Sécurité ISI - Devoir 1
CVE-2018-11235 : Remote Code Execution using submodules

Ke LI, Yassir IDHBI, Yohan GOUZERH

Contents

1 Introduction

1.1 Introduction

La faille de sécurité identifiée *CVE-2018-11235* concerne le système de gestion de version decentralisé Git. Elle permet l'exécution d'un script sur la machine de l'utilisateur[1], lors d'une banale opération de clone d'un dépot malveillant contenant des submodules.

1.2 Enjeux

La vulnérabilité vise à exploiter un bug lors de l'analyse du fichier `.gitmodules`.

Ce fichier contient les références de l'ensemble des sous modules utilisés dans un projet. Lorsque l'on souhaite récupérer le contenu de ces sous-modules lors d'un `git clone --recurse-submodules`, git va ici executer un script malveillant sur la machine de l'utilisateur, sans action de ce dernier.

2 Notions techniques

Dans cette partie, nous allons rappeler quelques notions du fonctionnement de git, pour bien comprendre cette faille.

2.1 Le dossier .git local

Lorsque vous clonez un dépot distant, git va créer un répertoire en local contenant les sources du projet, ainsi qu'un dossier ".git" à la racine de se répertoire.

```
root@c5f9a67b7622:~/home/poc/fake_submodule/.git# ls
COMMIT_EDITMSG  branches  description  index  logs     refs
HEAD            config    hooks        info   objects
```

Ce dossier contient toute la configuration nécessaire pour votre projet, comme des informations sur les commits, les adresses des dépôts distants,...

Ce dossier est créé en local par votre git, et non récupéré du serveur. Vous pouvez le configurer pour vous, il ne modifiera que votre version en locale.

Par exemple, vous pouvez créer un fichier `projet/.git/info/exclude`[2] qui a exactement le même fonctionnement qu'un `local/.gitignore`, et vous permet d'exclure des fichiers du projet que pour vous, sans impacter les autres utilisateurs.[3]

2.2 Les hooks

Un des éléments modifiables de ce dossier `.git`, c'est la possibilité de créer des "hooks" dans le dossier `.git/hooks`. Ce sont des fichiers executables qui vont se lançer à des évenements particuliers.

Par exemple, le fichier `.git/hooks/prepare-commit-message` sera lancé lorsque vous executerez la commande : `git commit`, et avant que vous ne rentriez le message de commit.

Le but de ces scripts est d'automatiser votre processus de développement. Exemple : lancer une vérification avant de commit que vous n'avez pas écrit dans le fichier contenant les variables d'environnement, pour prévenir le push en ligne d'informations confidentielles comme votre mot de passe.

[1] Communément appelée Remote Code Execution
[2] Ici local est le nom de votre projet en local
[3] Ce fichier ne sera pas pushé par exemple

Pour rappel, ces fichiers sont présents dans votre `.git`, c'est-à-dire modifiables normalement que par vous. Par conséquent, seul vous pouvez controler ce que vous exécutez.

Dans le cas de cette faille, nous allons découvrir qu'un attaquant peut justement rajouter des hooks et les executer sans même que vous ne vous aperceviez du changement.

2.3 Les submodules

Pour comprendre comment un attaquant peut vous faire executer ce qu'il souhaite sur votre machine, il faut comprendre par quel moyen il peut le faire. Ici, la partie vulnérable du système utilisé est le mécanisme de submodules.

Les submodules permettent d'incorporer dans votre projet d'autres dépôt. Un peu à la manière de ce qu'offre la notion de package en python par exemple, où vous avez un projet général, qui utilise différentes librairies indépendantes de vôtre projet, qui continuent leur développement en parallèle, et que vous pouvez mettre à jour quand vous le désirez.

Par exemple, imaginons que vous décidiez avec vos amis de créer un site internet. Vous allez créer un dépôt git, avec tous vos fichiers html, js etc...

Vous décidez de vous séparer les taches : certaines personnes s'occuperont principalement de la création du système de paiement pendant que vous vous concentrez sur le reste du site. Ces personnes n'auront pas besoin de votre projet initial, elles vont donc pour être plus indépendantes et gérer leur module comme elles l'entendent. Elles vont donc créer leur propre repertoire git, qui pourra être en plus comme ça utilisé pour différents autre projets.

Pour incorporer leur module dans votre projet, vous allez l'ajouter comme submodule.

Vous allez executer la commande `git submodule add git@github.com:other_team/module_de_paiement.git paiement`, qui va vous créer un dossier paiement contenant les fichiers sources de l'autre git.

Ce submodule est rattaché au répertoire parent grâce au fichier `.gitmodules` qui liste les submodules, un peu à la manière d'un package.json pour npm (nodejs) ou d'un requirements.txt pour pip (python). Ici, ce fichier vous indique le chemin du dépôt initial du submodule, ainsi que le dossier en local où le contenu sera téléchargé.

```
[submodule "vendor/plugins/demo"]
  path = vendor/plugins/demo
  url = ../plugin
```

Figure 1: Aperçu du fichier .gitmodules

Vous pouvez aussi commit / push sur le submodule, grâce à la création d'un `.git` pour le submodule dans `.git/modules/paiements`, qui contient tout le contenu normal d'un dossier `.git`, c'est à dire des hooks, des informations sur les branches du submodules, voir même un répertoire modules avec d'autres submodules dedans,... (récursif).

3 Analyse de la faille

Comme nous l'avons dit, il est possible normalement de créer des hooks seulement dans le dossier `.git/hooks`, ou bien dans les `.git` des submodules (rappel: `.git/modules/nom_du_module/hooks`),

dossier modifiable que par vous, et non modifiable par le dépôt distant.

Ici, la vulnérabilité consiste en la possibilité d'exécuter un hook que vous n'avez pas créeé vous-même, mais stocké dans un dossier à la racine du projet.

Pour ce faire, il est possible de modifier le nom du submodule pour que ce dernier ait un nom relatif, comme par exemple `../../modules/malicious_submodule`.

Comme ça, quand git va vouloir créer le `.git/` du submodule, au lieu de créer un dossier `project/.git/modules/malici` il va créer un fichier `project/.git/modules/../../malicious_submodule` (soit `project/modules/malicious_submodule` Ce dossier est un dossier à la racine, contenant des dossiers hooks, branches, ... comme un vrai `.git/`, et pouvant être donc pushé et récupéré par les utilisateurs.

Ainsi, lorsque nous allons cloner le projet , git va executer les hooks pré-fabriqués du dossier `project/modules/maliciou`

Lors de l'exécution de la commande `git clone --recurse-submodules`, les actions réalisées chez le client sont les suivantes :

1. Création du .git et récupération des sources.
2. Exécution du hook `project/.git/hooks/post-checkout` (hook exécuté après chaque checkout)
3. Analyse du fichier `.gitmodules` et création des `.git/modules/XXX` si pas déjà fait
4. Récupération des submodules en utilisant les `.git/modules/XXX`
5. Execution du hook `project/.git/modules/XXX/hooks/post-checkout` (soit le script `project/modules/malicious` ici)

4 Exploitation de la faille grâce à notre PoC

Notre PoC consiste en une image Docker contenant une image ubuntu, ainsi qu'une version de git non patchée (v2.14.1).

Le conteneur ainsi créé génère un dépôt malveillant local. L'utilisateur peut alors ouvrir une session shell sur ce conteneur, et cloner le répertoire avec la commande `git clone --recurse-submodules`, et ainsi s'apercevoir de l'execution d'un script malveillant lors du clone de sa machine.

4.1 Création du dépôt

Le PoC réalise les actions suivantes pour la création d'un dépot malveillant :

1. Création d'un répertoire `fake_submodule` qui ne fait rien de particulier mais qui va nous permettre de lancer la commande git add submodule du côté serveur, afin créer le répertoire malicieux.
2. Création de notre répertoire `malicious_submodule`, le répertoire qui sera cloné par le client.
3. Ajout du répertoire créé précédemment comme submodule, en utilisant un chemin relatif (`../fake_submodule`) : cela est permis car le répertoire existe. (au passage, change le nom pour `malicious_submodule`, vu que ce submodule va nous servir à des fins malicieuses)
4. Copie du contenu du `.git/modules/malicious_submodules` dans un dossier modules à la racine du projet.
5. Copie dans ce dossier du script en lieu et place du hook de post-checkout.
6. Rajout de ce dossier au HEAD pour que le client le récupère.
7. Modification du nom du module `malicous_submodules` pour `submodule.../../modules/malicious_submodule` afin de le faire pointer vers notre `.git/` modifié (`modules/malicious_submodule`)

8. Ajout du `fake_submodule` en tant que submodule afin d'avoir un `.gitmodules` cohérent.
9. Commit pour que le client puisse cloner le repo.

4.2 Exploitation

Notre exploitation permet ici d'exécuter un script qui va copier la clé publique du serveur dans le fichier /.ssh/authorized_keys du serveur, puis de renvoyer le nom d'utilisateur ainsi que son adresse ip. Cela permet au serveur de se connecter à la machine du client en ssh sans avoir à connaître le mot de passe utilisateur.

5 Analyse post-mortem

5.0.1 Pourquoi cette faille a-t-elle eu lieu ?

Cette faille a eu lieu principalement suite à un bug dans le client Git qui ne vérifiait pas les entrées utilisateurs notamment dans le fichier .gitmodules, ce qui permet de faire une attaque de type **Path Traversal** à l'aide des opérateurs "../" ou des liens symboliques.

Celà est dû probablement au manque de tests qui a fait que les développeurs ne se sont pas rendu compte de la faille qui existait dans leur code, et de la liberté laissée à l'utilisateur sur la manipulation des entrées, sans les avoir traitées et filtrées.

On remarque d'ailleurs qu'avant le patch correctif, on ne testait que le path correspondant aux sous modules:

```
if (lookup_name) {
                        submodule = submodule_from_name(commit_sha1, path_or_name);
                } else
                        submodule = submodule_from_path(commit_sha1, path_or_name);
                if (!submodule)
                        die_usage(argc, argv, "Submodule not found.");
```

En conséquent, les tests qui ont été faits par les développeurs ne vérifient que si on pouvait chercher le sous module par le nom ou path. Par exemple:

```
test_expect_success 'test parsing and lookup of submodule config by name' '
        (cd super &&
                test-submodule-config --name \
                        HEAD^ a \
                        HEAD a \
                        HEAD^ submodule \
                        HEAD submodule \
                                >actual &&
                test_cmp expect actual
        )
```

5.0.2 Comment éviter à l'avenir l'apparition d'une faille comme celle-ci ?

Une équipe de testeurs devrait être mise en place pour vérifier les patchs émis par les développeurs, en plus de la review.

Les contraintes sur les entrées utilisateurs doivent être spécifiées avant de commencer le développement.

Des outils d'analyse statique doivent être utilisés, ainsi que des tests de fuzzing pour augmenter la robustesse du code par rapport aux saisies utilisateurs.

6 Analyse du patch

En général, cette vulnérabilité repose sur le fait que les entrées de l'utilisateur ne sont pas bien gérées. Le path consiste à filtrer le nom du module entré par l'utilisateur afin que le nom du module ne contienne aucun chemin. Cette modification rend le cœur de cette vulnérabilité, la traversée de répertoires, non viable.

```
int check_submodule_name(const char *name)
{
        /* Disallow empty names */
        if (!*name)
                return -1;
         /*
          * Look for '..' as a path component. Check both '/' and '\\' as
          * separators rather than is_dir_sep(), because we want the name rules
          * to be consistent across platforms.
          */
        goto in_component; /* always start inside component */
        while (*name) {
                char c = *name++;
                if (c == '/' || c == '\\') {
in_component:
                        if (name[0] == '.' && name[1] == '.' &&
                            (!name[2] || name[2] == '/' || name[2] == '\\'))
                                return -1;
                }
        }
         return 0;
}
```

```
static int check_name(int argc, const char **argv, const char *prefix)
{
        if (argc > 1) {
                while (*++argv) {
                        if (check_submodule_name(*argv) < 0)
                                return 1;
                }
        } else {
                struct strbuf buf = STRBUF_INIT;
                while (strbuf_getline(&buf, stdin) != EOF) {
                        if (!check_submodule_name(buf.buf))
                                printf("%s\n", buf.buf);
                }
                strbuf_release(&buf);
        }
        return 0;
}
```

Une nouvelle fonction de vérification(`check_submodule_name`) est ajoutée au projet. Lors de l'importation d'un sous-module, le nom du sous-module est vérifié, et l'utilisation des opérateurs "../" ou encore les liens symboliques est interdite, ce qui corrige le bug et rend la faille irréalisable.

Cependant ce patch ne corrige pas tout : il a été découvert une faille RCE[4] très simple sur cette vérification du nom, non complète (Si le nom commence par un tiret, la chaîne est traitée comme une option, et exécutée).

[4]Voir la CVE-2018-17456 pour plus de détails : https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-17456

7 Les conséquences de cette faille

7.1 Agents concernés

Sont concernés tous les systèmes utilisant une version non patchée de git, soit git avant 2.13.7, 2.14.x avant 2.14.4, 2.15.x avant 2.15.2, 2.16.x avant 2.16.4, et 2.17.x avant 2.17.1.

Cette faille concerne donc aussi bien des postes utilisateurs que des serveurs, peu importe le système d'exploitation utilisé.

Cette faille utilise le port ssh (22) ou bien le port https (443), des ports qui doivent rester ouvert (sauf ssh qui peut être fermé, mais alors l'utilisateur ne pourra se connecter en ssh sur d'autres machines, ni ici cloner ses dépôts en ssh).

7.2 Contre mesure

Pour contrer cette faille, il suffit de mettre à jour son client git vers une version patchée.

7.3 Menaces possibles

Les menaces qui pèsent sur un SI à cause d'une faille de type Remote Code Execution (connues aussi sous le nom Arbitrary Code Execution) sont très importantes.

Le script s'exécute avec les droits de l'utilisateur qui a lancé la commande git clone.

Même sans être root, les menaces sont multiples. Ce script peut effectuer toutes les actions possibles qu'un utilisateur est susceptible e faire.

Vol de données, suppression de données, création de keylogger (récupération des touches de l'utilisateur), transformation de la machine en machine zombie lorsque l'utilisateur se connecte (ordinateur contrôlé à distance).

Une façon très simple de garder la main sur la machine est par exemple de modifier le bashrc de la machine, en y rajoutant une commande qui sera executée à chaque ouverture de terminal, ou bien de rajouter sa clé publique aux fichiers `authorized_keys` pour pouvoir se connecter sans mot de passe.

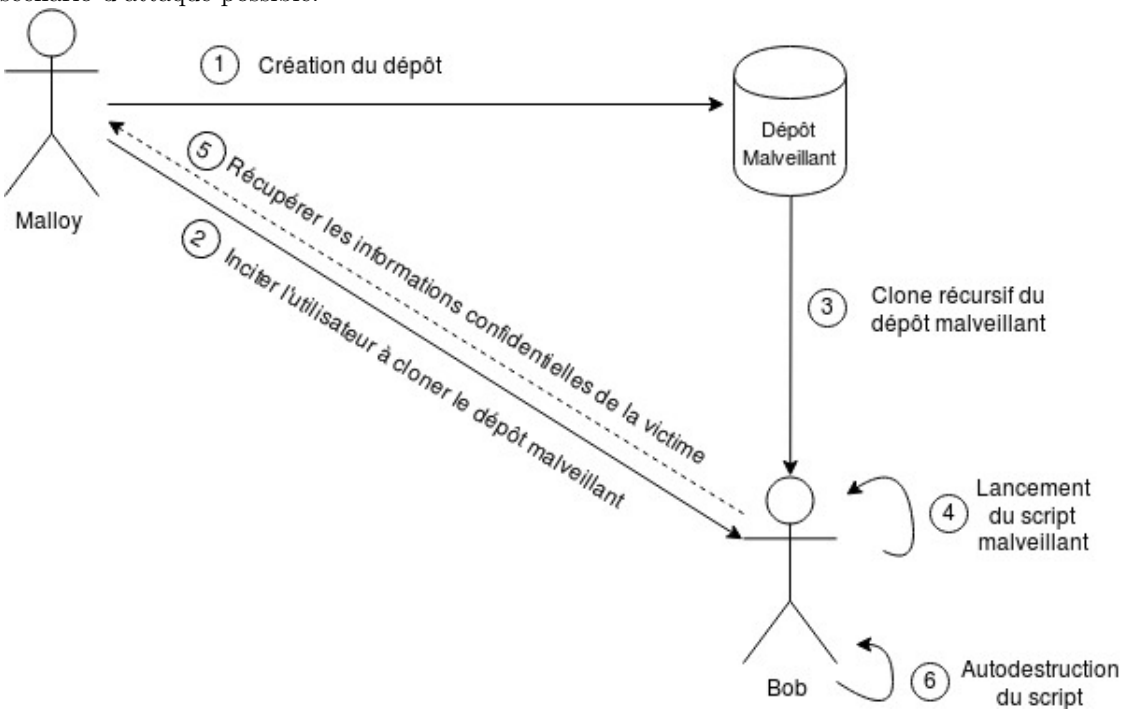
Les menaces sont plus importantes encore si la commande est exécuté en tant que root. L'attaquant peut réaliser toutes les actions qu'il souhaite. Si cet utilisateur est cependant un utilisateur lambda, mais qu'il est sudoer (utilisateur pouvant acquérir les droits root) sans mot de passe, ce script pourra acquérir les droits root aussi.

En étant root, l'attaquant a le contrôle total du système infecté.

8 Extrait de la Politique de Sécurité du Système d'Information

8.1 Scénario d'exploitation possible

Afin de contextualiser la Politique de Sécurité du Système d'Information, nous allons donner un exemple de scénario d'attaque possible.



Etape 2 : ici, il est possible d'inciter l'utilisateur à télécharger le dépôt grâce à de l'ingénierie sociale, ou de la simple publicité (le dépôt malveillant peut tout de même contenir du code utile).

Etape 5 : ici, lors d'un clone récursif sur un dépot malveillant fait par l'attaquant (Mallory dans notre exemple), le script malveillant sera lancé dans la machine victime de Bob ce qui va permettre à l'attaquant de parcourir la totalité des informations confidentielles exposées de Bob et qu'on peut y accéder par compte utilisateur, c'est-à-dire possiblement:

- Les fichiers de mots de passes sauvegardés par les navigateurs.
- L'accès aux variables d'environnement qui peuvent contenir des informations confidentielles (Identifiants de connexion pour certains sites web. Identifiants Amazon Web Services par exemple (souvent stockés dans le fichier `/.aws/credentials`))
- Accès à des fichiers de configurations
- Accès à des clés SSH privées, ce qui permettra à l'attaquant de se connecter en SSH à une autre machine si le réseau n'est pas assez isolé.
- Accès au fichier `.ssh/authorized_keys`, lui permettant de se connecter
- L'attaquant aura également accès aux applications de mails (Outlook, Thunderbird, ...) et pourra extraire les mails de l'utilisateur.

Etape 6 : Autodestruction du script (par exemple grâce à la simple commande `rm "$0"`), et effacement des logs pour supprimer ses traces.

8.2 Criticité de la faille

Au sein d'une entreprise, cette faille est critique car l'attaquant peut exécuter n'importe quel script malveillant, et éventuellement par exemple avoir la main sur le shell de la machine victime. En conséquence, l'attaquant peut tenter d'avoir un accès privilégié et tenter plusieurs manipulations qui peuvent nuire grandement à la continuation des services proposés par l'entreprise, ainsi qu'à son image.

8.3 Actions préventives

8.3.1 Ressources humaines

Une équipe de sécurité ayant pour rôle d'effectuer une veille sécurité et ainsi sécuriser le plus rapidement possible le SI doit être mise en place.

8.3.2 Architecture du SI

Le SI en entier doit être mis à jour régulièrement, de manière automatisée.

Les systèmes devront être isolés les uns des autres à l'aide de VLAN et de sous-réseaux afin de restreindre une possible infection d'un système.

Un IDS doit être mis en place pour observer les comportements inhabituels (connexions ssh,...).

8.3.3 Serveurs

Les ports des machines doivent être restreints au strict minimum.

Les serveurs ne doivent posséder que les informations minimales pour leur bon fonctionnement. (Configuration spécifique par serveur au lieu d'une configuration générale).

Les serveurs doivent être redondés, pour qu'il soit possible de les supprimer si ces derniers sont infectés, sans impacter le service.

8.3.4 Poste de travail

Les ports des machines doivent être restreints au strict minimum.

Des utilisateurs doivent être mis en place au lieu de laisser le compte root.

Tous les sudoers doivent avoir un mot de passe pour devenir sudo.

Les emails doivent être chiffrés, afin qu'un attaquant ne puissent les récupérer.

8.4 Actions curatives

8.5 Ressources humaines

Une équipe de sécurité ayant pour rôle de monitoré le système et préparé aux cas d'urgences devra être mise en place.

8.5.1 Continuité du service

Un backup doit être monté sur une autre machine dans un autre réseau, afin de redonder le service dans le cas d'un serveur.

La machine doit être isolée de tout autre système, au niveau physique s'il le faut.

8.5.2 Confidentialité

Chaque mot de passe utilisé par l'utilisateur devra être changé, ainsi que toutes les clés api qu'il a obtenue supprimées sur le service correspondant.

8.5.3 Légale

Informer le service juridique le plus tôt possible d'une attaque en cours. Ce service s'occupera de faire parvenir le formulaire de déclaration d'incident à l'ANSSI.

8.6 Formation des utilisateurs à envisager

Une équipe de sécurité ayant pour rôle de monitorer le SI et de réagir en cas d'attaques du système doit être mise en place.

8.6.1 Sécurisation des Password Managers

Les passwords managers doivent être sécurisés grâce à un master password, afin de stocker les passwords chiffrés et ainsi éviter une récupération des mots de passes en clair, dans le cas d'un accès à la machine.

8.6.2 Prévention des risques liés au clonage des répertories git

Les utilisateurs du SI doivent être conscients que ce genre de failles existe, et que cloner un répertoire git n'est pas sans risques. Il faut donc les former à bien utiliser ces systèmes de gestion de version afin de s'assurer que les dépots qu'ils vont cloner récursivement sur les machines du système d'information ne sont pas malveillants. Dans le cas de cette faille, il faut absolument vérifier le fichier .gitmodules pour être sur que les sous modules ne sont pas malicieux.

Les utilisateurs doivent être prévenus des risques associés à l'utilisation scripts hooks qui peuvent être lancés à des évenements particuliers. Pour un script récupéré de l'extérieur, l'utilisateur doit vérifier qu'il ne contient pas de manipulations dangereuses pour le système d'information.

8.7 Actions post-attaque

8.8 Conséquences sur le SI

Les machines infectés devront être reformatés pour éviter toute potentielles menaces. (Supprimer juste l'utilisateur infecté n'est pas assez sécurisé, une escalation de privilège a très bien pu avoir lieu).

8.8.1 Post-mortem

Une analyse post-mortem sera faite après l'attaque, avec le DSI, le RSI, les équipes de sécurité, les utilisateurs impactés. Suivant la gravité, des consultants experts pourront être engagés.

Cette analyse permettra de comprendre comment cette faille à pu avoir lieu et comment protéger le système d'une future attaque similaire.

Une analyse de l'action curative sera faite aussi, afin d'améliorer les processus de réactions aux attaques.

Cette analyse devra être documentée et archivée.

9 Glossaire

Analyse post-mortem : Analyse de la situation après qu'un évênement important à eu lieu et fut résolu.

ANSSI : Agence nationale de la sécurité des systèmes d'information.

Docker : Logiciel permettant de créer des conteneurs (entité logique isolé contenant la partie applicative d'une solution).

Git : Système de gestion de versions décentralisée.

DSI : Directeur des services informatiques.

Password manager: Logiciel de gestion de mot de passe.

PoC: Proof of Concept, un prototype créé pour démontrer la faisabilité de quelque chose.

Root : Nom donné à l'utilisateur qui possède toutes les permissions sur un système UNIX.

RSI : Responsable de la sécurité informatique.

SI : Système d'information.

Sudoers: Nom couramment utilisé pour désigner les utilisateurs pouvant acquérir les droits root.