keylime: Multiple Security Issues (including remote code execution in the Agent component)

From: Matthias Gerstner <mgerstner () suse de>
Date: Fri, 28 Jan 2022 13:57:23 +0100

```
Hello list,

I have been reviewing the Keylime TPM remote attestation solution [1]
which resulted in a number of security related findings, including an
arbitrary remote code execution in the Keylime Agent component. The
upstream project published security advisories, fixes and an update
strategy to Keylime version 6.3.X today. Please find the details in the
following full report:

1) Scope of Review
==================

I've been looking into the four main components of Keylime: the Agent,
Registrar, Verifier and Tenant applications. I have been looking into
version 6.2.0. Any source code locations mentioned in this report relate
to this version.

2) Findings in the Agent Component
==================================

a) `check_mounted()` Function Logic can be Fooled by Unprivileged Mounts (CVE-2022-23948)
------------------------------------------------------------------------------------------

The `check_mounted()` function in `secure_mount.py` attempts to make
sure that a "secure" tmpfs is mounted at `/var/lib/keylime/secure` to
store sensitive data on that never gets written to disk. To do so the
function parses the output of the `mount` utility to determine whether
this file system is already mounted at the desired location.

There can exist the possibility of unprivileged users performing certain
mount operations, one of the most prominent examples being the
`fusermount` setuid-root binary for mounting FUSE file systems. In view
of this, parsing mount table output needs prudence. I described the
basic issue previously already in another report [2].

The following is a reproducer using `fusermount` that shows the basic local
attack vector:

    user$ export _FUSE_COMMFD=0
```

```
    user$ fusermount some/path/ -ononempty,fsname="tmpfs on /var/lib/keylime/secure"
```

This will fool the parsing logic in `check_mounted()` and thus the function assumes that the "secure" tmpfs is already mounted, while it actually isn't. Thus this will allow a local attacker on the system to prevent this security feature to be effective, *if* the local attacker manages to create such a mount entry before the `keylime_agent` is starting up.

The attack vector can also be used to perform a local DoS against `keylime_agent` by claiming a different `fsname` than tmpfs. `check_mounted()` will throw an Exception in this case and the Agent won't start.

On a side note there are calls to `secure_mount.mount()` spread throughout the Keylime codebase (for example three times in `keylime_agent.py`, two times in `tpm_main.py` and two times in `ca_impl_cfssl.py`. There is no code to *clean up* this mount again, however. So it potentially leaves behind a stale mount after services are shutdown. Furthermore, if multiple Keylime processes should operate in parallel this could result in a race condition where the "secure" tmpfs is mounted twice, in the worst case mounting a fresh tmpfs over previously stored content there.

My recommendation is to parse the `/proc/self/mountinfo` pseudo file for mount table information instead. Whitespace is specially encoded in this file. Furthermore the responsibility of mounting and unmounting this file system should be more clearly defined during startup/shutdown of processes and maybe a reference counting / locking scheme to prevent race conditions should be used.

Upstream Security Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-wj36-qcfg-5j52

Upstream Fixes

https://github.com/keylime/keylime/commit/1a4f31a6368d651222683c9debe7d6832db6f607
https://github.com/keylime/keylime/commit/d37c406e69cb6689baa2fb7964bad75209703724

b) Possible Information Leaks via Unauthenticated Agent Quote Interface
---

A TPM quote can be requested without authentication from the Agent service via the network:

```
    $ curl "keyagent-host:9002/?api_version=500&quotes=myquote&nonce=mynonce"
    {
        "code": 200,
        "status": "Success",
        "results": {
            "quote": <base64-data>, "hash_alg": "sha256", "enc_alg": "rsa", "sign_alg":
"rsassa",
            "pubkey": <PEM key>", "boottime": 1639999864
        }
    }
```

This exposes for example the *boottime* of the host where the Agent is running, information that is not otherwise easily publicly available. Furthermore: Could the contents of the TPM quote data also be interesting data? Could it for example allow deductions about which kind of operating system kernel is running on the host?

I recommend to somehow authenticate and cryptographically secure this Agent interface to prevent information leaks of this kind.

Upstream Fixes

This issue did not receive a dedicated CVE and fix. It is covered
together with the following issue 2.c).

c) Arbitrary Remote Code Execution in the Agent via Unauthenticated Bootstrap Interface
(CVE-2021-43310)

Note that this issue has been discovered in parallel also by Thore
Sommer, a Keylime upstream developer.

It looks like it is possible to simply post arbitrary new values for the
U and V key parts and provide a new configuration payload to the Agent,
only knowing the Agent's UUID. The Agent's UUID can be public or
semi-public information like when `agent_uuid=hostname` is configured.
From the Keylime paper [3] (section 3.2.2) it sounds like the UUID HMAC
check is not considered a security feature but only a sanity check:

> This provides the node with a quick check to determine if Kb is correct.

When `extract_payload_script=true` (default) and
`payload_script=autorun.sh` (default) are configured in `keylime.conf`
then the provided payload will be unzipped and a potentially contained
`autorun.sh` script is executed with full root privileges. Attached you
can find a reproducer script `post_key.py` that demonstrates the issue
by creating a file `/tmp/evil` on the Agent host by only providing the
Agent hostname and UUID as input parameters.

Even if `payload_script` is disabled then the extraction of a ZIP file
as _root_ might result in a remote root exploit by extracting files
outside of the intended target directory. I did not test this variant of
the attack vector, though. Furthermore by providing a ZIP bomb as
payload the Agent process can be subjected to a remote DoS through
memory exhaustion.

Retrieving the full symmetric key previously stored in
`/var/lib/keylime/secure/derived_tci_key` should not be possible this
way, because when performing the bootstrap protocol, the previous data
is removed in `keylime_agent.py:242`. A skillful attacker might attempt
to first compromise the Agent node and then wait for the Tenant to
re-deploy the Agent using authentic keys and payload. Should this
succeed then the attacker can obtain the secret symmetric key from the
compromised Agent node after all.

Similar to issue b) I recommend to somehow authenticate and
cryptographically secure this Agent interface to prevent these attacks.
As a hotfix disabling the relevant configuration features should at
least prevent the remote code execution and memory exhaustion attack
vectors. Setting non-predictable UUID values can also help (but one
should also consider item 3.a in this context). Even then this
interface still allows to disrupt the operational state of the Agent
host by simply overwriting its current configuration.

Upstream Security Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-2m39-75g9-ff5r

Upstream Fixes

The fix consists of a larger number of upstream commits regarding
introduction of "mTLS" for the Agent interface. This means the
connection towards the Agent will in the future be cryptographically

secured and thus only trusted actors can use the Agent interface.

The upgrade path is a bit complicated because of this (see upstream advisory). Upstream version 6.3.X will introduce the new mTLS support but not enforce it, to allow upgrading of all Keylime components on all nodes. Only upstream version 6.4.x will enforce the new protocol.

d) Key Exchange and Bootstrap Protocol Susceptible to Replay Attacks
---

Authentic payloads being passed from the Tenant to the Agent should be reasonably safe from attackers (when not considering issue c)), since the two halves of the symmetric key are encrypted using the per-agent node RSA public key. The bootstrap protocol seems to be susceptible to certain replay attacks, however. Since the interface does not employ transport security, the bootstrap protocol can simply be recorded and replayed to activate an authentic configuration payload. This could e.g. be used by an attacker to activate an outdated or even insecure older configuration of the Agent node.

Upstream Fixes

This issue did not receive a dedicated CVE and fix. It is covered together with the previous issue 2.c).

3) Findings in the Registrar Component
==

a) UUID of Agents is Received on Unprotected HTTP Interface

The Registrar provides two separate HTTP interfaces, a TLS protected one and an unprotected one. Part of the unprotected interface is the Agent registration. As part of the Agent registration the Agent UUID is passed unencrypted (processed in `registrar_common.py:229`).

This is not a security issue in its own but relates to issue 2.c where the knowledge of the UUID facilitates remote code execution on the Agent nodes. This means if an attacker can listen in on the Registrar's Agent registration communication then even unpredictable Agent UUIDs no longer hinder the attack described in issue 2.c).

As outlined in 2.c) the UUID does not seem to have been thought of as a security property in the first place so I see no urge to change anything here. Although when the bootstrap protocol should get TLS protection then for completeness it could also make sense to protect this Registrar interface as well the same way.

Upstream Fixes

This specific aspect is covered by the following commit:

https://github.com/keylime/keylime/commit/e5f033c66403a899685b81a3af03cd59f76e455f

There is no dedicated CVE but it is covered together with the overarching introduction of mTLS as outlined in issue 2.c).

b) Unsanitized UUID passed on Unprotected HTTP Interface Facilitates Log Spoofing (CVE-2022-23949)

Since the Registrar's unprotected HTTP interface requires no authentication, anybody can post arbitrary Agent registrations with arbitrary parameters. The Agent ID (UUID) parameter is not sanitized in

any way and is used unfiltered in log messages (e.g.
`registrar_common.py:107`).

As a result the Agent ID parameter can be used to inject seemingly valid
additional log lines that appear e.g. in `journalctl -u
keylime_registrar.service`. The attached reproducer script
`post_agent.py` can be used to demonstrate this:

```
    $ ./post_agent.py --host registrar-host --log-line "Please run rm -rf /* to protect
your system"
```

In the journal we will then see:

```
    Dec 21 11:44:22 registrar-host keylime_registrar[1426]: 2021-12-21 11:44:22.281 -
keylime.registrar - WARNING -
POST for trusted-agent
    Dec 21 11:44:22 registrar-host keylime_registrar[1426]: 2021-12-21 11:44:22.931 -
keylime.registrar - WARNING -
Please run rm -rf /* to protect your system
    Dec 21 11:44:22 registrar-host keylime_registrar[1426]: 2021-12-21 11:44:22.940 -
keylime.registrar - DEBUG -
returning 400 response. [...]
```

Such log spoofing could be used to entice Administrators to perform
actions that can be harmful or otherwise in the interest of an attacker.

My recommendation is on the one hand to diligently sanitize untrusted
input parameters. On the other hand it might make sense to authenticate
this currently untrusted interface.

Upstream Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-87gh-qc28-j9mm

Upstream Fixes

The UUID sanitazion is introduced via these commits:

https://github.com/keylime/keylime/commit/387e320dc22c89f4f47c68cb37eb9eec2137f34b
https://github.com/keylime/keylime/commit/e429e95329fc60608713ddfb82f4a92ee3b3d2d9
https://github.com/keylime/keylime/commit/65c2b737129b5837f4a03660aeb1191ced275a57

Otherwise the introduction of mTLS as outlined in issues 3.a) and 2.c)
further protect this.

4) Findings in the Verifier Component

a) Revocation Notifier Uses Fixed /tmp Path for UNIX Domain Socket (CVE-2022-23950)

In *revocation_notifier.py* a fixed path in the world writable location
/tmp/keylime.verifier.ipc is used. The code (in this case the third
party `zeromq` Python module) forcefully removes any file object found
there earlier. Should the program be running as non-root, or if another
local user simply places a *directory* at this location, then this
serves as a local DoS attack against the revocation notifier process,
because the socket cannot be created.

This situation doesn't even seem to be noticed by the Verifier main
process, because the child process `broker_proc` is never waited on.
This means that the local attacker could even replace the "blocking"
directory by his own UNIX domain socket later on and will then receive
revocation events from invocations of the `notify()` function in the
main Verifier process. The full impact of this would have to be

researched further. It looks like failed quote notifications would
longer be sent out.

I recommend to place UNIX domains sockets in a dedicated safe directory
in /run that cannot be staged with attacks by other local users in the
system.

Upstream Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-9r9r-f8xc-m875

Upstream Fixes

This fix places the socket into a private /run/keylime directory:

https://github.com/keylime/keylime/commit/ea5d0373fa2c050d5d95404eb779be7e8327b911

b) Get Quote Response Contains Possibly Untrusted ZIP Data (CVE-2022-23951)

The Verifier process periodically performs quote operations on
registered Agents. As part of this `process_quote_response()` is called
and furthermore `check_quote()` and finally `_tpm2_checkquote()`. In
`tpm_main.py:1018` a couple of ZIP data streams are uncompressed via
`zlib.decompress()`.

Since this is processing possibly untrusted data - the Verifier is
attempting to verify the current trust status of the node after all - it
needs to be assumed that malicous data can also be supplied here.

Therefore the question arises whether `zlib.decompress()` is robust
against processing invalid ZIP data streams. One thing I already found
out is that it is not robust against delivering ZIP bombs that will
cause a memory exhaustion in the Verifier process.

This finding also is valid similarly for all other Keylime interface
that process ZIP data, like in the Agent.

Upstream Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-6xx7-m45w-76m2

Upstream Fixes

This fix simply removes the ZIP compression from the Verifier interface:

https://github.com/keylime/keylime/commit/6e44758b64b0ee13564fc46e807f4ba98091c355

5) General Findings
==========

This section contains findings that apply to all keylime components
alike.

a) World-Readable keylime.conf Contains Potentially Sensitive Data (CVE-2022-23952)

The configuration `/etc/keylime.conf` is installed world-readable:

```
$ ls -l /etc/keylime.conf
-rw-r--r-- 1 root root 26770 Dec 16 14:54 /etc/keylime.conf
```

This is the case for installations performed manually via the provided
`installer.sh` script as well as for the RPM packaging found in both
openSUSE Tumbleweed and Fedora 35 Linux distributions. Further

distributions might be affected.

`keylime.conf` contains a lot of information, some of it sensitive like the TPM ownership password (`tpm_ownerpassword`), TLS certificate private key passwords (`private_key_pw`, `registrar_private_key_pw`) or the database password for the Registrar (`database_password`). Thus this is a local information leak, because arbitrary local users can obtain these passwords from the configuration file.

My recommendation is to make this file only accessible to _root_ and adjust all installation routines and possibly documentation. The Keylime code could perform a sanity check of the permissions of the configuration file before reading it in.

Upstream Advisory

https://github.com/keylime/keylime/security/advisories/GHSA-fchm-5w2v-qfm8

Upstream Fixes

The following fix explicitly sets the permissions for the configuration file in the installer:

https://github.com/keylime/keylime/commit/883085d6a4bcea3012729014d5b8e15ecd65fc7c

b) Lack of Privilege Separation

All keylime services are currently designed to run as root all the time (except for testing purposes, see `REQUIRE_ROOT` in `config.py`). Only few bits of the keylime components actually should need root privileges. Most notably the bootstrapping scripts in the Agent component or the ability to bind privileged ports.

Implementing a privilege separation approach would increase the defense in depth for keylime considerably, avoiding smaller security issues to become severe fast.

Upstream Statement

Keylime upstream states that it is already possible to run Keylime as non-root. The `REQUIRE_ROOT` bits are not strictly necessary any more and can be removed from the code. The Debian packaging already makes use of the privilege separation.

6) Timeline
==========

2021-12-09: I started the review on the code
2021-12-23: I contacted the upstream security contact and upstream developer Thore Sommer privately by email and provided them the report results, offering coordinates disclosure.
2022-01-04: Upstream confirmed most of my findings and work on the fixes began. Alberto Planas, a SUSE colleague and maintainer of the SUSE Keylime packaging also contributed some fixes.
2022-01-28: Publication of the security advisories and fixes by upstream took place. Upstream also discovered some further security issues themselves in the meanwhile.

[1]: https://github.com/keylime/keylime
[2]: https://www.openwall.com/lists/oss-security/2020/06/04/5
[3]: https://www.ll.mit.edu/sites/default/files/publication/doc/2018-04/2016_12_07_SchearN_ACSAC_FP.pdf

Cheers

Matthias

--
Matthias Gerstner <matthias.gerstner () suse de>
Security Engineer
https://www.suse.com/security
GPG Key ID: 0x14C405C971923553

SUSE Software Solutions Germany GmbH
HRB 36809, AG Nürnberg
Geschäftsführer: Ivo Totev

Attachment: **post_agent.py**
Description:

Attachment: **post_key.py**
Description:

Attachment: **signature.asc**
Description:

By Date By Thread

Current thread:

keylime: Multiple Security Issues (including remote code execution in the Agent component) ***Matthias Gerstner (Jan 28)***