



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'environnement

L'environnement 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'échoserver avec la commande

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

Pour l'environnement 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 :

```

Windows PowerShell
secu2_exam on ? main [!?]
> podman run -it -p 22222:22222 --rm --cap-add=SYS_PTRACE --privileged secu2-exam
=====
==1==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x5020000000dd at pc 0x7fd0331869 bp 0x7ffe75c4550 sp 0x7ffe75c3d08
WRITE of size 8506 at 0x5020000000dd thread T0
#0 0x7fd0331868 in fread ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1013
#1 0x7fd0331867 in wPread src/port.c:142
#2 0x7fd0331866 in wolfSSH_SFTP_RecvRead src/wolfssh.c:3211
#3 0x7fd0331865 in wolfSSH_SFTP_read src/wolfssh.c:1334
#4 0x5614e7495fa8 in sftp_worker examples/echoserver/echoserver.c:925
#5 0x5614e749534b in server_worker examples/echoserver/echoserver.c:1009
#6 0x5614e7498906 in echoserver_test examples/echoserver/echoserver.c:1819
#7 0x5614e7498c01 in main examples/echoserver/echoserver.c:1862
#8 0x7fd0331869 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
#9 0x7fd0331868 in __libc_start_main_impl ../csu/libc-start.c:360
#10 0x5614e7493c24 in _start (/workspace/wolfssh/examples/echoserver/.libs/echoserver+0x4c24) (BuildId: dfa3c411ddbce5885dfb3000cdf688ed9df77d)

0x5020000000dd is located 1 bytes after 12-byte region [0x5020000000d0,0x5020000000dc)
allocated by thread T0 here:
#0 0x7fd0331867 in malloc ../../src/libsanitizer/asan/asan_malloc_linux.cpp:69
#1 0x7fd0331866 in wolfSSH_SFTP_RecvRead src/wolfssh.c:3205
#2 0x7fd0331865 in wolfSSH_SFTP_read src/wolfssh.c:1334
#3 0x5614e7495fa8 in sftp_worker examples/echoserver/echoserver.c:925
#4 0x5614e749534b in server_worker examples/echoserver/echoserver.c:1009
#5 0x5614e7498906 in echoserver_test examples/echoserver/echoserver.c:1819
#6 0x5614e7498c01 in main examples/echoserver/echoserver.c:1862
#7 0x7fd0331869 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
#8 0x7fd0331868 in __libc_start_main_impl ../csu/libc-start.c:360
#9 0x5614e7493c24 in _start (/workspace/wolfssh/examples/echoserver/.libs/echoserver+0x4c24) (BuildId: dfa3c411ddbce5885dfb3000cdf688ed9df77d)

SUMMARY: AddressSanitizer: heap-buffer-overflow ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:1013 in fread
Shadow bytes around the buggy address:
0x501fffffe000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x501fffffe010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x501fffffe020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x501fffffe030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x502000000000: fa fa 05 fa fa 05 fa fa fa fa fa fa fa fa fa fa
=>0x502000000008: fa fa fa fa fa fa fa fa fa fa 00[04]fa fa fa fa
0x502000000010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x502000000018: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x502000000020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa

```

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 :

```
=====
==1==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7f9f29500334 at pc 0x7f9f2b82e303 bp 0x7ffc85961898 sp 0x7ffc85961038
WRITE of size 5 at 0x7f9f29500334 thread T0
#0 0x7f9f2b82e302 in memcpy ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors_memintrinsics.inc:115
#1 0x7f9f2b6f5854 in wolfSSH_SFTP_RecvWrite src/wolfssftp.c:3034
#2 0x7f9f2b6ef13b in wolfSSH_SFTP_read src/wolfssftp.c:1340
#3 0x5624d2256fa8 in sftp_worker examples/echoserver/echoserver.c:925
#4 0x5624d225734b in server_worker examples/echoserver/echoserver.c:1009
#5 0x5624d2259906 in echoserver_test examples/echoserver/echoserver.c:1819
#6 0x5624d2259c01 in main examples/echoserver/echoserver.c:1862
#7 0x7f9f2b40f1c9 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
#8 0x7f9f2b40f28a in __libc_start_main_impl ../csu/libc-start.c:360
#9 0x5624d2254c24 in _start (/workspace/wolfssh/examples/echoserver/.libs/echoserver+0x4c24) (BuildId: dfa3c411ddbbce5885dfb3008cdf688ed9df77d)

Address 0x7f9f29500334 is located in stack of thread T0 at offset 52 in frame
#0 0x7f9f2b6f53c8 in wolfSSH_SFTP_RecvWrite src/wolfssftp.c:3002

This frame has 6 object(s):
[48, 52) 'fd' (line 3003) ← Memory access at offset 52 overflows this variable
[64, 68) 'sz' (line 3004)
[80, 84) 'outSz' (line 3009)
[96, 104) 'ofst' (line 3007)
[128, 145) 'err' (line 3013)
[192, 211) 'suc' (line 3012)
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow ../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors_memintrinsics.inc:115 in m
memcpy
Shadow bytes around the buggy address:
 0x7f9f29500080: f5 f5 f5 f5 00 00 00 00 00 00 00 00 00 00 00 00
 0x7f9f29500100: f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5
 0x7f9f29500180: f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 00 00 00 00
 0x7f9f29500200: f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5
 0x7f9f29500280: f5 f5 f5 f5 f5 f5 f5 f5 00 00 00 00 00 00 00 00
 0x7f9f29500300: f1 f1 f1 f1 f1 f1 f1 f1 04 f2 04 f2 00 f2 f2 f2
 0x7f9f29500380: 00 00 01 f2 f2 f2 f2 00 00 03 f3 f3 f3 f3 f3 f3
 0x7f9f29500400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x7f9f29500480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x7f9f29500500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x7f9f29500580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
```

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 données 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 données 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 :

```

secu2_exam on | main [??] took 13s
) podman run -it -p 22222:22222 --rm --cap-add=SYS_PTRACE --privileged secu2-exam
=====
==1==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7f8b79a03da0 at pc 0x7f8b7babf911 bp 0x7ffdf4c3ce90 sp 0x7ffdf4c3ce80
WRITE of size 1 at 0x7f8b79a03da0 thread T0
#0 0x7f8b7babf910 in wolfSSH_SFTP_RecvRealPath src/wolfssh.c:1132
#1 0x7f8b7bac0c9b in wolfSSH_SFTP_read src/wolfssh.c:1290
#2 0x560b61d3efa8 in sftp_worker examples/echoserver/echoserver.c:925
#3 0x560b61d3f34b in server_worker examples/echoserver/echoserver.c:1809
#4 0x560b61d419b6 in echoserver_test examples/echoserver/echoserver.c:1819
#5 0x560b61d41c01 in main examples/echoserver/echoserver.c:1862
#6 0x7f8b7b7e11c9 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
#7 0x7f8b7b7e128e in __libc_start_main_impl ../csu/libc-start.c:360
#8 0x560b61d3cc24 in _start (/workspace/wolfssh/examples/echoserver/.libs/echoserver+0x4c24) (BuildId: dfa3c411ddbbce5885dfb3000cdf688ed9df77d)

Address 0x7f8b79a03da0 is located in stack of thread T0 at offset 416 in frame
#0 0x7f8b7babf64e in wolfSSH_SFTP_RecvRealPath src/wolfssh.c:1104

This frame has 5 object(s):
[48, 52) 'rSz' (line 1107)
[64, 68) 'outSz' (line 1112)
[80, 128) 'atr' (line 1105)
[160, 416) 'r' (line 1106) ← Memory access at offset 416 overflows this variable
[480, 736) 'wd' (line 1136)
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
(Longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow src/wolfssh.c:1132 in wolfSSH_SFTP_RecvRealPath
Shadow bytes around the buggy address:
0x7f8b79a03b00: f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5
0x7f8b79a03b80: f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5 f5
0x7f8b79a03c00: f1 f1 f1 f1 f1 f1 04 f2 04 f2 00 00 00 00 00 00
0x7f8b79a03c80: f2 f2 f2 f2 00 00 00 00 00 00 00 00 00 00 00 00
0x7f8b79a03d00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
⇒0x7f8b79a03d80: 00 00 00 00[f2]f2 f2 f2 f2 f2 f2 f2 00 00 00 00
0x7f8b79a03e00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x7f8b79a03e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 f3 f3 f3
0x7f8b79a03f00: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
0x7f8b79a03f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x7f8b79a04000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00

```

- 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 recommandations

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

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