LogVM & Log Collection System

Introduction

Qubes OS is a reasonably secure operating system. Qubes takes an approach called security by compartmentalization, which allows to compartmentalize the various parts of someone's digital life into securely isolated compartments. This approach ensures that one compartment getting compromised won't affect the others.

It is an amazing idea with a pretty implementation but Qubes currently lacks a way to securely store and retrieve logs. There is no way to conveniently inspect logs from apps and services running across several virtual machines. This project aims to create an effective, robust and security-focused log collection system for Qubes OS.

Project goals

Priority goals of the project are following:

- Implement a log collection system that is itself working in its own separate VM.
- That system can receive logs from multiple logging systems, such as journald and syslog, for example. Additional bindings can be implemented if needed, including support for non-Unix guest operating systems.
- The log collection system is designed with security in mind.
- The system guarantees log integrity, including inability to modify previous logs and fake timestamps.
- The logs are persistent, despite coming from possibly ephemeral VMs such as DispVM.
- The system supports automatic log rotation with per-VM quotas to prevent DoS attacks.
- Implement some convenient GUI designed specially for the log collecting system.
- The system is extensively documented and tested from top to bottom.

Implementation

The aforementioned system is implemented in two parts. One of them runs on AppVMs and the other one on the LogVM (that runs a Unix-like operating system). A communication between them is going through vchan/qrexec channel, which already exists as a part of Qubes OS and is well-protected. Let's discuss the parts of the system separately in more detail.

The part on the AppVM side.

This part includes a daemon (named log-exporter) that retrieves logs in real time (in simple text format with necessary fields included, such as the log message, hostname, timestamp, PID and so on; the exact format can be defined later) from a logging system (such as journald for Linux guest systems). Data parsing is also implemented here (to define format of log entries and separate one from another in specific way). The daemon is started during the process of guest system boot-up. It is also responsible for creating a connection to the LogVM via vchan/qrexec and sending collected logs over it. It can be written as a Python script, a bash script or a C program.

The part on the LogVM side.

The remaining part of the system is not tied to any specific log format in any way (so it can be said to be log-format-agnostic). The only thing it knows about transmitted data is its text nature (successive lines of text).

The part on the LogVM side consists of the following:

- A program (named log-collector) that receives logs sent via the vchan/qrexec connection and saves them to a text file (with .log extension, for example). An instance of this program is spawned automatically each time an AppVM connects to the LogVM to transmit its logs. It's important to notice that log files received from two different VMs are saved to a separate directories by this tool (it's like any VM has the right to a separate directory, even DispVM). This tool is also responsible for prepending timestamps to log entries (this can be achieved with only very simple parsing to split lines). Can be implemented as a Python script or a bash script.

- A daemon (named log-compressor) that tracks the size of those directories and is responsible for intelligent and secure log rotation. This tool compresses medium-aged files to .gz, .xz etc, deletes the old ones and doesn't touch recent ones. The daemon is started during the process of the LogVM boot-up and works till it's shut down. Can be implemented as a Python script or a C program.

Also, there is the GUI for log collection system on the LogVM side. It allows to view a list of all collected logs sorted by their receiving time and see some of log attributes (such as their origin, size, etc.). At the same time it permits to open them with a suitable program in a DispVM. It can be implemented in Python using toolkits like Qt or GTK.

As soon as this minimal functionality will be implemented, more capability can be added. For example, it can be managing logs like sorting, deleting, copying them or so on.