SUBGRAPH

HUSH LINE FINDINGS REPORT - FINAL

Hush Line

April 9, 2024

Prepared for: Hush Line

Subgraph Technologies, Inc.
345 av Victoria Suite 400
Montreal, Quebec
https://subgraph.com

Contents

Overview

Subgraph conducted a security audit of Hush Line over two phases of its development. The first phase of our testing was focused on versions of Hush Line that may be deployed onto Raspberry Pi devices and self-hosted on any Debian-based server. This development line is `v1`. Hush Line provided us with a Personal Server device for testing purposes, and we also deployed the application ourselves.

Over the testing period, a managed version of Hush Line was released (`v2`). This version include different features, implementation details, and use cases. These will be described within the report.

First Phase - Personal Server and self-hosted

During this phase we tested multiple deployments including the Personal Server device provided by Hush Line and an installation of our own. We performed a source code review of the `v1` branches and a targeted analysis of the use of the PGPy OpenPGP implementation.

Use cases and attack surface

The `v1` versions are intended to provide the most secure options for deploying the Hush Line application. The Personal Server devices are configured to only provide access over the Tor network. `v1` can also be self-hosted and made as a Tor onion service and/or the public Internet.

It is designed to limit the possible attack surface. It does not include user accounts, or a back-end database. When configured with a PGP public key, no messages will be stored on deployments in plaintext. There is no need to store a private encryption key either.

A SMTP server must be configured to email messages, so SMTP credentials must be stored on the host. SMTP providers provide some options to mitigate the theft of SMTP credentials but those are specific to the providers and outside the scope of Hush Line itself.

The Personal Server implements HTTPS for connections via the local network using *mkcert*.

Positive observations

The minimalistic nature of this version of Hush Line eliminates potential attack surface.

The Personal Server implements some hardening at the operating system and network layer. During setup of the device, some measures are taken to limit unauthorized access to the server:

- Bluetooth radio is killed using `rfkill`
- SSH access is blocked using the `ufw` firewall
- USB access is disabled

The web application component of this versions does not employ JavaScript. This removes some attack surface in terms of not exposing users to untrusted third-party JavaScript or DOM-based (Document Object Model) cross-site scripting attacks.

Recommendations

The implementation can be further hardened to remove any unneeded and potentially unsafe features.

The SSH service could be disabled or the package could be removed from the operating system.

The Raspberry Pi installation instructions for self-hosted devices suggest using Raspberry Pi OS (64 Bit). During our tests, we installed the Lite version onto a Raspberry Pi and it worked as a base OS.

We recommend using a base OS with the least amount of software needed. Lite is a better option in this case.

There are further hardening opportunities that are worthy of consideration. These include configuring a custom kernel that removes unnecessary features or bootstrap a bare-bones host operating system using a framework such as Yocto or debootstrap.

Denial of service

There is some risk of denial of service attacks against devices. These can't be prevented entirely but some scenarios can be mitigated to some extent.

The main concern is that anonymous users over the Tor network could make a device inoperable by sending a large number of messages intended to fill available storage space. We tested this scenario and found that there was a hard limit of 2000 characters for each message.

We tested the *fail2ban* rules that are deployed with the device. We weren't able to force them to rate limit our incoming traffic.

If they do work as expected under the right conditions and block origins that are sending large amounts of requests, there is a potential risk of the rules inadvertently causing a denial of service when the device is accessed anonymously. If the rate limiting is done by IP address or globally, it may prevent legitimate users from accessing the device since legitimate and abusive traffic may originate from the same Tor exits. This form of denial of service protection may not be appropriate for Hush Line deployments over the Tor network, although it may be effective when Hush Line is accessed over the public Internet.

Second Phase - Managed Service

During this phase, we tested a demo deployment of the managed `v2` Hush Line application with both user and administrative accounts. We performed automated and manual testing. We also conducted a source code review of the `v2` code.

Use case and attack surface

Hush Line `v2` is a managed service with different features tiers intended for use cases. It implements sthird-party integrations such as Stripe for payments. `v2` can be self-hosted as well but it is designed as a managed service. It is also available as Tor onion service.

Messages

`v2` also provides different options for storing messages. In addition to the public key encryption option in `v1`, messages are encrypted at rest in the local database with a symmetric key. These will be rendered in

plaintext to the authenticated application user. The encryption key is stored on the host to facilitate this. Hush Line uses this scheme to store other sensitive and personally identifiable information in the database.

Additional components

`v2` includes additional components and functionality to facilitate running a managed service. This includes a back-end database (*MariaDB*), an administrative interactive, roles/privileges, and support for payments.

Additional risks

Due to the managed nature of `v2`, there are additional security risks that are not present in `v1`.

The database introduces the risk of SQL injection. The roles and privileges introduce a risk of privilege escalation to other user and administrative accounts. Personally identifiable information is also at risk of exposure. These are the normal risks of a managed web application.

The managed version is currently multi-tenant. Different organizations share the same server and database. Compromising the application or the underlying host may put all hosted organizations in such a scenario.

Recommendations

Hush Line should evaluate whether it makes sense to separate tenants. This can insulate organizations from one another in case one is compromised.

Positive Observations

Access control

We tested the application and review source code to determine if it was possible to escalate privileges such as unauthorized access between users and from users or from a regular user to an administrator.

We observed adequate protections. The application implements checks to ensure that a user must be authenticated and that their user ID may only access data within its own scope of privileges. Regular users are not able to access administrative functions and the same was observed with free versus paid accounts.

Database

We ran tests and examined application source code to determine if database queries were made safely. We didn't find any security vulnerabilities in this area. The database library (*SQLAlchemy*) was used in the default, safe manner, without any custom logic that would override safety features.

Recommendations

Perform database hardening such as restricting the ability of the database to read and write local files. This may help to limit privilege escalation attack vectors or unauthorized access to the symmetric encryption key if a SQL injection vulnerability is discovered.

This may be implemented by disabling the `local-infile` setting for *MariaDB* or revoking FILE privileges.

Web application security

Flask-WTF is integrated to provide enhanced security for form submissions. This includes Cross-Site Request Forgery (CSRF) protection. Our tests confirmed that this protection is effective – we were not able to submit forms with an incorrect or missing CSRF token.

All versions of Hush Line also implemented a hardened Content Security Policy (CSP).

Authentication

Hush Line implements strong password requirements for accounts. Two-factor authentication is also implemented as an additional measure. In addition to encrypting database contents at rest, account passwords are hashed using `bcrypt`.

Updates, dependencies and code security

The Personal Server performs operating system updates upon setup. Debian's `unattended-upgrades` feature is configured for Hush Line in general.

We observed that Dependabot was enabled on the Github repository to notify when dependencies are updated.

Recommendations

Static security analysis tools can help to eliminate security issues at the code level. We recommend that Hush Line investigates an appropriate tool to perform static security analysis.

Denial of service

Managed and self-hosted versions of Hush Line were less of a concern than the Personal Server in terms of the risk of denial of service attacks.

Recommendations

In terms of general denial of service attacks, third-party services may be implemented in these situations to provide distributed denial of service protection. Nginx configuration and tools such as *fail2ban* may also be fine-tuned to provide IP-based protections and enforce request size limits.

Comparison of OpenPGP libraries

The `v1` branch of Hush Line uses the *PGPy* library to implement encryption of messages. *PGPy* is a pure Python implementation of the OpenPGP protocol. This eliminates the need to invoke executables on the host operating system.

The `v2` branch switched to using the *GnuPG* library. This library is a wrapper to that uses the external `gpg` binary to perform its operations.

There are tradeoffs to either approach. *PGPy* avoids potential vulnerabilities associated with executing a local binary. In particular, it does not provide an opportunity for an attacker to subvert the calling arguments for the binary to execute their own commands. However, there may be vulnerabilities lurking in the library if the OpenPGP library has not been implemented correctly.

PGPy does not appear to be well maintained. If vulnerabilities are discovered, they may not be remediated in a timely manner or at all.

In the past, GnuPG has been affected by a vulnerability that allowed attackers to execute commands due to security flaws in how the `gpg` binary was invoked. This issue has been addressed. Furthermore, the dangerous operations (ie: opening a subprocess) are consolidated into a single method. This makes it easier to audit for future issues.

The `gpg` binary itself is a potential source of vulnerabilities. It is more complex in nature than *PGPy*, with more attack surface, even in this limited use case. It is also written in a language that is not memory safe and has a history of memory corruption vulnerabilities.

Lastly, the binary is also decoupled from the *GnuPG* library. It uses whatever version is installed on the host. In other words, the library may have no known vulnerability but it might be abused to exploit a known vulnerability in the binary itself.

Recommendations

Despite the potential issues with the *GnuPG* library, we would recommend using it over the *PGPy* library. The *PGPy* approach is preferred – especially because Hush Line uses a limited subset of the OpenPGP protocol. However, the lack of maintenance means library users may have little recourse if a security vulnerability is discovered.

Remediation Status

After receiving the initial report, Hush Line implemented our recommendations.

Minimal base operating system

Hush Line has switched to using Raspberry Pi OS Lite as the base operating system.

Multi-tenancy

Hush Line has begin offering a single tenant option for organizations that need this level of isolation.

Database hardening

Hush Line has changed the database implementation from **MariaDB** to *SQLite*. This greatly reduces the attack surface at the database layer.

In the case of an exploitable SQL injection vulnerability, there is still a potential to gain access to local files on the host operating through the file importation functions of *SQLite*. This risk should be documented with the threat models.

Static security analysis

Hush Line uses GitHub but is investigating other options such as *OWASP dep-scan*.

Rate limiting

Hush Line has implemented rate limiting using the *Flask-Limiter* library.

OpenPGP

Hush Line switched to using the *pysequoia* library. This library uses the **Sequoia** implementation of OpenPGP. The library itself is mostly implemented in Rust, making it more performant and addressing the potential of memory corruption vulnerabilities. It provides a Python interface for performing cryptographic operations.

pysequoia avoids the concerns with the *GnuPG* library. As it is better maintained, it is also a superior choice over the *PGPy* library.