VMware Workstation/Player Host Local Privilege Escalation Vulnerability

Background:

This vulnerability affects VMware Workstation/Player installations due to an incomplete fix for VMSA-2019-0002/CVE-2019-5511 which was reported by James Forshaw as issue 1733. The bypass has been reported privately to VMware PSIRT on July 30 2021, the team triaged but was unable to fix the vulnerability.

In March 3rd 2022, the team replied with the following statement :

"Your effort has been appreciated, but this is an accepted risk on VMware Workstation Fusion and they are fine with the current behavior of this functionality.

As a consequence, I am closing this as Informative.
Best regards,"

I replied claiming that such decision for a valid vulnerability is not recommended, I insisted that this is a valid privileges escalation bug that must be fixed.

The team replied again, stating:
"We have reviewed the new information provided by you. We see hijacking the C: drive to be outside our threat boundary. Attacker's access to the system can allow any tampering to mount points and are beyond the control of product's fix. Hence we are not considering this as a product security vulnerability and closing this ticket as informative.
We appreciate your sincere effort and would be happy to work with you in the future.
Thanks,"

I replied back claiming that hijacking C: is not a security boundary as long as it doesn't lead to security vulnerabilities.
The team did not reply back which led me to dismiss the case.

As of the time of writing this (05/03/2022) VMware team has shown incompetency and left a valid high severity vulnerability unfixed in their products.

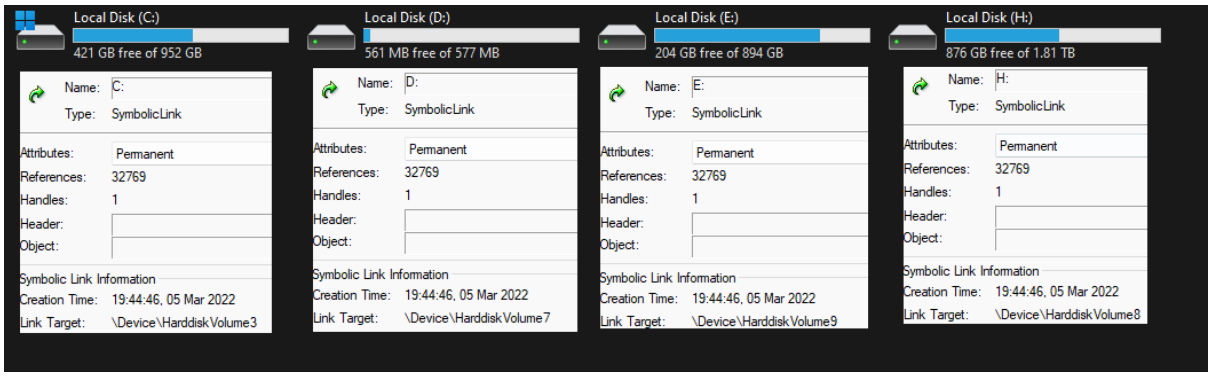
Details:

The first bug to hijack VMware-vmx process was first found by a research which I believe named Sun Bing, he demonstrated his research in CanSecWest in 2008.
The first bug occurred because VMware daemon service used to read the path of VMware-vmx binary from a user-writeable configuration file, the vulnerability has been fixed as CVE-2008-1363.
Forshaw later found that a user can still hijack the process leading to elevation of privileges.

But how ?

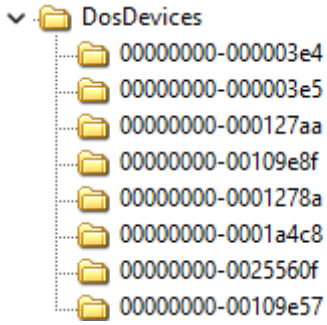
This is related to the symbolic link mapping of drives.
I will try to give a short explanation of this, noting that this is mainly based on [Forshaw blog]

Each drive that exists on Windows is not actually a “real” drive but it is just an object manager symbolic link located in “\Global??” which points to a device object.



Symbolic links were implemented here because the OS needed to recognise each drive, because relying on enumerating each device object or choosing a specific name can lead to undefined behaviour because device names changes depending on the changes made to partitioning.
This implementation did know some big changes, when the major version of Windows XP was released, it allowed multiple users to be logged in, in the same time and new features such as Remote Desktop Protocol (RDP) and Terminal Services with that new issues were introduced.
The problem was, a user needed to map a drive or a specific network resource, this was not an issue for pre-XP as they didn’t support multiple users to be logged in.

The solution was to create a DosDevices object manager directory which is by default located in \Sessions\0\DosDevices which contains multiple directories, the names may look random but each name points to a windows token Authentication ID. You can call GetTokenInformation with TokenStatistics to retrieve the Authentication ID.



This solution was successful to separate each user dosdevice, now if a user needed to map a drive, he could map it for himself because giving a user to map a drive globally is considerably dangerous.
Windows kernel creates this directory for every token that has been impersonated or used to create a process in the first time to ensure the separation.
Windows kernel also has a powerful mechanism when it comes to resolving paths, when opening a native path \??\D: , the kernel first look in the DosDevices directory with the corresponding authentication id for the process token. But if we actually looked inside it, it may only contain the Global symbolic link which points to \Global??
The kernel actually checks for the symbolic link in the corresponding directory if doesn't exist, then it proceeds to look in \Global??
While a user can create a non-existing drive for himself, he can also override an existing one by calling NtCreateSymbolicLink. While this overriding functionality is might seem completely unnecessary, it is maintained to ensure the stability if a user had his own drive mapped then later a global drive is mapped with the same name so the kernel doesn't have to worry about having two symbolic links with the same name.
This has actually introduced several issues, by example this bug which was found by Forshaw. If a service was actually impersonating a user it may follow its mapped drive and may lead to being redirected to a fake resource as for the bug the resource was the dll which led to compromising a process running as LocalSystem.

For us this means a lot, the VMware-vmx process is created by the daemon service vmware-authd.exe a.k.a VMware authorisation service. The vmware-vmx process is created in the context of the current user but with a modified token integrity which is raised to SYSTEM.
This is done to make sure that certain virtual machine properties and objects may still be accessible to the process but also prevent the user from tampering with them.
Beside the integrity level, the vmware-vmx also has special trust to certain vmware components which can be abused if the process was compromised.

Now going back to the actual bug, the vmware-vmx is by default installed in a user read-only location : “C:\Program Files (x86)\VMware\VMware Workstation\x64\vmware.exe”
So hijacking it replacing the executable file may not work, but looking at Forshaw bug, he abused the fact that the process creation is done while impersonating the user.
This means that a user can override the C: drive to make sure it points to a fake location and hence pointing to a user controllable binary vmware-vmx.
VMware implemented path checking to mitigate the vulnerability as observed below

This code create a new primary token and raise its integrity level to SYSTEM

```
if ( !DuplicateTokenEx(v17, 0xF01FFu, 0, SecurityImpersonation, TokenPrimary, &phNewToken) )// Duplicate token
{
  v18 = GetLastError();
  sub_4027C0(2, "DuplcateTokenEx failed: %d.\n", v18);
  free(0);
  goto LABEL_25;
}
v19 = phNewToken;
*(_DWORD *)pIdentifierAuthority.Value = 0;
*(_WORD *)&pIdentifierAuthority.Value[4] = 4096;
ReturnLength = 0;
v20 = 0;
v49 = 0i64;
if ( AllocateAndInitializeSid(&pIdentifierAuthority, 1u, 0x4000u, 0, 0, 0, 0, 0, 0, 0, (PSID *)&ReturnLength) )
{
  v49 = ReturnLength | 0x6000000000i64;
  if ( SetTokenInformation(v19, TokenIntegrityLevel, &v49, 8u) )// Raise token integrity to SYSTEM
  {
    v20 = 1;
  }
  else
  {
    v22 = GetLastError();
    Warning("%s: SetTokenInformation failed: %u\n", "SetTokenIntegrityLevel", v22);
  }
}
```

The newly created token is used for the next operations, the service first open the vmware-vmx file without impersonation look for its path using GetFinalPathNameByHandle

```
NtFinalPath = (void *)File_GetNTGlobalFinalPath(a3);// Open vmware-vmx file and call GetFinalPathNameByHandle
if ( !NtFinalPath )
{
  v24 = GetLastError();
  sub_4027C0(2, "Before: File_GetNTGlobalFinalPath failed for %s: %d\n", a3, v24);
  free(0);
  free(0);
  goto LABEL_61;
}
if ( !ImpersonateLoggedOnUser(phNewToken) )   // Impersonate the newly created token
{
  v25 = GetLastError();
  sub_4027C0(2, "ImpersonateLoggedOnUser failed: %d\n", v25);
  free(NtFinalPath);
ABEL_25:
  free(0);
  goto LABEL_61;
}
NTFinalPath2 = (const char *)File_GetNTGlobalFinalPath(a3);// Open vmware-vmx file and call GetFinalPathNameByHandle while impersonating the use
v27 = (char *)NTFinalPath2;
ReturnLength = (DWORD)NTFinalPath2;
if ( !NTFinalPath2 )
{
  v28 = GetLastError();
  sub_4027C0(2, "After: File_GetNTGlobalFinalPath failed for %s : %d.\n", a3, v28);
  free(NtFinalPath);
  free(0);
  goto LABEL_61;
}
if ( stricmp((const char *)NtFinalPath, NTFinalPath2) )
{
  sub_4027C0(2, "Failed to validate vmx file \"%s\", abort to launch it.\n", a3);
  free(NtFinalPath);
  free(v27);
  goto LABEL_61;
}
AllocBytes = (WCHAR *)Unicode_GetAllocBytes(a3, 1);
v30 = (WCHAR *)Unicode_GetAllocBytes(v52, 1);
v42 = CreateProcessAsUserW(                   // if the both paths matches, the process is created
        phNewToken,
        AllocBytes,
        v30,
        0,
        0,
        0,
        0x414u,
        Block[3],
        0,
        &StartupInfo,
        &ProcessInformation);
free(AllocBytes);
free(v30);
```

Then it impersonated the user and checked for the file path again, if they match, the process is created.

This check is vulnerable to a TOCTOU, a user may replace the C: drive after check is done which results in being redirected and executing arbitrary binary with elevated privileges.

The proof of concept which I submitted to VMware team was instable because it was a race condition, developing proof of concept isn't something easy thus VMware team was not able to investigate the issue any further because they were unable to reproduce the issue.

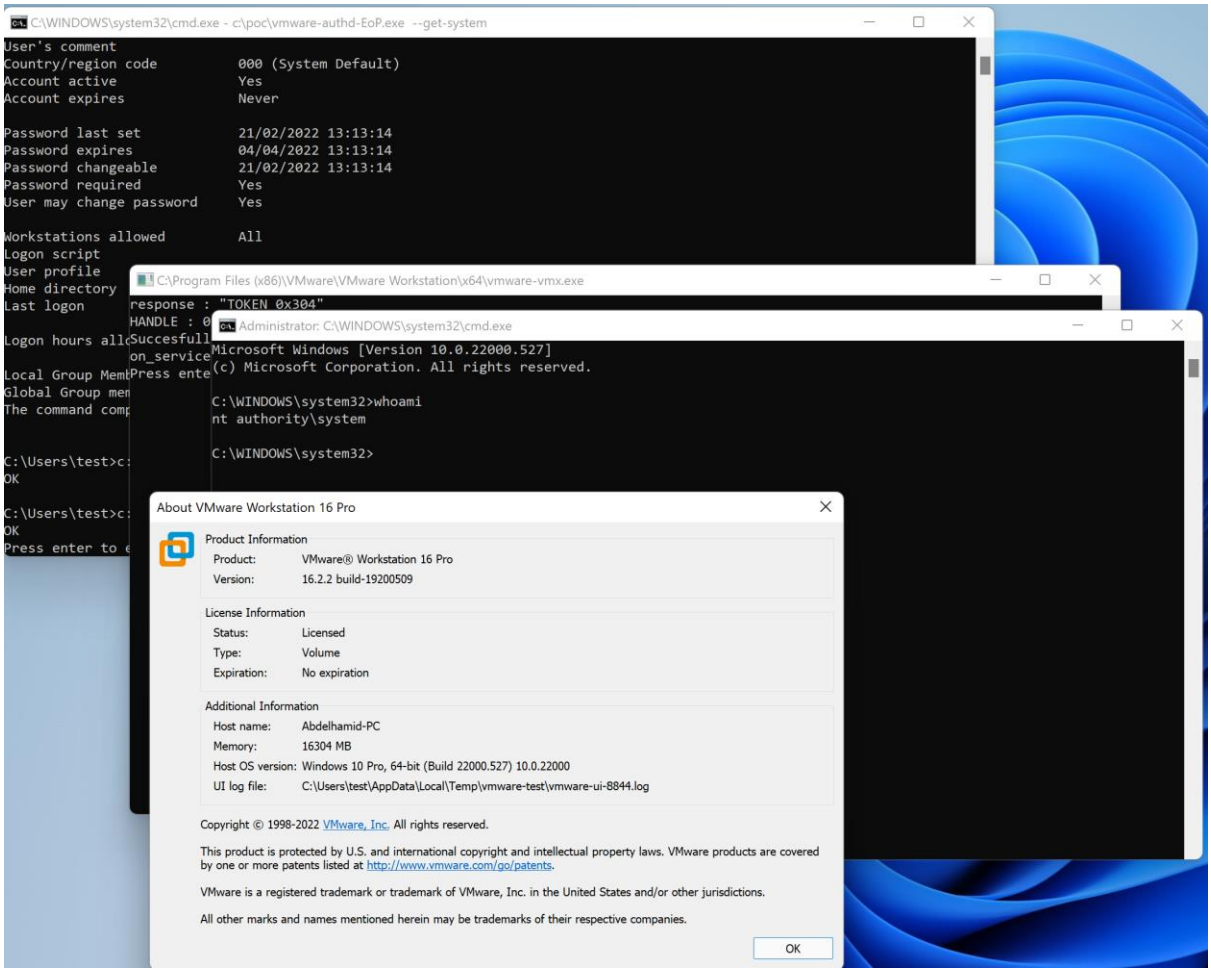
After a considerable amount of time, I decided to revisit the bug and I created a stable PoC which actually hijacked dll load in vmware-vmx process instead of relying on hijacking the creation, this worked surprisingly well.

Now what ? I mean we are running with SYSTEM integrity but still running as the user account, this actually may allow us to bypass a lot of windows kernel access checks but I really couldn't remember where I found those checks :((((

For now, we will be moving to abusing the VMware authorisation service instead of abusing windows for elevation of privileges.
The VMware authorisation service exposes a named pipe named vmware-authdpipe which allows authenticated users to execute commands.
One of the interesting commands in opensecurable which allows you to open any file as SYSTEM, of course this isn't allowed for anyone but only for vmware-vmx process.
If the pid of the trusted vmware-vmx process matches yours, the vmware-authd duplicate the handle into your process and reply thought the named pipe with something like "TOKEN: 0x1F2"
Where 0x1F2 is the address of the file handle, opening files as SYSTEM is easily abuse-able, of course we can simply just open any file then rewrite it and execute as system.
For this one, I will just rewrite Microsoft Edge elevation service then execute through COM, I know this may not work in every windows but you may figure out how to elevate by yourself.

PoC time !

The proof of concept which I made assume that VMware workstation is installed in “C:\Program Files (x86)\VMware” which is the default.
The PoC takes arguments, by example
vmware-authd-EoP.exe c:\users\desktop.ini c:\windows\win.ini
You can also specify “—get-system” in order to spawn a SYSTEM shell, this may only work if Microsoft edge elevation service existed.



TIMELINE

[30/07/2021][>] : The bug has been reported privately to VMware team.
[02/08/2021][<] : The team has confirmed that their discussing the issue internally.
[08/08/2021][<] : The team was unable to reproduce the issue using the provided PoC.
[10/08/2021][>] : PoC updated.
[10-24/08/2021][<] : The team was unable to reproduce the issue and asked for the reproduction environment.
[26/08/2021][>] : Provided more information about the reproduction environment.
[02/09/2021][<] : The team is still unable to reproduce the issue due to the complexity of the issue.
[01-16/10/2021][<] : The team asked for updates.
[18/10/2021][>] : Told the team that I will unable to provide an update at the moment, but I will follow with more details as soon as possible.
[24/01/2022][>] : Provided a stable PoC
[08/02/2022][<] : The team asked for the dll used in the PoC
[08/02/2022][>] : Told the team that the dll just call MessageBox as it is nothing special
[03/03/2022][<] : Won’t fix. The team has claimed that they are comfortable with having privileges escalation weakness in VMware Workstation/Player.