Rapport d’analyse et détection de vulnérabilité

RANDRIANAHARISON Tolojanahary

STD22081

NB : La majorité des matériaux de cet analyse (scripts, vidéos, …) sont dans le repo github suivant : https://github.com/kirisaki-vk/wolfssh-devcontainer

Préparation de l'environement

L'environement requis pour commencer à expérimenter sur ces exploits est déjà déclaré dans *ce fichier docker*.

Pour commencer, cloner le repo contenant les fichiers nécessaires pour le serveur et les scripts des exploits.

```
$ git clone https://github.com/kirisaki-vk/wolfssh-devcontainer.git
```

Entrez dans le repo cloné

Ensuite buildez l'image Docker avec ce commande le tag de l'image vous permettra de démarrer le container plus facilement

```
$ docker build -t <tag-de-l'image> .
```

Après avoir exécuté cette commande, vous pouvez exécuter l'echoserver avec la commande

```
$ docker run -it -p 22222:22222 --rm <tag-de-l'image>
```

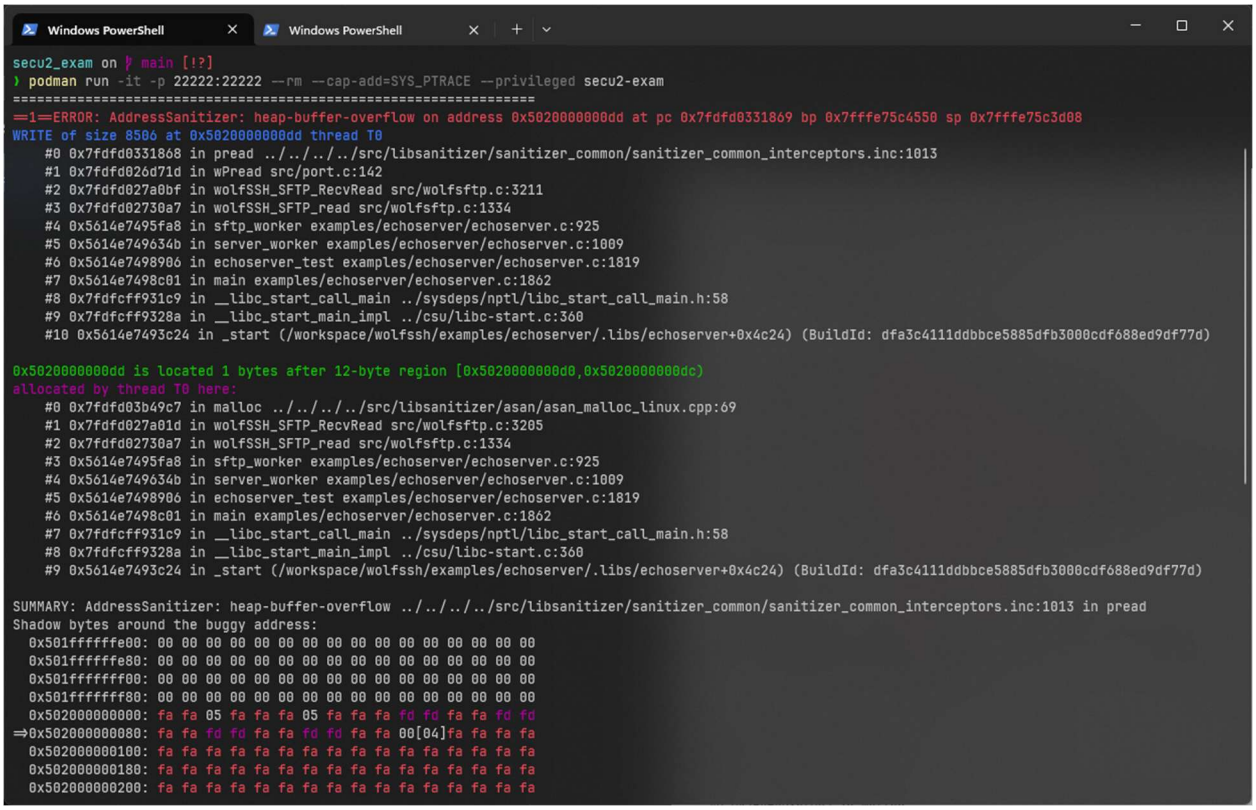
Pour l'environement python créez seulement un venv dans le dossier que vous désiriez et installez paramiko dessus et vous pouvez exécuter ces scripts d'exploit avec python

```
$ python **/exploits/*.py
```

Analyse des vulnérabilités

1. wolfSSH_SFTP_RecvRead

a. Script d'exploit : wolfSSH_SFTP_RecvRead_exploit
b. Type de vulnérabilité : Heap buffer overflow
c. Message ASAN :



d. Cette vulnérabilité est causé à cause d'un integer overflow sur la taille de la lecture à effectuer sur le fichier

```
3201    /* get length to be read */
3202    ato32(data + idx, &sz);
3203
3204    /* read from handle and send data back to client */
3205    out = (byte*)WMALLOC(sz + WOLFSSH_SFTP_HEADER + UINT32_SZ,
3206            ssh->ctx->heap, DYNTYPE_BUFFER);
3207    if (out == NULL) {
3208        return WS_MEMORY_E;
3209    }
3210
3211    ret = WPREAD(fd, out + UINT32_SZ + WOLFSSH_SFTP_HEADER, sz, ofst);
3212    if (ret < 0 || (word32)ret > sz) {
3213        WLOG(WS_LOG_SFTP, "Error reading from file");
3214        res  = err;
3215        type = WOLFSSH_FTP_FAILURE;
3216        ret  = WS_BAD_FILE_E;
3217    }
```

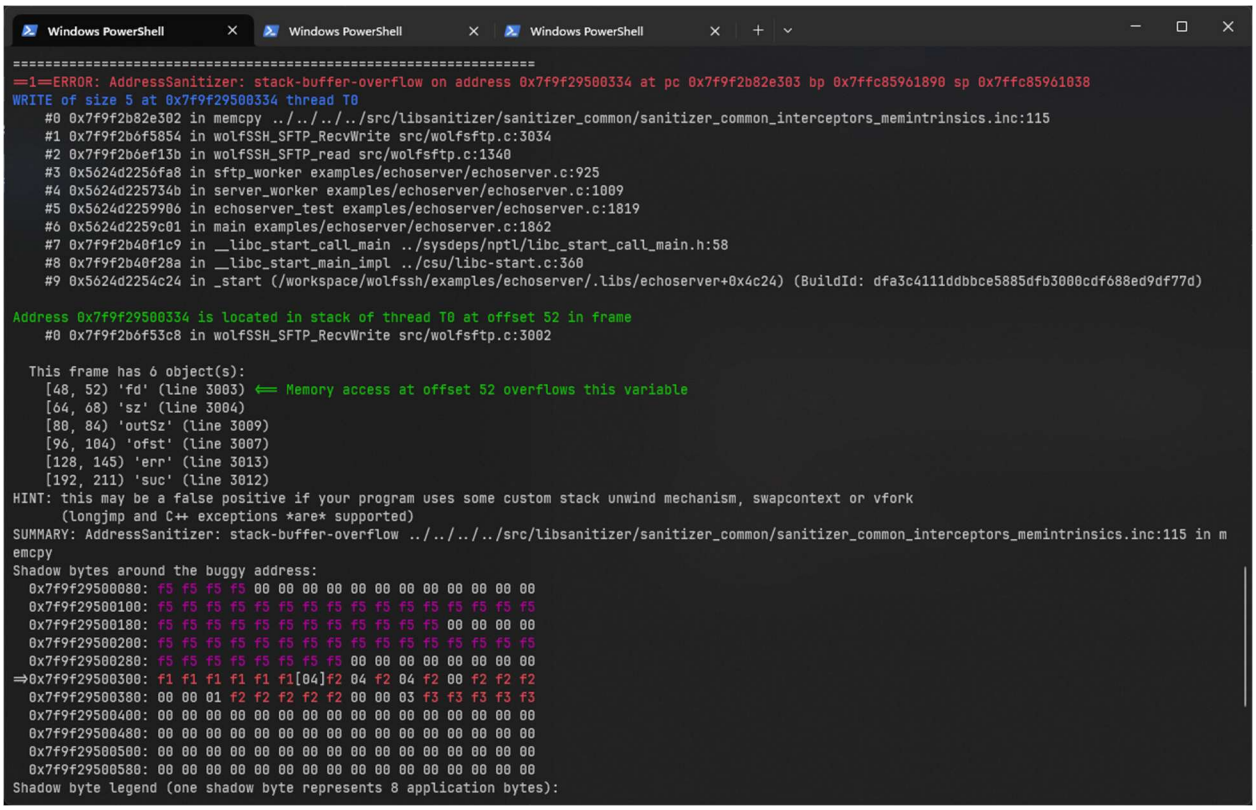
e. Sur la ligne 3202 on prends la taille du buffer dans la variable sz qui est de type word32 alors il ne peut contenir que les nombres unsigned de 0-255
Hors sur la ligne 3205 on effectue une addition sur la variable sz ce qui va causer le débordement d'entier et va donner un résultat erroné pour WMALLOC qui va allouer une taille erroné pour out. Puis out va être écrit avec WPREAD avec une taille de sz qui est largement plus grand que le buffer alloué (out) ce qui va causer l'overflow.

2. wolfSSH_SFTP_RecvWrite

a. Script d'exploit : [wolfSSH_SFTP_RecvWrite_exploit]

b. Type de vulnérabilité : Stack buffer overflow

c. Message ASAN :



d. Cet erreur est causé en reportant au server un file_handle plus de 4octets alors que cet valeur est stocké dans une variable de taille de 4octets.

```
3023    /* get file handle */
3024    ato32(data + idx, &sz); idx += UINT32_SZ;
3025    if (sz + idx > maxSz || sz > WOLFSSH_MAX_HANDLE) {
3026        WLOG(WS_LOG_SFTP, "Error with file handle size");
3027        res  = err;
3028        type = WOLFSSH_FTP_FAILURE;
3029        ret  = WS_BAD_FILE_E;
3030    }
3031
3032    if (ret == WS_SUCCESS) {
3033        WMEMSET((byte*)&fd, 0, sizeof(WFD));
3034        WMEMCPY((byte*)&fd, data + idx, sz); idx += sz;
3035
3036        /* get offset into file */
3037        ato32(data + idx, &ofst[1]); idx += UINT32_SZ;
3038        ato32(data + idx, &ofst[0]); idx += UINT32_SZ;
3039
3040        /* get length to be written */
3041        ato32(data + idx, &sz); idx += UINT32_SZ;
3042
3043        ret = WPWRITE(fd, data + idx, sz, ofst);
```

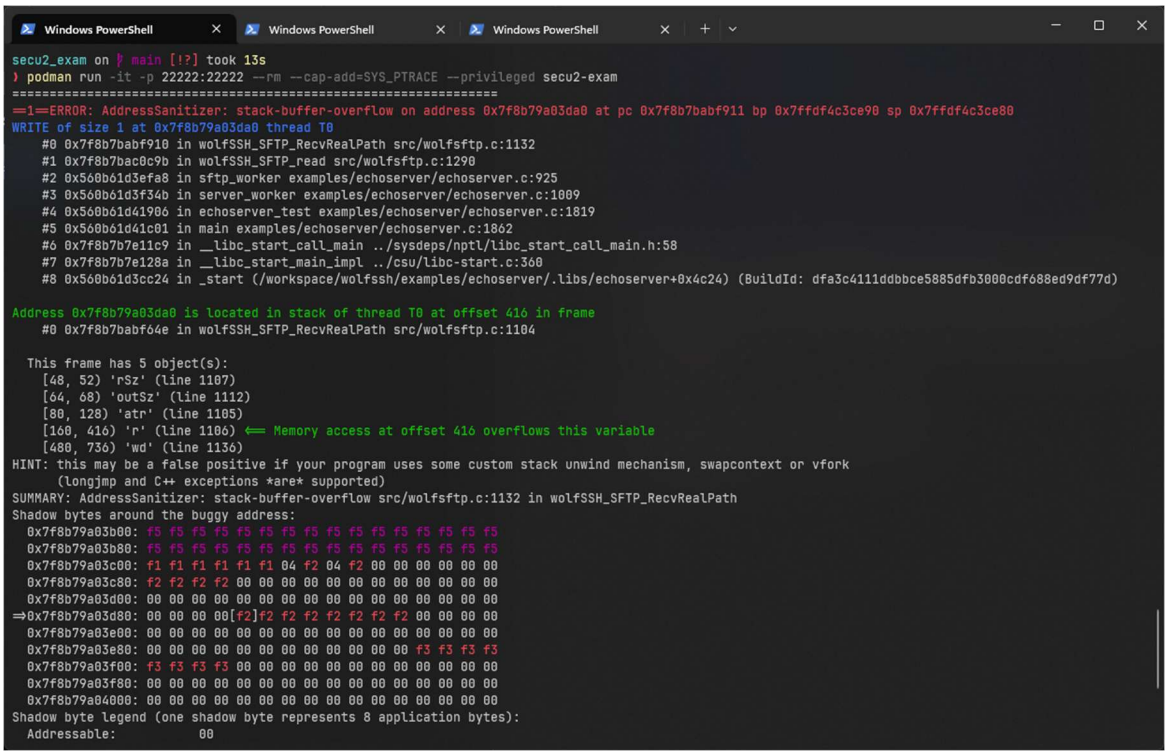
Le code stocke la taille du file handle dans la variable sz puis ensuite sur la ligne 3033 va initialiser la valeur de fd en 4octets, alors durant l'instruction suivant WMEMCPY va copier des donnés du data vers fd de taille sz alors que si on déclare la taille de file handle du côté client à une valeur plus de 4octets alors fd va causer une overflow car elle n'est initialisé que pour contenir 4octets de donnés ce qui cause l'overflow.

3. wolfSSH_SFTP_RecvRealPath

a. Script d'exploit : [wolfSSH_SFTP_RecvRealPath_exploit]

b. Type de vulnérabilité : Stack buffer overflow

c. Message ASAN :



d. Cette vulnérabilité est causé à cause de l'écriture en dehors de la variable r dans le cas où la taille du chemin du fichier est exactement de 256

```
1126    ato32(data + lidx, &rSz);
1127    if (rSz > WOLFSSH_MAX_FILENAME || (int)(rSz + UINT32_SZ) > maxSz) {
1128        return WS_BUFFER_E;
1129    }
1130    lidx += UINT32_SZ;
1131    WMEMCPY(r, data + lidx, rSz);
1132    r[rSz] = '\0';
```

ce qui va copier 256 bytes dans la variable r qui a comme taille maximum 256 puis on accède à r[rSz(256)] qui est un index en dehors du buffer de taille 256 ce qui a causé l'overflow.

Conclusion et recommendations

La majorité de ces failles sont causé à cause manque validation des tailles des donnés spécifié par les clients, surtout sur la taille des buffer où on compte stocker les donnés venant du client.

La solution proposer est de renforcer chaque vérification de la taille de chaque buffer mais aussi de bien valider les entrés données par les clients pour éviter ces comportements du programme indésirables.