Twenty Years of LastPass, One Afternoon of Vaultwarden

August 28, 2026 / Gavin Jackson

vaultwarden bitwarden passwords self-hosted ansible docker nginx security linux rbw

I have used LastPass for the better part of two decades. It was the default answer for long enough that I stopped thinking about it. Then 2022 happened: source code taken from the development environment in August, and by November the attacker had used that access to take encrypted customer vault backups from cloud storage. The vaults were encrypted, so the practical risk was low. The way the incident was disclosed, slowly and in stages, bothered me more than the technical detail. I stayed anyway, because moving a vault is a job that is always scheduled for next month.

The push came from work. We needed a shared vault for the team: service accounts, API keys and the other secrets that had been living in too many places. The official self-hosted Bitwarden server is a heavy .NET and MS SQL stack. Vaultwarden is an unofficial Bitwarden-compatible server written in Rust. It runs as one container with a SQLite database, and the official Bitwarden clients work against it unmodified.

I ran it up on an internal VM, not reachable from the internet, using Docker Compose with an nginx reverse proxy in the same compose file. It took an afternoon, and most of that was reading the vaultwarden wiki.

Running it with Docker Compose

Both containers live in the one compose file:

```
services:
 vaultwarden:
 image: vaultwarden/server:latest
 container_name: vaultwarden
 restart: unless-stopped
 environment:
 DOMAIN: "https://vault.internal.example.com"
 INVITATIONS_ALLOWED: "true"
 ADMIN_TOKEN: "$argon2id$v=19$m=65540,t=3,p=4$..." # argon2 hash, not plaintext
 SMTP_HOST: "smtp.internal.example.com"
 SMTP_FROM: "vaultwarden@internal.example.com"
 SMTP_PORT: "25"
 SMTP_SECURITY: "off"
 EXPERIMENTAL_CLIENT_FEATURE_FLAGS: "ssh-key-vault-item,ssh-agent"
 volumes:
 - ./data:/data
 nginx:
 image: nginx:stable
 container_name: vaultwarden-nginx
 restart: unless-stopped
 ports:
 - "443:443"
 volumes:
 - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro
 - ./certs:/etc/nginx/certs:ro
 depends_on:
 - vaultwarden
```

Vaultwarden does not publish a port on the host at all. Nginx is the only way in, and it reaches Vaultwarden over the compose network using the service name.

A few of these lines are worth knowing about:

- `ADMIN_TOKEN` protects the `/admin` panel. Use the argon2 hash that `vaultwarden hash` generates, not a plaintext string.
- SMTP settings matter because invitations go out by email. Without a working relay, nobody can join.
- `DOMAIN` should be the real URL, or WebAuthn gets unhappy.
- The experimental feature flags turn on SSH key storage and the SSH agent, covered further down.
- I have not set `SIGNUPS_ALLOWED` at all, so registration is open on my install. The server is only reachable inside our network, so anyone who can see it is already supposed to be there. If it were ever exposed beyond that, `SIGNUPS_ALLOWED: "false"` with invite-only accounts is the setting you want.

The nginx config is mounted into the proxy container:

```
server {
 listen 443 ssl;
 server_name vault.internal.example.com;
 ssl_certificate /etc/nginx/certs/vault.internal.crt;
 ssl_certificate_key /etc/nginx/certs/vault.internal.key;
 client_max_body_size 128M;
 location / {
 proxy_pass http://vaultwarden:80;
 proxy_set_header Host $host;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_set_header X-Forwarded-Proto $scheme;
 }
 location /notifications/hub {
 proxy_pass http://vaultwarden:80;
 proxy_http_version 1.1;
 proxy_set_header Upgrade $http_upgrade;
 proxy_set_header Connection "upgrade";
 proxy_set_header Host $host;
 }
}
```

The `proxy_pass` lines use the compose service name, which Docker's internal DNS resolves. The second location block passes the websocket endpoint through; that is what gives the desktop and browser clients live sync. HTTPS is not optional even on an internal network, because WebAuthn and the browser extension both want a secure context. The certificates come from our internal CA, which the machines already trust.

Backups are a nightly copy of the `data` volume: the SQLite database, the attachments and the RSA keys. I did one test restore to a scratch VM to confirm the backup actually restores. Upgrades are `docker compose pull` and a restart.

Getting the team in

I expected this to be the painful part. It was not. I created an organisation, made collections roughly matching our systems, and sent everyone the URL. They registered an account, I invited them into the organisation, and they were in. The only account that existed before the invites went out was mine.

Two organisation policies are worth enabling early. **Require two-step login** refuses access to the organisation until a member has MFA enabled, and **master password requirements** puts a floor under what people choose. Account recovery handles a forgotten master password, provided mail is working.

The whole process was quick. Naming the collections took longer than standing the server up.

The clients

Because Vaultwarden implements the Bitwarden API, the clients are the official ones. Every client has a self-hosted option: on the login screen, change the region from `bitwarden.com` to **self-hosted** and enter the server URL. That is the entire configuration.

Chrome extension

The extension is the one I use most. Against our server it behaves the same as it does against Bitwarden cloud: autofill, inline suggestions, the badge count on the icon. I keep the vault timeout short, because it now holds more than my own accounts.

Desktop app

Same setup, plus it is where the SSH agent lives. It runs fine on Ubuntu.

The bw CLI

Bitwarden's official CLI works once you aim it at the right server:

```
bw config server https://vault.internal.example.com
bw login
export BW_SESSION="$(bw unlock --raw)"
bw sync
bw get password "prod-db-01"
```

It does the job, but it is stateless. Every shell that wants vault access needs `BW_SESSION` exported into it, sessions expire, and you have to remember `bw sync` because the CLI works from a local copy that quietly goes stale. Fine for a one-off lookup. Annoying for anything scripted.

rbw, which is the good one

rbw is an unofficial Bitwarden CLI written in Rust. It runs a background agent that holds your keys in memory, the same way `ssh-agent` and `gpg-agent` do. You unlock once, and every call after that is a plain command with no session strings:

```
rbw config set base_url https://vault.internal.example.com
rbw config set email gavin@example.com
rbw register
rbw unlock
rbw get "prod-db-01"
rbw get --field notes "prod-db-01"
```

It prompts through pinentry, locks itself after a timeout, and Debian and Ubuntu both package it. If you script against a password vault at all, this is the client you want. The Ansible section below exists mostly because rbw exists.

What Vaultwarden supports

The feature list is longer than I expected from one container. These are the parts that matter to me.

Multi-factor authentication. TOTP authenticator apps, email codes, FIDO2 WebAuthn (YubiKey, Nitrokey, SoloKey) and Duo all work. The organisation policy to *require* two-step login works too, so MFA can be a condition of membership rather than a suggestion. Vault items can also store TOTP secrets, so the code sits in the same entry as the password.

> ### *No Premium Tier Required*
>
> *On Bitwarden cloud, some MFA methods need a paid plan. On Vaultwarden they all work out of the box: WebAuthn keys, Duo, TOTP and email codes. There are no tiers on a server you run yourself.*

SSH keys and an SSH agent. Still experimental, hence the feature flags in the compose file. `ssh-key-vault-item` adds SSH keys as a vault item type. `ssh-agent` lets the desktop app act as an SSH agent using keys stored in the vault: enable it in the desktop app's settings, point `SSH_AUTH_SOCK` at the socket it creates, and `ssh` authenticates with a key that is synced everywhere your vault is, passphrase included. It currently needs the desktop client rather than the web vault, and it works on Linux and macOS. If you have been copying an `id_ed25519` file between machines for years, this is the feature that makes the vault the single home for both halves of your identity.

Bitwarden Send. One-off sharing of text or files with people outside the vault, with expiry and download limits. It gets a secret to someone without putting it in an email thread.

Emergency access. A trusted contact can request access to your vault, with a waiting period before it is granted. Grim feature, necessary feature.

The rest. Attachments, folders, favourites, collections, trash with soft delete, master password re-prompt on sensitive items, personal API keys, account recovery and live sync. Groups and event logs exist too, each behind an environment flag.

What is missing: the Bitwarden Public API and organisation API keys (implemented only enough to support Directory Connector), login with passkeys, custom roles, a few enterprise policies, and Bitwarden Secrets Manager. The Secrets Manager gap matters for the Ansible plan further down.

Who is behind Vaultwarden, and can it be trusted?

Fair questions to ask before handing a password server to a whole team. I asked them too.

> ### *How Secrets Are Actually Stored*
>
> *Nothing readable is stored on the server. All encryption happens in the clients, using Bitwarden's zero-knowledge design. Your master password is run through a key derivation function (PBKDF2-SHA256 with 600,000 iterations by default, or Argon2id) to produce the key that protects the vault's symmetric key. Items are encrypted with that key, and the server only ever holds ciphertext. Organisation vaults use a shared key, delivered to each member encrypted with that member's RSA public key. The SQLite database on my server could be published tomorrow and it would be useless without the master passwords. Vaultwarden follows this model because it has no choice: the official clients do the cryptography, and the server just stores what they send it.*

How do we know it does what it says? Two ways. The code is GPLv3 open source and is read by plenty of people who are not me. More usefully, it has been independently audited. Germany's Federal Office for Information Security (BSI) published a static and dynamic code analysis of Vaultwarden in 2024, and security firm ERNW ran a penetration test the same year. The ERNW test found three real vulnerabilities, including an authentication bypass, which were responsibly disclosed and fixed. I find that more reassuring than a clean report: it means people with actual skill are looking, and the fixes land.

CVEs. A handful over the years, and I watch them. Recent examples are CVE-2026-26012, where an organisation member could reach ciphers in collections they did not belong to, and CVE-2026-43914, where the email 2FA endpoint could be abused as a credential oracle to bypass brute-force protection. Both were patched promptly. This is the normal background noise of running software, and the answer is the same as for everything else I run: subscribe to releases, apply updates, do not expose the service more than necessary. Nothing in the project's history looks like the LastPass incident.

Who develops it. The project started as `bitwarden_rs` and was renamed in 2021 after Bitwarden raised trademark and confusion concerns. It is maintained by Dani García with a steady group of contributors. There is no company behind it: support is the GitHub issue tracker, a Discourse forum and a Matrix room, and funding is sponsorships. Releases are frequent, and both security patches and client compatibility fixes arrive quickly.

Long-term support. There is no LTS branch and no support contract to buy. You run `latest` and you keep up. The honest way to think about that risk: if the project stopped tomorrow, nothing is stranded. Every client can export the vault, and that export imports into official Bitwarden or another Vaultwarden instance. The exit cost is low, which is more than I can say for most hosted services.

What does Bitwarden think of it? Tolerance without endorsement, as far as I can tell. Their hosting FAQ says they expect most client functionality to work with unofficial servers such as Vaultwarden, but they cannot guarantee the official clients will work perfectly with them. They will not take bug reports from Vaultwarden users, which is fair enough. They asked for the 2021 rename rather than reaching for lawyers. And their clients remain GPLv3 open source, which is what makes Vaultwarden

possible at all.

Could Bitwarden break compatibility on purpose? Yes, and nothing contractual stops them. They could add server version checks or move part of the API somewhere proprietary. In practice the risk looks smaller than that sounds. The clients are open source, so a hostile change can be forked. The vault data is portable. And the track record so far is accidental breakage, not malice: new client releases have broken Vaultwarden compatibility a couple of times, and each time Vaultwarden shipped a fix within days. The noticeable effect for us is that new Bitwarden features sometimes arrive on Vaultwarden later than on the cloud service, which is exactly the situation with the SSH agent feature flags in the compose file.

Using it with Ansible

The longer-term plan is our fleet of 100+ Ubuntu servers. The secrets situation there is the usual mix of ansible-vault files and worse habits. I want Vaultwarden as the source of truth, with playbooks pulling secrets at runtime.

The official Bitwarden Ansible integration does not work here. It is built on **Bitwarden Secrets Manager**, which talks to the Bitwarden Public API, and Vaultwarden does not implement that API. The community options are better anyway.

Option one is the `community.general.bitwarden` lookup plugin. It shells out to the `bw` CLI on the control node, so it inherits the same session handling: log in, export `BW_SESSION`, keep it alive for the run.

```
- hosts: localhost
 gather_facts: false
 tasks:
 - name: Show a secret from the vault
 ansible.builtin.debug:
 msg: "{{ lookup('community.general.bitwarden', 'prod-db-01', field='password')[0] }}"
```

The lookup returns a list, hence the `[0]`.

Option two, and the one I will standardise on, is rbw with the humble `pipe` lookup:

```
- hosts: webservers
 become: true
 tasks:
 - name: Render the application environment file
 ansible.builtin.template:
 src: app.env.j2
 dest: /etc/myapp/app.env
 owner: root
 group: root
 mode: "0600"
 vars:
 db_password: "{{ lookup('ansible.builtin.pipe', 'rbw get prod-db-01') }}"
 no_log: true
```

No session variables, no login state in the playbook. The operator runs `rbw unlock` at the start of the day and the agent handles the rest. The lookup runs on the control node, the secret travels to the target inside the task and nowhere else, and `no_log: true` keeps it out of the output. Nothing is written down, so nothing ends up in git by mistake.

The decisions around the lookup matter more than the syntax:

- Use a dedicated automation account with access to an organisation collection, not personal logins. Offboarding becomes one permission change.
- Rotate secrets in the vault and nowhere else. The next playbook run picks up the new value.
- The bootstrap question is real: the master password has to come from somewhere. Interactive runs use `rbw unlock` at the keyboard. Unattended CI runs need something else; a short-lived injected credential that only unlocks the automation account is the current favourite.
- Put `rbw sync` at the start of long plays, because a stale local copy fails in unhelpful ways.

None of this is rolled out yet. It is the next project, and it will probably get its own post once I have broken something properly.

The LastPass question

The internal server has been boring, which is what I want from infrastructure. That has raised a question I did not expect: if I trust this with the team's secrets, why is my own vault still at LastPass?

I am genuinely considering a second instance on a small VPS, with the LastPass export imported and the family moved over. The things holding me back are the right things to be held back by: a tested restore procedure, an offline encrypted export kept somewhere safe, emergency access for Jo, monitoring so I find out it is down before I need a password, and the fact that an internet-facing vault server is a target in a way a LAN-only one is not. None of those are reasons not to do it. They are the checklist.

Nearly twenty years of LastPass, and the thing most likely to end the relationship is a weekend project that took an afternoon. The Ansible rollout comes first, then the VPS decision. I will write up both.

References

- Vaultwarden on GitHub and the vaultwarden wiki
- Official Bitwarden clients
- rbw: the unofficial Bitwarden CLI
- community.general.bitwarden lookup documentation
- Using Ansible to read passwords from Vaultwarden
- Bitwarden SSH agent documentation
- Vaultwarden audits wiki page
- ERNW: authentication bypass in Vaultwarden (2024 disclosure)
- Bitwarden self-hosting FAQ, including unofficial servers
- LastPass: notice of the 2022 security incident