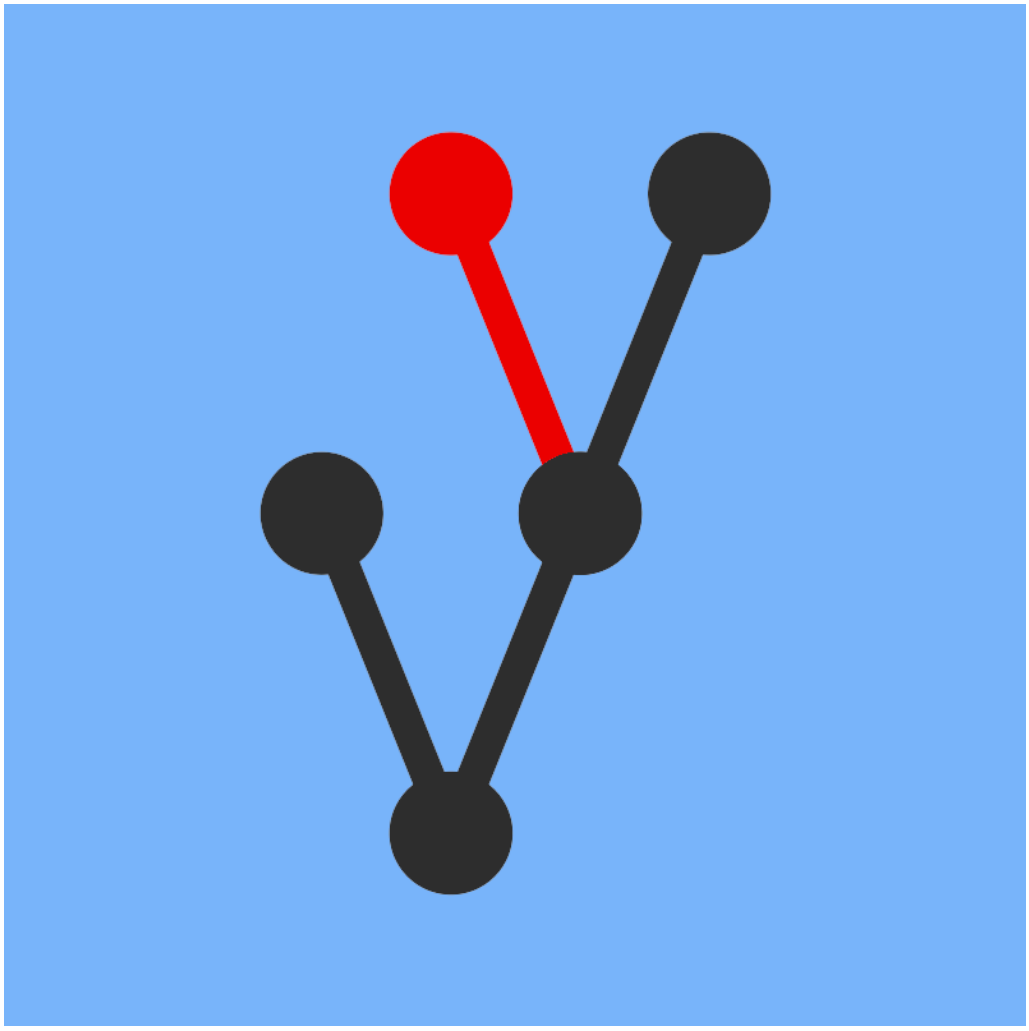

VESvault

Security Assessment

by Jim Zubov <jz@vesvault.com>

Introduction

This document covers security aspects of VESvault (https://vesvault.com) and VES related systems.
The *VES Ecosystem* section provides a detailed description of data types stored in VES Repository, on client devices and in VESlocker Backend. The section explains the relationship between the data and the access permissions.
The *Attack Scenarios* section describes potential attacks targeted at retrieving the private data, or at blocking the legitimate users from accessing their data (denial of service). The section describes the conditions at which each type of attack could succeed, and the ways of mitigating the attacks.

VES Ecosystem

The ecosystem involved in VES operations can be subdivided into the following compartments:

- VES Repository, a centralized database controlled by VESvault
- VES Local Storage, a repository of data pertaining to VES stored locally on the user's device
- VESlocker Backend, a centralized database that pertains to VESlocker split secrets

VES Ecosystem structure, types of stored data and access control measures for each compartment are covered in the following subsections of this document.

VES Repository

Users

VES Repository stores a numeric ID and a unique email address for each VES user. Optionally, first and last name can be stored. The latter feature is not reflected in the VES UI, and is available only on the API level.

API Access

- No authentication is needed to retrieve the current Primary and Shadow Vault Key objects by ID or email.
- An authenticated API call can retrieve emails and optional first/last names for Users that own Vault Keys or Vault Items accessible by the authenticated session.
- An API call authenticated with the User's session can retrieve all details and related objects pertaining to the authenticated User.
- An API call authenticated with the User's session can update the first and last name fields to arbitrary strings up to 48 bytes long, or to *null*.

Vault Keys

A Vault Key is an asymmetric keypair that encrypts Items stored in the corresponding Vault. The public key is stored as SPKI PEM.The private key of the keypair is stored in a PKCS #5 encrypted form. The API does not make any effort to validate Private Keys, and custom implementations may choose to store the private keys outside of the VES Repository, although it may affect interoperability.

Vault Types

The following types of Vaults are defined:

- **Primary:** The master Vault, stores any keys and recovery tokens pertaining to the User. There is one Primary key per User. The Primary Private Key is encrypted with VESkey, the User's master passphrase.

- **Shadow:** A mirror image of the Primary Vault. The Shadow Private Key is encrypted with the Recovery Key, which is never stored explicitly. The Recovery Key is reconstructed from Recovery Tokens during the VESrecovery process. There is one active Shadow Vault per User. Multiple Shadow Vaults in a Recovery state, pertaining to lost vaults, may exist per User. Shadow vaults are automatically recycled once the Recovery is completed.
- **Secondary:** An app level Vault, referred to as Secondary Vault or App Vault. identified by a combination of a VES Domain name and an External ID, owned by a specific User. The passphrase to the Secondary primary key, the AppKey, is stored encrypted in the User's Primary Vault.
- **Temp:** A provisional vault created when an item is requested to be shared with another User, and the recipient User does not have a matching Primary or Secondary Vault. The passphrase to the Temp primary key, the TempKey, is stored in
    - The sender's Primary Vault,
    - The sender's Secondary Vault, if the sharing was requested on a secondary vault level,
    - The sender's Propagator Vault, a special type of a Secondary Vault used for key propagation,
    - The recipient's Primary Vault, if the Temp Vault is secondary and the Primary Vault existed at the time of sharing.
- **Anonymous:** A variant of a Secondary Vault that is not associated with any User. The External ID must start with '!'. For use with automated services and bots.
- **Lost:** A former Primary Vault after the User initiated VESrecovery, or a Secondary Vault explicitly replaced by the User because the AppKey is stored in a lost Primary Vault and is inaccessible.
- **Deleted:** A former Vault that has been replaced or recycled by an appropriate action. Not guaranteed to be stored permanently in the VES Repository.

API Access

- The public key, metadata and the key type can be accessed by any request that accesses the Vault Key.
- Any Vault Key can be matched by an internal ID.
- Primary and Shadow keys can be matched by the User's ID or email.
- Secondary and Anonymous keys can be matched by a combination of VES Domain and External ID.
- An authenticated API session can access Vault Keys with which the items the session has access to are shared.
- The encrypted private key can be retrieved as follows:
    - For a Secondary or an Anonymous Vault Key – by any request that accesses the Vault Key.
    - For a Primary or Lost Vault – by an API session authenticated with the User who owns the key.

    - For a Temp Vault – by an API session authenticated with the recipient User of the key, or with the Secondary Vault having the same Domain + External ID as the Temp Vault.
    - For a Shadow Vault – by an API session authenticated with the owner User, *only if* the Vault is in VESrecovery, *and* the Security Time Delay has expired or was not set.
    - For a Deleted Vault – by an API session authenticated with the owner User, *only if* the deleted Vault was *not* a Shadow Vault.
- An API session authenticated by the owner User, or by the Secondary Vault having the same Domain + External ID as the accessed Vault Key, can retrieve:
    - The owner User
    - The Domain and the External ID
    - The Vault Items decryptable by the key
    - The Vault Items that store the AppKey (for Secondary or Anonymous vaults), or the Recovery Tokens (for Shadow vaults).
- A session authenticated by the User's Primary Vault Key or through the User's OTP, can create or replace the User's Primary, Shadow or Secondary keys.
- An unauthenticated API request can create an Anonymous Vault Key, *if* the Domain + External ID combination was not used yet, *if* the Domain permits anonymous vaults, *and if* the External ID starts with an '!' and does not contain any other '!'.
- **Anonymous Hierarchy:** An API session authenticated by an Anonymous Key can create another Anonymous Key in the same VES Domain *if* the External ID of the new Anonymous Key contains an '!' at some position besides the first character *and* the part of the External ID preceding such '!' exactly matches the External ID of the key the session is authenticated with.
- An API session authenticated by a user or by a key, except for an Anonymous key, can create a Temp Key, provided that the recipient user doesn't have a Primary or Secondary Key that corresponds to the Temp Vault for the key being created.
- The public key, encrypted primary key, the owner User assignment and metadata are immutable, and cannot be changed after creation.
- The key type is appropriately updated by the API when the key is replaced, or when VESrecovery is initialized.

Vault Items

A Vault Item stores a short sequence of bytes, encrypted with one or move public Vault Keys, and associated metadata. Therefore, the content can be decrypted only by the corresponding private Vault Keys.
Due to end-to-end nature, the API is not capable of validating the content of Vault Items. The responsibility for the integrity of the content lies on the client-side libraries.

Vault Item Types

VES Repository stores Vault Items of the following types:

- **String.** An arbitrary string or binary sequence of bytes. The length of the plaintext is limited to 16384 bytes. The item is identified by a combination of Domain + External ID, the namespace is separate from Vaults.
- **File.** A symmetric encryption key for some externally stored ciphertext, and associated metadata. Note that the File metadata is stored in an encrypted form, unlike the Vault Item metadata that is stored unencrypted. Like for a String Vault Item, a File is identified by a combination of Domain + External ID, the namespace is separate from Vaults but is shared with String Vault Items. The plaintext Vault Item metadata contains a key “a”, which identifies an encryption algorithm, understandable to libVES on the user side.
- **Password.** A passphrase (AppKey) for a Secondary or Temp Vault. The item is associated with a Secondary or Temp Vault Key, or with a Lost or Deleted Vault Key that was originally created as Secondary or Temp.
- **Secret.** A Recovery Token for use with VESrecovery. The item is associated with a Shadow Vault Key. The plaintext Vault Item metadata contains keys “v”, “n” and “b” understandable to libVES on the client side, and a key “t” which defines the Security Time Delay and controls the private key accessibility through the API.

API Access

- Vault Item fields and related objects accessible through the API include:
    - The plaintext metadata,
    - The internal ID,
    - The Creator information,
    - Domain + External ID for String and File Items,
    - The Vault Key for Password and Secret Items
    - Ciphertext data entries encrypted by one or more Vault Keys, and information on each corresponding Vault Key.
- All fields for a Vault Item listed above can be retrieved by an API session authenticated by
    - a Vault Key associated with one of the ciphertext data entries for the Vault Item,
    - *Or*, by a Vault Key associated with a Password or Secret Item,
    - *Or*, by a User that owns one of Vault Keys mentioned in the former clauses.
- **Creator:** The User (or the owner of the Vault Key) pertaining to the API session that created the Item is assigned as the Creator. If the API session is authenticated by an Anonymous Vault Key, no creator is assigned, therefore such Item is considered anonymous.
- **Admins:** Ciphertext entries associated with Secondary Vault Keys in the special Domain “.admin” assign Admins to the Item. The External ID of such Vault Key must match the lowercase email address of the Admin User. Admins have the same permissions as the Creator, except that they cannot remove the ciphertext entry associated with the Creator’s Vault.
- A String or File Item can be created by an authenticated API session.

    - The Item must have a ciphertext data entry encrypted by the Vault Key the session is authenticated with, or by a Vault Key owned by the User the session is authenticated with.
    - The Domain + External ID combination must be available, i.e. not used by an existing (non-deleted) Vault Item.
    - If the External ID contains a '!' character, the value preceding the '!' must match the External ID of the Secondary Vault Key the session is authenticated with.
    - If a Domain for a Vault Key associated with a ciphertext entry is different that the Domain of the Item, cross-domain rules and quotas may apply.
- A String or File Item can be replaced or deleted by
    - An API session authenticated with the Creator User, or with a Vault Key owned by the Creator that has a ciphertext entry for the Item,
    - *Or*, by an Admin of the Item. When an Admin replaces an Item, they become the Creator of the new version.
    - A new version of the replaced Item gets a new internal ID. Replaced or deleted Items become historical, and can be accessed through Events.
- A Password or Secret Item can be created only together with the corresponding Vault Key, subject to the API policies for the Vault Key creation. The Item cannot be replaced or deleted.
- A Creator or an Admin of a Password Item can create or delete ciphertext entries pertaining to other Vault Keys, thus delegating access to the underlying Vault the Password Item pertains to. Access delegation is subject to VES Enterprise terms, and can be restricted.
- A Secret Item, when created, is not supplied with a ciphertext entry for the Creator's Vault. Instead, the ciphertext entry is for a Primary Vault of the Assistance Friend (normally one Friend per Item). The Assistance Friend can create or delete a ciphertext entry for the Creator's Primary Vault as a part of VESrecovery, only if the underlying Shadow Vault Key is in the Recovery state.
- **Key Rotation:** When a new Primary, Shadow or Secondary Key is created through an API call and the corresponding old key is accessible, the API client is required to supply ciphertext entries associated with the new key that pertain to all currently active Vault Items associated with the old key. Ciphertext entries for deleted or replaced Items are permitted but are not required.
- **Item Propagation:** An API session authenticated with a Vault Key that can decrypt the Item, and also can decrypt a Password for a Temp Key that can decrypt the Item, can create a ciphertext entry for the Item with a Primary, Shadow or Secondary Key, if such Key has same User and/or Domain + External ID as the Temp Key. The item propagation is usually performed by the owner of the Temp Key, but can be also performed by any other user if the conditions are met.
- **Item Recovery:** An API session authenticated with a Vault Key that can decrypt the Item can create a ciphertext entry for the Item with a Primary, Shadow or Secondary Key, if such Key has same User and/or Domain + External ID as a Lost Key that has a ciphertext entry for the Item.

Sessions

Each VES session is identified by a unique high entropy Session Token. The Token is passed with every authorized API call.

- **User Sessions:** Created by VES UI or CLI when adding a VES account to a device. User Sessions have indefinite lifetime, but are revoked when the User rekeys their Primary Vault. A User Session allows full read access to all Vaults owned by the User, and limited write access. Most updates on the Primary and Shadow Vaults require an Elevated Session.
- **App Sessions:** Authorized by a Secondary or an Anonymous Vault Key. The API returns a Session Token encrypted by the corresponding public key, therefore the client needs to have the AppKey to decrypt the private key which decrypts the Session Token. App Sessions usually have a lifetime of 4 hours, but the client may optionally request a permanent App Session Token for specific cases.
- **Elevated Sessions:** Authorized by the User Session and the Primary Vault Key, or in some cases by a Lost or Deleted Vault Key. The key authorization process involves an encrypted Session Token, same as for App Sessions. Elevated sessions are short lived, and are used for sensitive operations, such as creating or rekeying Primary, Shadow and Secondary Vaults.
- **Domain Sessions:** A special type of sessions for VES Domain Administrators, automated services and bots. Normally have an indefinite lifetime, unless terminated explicitly. A Domain Session gives read permission on all Vault Keys, Vault Items, Sessions and Events pertaining to the Domain. Due to end-to-end nature, the client can only decrypt ciphertext entries if in possession of the appropriate AppKey or passphrase. A Domain Session can delete Keys, Items or ciphertext entries pertaining to the Domain, if the API constraints are satisfied. A Domain Session can be generated by an assigned Administrator of the Domain or by the creator of an experimental Domain. The Session Token creation process is similar to App Sessions and Elevated Sessions.
- **OTP Authorization:** A special type of authorization involving an OTP token sent by VES to the User's email. Used when the User's Primary Vault / VESkey is not available – initial account creation and account VESrecovery initiation.

Domains

VES Domains are namespaces for Vault Keys and Items. A Domain usually pertains to a specific type of App. Experimental domains start with "x-", and are created automatically when used in proper API calls or in libVES unlock requests.

Events

Events represent a historical log of actions on VES Repository objects – creation, deletion and modification. Event logs for Vault Keys, Vault Items, Users and Domains can be retrieved with a proper API authorization. Client side libraries offer real time event tracking via HTTPS long polling.

VES Local Storage

VESvault and VES enabled apps may store essential information on clients' devices in ciphertext and plaintext form.
VESvault.com Web UI, Web and Node-based VES enabled apps use JS storage APIs, localStorage and IndexedDB.
VES CLI and native VES enabled apps use libVES.c KeyStore extendable API. The default KeyStore uses a local directory to store VES related content.

Web UI

VESvault.com uses JS localStorage API to store user session tokens and VESlocker entries.

- User Session Tokens for the connected user accounts are stored in the localStorage in a plaintext form
- VESkeys are stored in a form of VESlocker entries. Each entry consists of randomly generated 256-bit VESlocker ID and Seed, and a ciphertext data. The key to the ciphertext is returned by the VESlocker backend API in response to the ID and the Challenge. The Challenge is a hash of the Seed and a user supplied PIN, that is not stored anywhere.
- Additional information, such as VESlocker consolidation counters, VES account metadata and user preferences, may be stored in the localStorage and/or sessionStorage.

VES enabled Apps that use libVES Flow functions store an unencrypted private exchange key in the sessionStorage. The exchange key encrypts VES related tokens, the ciphertext is passed between the App pages and VESvault pages in the hash part of the URL. When navigating between the app pages, tokens encrypted by the exchange key may be temporarily stored in the sessionStorage too.
Additionally, an App may handle and store information related to the App Vault of various sensitivity levels on its own discretion.

Service Workers

VES Service Worker listens to WebPush notifications from the VES backend. It performs Temp Key propagation and/or raises VES alerts to the user, depending on the notification type. If a WebPush notification requested a Temp Key propagation and the Worker has completed the propagation successfully, the user alert is suppressed if the device permits silencing.
VES Service Worker uses indexedDB API to store the following data:

- A copy of VESvault.com localStorage, including plaintext Session Tokens and VESencrypt Tokens.
- Plaintext Propagator AppKeys for the connected VES accounts. A Propagator Vault, a special App Vault with the Domain ".propagate" and the External ID equal to the User's lowercase email, stores the copies of TempKeys created by the corresponding VES account User.
- Redundant plaintext copies of TempKeys created by the connected VES account Users

VES does not provide standard means or guidelines for implementing Service Workers for VES enabled Apps. If implemented at the App's discretion, such Worker might store a Session Token and/or AppKey to perform certain tasks in conjunction with WebPush.

Browser Add-Ons

VESvault provides optional browser add-ons for certain platforms. Such add-ons store a backup copy of the localStorage, and automatically restore the backup in case if the localStorage was cleared.
Custom add-ons may implement some VES app functionality and store some sensitive data at the implementor's discretion.

Node Apps

Node JS Apps can be integrated with VES using libVES.js, similar to Web-based apps. The Apps may store some sensitive data locally at the implementor's discretion.

VES CLI and Native Apps

libVES.c provides a KeyStore API for storing VES related data locally. A standard implementation, KeyStore_cli, stores the data in a ".ves" subdirectory of the Unix $HOME directory. Session Tokens and AppKeys are stored in a plaintext form, while VESkeys are stored as VESlocker entries, in the same format as in VESvault UI.
The library does not store any keys unless explicitly directed by VES CLI arguments or by KeyStore API calls from a custom App.

VESlocker Backend

For each VESlocker ID generated by the client library, VESlocker Backend database stores a randomly generated VESlocker Secret and the access counter for bruteforce throttling. The Secret is never exposed through the API. The client receives a Key in the API response, which is deterministically generated from the Secret and the client supplied Challenge.

Attack Scenarios

This section covers special API use scenarios, data breaches to each of the parts of the VES ecosystem, malware actions, intercepted communications, user errors and social engineering scenarios.

Public API / UI Attack

Using public VES API and UI, the Attacker may attempt several brute force and denial of service approaches. Note that if the Attacker discovers any currently unknown vulnerabilities, additional attack vectors may apply.

Public Key Reversal

VES Repository exposes most of the public keys pertaining to Vaults. If a public key was reversed to derive a corresponding private key, the Attacker would gain immediate full access to an App or Anonymous Vault, and significantly compromise other types of Vaults. According to the industry consensus, reversing a public key for standardized algorithms is not feasible. In normal operations, VES libraries generate keypairs that meet current security standards. A possibility of compromised keypairs would arise if the client explicitly instructed libVES to use a weak algorithm (such as RSA with < 1024 bits key size), or if the client explicitly used one of experimental postquantum algorithms and such algorithm was later shown to be unsecure (an example would be SIDH).

AppKey Bruteforce

In normal operations, VES libraries generate AppKeys and other passphrases using a cryptographically safe randomizer with at least 192 bits of entropy. Private keys in VES Repository are stored as PKCS #5 with secure key derivation functions on top of cryptographically safe passphrases. Bruteforce decryption of such a key is not feasible, according to the industry consensus. An exception is when a client explicitly instructs libVES to use a custom supplied weak passphrase.

OTP Bruteforce

VES backend generates an OTP code with at least 30 bits of entropy, upon a User's request with a captcha verification. The code is sent to the User over email. VES UI receives a VESlocker entry that is unlockable by the OTP code, and contains an encrypted temporary password of at least 144 bits of entropy with a 1 hour lifetime. The VESlocker entry with additional measures limits the number of OTP attempts to 3. A new OTP code can be re-sent to the same email address in at least 15 minutes, and the time delay is increased on repeated attempts, making the bruteforce approach practically unfeasible.

Session Token Bruteforce

Session Tokens are generated by the VES backend, and have at least 256 bits of entropy. Therefore, bruteforce hacking is not feasible for Session Tokens. Moreover, while Session Tokens give certain access rights to VES Repository, the Attacker would need additional keys to decrypt any ciphertext. Also, any changes done to VES Repository by an authorized API or UI session are reversible within a reasonable timeframe.

OTP DoS

An OTP code is useful in conjunction with a VESlocker token that encrypts a temporary password. The VESlocker token is delivered to the UI frontend at the time when the OTP code is sent to the User's email, after the User completes a captcha. Re-sending a different OTP to the same email is limited to 15 minute delay, the delay increases on repeated attempts. An Attacker repeatedly requesting an OTP for a certain email address may cause some minor inconvenience to the owner of the email, but the cost to the Attacker is repeatedly passing a captcha validation. Supplying a correct OTP for a certain User to the VES backend without having a VESlocker token will immediately trigger generating and sending a new OTP code for this User. It may potentially interfere with the User's legitimate operations. However, even though the entropy of an OTP code is relatively low, any attempts of a DoS attack by OTP flooding is impractical, and will result in source IP throttling for the Attacker.

Import Code DoS

VES UI and CLI use Import Codes for end-to-end syncing of VES accounts. Requesting account import via UI involves captcha verification. Account import via CLI can be requested without a captcha, unless there are repeated attempts on the same account. If there are too many active Import Codes for the User (more than 8), VES UI export section (Add a New Device/App) will show a text entry field for the Import Code instead of a selection list. Therefore, an Import Code DoS attempt would only result in a minor inconvenience to the User.

Item Sharing DoS

API requests that create ciphertext entries for the same Vault and multiple Items in a short timeframe by non-owners of the Vault may trigger a recipient quota, and result in some of the API requests being denied. The recipient User (the owner of the Vault) can mitigate this problem by having some Vault Items owned by the User shared with (i.e. having ciphertext entries for some Vault of) certain trusted users. The sharing action puts the trusted users in a separate bucket with higher quota limits.

VESlocker Bruteforce and DoS

A meaningful attack via the VESlocker API is only feasible when the Attacker has a valid VESlocker ID, a 256-bit entropy value which is a part of a VESlocker Entry. Calls to the VESlocker API with VESlocker IDs that do not correspond to any currently used VESlocker Token will not have any effect outside of a generic server level DoS.

VES Repository Data Breach

A read-only access to VES Repository database, or its backup copy, would not give immediate ability to decrypt any encrypted data. However, it would give a certain advantage in combination with other attacks.

VESrecovery Friends' Collusion

Having an image of the VES database, the Attacker can retrieve any User's encrypted private Shadow Vault Key, a list of the User's VESrecovery Friends, and all ciphertext entries pertaining to the User's vaults. With the cooperation from the sufficient number of the Recovery Friends (or separate attacks to decrypt their Vaults by some means), the Attacker can reconstruct the Recovery Key, unlock the User's keychain and decrypt all items.

Data Collection

The Attacker can gather non-public information and statistics about email addresses, Vaults, VES Domains, Items and Sessions. The provided insights might be useful in conjunction with other attacks.

Session Token Reversal

Session Tokens are generated with at least 256 bits of entropy and are stored in a hashed form. Reversal of Session Tokens is not feasible.

OTP Interference and Temporary Password Reversal

OTP Codes and temporary passwords are short lived. Reading realtime or very recent data could give the Attacker ability to interfere with users' OTP operations, causing minor inconvenience. Temporary passwords are short lived and are stored in a hashed form. Reversing a temporary password in conjunction with the right timing could allow the Attacker to replace the User's key and/or start the account recovery. However, reversing a 144-bit value from a KDF based hash within a short period of time is not feasible. Also, any further actions would trigger a notification to the legitimate User and can be addressed by the User through the VES UI.

Write Access

Write access to the VES Repository Database could give the Attacker more capabilities, like creating, deleting and altering Users, Vaults, Items and Sessions. In particular, the Attacker could perform a Man in the Middle attack by replacing a Vault Key pair of a certain User, thus creating an impostor Vault and gaining the ability to decrypt any new ciphertext entries deposited under the new Vault Key.
Although the VES Repository backend is protected by strong access control, VES enabled apps seeking additional security may implement their own additional checks by keeping track of

public Vault Keys the data is shared with, and triggering specific actions if any public keys were changed.

Client System Exploits

Local storage breaches on various platforms, and deploying malware on client devices, can give the Attacker certain levels of advantage.

Client Storage Breach

Session Tokens

Collecting Users' Session Tokens from the JS localStorage or from the CLI KeyStore would give the Attacker some access rights. However, a Session Token alone doesn't give the ability to decrypt any ciphertext entries or to perform account operations such as rekeying or starting the recovery. Any changes done to the Items on the account are reversible if addressed within a reasonable timeframe.

VESlocker Entries

Collecting VESlocker entries from the JS localStorage or from the CLI KeyStore gives the Attacker an opportunity to decrypt a VESkey from a VESlocker entry. With no other access privileges, the Attacker is limited by the VESlocker backend to about 19 PIN attempts during the first weeks, and 6 more attempts during the year. Considering a 4 digit numeric PIN, the probability of guessing the correct PIN is within a fraction of a percent. Longer PINs make this probability even smaller. However, if the Attacker has some insight into possible values of the PIN, this probability may significantly increase.
Repeated VESlocker attempts will increment the throttling count for the corresponding VESlocker ID, and may effectively cause a denial of service when the legitimate User tries to use the VESlocker Entry. Due to the nature of VESlocker, there is no mitigation for this DoS type. The affected User would have to use other means to access their VES account, such as using another device where the VES account is set up, a backup VESkey or VESrecovery.
Multiple VESlocker entries for different accounts stored on the same device would proportionally increase the probability of a successful attack on one of the accounts. A particular risk is involved in having multiple VESlocker entries under the same PIN without VESlocker consolidation, i.e. having entries with the same PIN stored under different VESlocker IDs. In this case, the risk of successfully hacking the shared PIN would proportionally increase.
VESlocker entry access combined with the VESlocker backend breach presents a particularly high threat. In this case, any PIN can be relatively easily bruteforced, and the corresponding VESkey recovered, without VESlocker throttling.
Since VESlocker entries are stored together with Session Tokens for the corresponding accounts, a successful VESkey reversal from a VESlocker entry is a major security threat, and would allow decrypting all ciphertext entries for the affected account.

Service Worker IndexedDB

Breaching the IndexedDB storage of a service worker would expose a backup of the localStorage that stores Session Tokens and VESlocker entries, the corresponding consequences are described in the above subsections. Additionally, it would expose the Propagator AppKey and the TempKeys.
Access to the Propagator Vault would not provide an additional advantage, because the Propagator Vault normally only stores TempKeys created by its owner.
The TempKeys can only give a limited advantage. A TempKey unlocks a Temp Vault, and the encrypted private key for the Temp Vault is only accessible to the authenticated owner of the Temp Vault, who is normally different from the creator of the Temp Key.

Plaintext AppKeys

Web localStorage does not normally store plaintext keys, unless a VES enabled app does so on its own discretion. However, non-interactive CLI and native backend apps can use the KeyStore API to store plaintext AppKeys. While plaintext keys are essential for automation of the services, they can pose security risk in case of a data breach, in particular if the disk image of the backend server is read by an unintended party.
To minimize the security risks, it is recommended to avoid disk backups of servers that run automated VES processes. Such a server should be redeployed from the distribution images in case of failure, and VES setup should be reinitialized in a secure way.

Keystroke Logger

A successful deployment of a keystroke logger on a client device would allow the Attacker intercepting PINs. While a PIN is not of much use on its own, it’s a high security threat when combined with access to the VESlocker entries and Session Tokens.

Traffic Sniffer

VES libraries use TLS to communicate with VES and VESlocker servers. Intercepting the TLS traffic does not pose any security threat.
However, a malware that can intercept socket communications outside of TLS encryption may incur certain risk. Communications with the VES API and backend do not contain passphrases to any keys, but may contain a Session Token, and in some cases a temporary password. Intercepting these values may give the Attacker certain access to the account, as mentioned in other sections of this document.
Intercepting VESlocker traffic outside of TLS encryption would give the Attacker the Key that decrypts the corresponding VESlocker entry. In combination with accessing VESlocker entries and Session Tokens, the Attacker would be able to retrieve and decrypt all ciphertext entries pertaining to the VES account.

VESlocker Backend Data Breach

If the Attacker can access the VESlocker Backend, they could retrieve VESlocker Secrets. The Secrets are not useful on their own, but in combination with a client storage breach they would give a possibility to bruteforce the PINs associated with the VESlocker Entries stolen from the client side. The potential consequences are described in the *Client System Exploits* subsection.

Data Loss

Data loss may occur by the Attacker's malicious action, by an accident, or resulting from User's actions on client side devices.

VES Repository Data Loss

In case of VES Repository data loss the User may lose their ciphertext entries, or the keychain necessary to decrypt them. While VES Repository is redundant and strictly access controlled, some VES enabled Apps may consider storing copies of the ciphertext entries and the encrypted private Secondary Key pertaining to the User. Some custom systems may consider backing up the encrypted private Primary Key too.
The additional risk of an app level VES backup breach is limited to exposing some metadata, such as the list of Vault Items the App Vault can access.
A backup copy of the encrypted private Primary Key and of the Primary Vault ciphertext entries, if breached, would result in more metadata exposure. It would also give the Attacker the ability to decrypt the keychain having User's VESkey, while VES API and UI require additional email verification to access the User's encrypted private Primary Key.

Client Data Loss

Data loss on a client system may be caused by loss or damage of a device, malware actions, automatic actions such as browser's localStorage cleanup, User actions or Attacker actions. User Session Tokens can be easily regenerated, but losing VESlocker Entries may result in the User permanently losing the ability to decrypt their VES keychain.
To mitigate potential loss of VESlocker Entries, the User should follow VES Redundancy guidelines in the VES UI.
If a VES enabled App stores data encrypted by VES File Items on the client system, it should consider backups, such as cloud storage.

VESlocker Backend Data Loss

Losing VESlocker Secrets would render all affected VESlocker Entries useless and result in Users losing the ability to decrypt their VES keychain. VESlocker Backend is redundant and strictly access controlled, however, Users are advised by the VES UI to store a copy of their VESkey in a safe place.

Email Inbox Breach

Having an access to a User's email account allows the Attacker to start VESrecovery on the User's behalf, or to create an impostor VES account linked to the compromised email address, such account being fully under the Attacker's control. If the Attacker has write access to the User's inbox or can redirect the mail, they can also erase or hide the email notifications from VES, leaving the legitimate User unaware of the intrusion.
To mitigate the possibility of VES account takeover, the User should maintain reliable VESrecovery Friends that will receive notifications and alert the User in case of an intrusion attempt. One of the options is to set the User's alternate email address as a VESrecovery Friend, and make sure both primary and alternate email accounts are continuously accessible. VESrecovery Friends are strongly advised to contact the User prior to providing a VESrecovery Assistance to mitigate any potential attack.
Impostor accounts are harder to manage because the legitimate owner of the email might be unaware of VES at all. For additional security, VES enabled applications are recommended to maintain a list of trusted VES accounts, and appropriately advise the User when sharing an item with a new account. When a User selects a new VESrecovery Friend, a direct verification with the person is recommended after setting VESrecovery.

Stolen or Shared Device

When an Attacker gets access to the User's device and passes the device security, they may take advantage of all Client System Exploits, and additionally of Email Inbox Breach, if the User's email account is connected to the device. The remaining safety in this case is VES PIN. Therefore, if the device is PIN protected and the device PIN is shared with other users, it is important to keep the VES PIN different from the device PIN.

User Errors and Social Engineering

Outside of data breaches and malware, security risk may arise from inadequate actions of a User. Such mistakes may be caused either by lack of User's awareness, or by an Attacker convincing the User to perform dangerous actions under false premises.

Improper Sharing

Improper Item sharing involves creating a ciphertext entry for a Vault that is not owned by the intended recipient. The two possibilities are

- Sharing with a wrong email address or Vault ID,
- Sharing with an impostor Vault.

The former case may involve either mistyped email address, or an address or Vault ID copy-pasted from a phishing message.
Impostor Vaults are discussed in the Email Inbox Breach section of this document.
In either case, the User should be made aware of the dangers of sharing secure information with a wrong recipient. A secure VES enabled App should consider maintaining a special Vault

Item for the User that has ciphertext entries for all trusted Vaults, and alert the User when sharing is requested outside of the trust list.

Improper VESrecovery Assistance

VES UI strongly advises a VESrecovery Friend to verify with the User via a side channel prior to providing the Assistance. If a Friend disregards this advice, a VESrecovery started by an Attacker gets a higher chance of success. However, even if the Attacker gets enough Friends assisting them, a properly set Security Time Delay would give the legitimate User time to stop the VESrecovery launched by the Attacker and invalidate all related tokens.
As a variant, the Attacker may send phishing emails to the Friends pretending to come from the legitimate User, or even multimodal messages from other User's accounts if the Attacker is in control of the User's personal device.
In addition to setting a proper Security Time Delay, for additional security, the User can select their own secondary email address as a VESrecovery Friend. In this case the User will get a VESrecovery alert even if the primary email is fully controlled by the Attacker.

Improper Export

In the VES UI, *Add a Device / App* page displays the Import Codes that are listening for the User's account to be imported. If the User clicks on a wrong Import Code pertaining to an Attacker, the Attacker will receive a special type of VESlocker entry indirectly protected by the PIN selected by the User. The Attacker has 3 PIN attempts before the entry is invalidated. The User can also immediately invalidate the export entry if the mistake is identified.
The probability of the Attacker guessing the right PIN is very low, however it can significantly increase if the Attacker has any insights into the possible PIN values.
If the Attacker was successful with the PIN, the User would see a notification of a successful export. At this point, the User would need to rekey the Primary Vault to mitigate the consequences.

VESrecovery Friends' Collusion

In this scenario, a subgroup of VESrecovery Friends selected by the User colludes to get illegitimate access to the User's VES account. Having enough members in the subgroup, the malicious Friends can reconstruct the User's Recovery Key from the tokens stored in their Vaults. However, the User's primary encrypted Shadow Key is needed in conjunction with the Recovery Key to decrypt any information. The Attacker needs either to have access to the VES Repository backend, or to complete a VESrecovery on the User's account, including email verification, to be successful.

Impostor Apps

Using VES libraries, any web-based app can request a User to grant permissions to an App Vault pertaining to any specified VES Domain. Therefore, malicious Apps can attempt to gain access to a certain App Vault pertaining to the User. To mitigate this threat, VES UI implements the App Trust Manifest system.

By default, any App requesting access to any VES Domain triggers a standard warning. The UI displays a banner that includes the URL of the requesting App and the warning text that alerts the user that the App is untrusted or experimental, and advises to proceed with caution.
The root of the App Trust Manifest is managed by VESvault. The publisher of the App can request VESvault to set specific trust levels with custom messages for the specific URLs when requesting access to the VES Domain pertaining to the App. The Trust Manifest for a specific URL/Domain combination can redirect to the Publisher server for finer granularity access control, such as allowing only specific VES Users to proceed.
A secure App publisher should request to display a trusted banner for the authorized production URL, and to block access (disable the UI PIN entry) for any URLs except for authorized production and development URLs pertaining to the App.

Conclusion

The analysis provided in this document highlights the measures that can be taken to secure VES accounts and encrypted data:

- Protect your devices, infrastructure and servers from malware and exploits;
- Use caution when sharing VES items with other Users, be aware of possible impostors and dangers of mistyped emails;
- Follow the VES Redundancy guidelines in the VES UI to securely protect the VES account from key loss;
- Choose VESrecovery Friends wisely, maintain a contact with them;
- Contact the User before providing VESrecovery, beware of possible attacks;
- Use special caution when setting VES account on shared devices, never share the VES PIN;
- Pay attention to VES alerts;
- Do not disregard warnings when entering VES PIN to grant access to an App Vault, double check the URL of the App;
- When publishing a VES enabled App, contact VESvault to set the proper Trust Manifests;
- Use proper access control for servers and devices that run automated VES enabled processes and bots, avoid automated backups for such servers.