

GreyHatHacker.NET

Malware, Vulnerabilities, Exploits and more ...

ABOUT

Posted by Parvez on November 18, 2013

Elevating privileges by exploiting weak folder permissions

Posted in: All, Vulnerabilities. Tagged: Elevate, Hijack. 13 comments

Securing machines is always an on-going process whether it is by locking down settings, blocking applications, disabling Windows Services, making sure user privileges are kept to a minimum and so on. If we don't then users will end up installing non-standard software, making changes to the system, malware doing more damage once getting compromised, etc. This post is about weaknesses in folder permissions leading to elevation of privilege by using DLL hijacking vulnerabilities in Windows Services.

What is DLL hijacking?

A few years ago there was quite a bit of hype being able to load malicious DLLs remotely or locally from the current working directory. The Microsoft article [1] explains it clearly

"When an application dynamically loads a dynamic-link library without specifying a fully qualified path name, Windows attempts to locate the DLL by searching a well-defined set of directories in a particular order. If an attacker gains control of one of the directories on the DLL search path, it can place a malicious copy of the DLL in that directory. This is sometimes called a DLL preloading attack or a binary planting attack. If the system does not find a legitimate copy of the DLL before it searches the compromised directory, it loads the malicious DLL. If the application is running with administrator privileges, the attacker may succeed in local privilege elevation."

So if an application loads a DLL just by its name it goes through the search order below (32bit OS) to find the library

1. The directory from which the application loaded
2. 32-bit System directory (C:\Windows\System32)
3. 16-bit System directory (C:\Windows\System)
4. Windows directory (C:\Windows)
5. The current working directory (CWD)
6. Directories in the PATH environment variable (system then user)

What are we exploiting?

The goal here is to get local admin rights on the machine. In order to achieve this we need three things to make this work

- Windows DLL search order
- DLL hijacking vulnerability
- Weak folder permissions

Windows DLL search order

In Windows DLL search order the directories of the path environment variable are the last search it carries out starting with the system variable path and then the user variable path. Unless the application hasn't used a fully qualified path name for its DLL it will try to find the DLL through the search order even with certain mitigations in place.

DLL hijacking vulnerability

A quick way to find DLL hijacking vulnerabilities is to start Process Monitor, setup the relevant filtering and carry out some actions. Here we will be exploiting Windows Services as a large number of services run on SYSTEM privileges, just by stopping and starting the services and observing the search patterns. Keep in mind that Services running under SYSTEM does not search through user path environment. After stopping and starting Services a number of vulnerabilities had been discovered.

Recent Posts

- Dokany/Google Drive File Stream Kernel Stack-based Buffer Overflow Vulnerability
- Exploiting STOPzilla AntiMalware Arbitrary Write Vulnerability using SeCreateTokenPrivilege
- Exploiting System Shield AntiVirus Arbitrary Write Vulnerability using SeTakeOwnershipPrivilege
- IKARUS anti.virus and its 9 exploitable kernel vulnerabilities
- Exploiting Vir.IT eXplorer Anti-Virus Arbitrary Write Vulnerability

Categories

- All
- Bugs
- Exploits
- Malware
- Mitigation
- Other
- Vulnerabilities

Tags

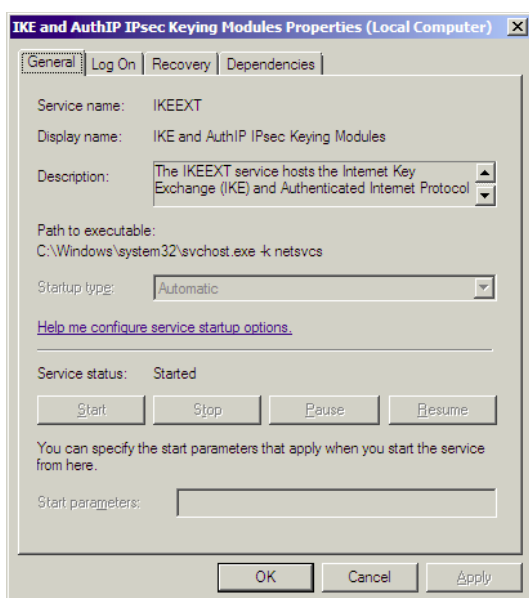
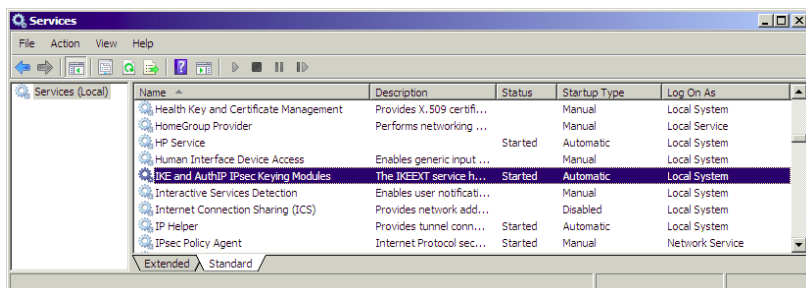
ActionScript ActiveX Adobe Anti-Rootkit
ASLR Autorun BHO BlazeDVD
Download and Execute **Elevate**
EMET FakeAV heapspray
Hidden Hijack IrfanView Java
Kernel Macros McAfee MSI
MSWord PGP pif RemoteExec
Return to Libc **ROP** Sandbox
Skype SureThing Symantec trailing UAC
URI Vista

Archives

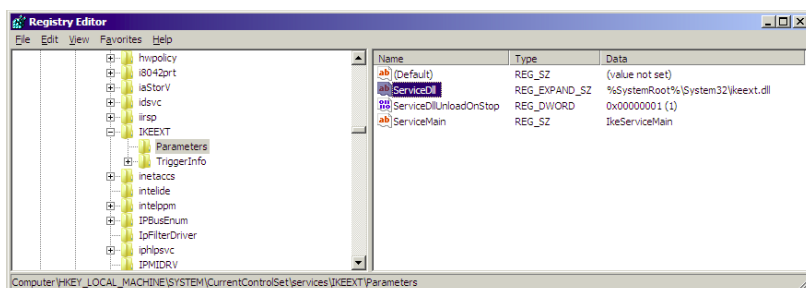
- January 2019 (1)
- September 2018 (1)
- January 2018 (1)
- November 2017 (2)
- September 2016 (1)
- December 2015 (2)
- July 2015 (1)
- January 2015 (1)
- December 2014 (1)
- June 2014 (1)
- January 2014 (1)
- November 2013 (1)
- September 2013 (1)
- February 2013 (1)
- December 2012 (1)
- August 2012 (1)
- June 2012 (1)
- February 2012 (1)
- January 2012 (1)

One Windows Service being the "IKE and AuthIP IPsec Keying Modules" This service is not started and set to manual by default but might be started or set to Automatic by VPN clients, policies, other Services, etc. For someone trying to obtain local admin rights starting Process Monitor will not be possible with limited permissions so let's go through the steps if we didn't have rights. In this example the IKE service is used but it can be any service for software that you may not have direct access to and need to audit.

First let's take note of the service executable through Windows Services (say services.msc via run command) checking to see if its status has started and running under localsystem.



Now checking in the registry to see if there are any service dlls being loaded by the service



We can copy these files (svchost.exe and IKEEXT.DLL) off to another machine to do our static analysis. After loading in IDA and simply searching for loadlibrary and jumping to the call will show what library is going to load. If a fully qualified path is not specified then we may be in luck. Here in IKEEXT.DLL LoadLibraryW will try to load "wlbsctrl.dll"

- December 2011 (1)
- November 2011 (1)
- August 2011 (2)
- July 2011 (1)
- April 2011 (1)
- March 2011 (1)
- October 2010 (3)
- June 2010 (1)
- May 2010 (1)
- March 2010 (2)
- February 2010 (1)
- December 2009 (1)
- September 2009 (1)
- May 2009 (1)
- April 2009 (1)
- September 2008 (1)
- November 2007 (2)

Meta

- Log in
- Entries [RSS](#)
- Comments [RSS](#)
- [WordPress.org](#)

```

.text:700813D1 ; ===== S U B R O U T I N E =====
.text:700813D1
.text:700813D1
.text:700813D1 ; int __stdcall IkeLoadNlb()
.text:700813D1 _IkeLoadNlb@0 proc near ; CODE XREF: IkeInit()+3D01p
.text:700813D1 mov edi, edi
.text:700813D3 push edi
.text:700813D4 push offset LibFileName ; "wbctrl.dll"
.text:700813D9 call ds: __imp_LoadLibraryW@4 ; LoadLibraryW(x)
.text:700813DF mov ecx, _gIkeExtGlobals
.text:700813E5 mov [ecx+0AC8h], eax
.text:700813E8 mov eax, _gIkeExtGlobals
.text:700813F0 add eax, 0AC8h
.text:700813F5 xor edi, edi
.text:700813F7 cmp [eax], edi
.text:700813F9 jnz loc_70094895
.text:700813FF
.text:700813FF loc_700813FF: ; CODE XREF: IkeProcessZombieQMContext
; IkeProcessZombieQMContextByQMPolicyI
.text:700813FF xor eax, eax
.text:70081401 pop edi
.text:70081402 retn
.text:70081402 _IkeLoadNlb@0 endp
.text:70081402 ; -----
.text:70081403 align 4
.text:70081404 ; const WCHAR LibFileName
.text:70081404 LibFileName: ; DATA XREF: IkeLoadNlb()+3To
.text:70081404 unicode 0, <wbctrl.dll>,0
.text:7008141E db 5 dup(90h)
.text:70081423

```

Note: It is not always as straight forward as in this example as the dll called might be using fully qualified path name but linked at compile time with another dll which will try to load this at load time which might be vulnerable due to being in another folder or not available.

Lastly we search for the library wbctrl.dll on the system to see if it exists and if so take note as to where it is located.

```
C:\>dir wbctrl.dll /s
```

In this case wbctrl.dll does not exist on the system so it will go through the entire search order.

Weak folder permissions

Now for the most important part "Weak folder permissions". When new folders are created in the root it is writeable for all authenticated users by default. The "NT AUTHORITY\Authenticated Users:(I)(M)" gets added to the folder where M stands for modify access. So any application that gets installed on the root can be tampered with by a non-admin user. If binaries load with SYSTEM privileges from this folder it might just be a matter of replacing the binary with your own one.

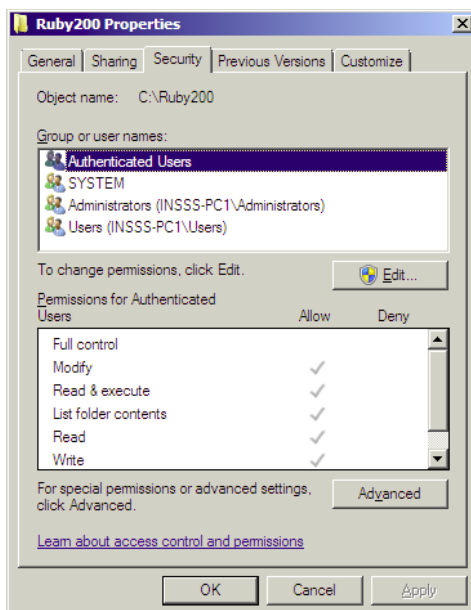
It gets interesting when applications gets installed in the root and add its path to the system path environment. This now opens the attack surface for a large number of applications that may have DLL hijacking vulnerabilities. One scenario is software getting pushed onto machines, with the likes of Marimba, Landesk, etc. which use a Windows service running with system privileges to install the software. Since it runs with system privileges software pushed onto machines such as Perl, Python or Ruby it will add to the system path environment if adding the path had been set in the package along with being installed on the root as default. Or it could be an IT support personnel installs the software with their admin rights for the user. If a user installs manually (if possible) with non-admin rights then it may be added to user path environment and then exploitation would not be possible. We can use icacs.exe to check the permissions of the folder or by the folder properties security tab.

```

C:\Windows\system32\cmd.exe
C:\>icacls c:\Ruby200
c:\Ruby200 BUILTIN\Administrators:(I)(F)
           BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
           NT AUTHORITY\SYSTEM:(I)(F)
           NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
           BUILTIN\Users:(I)(OI)(CI)(RX)
           NT AUTHORITY\Authenticated Users:(I)(M)
           NT AUTHORITY\Authenticated Users:(I)(OI)(CI)(IO)(M)

Successfully processed 1 files; Failed processing 0 files
C:\>

```



Pwning the box

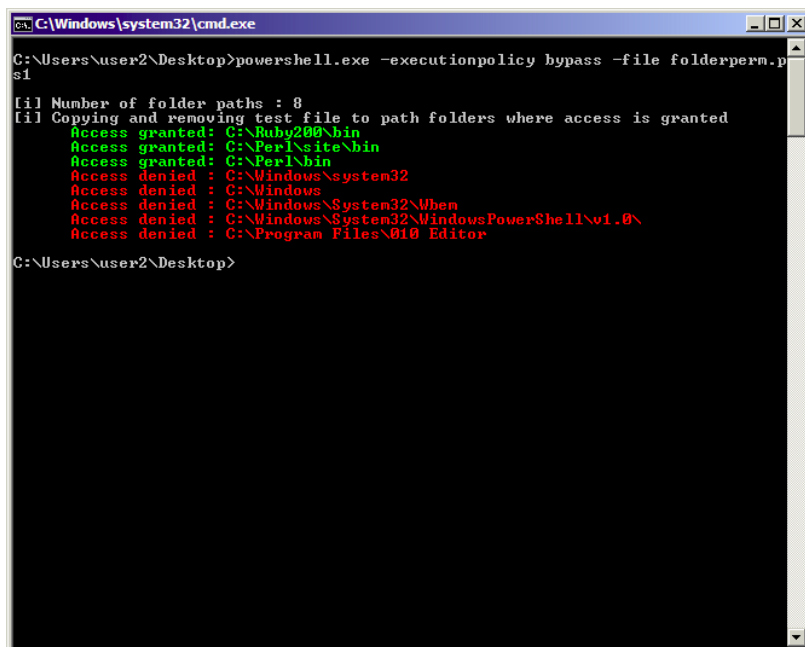
From our previous sections what we know now are

- Service "IKE and AuthIP IPsec Keying Modules" loads service dll IKEEXT.DLL
- IKEEXT.DLL will try to load wlbsctrl.dll
- OS will carry its search order to find wlbsctrl.dll
- We have a writeable folder C:\Ruby200\bin which is in the search order

All we need to do now is drop our malicious crafted DLL wlbsctrl.dll in C:\Ruby200\bin, reboot the machine and it will carry out its action under SYSTEM privileges. Users requesting Ruby, Perl, etc. are probably developers and have rights anyway but there may be other software which gets installed on the root and adds to the system path where limited users might take advantage of and this where we need to do our assessment and make any changes before being deployed.

Testing folder paths

I wrote a simple PowerShell script you can download from [here](#) that can be used to quickly check vulnerable path folders. System path environment variable comes first and then user path environment variable. Running it in a medium integrity shell for an admin or non-admin user will give the same results.



```

C:\Windows\system32\cmd.exe
C:\Users\user2\Desktop>powershell.exe -executionpolicy bypass -file folderperm.ps1

[i] Number of folder paths : 8
[i] Copying and removing test file to path folders where access is granted
Access granted: C:\Ruby200\bin
Access granted: C:\Perl\site\bin
Access granted: C:\Perl\bin
Access denied : C:\Windows\system32
Access denied : C:\Windows
Access denied : C:\Windows\System32\Wbem
Access denied : C:\Windows\System32\WindowsPowerShell\v1.0\
Access denied : C:\Program Files\010 Editor

C:\Users\user2\Desktop>

```

Vulnerable Windows Services

Here are Windows Services that have been found to be vulnerable and could be exploited on Windows 7 (32/64)

IKE and AuthIP IPsec Keying Modules (IKEEXT)	– wlbsctrl.dll
Windows Media Center Receiver Service (ehRecvr)	– ehETW.dll
Windows Media Center Scheduler Service (ehSched)	– ehETW.dll

The Windows Media Center Services startup type is set to manual and status not started and will only give us only Network service privileges so I cannot see it to being much use especially with its limited privileges. It can however be started temporarily via certain scheduled tasks.

```

schtasks.exe /run /i /TN "Microsoft\Windows\Media Center\mcupdate"
schtasks.exe /run /i /TN "Microsoft\Windows\Media Center\MediaCenterRecoveryTask"
schtasks.exe /run /i /TN "Microsoft\Windows\Media Center\ActivateWindowsSearch"

```

A quick check on Windows XP has shown that these Services are vulnerable

Automatic Updates (wuauserv)	– ifsproxy.dll
Remote Desktop Help Session Manager (RDSessMgr)	– SalemHook.dll
Remote Access Connection Manager (RasMan)	– ipbootp.dll
Windows Management Instrumentation (winmgmt)	– wbemcore.dll

Other Services that might be installed are also vulnerable

Audio Service (STacSV)	– SFFXComm.dll SFCOM.DLL
Intel(R) Rapid Storage Technology (IAStorDataMgrSvc)	– DriverSim.dll
Juniper Unified Network Service(JuniperAccessService)	– dsLogService.dll
Encase Enterprise Agent	– SDDisk.dll

No dll hijacking vulnerabilities were found on a clean default installation of Windows 8 OS (64) so another good reason to start migrating to Windows 8.

Mitigation

There are a number of mitigations available to prevent this vulnerability to be exploited by using certain API's, changing registry settings, applying updates, etc. it does start to get confusing as to what we are mitigating so hopefully this section will make it a bit clearer.

CWDIllegalInDllSearch

This update [2] at the time introduced a new registry entry CWDIllegalInDllSearch that allowed users to control the DLL search path algorithm. Tested on a fully patched Windows 7 machine this update is no longer required so it might have been later included in some security update. Once the patch is installed (if applicable) you will need to add the DWORD name CWDIllegalInDllSearch with a value in the registry key location

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager]
```

The value data can be 1, 2 or ffffffff. If the value name CWDIllegalInDllSearch does not exist or the value data is 0 then the machine will still be vulnerable to CWD attack. Please be aware that the value ffffffff could break certain applications. The search order is the same but this time if a malicious DLL is located in the current working directory the library is not loaded.

1. The directory from which the application loaded
2. 32-bit System directory (C:\Windows\System32)
3. 16-bit System directory (C:\Windows\System)
4. Windows directory (C:\Windows)
5. **The current working directory (CWD)** [dlls not loaded]
6. Directories in the PATH environment variable (system then user)

Previously I carried out tests on CWDIllegalInDllSearch values which you can check the results on my earlier [blog post](#).

SetDllDirectory

This function [3] removes the current working directory (CWD) from the search order when loading DLLs. For instance, the DLL search order after calling SetDllDirectory("C:\program files\MyApp\") becomes:

1. The directory from which the application loaded
2. **C:\program files\MyApp** [added]
3. 32-bit System directory (C:\Windows\System32)
4. 16-bit System directory (C:\Windows\System)
5. Windows directory (C:\Windows)
6. **The current working directory (CWD)** [removed]
7. Directories in the PATH environment variable (system then user)

Passing an empty string to SetDllDirectory("") the current working directory (CWD) is removed from the search order

1. The directory from which the application loaded
2. 32-bit System directory (C:\Windows\System32)
3. 16-bit System directory (C:\Windows\System)
4. Windows directory (C:\Windows)
5. **The current working directory (CWD)** [removed]
6. Directories in the PATH environment variable (system then user)

If this parameter is NULL, the function restores the default search order.

SafeDllSearchMode

Safe DLL search mode [4] is enabled by default. To disable this feature we can create a DWORD name SafeDllSearchMode with value 0

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager]
```

If SafeDllSearchMode is enabled, the search order is as follows:

1. The directory from which the application loaded
2. 32-bit System directory (C:\Windows\System32)
3. 16-bit System directory (C:\Windows\System)
4. Windows directory (C:\Windows)
5. The current working directory (CWD)
6. Directories in the PATH environment variable (system then user)

If SafeDllSearchMode is disabled, the search order is as follows:

1. The directory from which the application loaded
2. **The current working directory (CWD)** [moved up the list]
3. 32-bit System directory (C:\Windows\System32)
4. 16-bit System directory (C:\Windows\System)
5. Windows directory (C:\Windows)
6. Directories in the PATH environment variable (system then user)

Calling the `SetDllDirectory("")` or `SetDllDirectory("C:\\program files\\MyApp\\")` disables `SafeDllSearchMode` and uses the search order described for `SetDllDirectory`.

`LoadLibraryEx` function [5] takes another argument where a flag can be set to change the search order but I didn't get round to test it.

Mitigation for developers

For software developers there are a number of actions they can take

- Use `SetEnvironmentVariable(TEXT("PATH"),NULL)` API which removes the path environment variable from its search order
- Change default installation folder to `C:\\Program Files`
- Use fully qualified path when loading DLLs, i.e. `LoadLibrary("C:\\program files\\MyApp\\mylibrary.dll");`
- Use `SetDllDirectory("")` API removing the current working directory from the search order

Mitigation for IT professionals

For IT support professionals there are also a number of actions that can be taken

- When packaging and deploying software via deployment tools such as Marimba, Landesk, etc. or manually installing software change the installation folder to `C:\\Program Files`
- If software needs to be installed on the root check there are no binaries needing `SYSTEM` privileges
- If `SYSTEM` privileges are required then change the ACL's of the folder
- Remove the path entry from the `SYSTEM` path variable if not needed

Conclusion

This post shows us how easily elevated privileges can be achieved with very little effort. Ultimately the solution is simple by just making sure all software gets installed in the `C:\\Program Files` folder which will then inherit its more secure folder permissions. Malware could take advantage of this weakness not only to obtain system privileges but also to automatically load its malware making it that much harder to pinpoint its auto start entry points.

References

- [1] [http://msdn.microsoft.com/en-us/library/windows/desktop/ff919712\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ff919712(v=vs.85).aspx)
- [2] <http://support.microsoft.com/kb/2264107>
- [3] <http://msdn.microsoft.com/en-us/library/ms686203%28v=vs.85%29.aspx>
- [4] <http://msdn.microsoft.com/en-us/library/ms682586%28v=vs.85%29.aspx>
- [5] <http://msdn.microsoft.com/en-us/library/ms684179%28v=vs.85%29.aspx>

← Heap spraying browsers using Adobe Flash's ActionScript

Bypassing Windows ASLR using "Run without permission" Add-ons →

13 comments on "Elevating privileges by exploiting weak folder permissions"



gregory90

on November 20, 2013 at 5:10 pm said:

Hi,

As the name of the software you use to display dll libraries?
As shown in the fourth screenshot of this post.

Regards!



Parvez

on November 20, 2013 at 5:30 pm said:

Hi Gregory, IDA is an "Interactive DisAssembler", you can download a demo from there site
https://www.hex-rays.com/products/ida/support/download_demo.shtml

**gregory90**

on November 20, 2013 at 7:49 pm said:

But could unburden the freeware version?

https://www.hex-rays.com/products/ida/support/download_freeware.shtml

It would be the same?

**Parvez**

on November 20, 2013 at 9:06 pm said:

Not sure what you mean, try it out and see for yourself

**gregory90**

on November 22, 2013 at 9:46 am said:

Hi,

I tried with IDA Pro Free v6.0 and is very much like IDA Pro (shareware).

It is also a great option OllyDbg.

What do you think?

**Parvez**

on November 22, 2013 at 9:28 pm said:

Yes OllyDbg is a great portable debugger, a must have in your toolkit.

**altonius**

on January 16, 2014 at 12:07 am said:

FYI, I reached out to Microsoft about these services and the dll's not being fully qualified. They responded with:

"I believe that all of these binaries live in directories that require Admin privileges to write to, such as System32 and "Program Files". As such, we don't consider the non-fully qualified path to be a vulnerability in these cases."

**altonius**

on January 25, 2014 at 1:26 pm said:

I asked Microsoft about services looking for DLLs that don't exist on the machine and this is their response:

"We fully qualified the paths for those binaries in Windows 8 as a Defense-in-Depth measure. For earlier versions, the attack can only be performed if the user can change the PATH environment variable, which requires administrator privileges. Thus, it doesn't meet our bar for a security bulletin."


Looks like they're improving Windows 8 and beyond to use fully qualified paths, but as you need local admin to alter the PATH variable they're not too worried.

I did some work on unquoted paths in services earlier (MS13-058), and because you needed local access it took a while to get it recognised. This is different than yours as you point out, as weakened permissions are the default (even on Windows 8) and could occur when other 3rd party apps are updating the PATH variable.

Thanks again for publishing this.

Altonius

**Parvez**

 on January 28, 2014 at 10:42 am said:

Thanks for sharing your feedback Altonius.



b33f

on February 20, 2014 at 2:31 am said:

Great stuff as always Parvez!



Parvez

on February 22, 2014 at 2:55 pm said:

Thanks b33f



n0ne

on September 21, 2015 at 11:26 pm said:

So I followed your directions above and ensured my python directory had the correct permissions. Environment variables are properly set, I am using a Windows 7 SP1 x64 VM and reboot. When I come up the service is started and everything looks like it should work but I have 0 results.

Any ideas?



Parvez

on September 22, 2015 at 6:37 pm said:

Unless Microsoft has silently fixed it should work. Make sure you are using x64 compiled dll

Leave a Reply

*Your email address will not be published. Required fields are marked **

* Name

* Email

Website

I'm not a robot

reCAPTCHA
[Privacy](#) - [Terms](#)

Proudly powered by WordPress Theme: Parament by Automattic.