXT record with a single `"` sign as data, and the Tezos Profiles worker validates the domain, causing the worker to crash.

Recommendations
Short term, fix the Tezos Profiles worker so that it validates the data fetched from the Twitter and Cloudflare APIs. This will make the Tezos Profiles service more resilient.

Long term, ensure that data coming from third-party services is always validated.

40. Kepler logs authentication data

Severity: Informational
Type: Data Exposure
Target: `kepler/src/auth.rs`

Difficulty: High
Finding ID: TOB-SPRUCE-040

Description
The Kepler authentication process logs authentication data coming into the `Authorization` HTTP header from users (figure 40.1). At the moment, the data does not appear to contain session keys or secrets, as the tokens are verified based on a signature. In the event of a log data leak, the API could become vulnerable if it is not sufficiently protected from replay attacks, or if the authentication process supports a form of session keys in the future.

```
async fn extract_info<T>(
    req: &Request<'_>,
) -> Result<(Vec<u8>, AuthTokens, config::Config, Cid), Outcome<T, anyhow::Error>> {
    // TODO need to identify auth method from the headers
    let auth_data = match req.headers().get_one("Authorization") {
        Some(a) => a,
        None => "",
    };
    info_!("Headers: {}", auth_data);
    …
}
```

Figure 40.1: [kepler/src/auth.rs#L107-L115]

Recommendations
Short term, remove the logging mechanism or change it to a debug level and ensure that it is not executed in production builds.

Long term, do not log any sensitive data in production. The logs might be shipped to a third-party service for analysis, making them susceptible to a data leak.

41. Kepler uses vulnerable Rust dependencies

Severity: High
Type: Patching
Target: `kepler/src/auth.rs`

Difficulty: High
Finding ID: TOB-SPRUCE-041

Description
Kepler uses the following vulnerable dependencies:

Dependency	Version	ID	Description
`chrono`	0.4.19	RUSTSEC-2020-0159	Potential segfault in `localtime_r` invocations
`libsecp256k1`	0.3.5	RUSTSEC-2021-0076	`libsecp256k1` allows overflowing signatures
`prost-types`	0.7.0	RUSTSEC-2021-0073	Conversion from `prost_types::Timestamp` to `SystemTime` can cause an overflow and panic
`time`	0.1.43	RUSTSEC-2020-0071	Potential segfault in the `time` crate
`zerioze_derive`	1.1.0	RUSTSEC-2021-0115	`#[zeroize(drop)]` doesn't implement `Drop` for enums

Other than `chrono`, all the dependencies can simply be updated to their newer versions to fix the vulnerabilities. The `chrono` crate issue has not been mitigated and remains problematic. A specific sequence of calls must occur to trigger the vulnerability, which is discussed in this GitHub thread in the `chrono` repository.

Exploit Scenario
An attacker exploits a known vulnerability in Kepler and performs a denial-of-service attack by taking down the worker.

Recommendations
Short term, update the Kepler dependencies to their newest versions. Monitor the referenced GitHub thread regarding the `chrono` crate segfault issue.

Long term, run `cargo-audit` as part of the CI/CD pipeline and ensure that the team is alerted to any vulnerable dependencies that are detected.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
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 protecting the privacy or integrity of 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 a 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
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose

	reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Cryptography	Related to the proper use of cryptographic primitives and protocols
Data Exposure	Related to the protection of sensitive data
Data Validation	Related to the validation of untrusted input
Dependency Management	Related to the existence of clear processes for dependency management
Error Handling	Related to the handling of edge cases and error states
Function Composition	Related to separation of the logic into functions with clear purposes
Specification	Related to the expected codebase documentation
Testing and Verification	Related to the use of testing techniques (unit tests, property testing, fuzzing, symbolic execution, etc.)

C. Code Quality Recommendations

This appendix contains findings that do not have any immediate or obvious security implications. We discovered most of these issues by running Clippy, Dylint, `cargo-audit`, and Semgrep rules on the codebase.

SSI

- Running Clippy on the codebase produces a large number of code quality findings. These should be addressed.
- Unsafe code blocks (line 91 in `src/bbs.rs`) should be prefixed by a comment starting with the keyword `SAFETY`, indicating why the code is considered safe.
- The assignment `passed_seed = true;` (line 229 in `src/bbs.rs`) is redundant.
- There are 128 to-do comments throughout the codebase. These should be addressed.
- `DIDURL` implements `TryFrom<String>` (line 649 in `src/did.rs`) but then passes the string by reference to `DIDURL::from_str` to create a new `DIDURL`. This may require the caller to create a copy of the input, which would then immediately be discarded.
- The `SSI` library depends on two crates, `difference` and `failure`, which are both unmaintained according to `cargo-audit`.
- The `curve` field in `ECParams` and `OctetParams` (in `src/jwk.rs`) should be implemented as an enum for improved type safety.
- The `SSI` crate does not compile with all features enabled. This should be fixed.

DIDKit

- Running Clippy on the codebase produces a large number of code quality findings. These should be addressed.
- Avoid using `unwrap` (as in `Ok(env.new_string(...).unwrap().into_inner()))` when returning a result to JNI (in `lib/src/jni.rs`). Since the function already returns a `Result`, use the `?` operator.

Tezos Profiles

- Most TypeScript variables in the web app under `dapp` are declared using `let`. Since many of these are never reassigned, consider using `const` instead.
- The string on line 468 in `dapp/src/store.ts` does not require escaped quotation marks.
- Domain name validation (using `isValidUrl`) should be performed before calling `checkIsWebsiteLive` during DNS verification step 1 (in `dapp/src/routes/DnsVerification/DnsVerification.svelte`).

- The `dapp/src/store.ts` file uses five different variables—`network`, `networkStr`, `strNetwork`, `networkStrTemp`, and `localNetworkStr`—to track which Tezos network is used. This is error-prone and should be refactored.
- The codebase contains a large number of to-do comments. These should be addressed.
- The Tezos Profiles `dapp` is missing `package-lock.json`.

Kepler

- Running Clippy on the codebase produces a large number of code quality findings. These should be addressed.

D. Fix Log

After the initial assessment, the Trail of Bits audit team reviewed each fix to ensure that the underlying issue was correctly addressed.

#	Title	Severity	Status
1	The Tezos DID resolver accepts invalid input that can crash the program	High	Fixed
2	The program can crash when parsing EIP-712 types	High	Fixed
3	Potentially unsafe dependency on the internal representation of types from the bbs crate	Informational	Partially fixed
4	Potential panic when creating a new BLS key pair	Informational	Fixed
5	Tezos DID resolver does not take the network into account	Undetermined	Fixed
6	PassthroughDigest reduces the entropy of the output for digests that are not 32 bytes	Informational	Partially fixed
7	HTTPS is not enforced when loading a revocation list	Medium	Fixed
8	Potential resource exhaustion when loading a revocation list	Medium	Fixed
9	JWT encoding may produce an invalid credential	Undetermined	Fixed
10	Issuer that is used for testing is exposed in release builds	Informational	Fixed
11	DIDKit CLI option to change tzkt_url is not documented	Informational	Partially fixed
12	Smart contract address–based Tezos DIDs are not implemented	Informational	Fixed
13	The DID manager resolution process supports only the default TZIP-19 contract	Informational	Not fixed

14	Verifiable credentials with invalid revocation list indices are accepted by default	Informational	Fixed
15	DIDKit HTTP server is vulnerable to slowloris attacks	Undetermined	Fixed
16	DIDKit HTTP server is vulnerable to memory resource exhaustion	Undetermined	Fixed
17	Private key material is not cleared from memory when no longer needed	Medium	Fixed
18	Credible lacks protection against unauthorized user access	Medium	Fixed
19	Credible does not protect sensitive data when switching applications	Medium	Fixed
20	Consider using IOSAccessibility.passcode to protect keychain items on iOS	Medium	Fixed
21	Credible does not validate the length of the recovery phrase	Informational	Fixed
22	The QR code handler is vulnerable to denial-of-service attacks	Low	Fixed
23	Credible disables TLS certificate verification	High	Fixed
24	Credible does not prompt users to write down generated mnemonics	Informational	Fixed
25	The bip39.generateMnemonic function generates non-uniform entropy	Low	Fixed
26	The Credible iOS client allows third-party keyboards	High	Fixed
27	The Credible Android client allows backups of stored credentials	Undetermined	Fixed
28	The checkIsWebsiteLive function checks only for pages served over HTTPS	Informational	Fixed
29	The isValidUrl function does not conform to RFC 1034	Low	Fixed
30	Tezos Profiles uses vulnerable dependencies	High	Risk accepted

31	The Tezos Profiles worker is vulnerable to URL injection attacks	High	Fixed
32	The Tezos Profiles Instagram attestation is broken	Informational	Fixed
33	The Tezos Profiles web app is vulnerable to URL injection attacks	High	Fixed
34	The Tezos Profiles API is vulnerable to URL parameter injection attacks	Undetermined	Fixed
35	The Tezos Profiles API reports internal errors to users	Low	Fixed
36	Too few confirmations required when deploying new smart contracts	Informational	Risk accepted
37	Tezos Profiles does not validate the credential subject or issuer	High	Fixed
38	The Tezos Profiles contract code is duplicated	Informational	Not fixed
39	The Tezos Profiles worker insufficiently validates data coming from third-party APIs	High	Fixed
40	Kepler logs authentication data	Informational	Fixed
41	Kepler uses vulnerable Rust dependencies	High	Partially fixed

Detailed Fix Log

Finding 1: The Tezos DID resolver accepts invalid input that can crash the program
Fixed. The implementation of resolve for DIDTz now uses String::get to obtain the prefix of the address.

Finding 2: The program can crash when parsing EIP-712 types
Fixed. The implementation of try_from::<String> for EIP712Type now returns a TypedDataParseError::UnmatchedBracket error if the input string does not contain the character [.

Finding 3: Potentially unsafe dependency on the internal representation of types from the bbs crate
Partially fixed. The implementation of From<SecretKey> for BlsSecretKey still implicitly depends on the internal representation of the SecretKey type defined by the bbs crate. However, the implementation now contains a comment detailing the property that needs to be upheld, reducing the risk posed by the unsafe block.

Finding 4: Potential panic when creating a new BLS key pair
Fixed. The implementation of bls_generate_keypair now returns a BlsGenerateKeyPairError::DeserializeBlinder error if it fails to deserialize the optional blinding value.

Finding 5: Tezos DID resolver does not take the network into account
Fixed. The implementation of resolve for DIDTz now uses the format macro to build the URL from the network.

Finding 6: PassthroughDigest reduces the entropy of the output for digests that are not 32 bytes
Partially fixed. The Spruce team has reimplemented the ProofSuite trait for the EthereumPersonalSignature2021 type to avoid using PassthroughDigest. However, the PassthroughDigest type is still used for JSON web signature generation and verification in ssi/src/jws.rs, and calling the PassthroughDigest::update method still results in a low-entropy value for inputs that are not exactly 32 bytes long. According to the Spruce team, this type will be used only to generate Tezos signatures.

Finding 7: HTTPS is not enforced when loading a revocation list
Fixed. The implementation of CredentialStatus::check for RevocationList2020Status now checks that the revocation list is loaded over HTTPS.

Finding 8: Potential resource exhaustion when loading a revocation list

Fixed. The `load_resource` function now reads the revocation list payload in chunks and returns an error if the payload is too large.

Finding 9: JWT encoding may produce an invalid credential
Fixed. Verifiable credentials with `credentialSubject` arrays containing a single element are now encoded correctly.

Finding 10: Issuer that is used for testing is exposed in release builds
Fixed. The HTTP issuer used for the `vc-test-suite` is now disabled by default and can be enabled by adding the `example-http-issuer` feature.

Finding 11: DIDKit CLI option to change tzkt_url is not documented
Partially fixed. The `tzkt_url` parameter is now documented in the README file, but it is not clear from the README how the parameter is passed to the application using the CLI.

Finding 12: Smart contract address–based Tezos DIDs are not implemented
Fixed. Smart contract address–based Tezos DIDs are now supported by `ssi`.

Finding 13: The DID manager resolution process supports only the default TZIP-19 contract
Not fixed. The Spruce team has postponed adding support for general `TZIP-19` contracts because the TzKT API does not support `TZIP-16` metadata.

Finding 14: Verifiable credentials with invalid revocation list indices are accepted by default
Fixed. The implementation of `CredentialStatus::check` for `RevocationList2020Status` now returns an error if the revocation list index is too large.

Finding 15: DIDKit HTTP server is vulnerable to slowloris attacks
Fixed. The Spruce team has added documentation to the project README recommending the use of a reverse proxy if the HTTP server is deployed in a potentially hostile production environment.

Finding 16: DIDKit HTTP server is vulnerable to memory resource exhaustion
Fixed. The HTTP server implementation now reads the request body by using the chunked API `Body::data` and limits the request size to 2 MB.

Finding 17: Private key material is not cleared from memory when no longer needed
Fixed. Each `Params` variant now implements `Drop` to zeroize any private key material contained in the structure using the implementation of `Zeroize` on `Base64urlUInt` and `Prime`. The derived implementation of `Zeroize` on the `Params` type does not appear to be used and may be removed.

The implementations of `generate_ed25519`, `generate_secp256k1`, and `generate_p256` have also been updated to prevent the secret key from being leaked to the stack.

Finding 18: Credible lacks protection against unauthorized user access
Fixed. Credible now uses the `local_auth` dart plugin to implement local authentication.

Finding 19: Credible does not protect sensitive data when switching applications
Fixed. Credible now uses the `secure_application` dart plugin to secure sensitive data in the application switcher.

Finding 20: Consider using IOSAccessibility.passcode to protect keychain items on iOS
Fixed. The secure storage implementation now uses the more restrictive `IOSAccessibility.passcode` access level.

Finding 21: Credible does not validate the length of the recovery phrase
Fixed. The `OnBoardingRecoveryPage` now checks the length of the mnemonic.

Finding 22: The QR code handler is vulnerable to denial-of-service attacks
Fixed. The response size is now limited to 4 MB.

Finding 23: Credible disables TLS certificate verification
Fixed. The code allowing invalid TLS certificates has been removed.

Finding 24: Credible does not prompt users to write down generated mnemonics
Fixed. The application now checks that the user has saved the mnemonic by asking her to input three of the words before continuing.

Finding 25: The bip39.generateMnemonic function generates non-uniform entropy
Fixed. The `bip39` dependency has been replaced with a fork in which the issue has been addressed.

Finding 26: The Credible iOS client allows third-party keyboards
Fixed. The application now prohibits third-party keyboards in `AppDelegate.swift`.

Finding 27: The Credible Android client allows backups of stored credentials
Fixed. The application now has backups disabled in the Android manifest.

Finding 28: The checkIsWebsiteLive function checks only for pages served over HTTPS
Fixed. The website verification code has been removed from the implementation.

Finding 29: The isValidUrl function does not conform to RFC 1034
Fixed. The `isValidUrl` function now disallows URLs starting with a digit and allows URLs with uppercase characters.

Finding 30: Tezos Profiles uses vulnerable dependencies
Risk accepted. The vulnerable Rust dependencies remain, and a number of `npm` dependencies under `dapp`, `api/service`, and `contracts` are still vulnerable. The Spruce team considers the risk of exploitation to be small since the Cloudflare worker is compiled to Wasm.

Finding 31: The Tezos Profiles worker is vulnerable to URL injection attacks
Fixed. The Tezos Profiles worker now validates user-provided URL parameters by using a regular expression before requesting the corresponding resource.

Finding 32: The Tezos Profiles Instagram attestation is broken
Fixed. The Spruce team has disabled support for Tezos Profiles Instagram for now.

Finding 33: The Tezos Profiles web app is vulnerable to URL injection attacks
Fixed. The network name is now required to be either `mainnet`, `hangzhounet`, or `custom`.

Finding 34: The Tezos Profiles API is vulnerable to URL parameter injection attacks
Fixed. The Tezos Profiles API now validates the user-provided address parameter by using a regular expression.

Finding 35: The Tezos Profiles API reports internal errors to users
Fixed. The Tezos Profiles API logs errors using `console.log`, but internal errors are not reported back to the user.

Finding 36: Too few confirmations required when deploying new smart contracts
Risk accepted. The Spruce team considers three confirmations to be enough, given the low-severity consequences of a failed confirmation.

Finding 37: Tezos Profiles does not validate the credential subject or issuer
Fixed. The page now verifies that the credential subject corresponds to the contract creator. The indexer verifies the verifiable credential issuer.

Finding 38: The Tezos Profiles contract code is duplicated
Not fixed. The Michelson code is still duplicated under `contract/src` and `contract/lib`.

Finding 39: The Tezos Profiles worker insufficiently validates data coming from third-party APIs
Fixed. The worker now removes quotes from the DNS response in two steps, preventing the issue. The worker also ensures that the Twitter response includes at least one user before obtaining the username from the response.

Finding 40: Kepler logs authentication data
Fixed. The `Authorization` headers are no longer logged in `extract_info`.

Finding 41: Kepler uses vulnerable Rust dependencies
Partially fixed. The `libsecp256k1` and `zeroize_derive` dependencies have been updated, but the other vulnerable dependencies remain. The Spruce team indicated that it may address these dependencies at a later date, as Kepler is still under active development.