

TokenImp plugin

Shachaf Atun

Introduction

TokenImp plugin is a volatility plugin that will allows mapping the processes' and threads' tokens. This document describes the idea behind *TokenImp*, its' usage, its' goal and how important it is for memory forensics investigations.

Contents

Introduction	1
General Information.....	2
What is a Token?	2
User Account Control (UAC).....	2
Tokens in our everyday lives	2
Malicious usage of Token Impersonation	2
The Research.....	3
TokenImp Goal	3
Optional Flags.....	3
Let's Hunt! (Usage examples)	4
Mimikatz	4
Tokenvator	5
Invoke-TokenManipulation.....	7
Summary.....	12

General Information

What is a Token?

An access token is an object that describes the security context of a process or a thread. The information inside a token includes the identity and privileges of the user account associated with the process or thread. Every process has a primary token that describes the security context of the user account associated with the process. By default, the system uses the primary token when a thread of the process interacts with a securable object. Moreover, a thread can impersonate a client account. Impersonation allows the thread to interact with securable objects using the client's security context. A thread that is impersonating a client has both a primary token and an impersonation token (taken from [MSDN](#)).

User Account Control (UAC)

Since Windows Vista, UAC became a part of Windows security features. It basically means that even though an account is administrative, every application won't run in administrative context until it will be approved by the user itself or inherit the high context from the parent process.

There are many techniques to bypass the UAC in the wild, examples can be found [here](#).

A method worth mentioning is utilizing the fact that Microsoft's signed executables can run in high privileged context without the user's permission and interaction when they use operations such as *IFileOperation COM* object. (Presented in [Cobalt Strike's bypassuac](#) command).

Tokens in our everyday lives

Tokens are everywhere. Every running application uses a token and some of them use impersonation. The most common tool that allows the impersonation of another user is a built-in tool called *RunAs* and allows you to run an application as other users if you know their credentials.

Another example is a *system* context *svchost/WMI* process that needs to perform an action as the user itself, often changes the thread's token to the user's token. Even when an application crashes system context *svchost* process will create a *Wermgr* task in the crashed application with the user's token to collect information about the error and sometimes to take memory dump.

Malicious usage of Token Impersonation

Token impersonation is a technique used often by red teams and attackers in order to impersonate another user logged on in order to commit some tasks as a legitimate user, or to perform privilege escalation into *SYSTEM* account.

An example for usage in the wild can be found in *APT28*, *Azorult*, *Lazarus Group*, *Duqo* and more.

The Research

It all started when I faced a memory dump of a machine infected with a malware of a group that used token impersonation and had no existing way to analyze it properly. Moreover, I couldn't see the UAC status for every process in the image. I started looking for some useful structures that can help me with the mission.

I found out I can get a token of an impersonated thread from *ClientSecurity* inside the *_ETHRAED*, but the problem was that even though the token was valid sometimes it wasn't in use, and that could be detected by searching an impersonation flag inside the *CrossThreadFlags*.

Now I had the ability to detect active impersonation and yet if a process was created with an impersonation token, I was still unable to alert it - I can't check the Parent Process ID due to PPID manipulation technique or a simple case of a closed process. The solution for this problem was the *CrossSessionCreate* flag lying inside the *_EPROCESS* structure, but this flag can cause a few false positive alerts for normal Windows behavior such as *.NET* processes (i.e. written in C#) and processes that were created using elevated mode via explorer (normal UAC) and with some other flags inside the *_EPROCESS* structure I managed to identify the mentioned behaviors and get only true impersonation creation.

Finally, I discovered an undocumented mask value inside the Token Flags that can indicate about the UAC mode.

TokenImp Goal

The plugin's goal is to detect token impersonation attacks, and perhaps detect suspicious behavior that can lead to zero day attacks.

TokenImp plugin comes in handy when you want to map administrative user's UAC status or every other user's session, detect token impersonation inside existing processes (i.e. *ImpersonateLoggedOnUser*, *SetThreadToken* API) and new processes related to impersonation without explorer (i.e. *CreateProcessWithTokenW* API).

Optional Flags

-p (--PID)	Runs for one specific process or a group of processes.
-v (--verbose)	Prints each process with all of its impersonating threads regardless to its token. (This flag can also help you to get information about UAC status of all the processes in the system).
-u (--SELECT-USERS)	Performs the plugin's checks only to processes owned by the selected users.
-c (--DETECT-CTRATION)	Search for impersonate process creation as well.

Let's Hunt! (Usage examples)

The plugin supports all windows platforms from Windows vista and higher and tested on Windows 7, Windows 2012 and Windows 10.

The tools that will be used in the demonstration are the most common tools for token manipulation:

- *Mimikatz* ([Mimikatz source code](#)).
- *Tokenvator* ([Tokenvator source code](#)).
- *Invoke-TokenManipulation* ([Invoke-TokenManipulation source code](#)).

With these tools we will impersonate other users and create new processes as well.

In the [Image file](#) there are several infected processes:

Mimikatz

One of *Mimikatz's* features is to elevate using token impersonation.



```
mimikatz # token::elevate
Token Id : 0
User name :
SID name : NT AUTHORITY\SYSTEM

244      {0;000003e7} 0 D 34279      NT AUTHORITY\SYSTEM      S-1-5-18
(04g,30p)      Primary
-> Impersonated !
× Process Token : {0;00078aa0} 2 F 1301309      WIN-GGDN0U3UUDC\user      S-1-5-21-1924014280-1981944623-3690795012-1000 (12g,23p)      Primary
× Thread Token : {0;000003e7} 0 D 1331511      NT AUTHORITY\SYSTEM      S-1-5-18
(04g,30p)      Impersonation (Delegation)

mimikatz #
```

Figure 1 - MImikatz's command line

Tokenvator

With *Tokenvator*, you can impersonate in the same process and create new processes using impersonation.

Tokenvator allows to select the token by a PID.

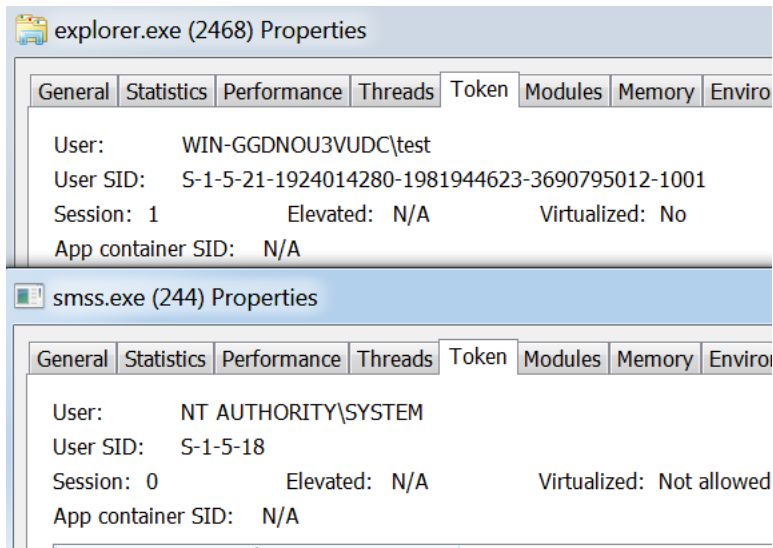


Figure 2 - processes that was used by Tokenvator

I used two instances of *Tokenvator*.

In the first, I only created a new process (*calc.exe*).

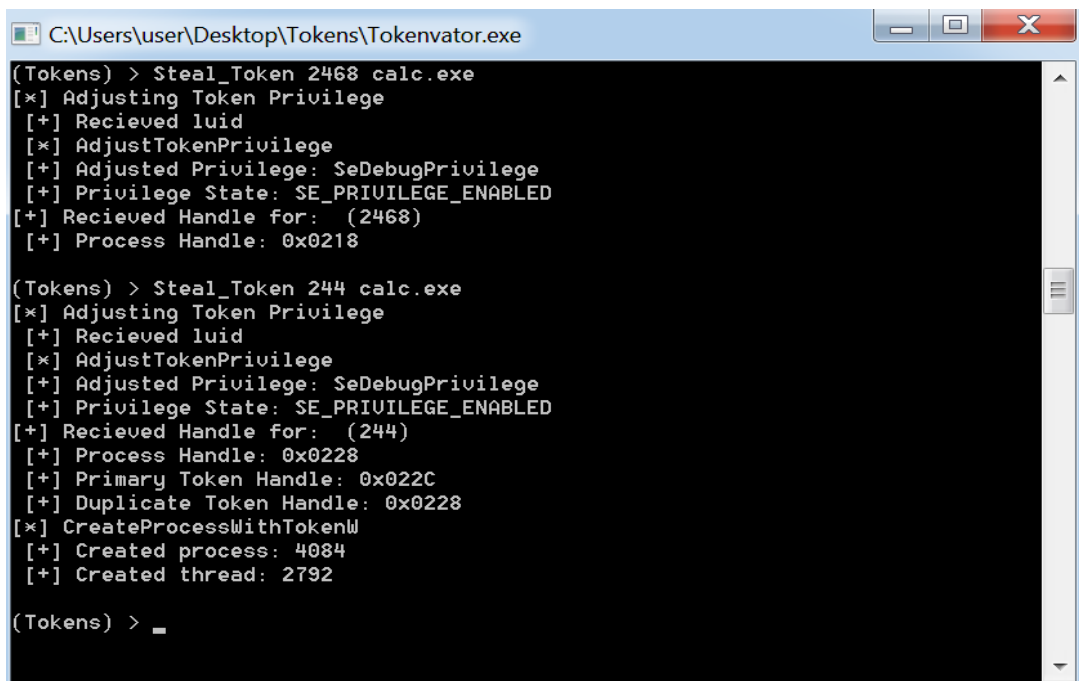
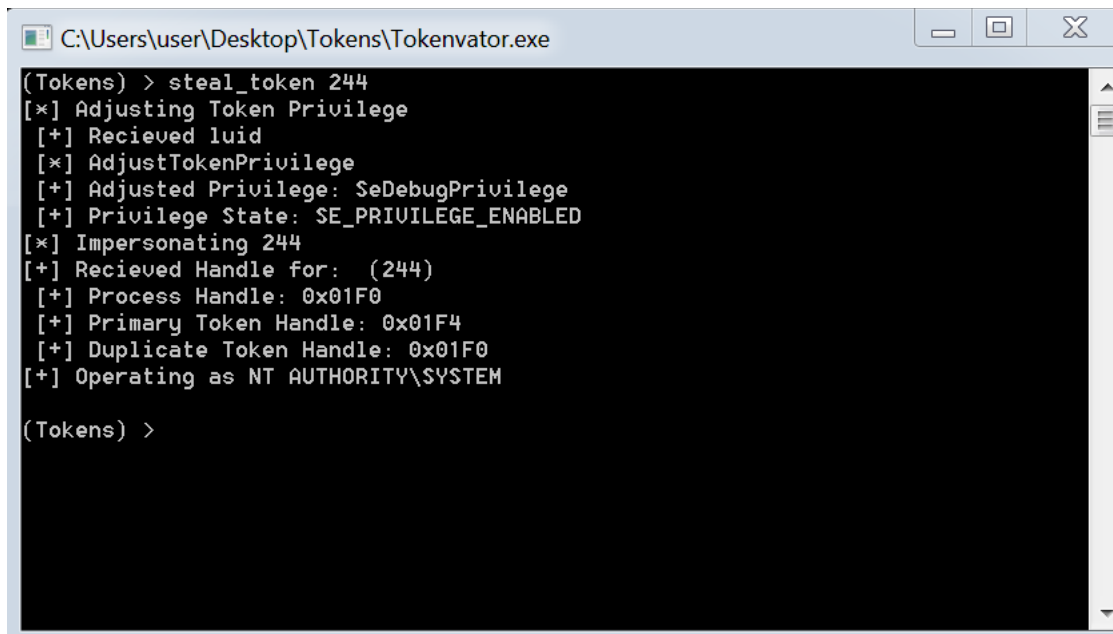


Figure 3 - Tokenvator used to create new processes

In the second, I used impersonation in *Tokenvator* itself.



```
(Tokens) > steal_token 244
[*] Adjusting Token Privilege
[+] Recieved luid
[*] AdjustTokenPrivilege
[+] Adjusted Privilege: SeDebugPrivilege
[+] Privilege State: SE_PRIVILEGE_ENABLED
[*] Impersonating 244
[+] Recieved Handle for: (244)
[+] Process Handle: 0x01F0
[+] Primary Token Handle: 0x01F4
[+] Duplicate Token Handle: 0x01F0
[+] Operating as NT AUTHORITY\SYSTEM

(Tokens) >
```

Figure 4 - Tokenvator was used to impersonate SYSTEM account

Invoke-TokenManipulation

With *Invoke-TokenManipulation* you can create new processes and impersonate in the same processes as well, but with *Invoke-TokenManipulation* you can also provide the username we want to impersonate.

Same as earlier, I used two instances of the tool above for the demonstration:

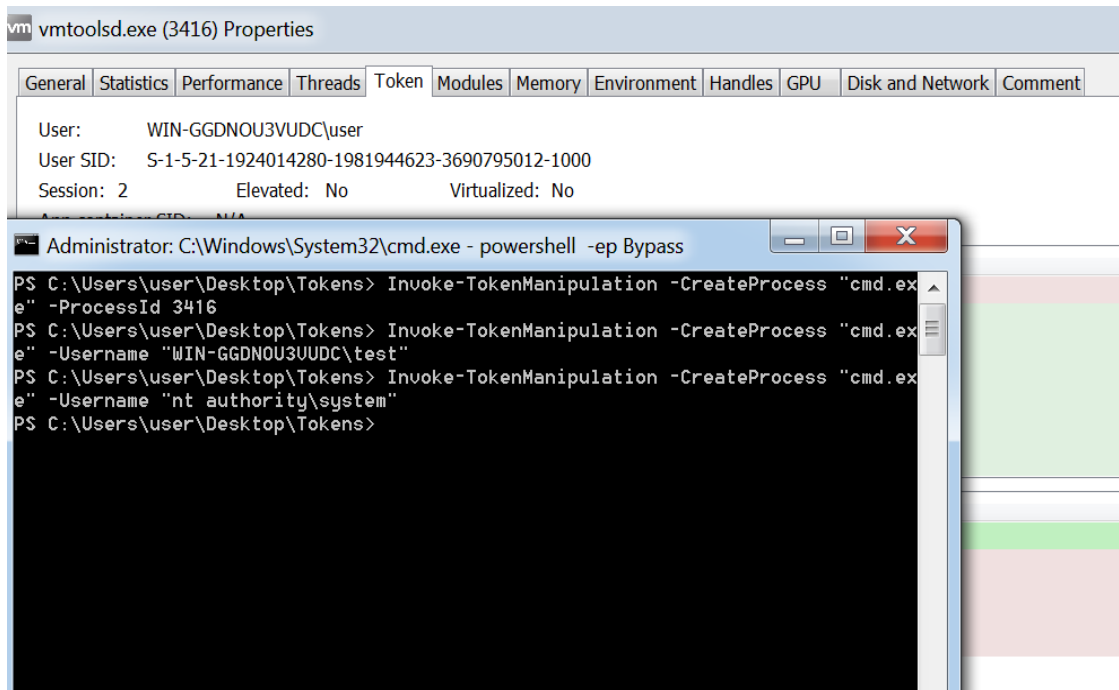


Figure 5 - Invoke-TokenManipulation for creating new processes

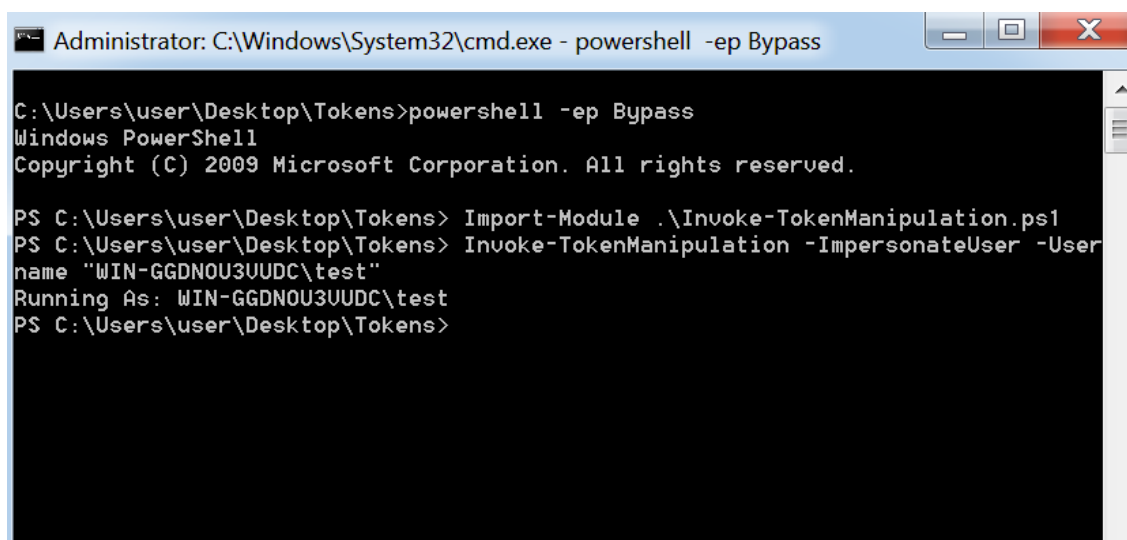


Figure 6 - Invoke-TokenManipulation for impersonation

And now let's try to find them 😊

Example 1:

TokenImp with no flags:

Detects only impersonation for active threads.

```
C:\Users\atun8\Desktop\Tools\volatility-master>vol.py -f tokensImg.vmem --profile=Win7SP1x64 tokenimp
Volatility Foundation Volatility Framework 2.6

Token Impersonation Information:
Proc - powershell.exe, Pid - 2072, PPid - 3188
Detection Reasons:
  Active Impersonated Thread(s) With Different Token(s) found
Token Info: WIN-GGDN0U3VUDC/user (Elevated token)
Threads With Different token:
Thread Id:4056
  Token Info:WIN-GGDN0U3VUDC/test (Non elevated token), source - User32
-----
Proc - Tokenvator.exe, Pid - 1272, PPid - 3312
Detection Reasons:
  Active Impersonated Thread(s) With Different Token(s) found
Token Info: WIN-GGDN0U3VUDC/user (Elevated token)
Threads With Different token:
Thread Id:3840
  Token Info:WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token), source - *SYSTEM*
-----
Proc - Not_mimikatz.e, Pid - 1932, PPid - 3312
Detection Reasons:
  Active Impersonated Thread(s) With Different Token(s) found
Token Info: WIN-GGDN0U3VUDC/user (Elevated token)
Threads With Different token:
Thread Id:3964
  Token Info:WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token), source - *SYSTEM*
```

Process' general information

All detection reasons will be shown here

Process' token

All suspicious threads with their token information

Detection1 - normal detection

We can see that we found only the ones that used thread impersonation, but we haven't found all the suspicious processes yet.

Example 2:

TokenImp with the *Detect-Creation* flag

```
C:\Users\atun8\Desktop\Tools\volatility-master>vol.py -f tokensImg.vmem --profile=Win7SP1x64 tokenimp -c
Volatility Foundation Volatility Framework 2.6

Token Impersonation Information:

Proc - powershell.exe, Pid - 2072, PPid - 3188
Detection Reasons:
    Active Impersonated Thread(s) With Different Token(s) found

Token Info: WIN-GGDN0U3VUDC/user (Elevated token)

Threads With Different token:
Thread Id:4056
    Token Info:WIN-GGDN0U3VUDC/test (Non elevated token), source - User32
-----
Proc - cmd.exe, Pid - 1004, PPid - 3696
Detection Reasons:
    Impersonate Process Creation Detected

Token Info: WIN-GGDN0U3VUDC/user (Non elevated token)
-----
Proc - cmd.exe, Pid - 1268, PPid - 3696
Detection Reasons:
    Impersonate Process Creation Detected

Token Info: WIN-GGDN0U3VUDC/test (Non elevated token)
-----
Proc - cmd.exe, Pid - 3360, PPid - 3696
Detection Reasons:
    Impersonate Process Creation Detected

Token Info: WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token)
-----
Proc - Tokenvator.exe, Pid - 1272, PPid - 3312
Detection Reasons:
    Active Impersonated Thread(s) With Different Token(s) found

Token Info: WIN-GGDN0U3VUDC/user (Elevated token)

Threads With Different token:
Thread Id:3840
    Token Info:WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token), source - *SYSTEM*
-----
Proc - calc.exe, Pid - 4084, PPid - 3852
Detection Reasons:
    Impersonate Process Creation Detected

Token Info: WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token)
-----
Proc - Not_mimikatz.e, Pid - 1932, PPid - 3312
Detection Reasons:
    Active Impersonated Thread(s) With Different Token(s) found

Token Info: WIN-GGDN0U3VUDC/user (Elevated token)

Threads With Different token:
Thread Id:3964
    Token Info:WORKGROUP\WIN-GGDN0U3VUDC$ (Elevated token), source - *SYSTEM*
-----
```

Detection 2 – detects all malicious programs.

As we can see, now we detected all malicious processes in the image.

Example 3:

Now let's say we only need to check if a certain user is compromised, we can use the user flag:

```
C:\Users\atun8\Desktop\Tools\volatility-master>vol.py -f tokensImg.vmem --profile=Win7SP1x64 tokenimp -c -u WIN-GGDNOU3VUDC/test
Volatility Foundation Volatility Framework 2.6

Token Impersonation Information:

Proc - cmd.exe, Pid - 1268, PPid - 3696
Detection Reasons:
    Impersonate Process Creation Detected

Token Info: WIN-GGDNOU3VUDC/test (Non elevated token)
-----
C:\Users\atun8\Desktop\Tools\volatility-master>^S_
```

And if we check who is the parent process with some additional user information:

```
C:\Users\atun8\Desktop\Tools\volatility-master>vol.py -f tokensImg.vmem --profile=Win7SP1x64 tokenimp -v -p 3696
Volatility Foundation Volatility Framework 2.6

Token Impersonation Information:

Proc - powershell.exe, Pid - 3696, PPid - 3920
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Elevated token)
-----
C:\Users\atun8\Desktop\Tools\volatility-master>_
```

Now we can see that this is not a False Positive, but a token manipulation attack.

Example 4:

Map UAC status of specific user's processes with the verbose flag:

```
C:\Users\atun8\Desktop\Tools\volatility-master>vol.py -f tokensImg.vmem --profile=Win7SP1x64 tokenimp -u WIN-GGDNOU3VUDC/user -v
Volatility Foundation Volatility Framework 2.6

Token Impersonation Information:

Proc - taskhost.exe, Pid - 2612, PPid - 492
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Non elevated token)
-----
Proc - dwm.exe, Pid - 3300, PPid - 868
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Non elevated token)
-----
Proc - explorer.exe, Pid - 3312, PPid - 3292
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Non elevated token)
-----
Proc - vmtoolsd.exe, Pid - 3416, PPid - 3312
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Non elevated token)
-----
Proc - cmd.exe, Pid - 3920, PPid - 3312
Detection Reasons:
    Verbose Mode

Token Info: WIN-GGDNOU3VUDC/user (Elevated token)
-----
```

Detection 3 – discover UAC status of a specific user (output snippet)

Summary

TokenImp plugin can be very helpful in your next investigation, false positive rate is extremely low and a false positive (if alerted) is easily recognizable. You need to be able to distinguish between an IT guy who used *RunAs* tool for a legitimate procedure, and an attacker who used it for malicious purposes. The plugin will detect all the common tools, and even custom ones due to the use of the same APIs, the plugin detects behavior and not tools.

You can use the plugin for any reason you may find:

- Get more information of a certain user/process
- Detect token manipulation attacks
- Map UAC status for certain session/process

Hope you'll enjoy it.