Protecting Privileged Domain Accounts: PsExec Deep-Dive

By Mike Pilkington

Published: 2012-12-17 · Archived: 2026-04-05 17:58:10 UTC

My primary goal is to help incident responders protect their privileged accounts when interacting with comprised hosts, though I also believe this information will be useful to anyone administering and defending a Windows environment.]

PsExec is an extremely powerful tool and is used commonly in enterprise networks, for both good and evil. Systems administrators and incident responders use it for its flexibility in interacting with remote machines, including a telnet-like ability to run command-line tools on remote machines and receive the output on their local console. Attackers utilize it for the same reasons, providing a convenient way to move laterally and interact with remote machines using compromised credentials.

Given its power, you might wonder what the ramifications are of using this tool on a compromised machine. In other words, could it lead to your credentials being compromised? In this article, I'll discuss the two "native" methods of logging onto a remote machine with *PsExec* and why you should always avoid one of the two. I'll also discuss possible workarounds to the second, more dangerous logon. Finally, since attackers have been known to use this tool for lateral movement, I'll follow up the logon discussion with a brief forensic analysis of the artifacts you will typically find from *PsExec* usage.

2 Types of Logons with *PsExec*

It's easy to overlook that fact that there are two distinct ways to logon to a remote host with *PsExec*. The first method is to run *PsExec* under the context of the currently logged-on user. This requires no special switches or specification of an account. It simply uses the logged-on account to authenticate to the remote machine. As we'll see in a moment, this results in a network logon to the remote machine.

The second method is to specify an alternate account using the "-u" switch and optionally the "-p" switch (if you don't supply "-p", it behaves the same but will prompt you for the password). This allows you to authenticate to the remote machine as a different user. This would commonly be a user with higher privileges than your currently logged-on account. While this is often convenient, it provides more than just convenience?it also provides a full interactive logon, which loads all the user's credentials on the remote host. This has the benefit of allowing a user to authenticate to another system *from* the remote host, but it also carries the significant drawback of exposing NT & LM hashes, Kerberos Ticket Granting Ticket (TGT), and clear-text passwords to malware on the remote host. We clearly don't want to do this on an untrusted host!

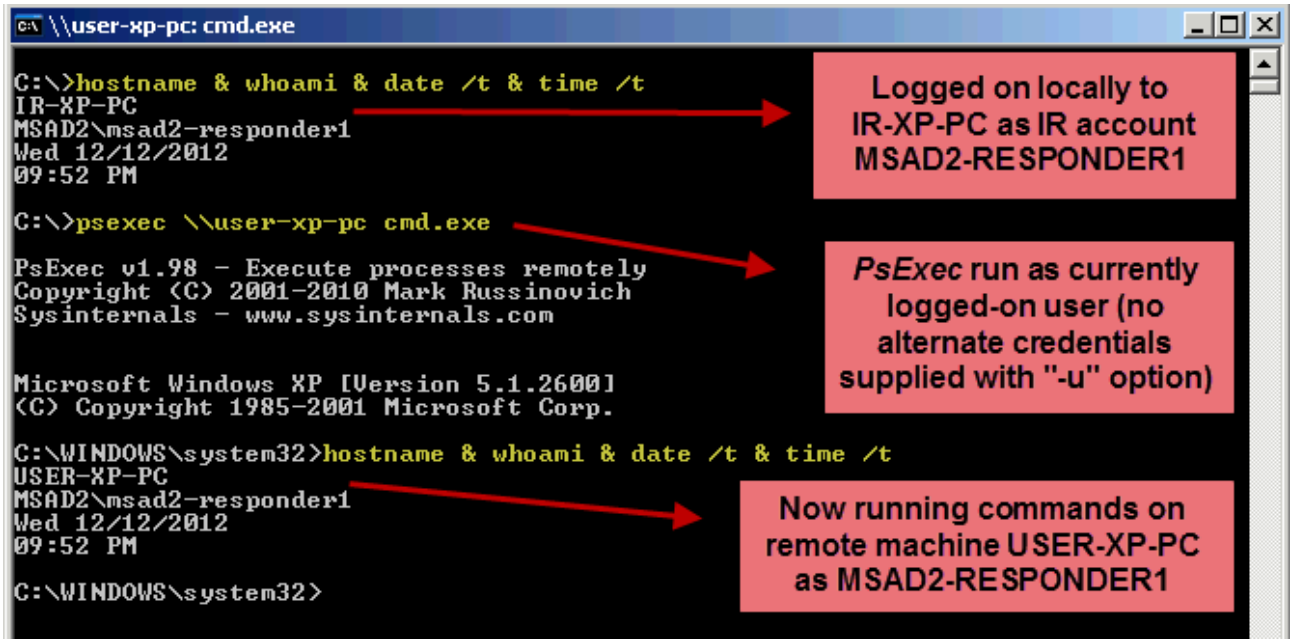
An important side note here is to be aware that both logon methods for *PsExec* have the potential of creating the delegate-level access token, if the remote system is trusted for delegation. This is also a significant risk, though not as commonly exploited. As discussed in my article on access tokens, we fortunately have a simple fix

available by enabling the setting "Account is sensitive and cannot be delegated", which is recommended by Microsoft for sensitive accounts.

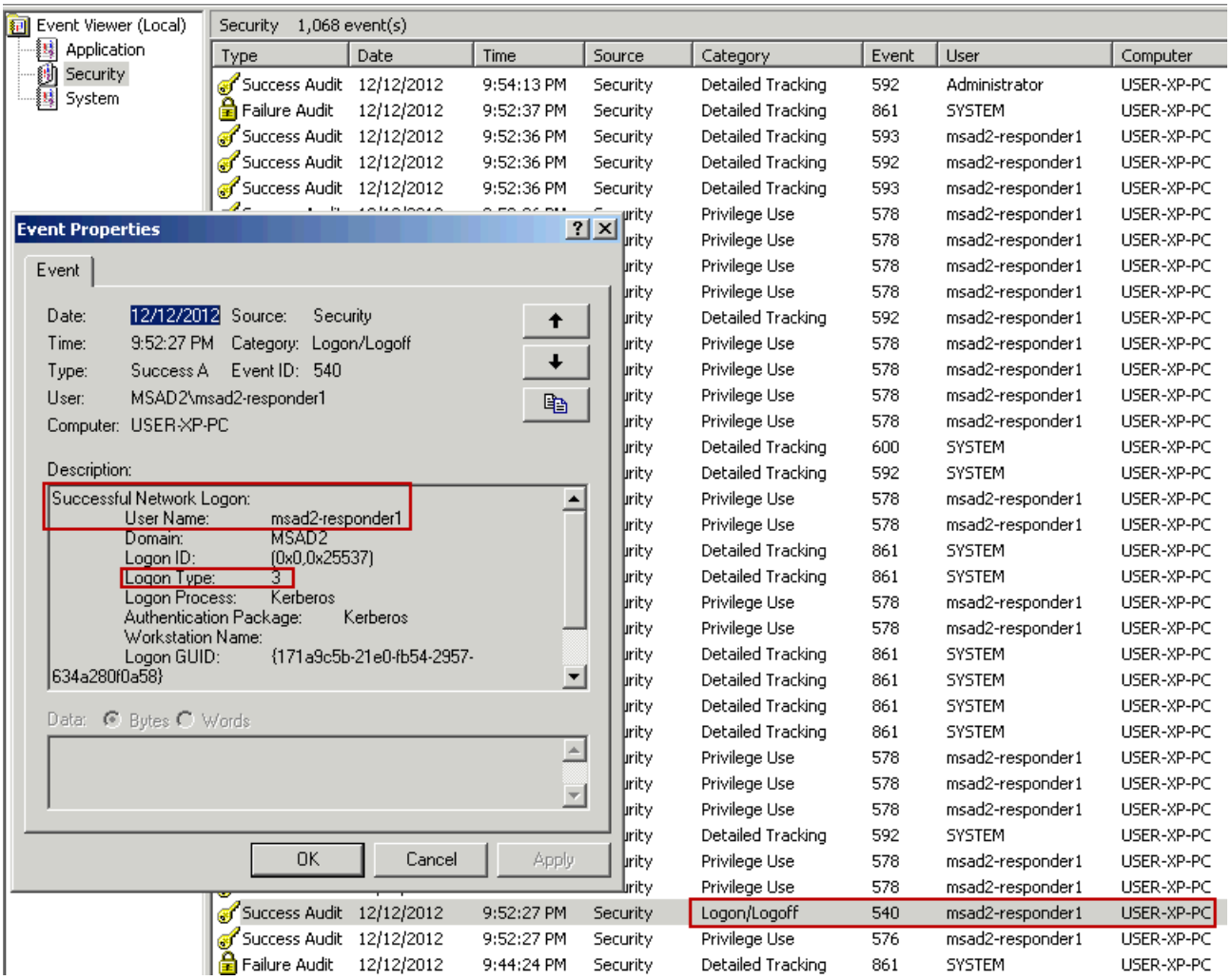
Ok, so let's have a look at these two logon methods in action.

Testing: *PsExec* with logged-on user account

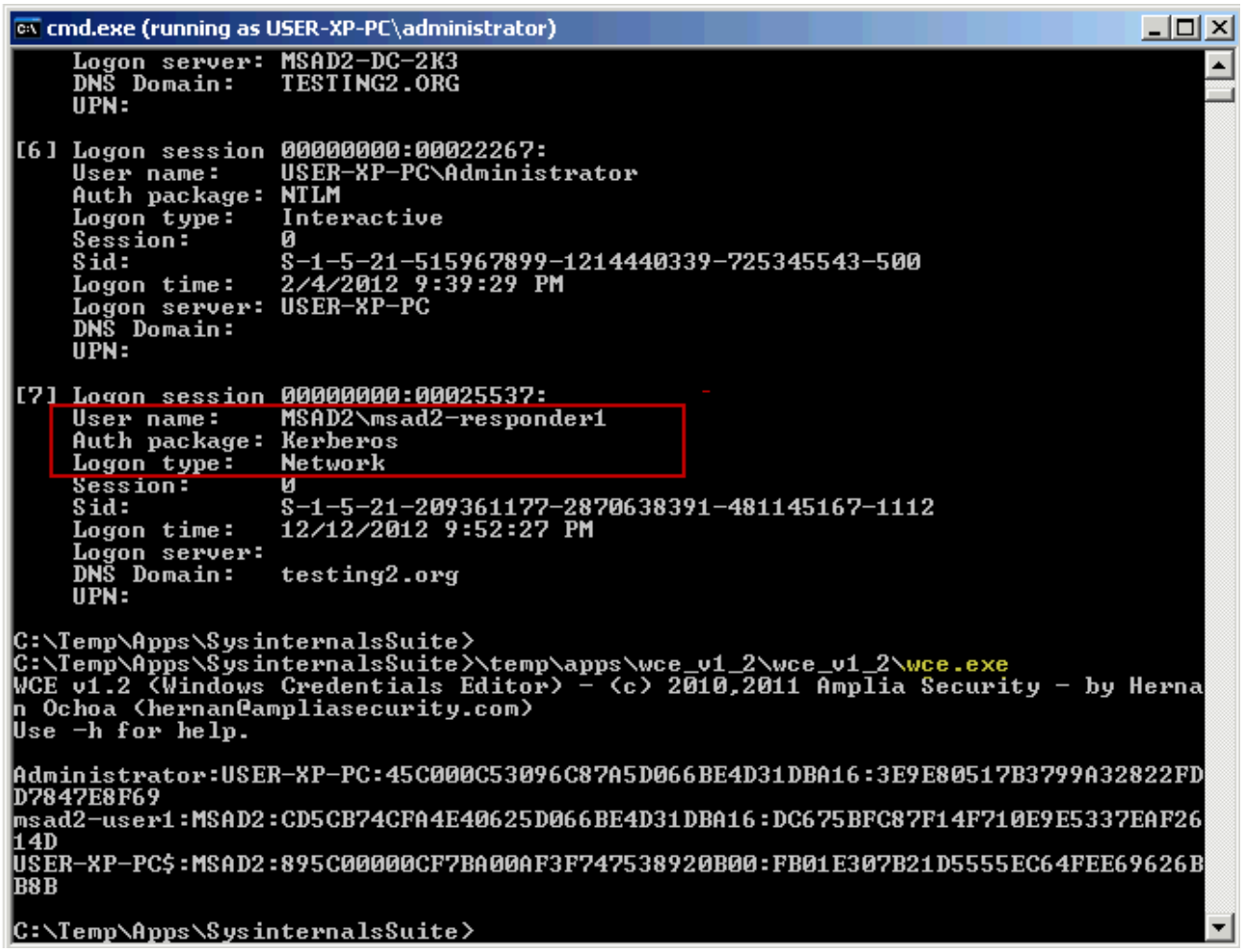
In the following screenshot, I'm logged onto machine IR-XP-PC as MSAD2-RESPONDER1. I've initiated a *PsExec* logon from the local host to the remote host named USER-XP-PC. Since I did not specify "-u", the currently logged-on user will use standard Kerberos or NTLM integrated authentication to connect to the remote host. Once connected, I ran a couple of commands to show that I am still MSAD2-RESPONDER1 and am now executing commands on USER-XP-PC.



The following screenshot shows the relevant Event Log from the *PsExec* logon. This connection created only a Type 3 network logon, which is the safe way to connect to a remote host.



To confirm the logon type and the fact that password hashes are not revealed, here's the relevant output from Sysinternals' *logonsessions* tool, again showing a network logon for MSAD2-RESPONDER1. The screenshot also shows the output of *Windows Credentials Editor (WCE)*, which verifies the hash is not in memory for the MSAD2-RESPONDER1 account.

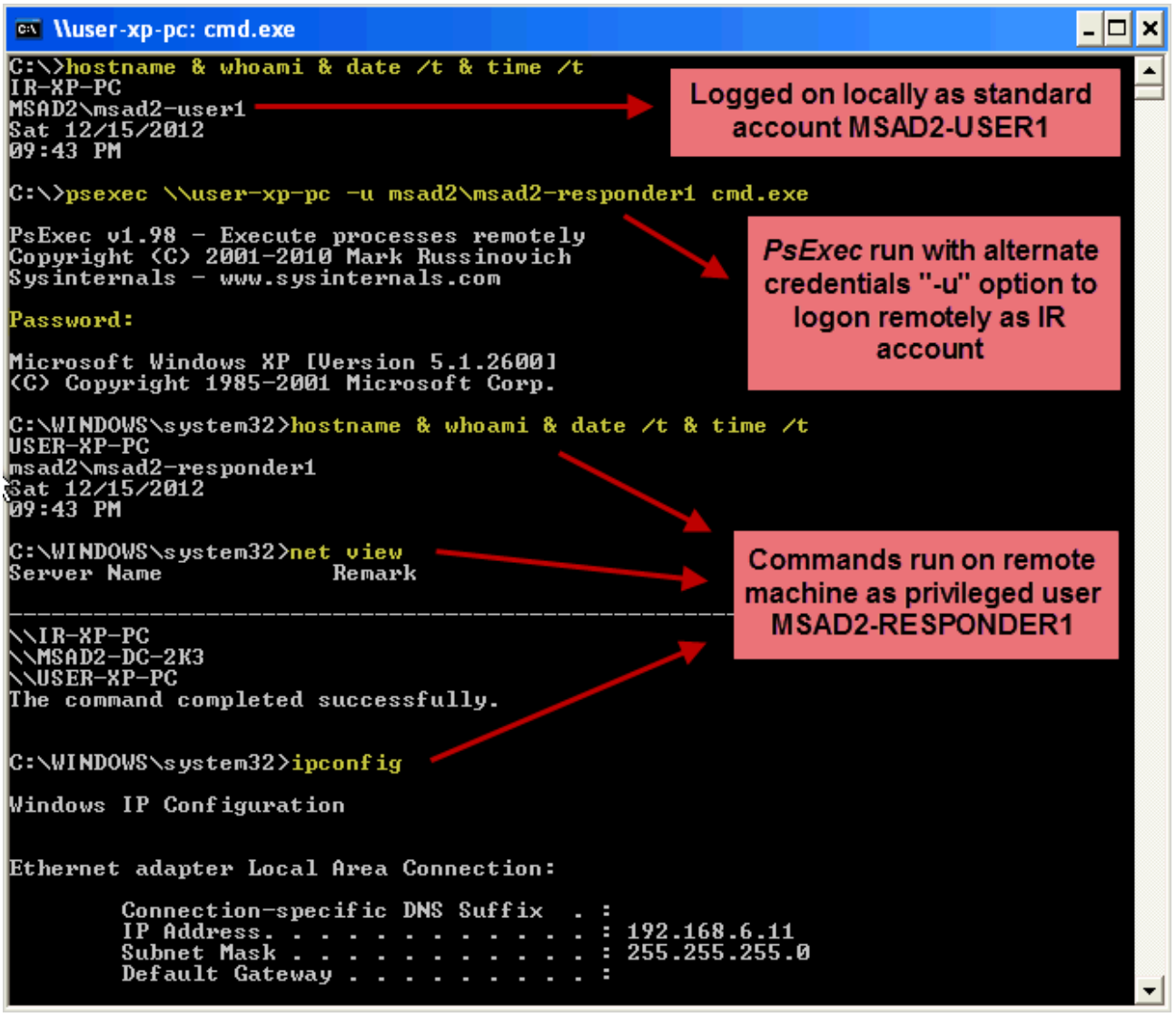


So the good news is that we've seen that a standard connection with *PsExec*, using the currently logged-on user's credentials, results in only a network logon. This is the safe way to execute *PsExec* remotely.

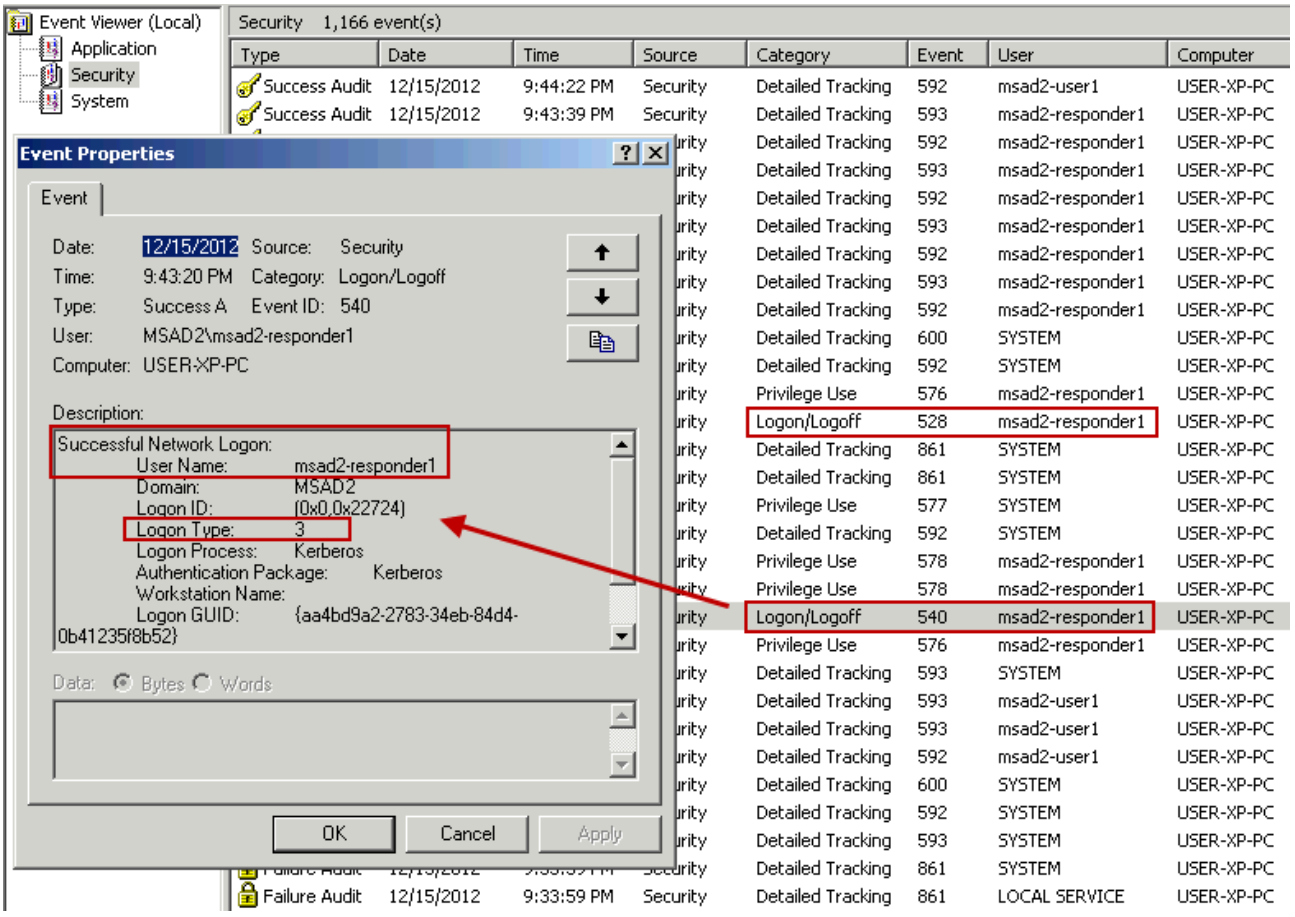
Now let's look at the other authentication method with *PsExec*.

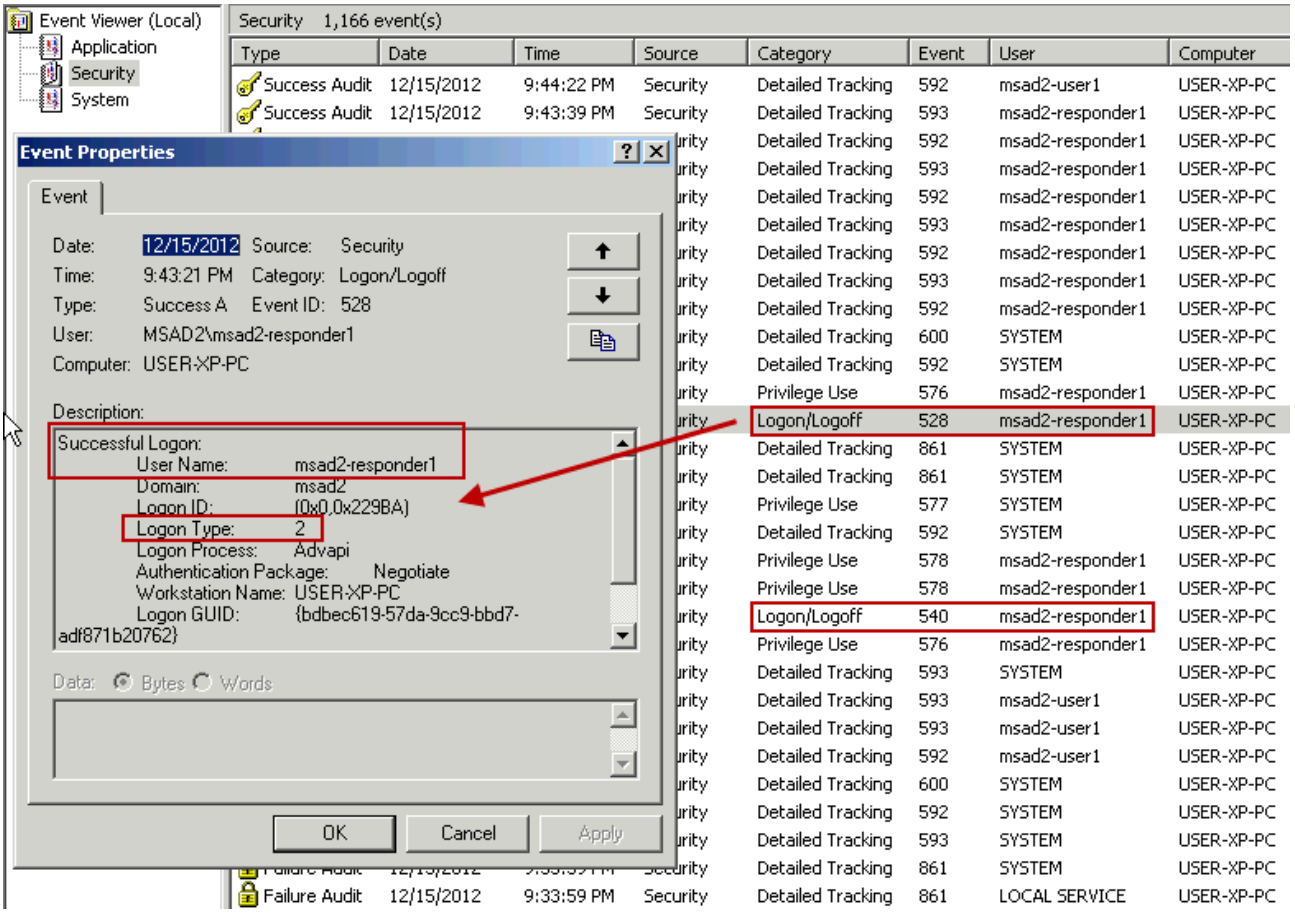
Testing: *PsExec* with Alternate Credentials "-u" option

This time I'm logged on locally with a non-privileged account and will then log on remotely using a privileged account, MSAD2-RESPONDER1.

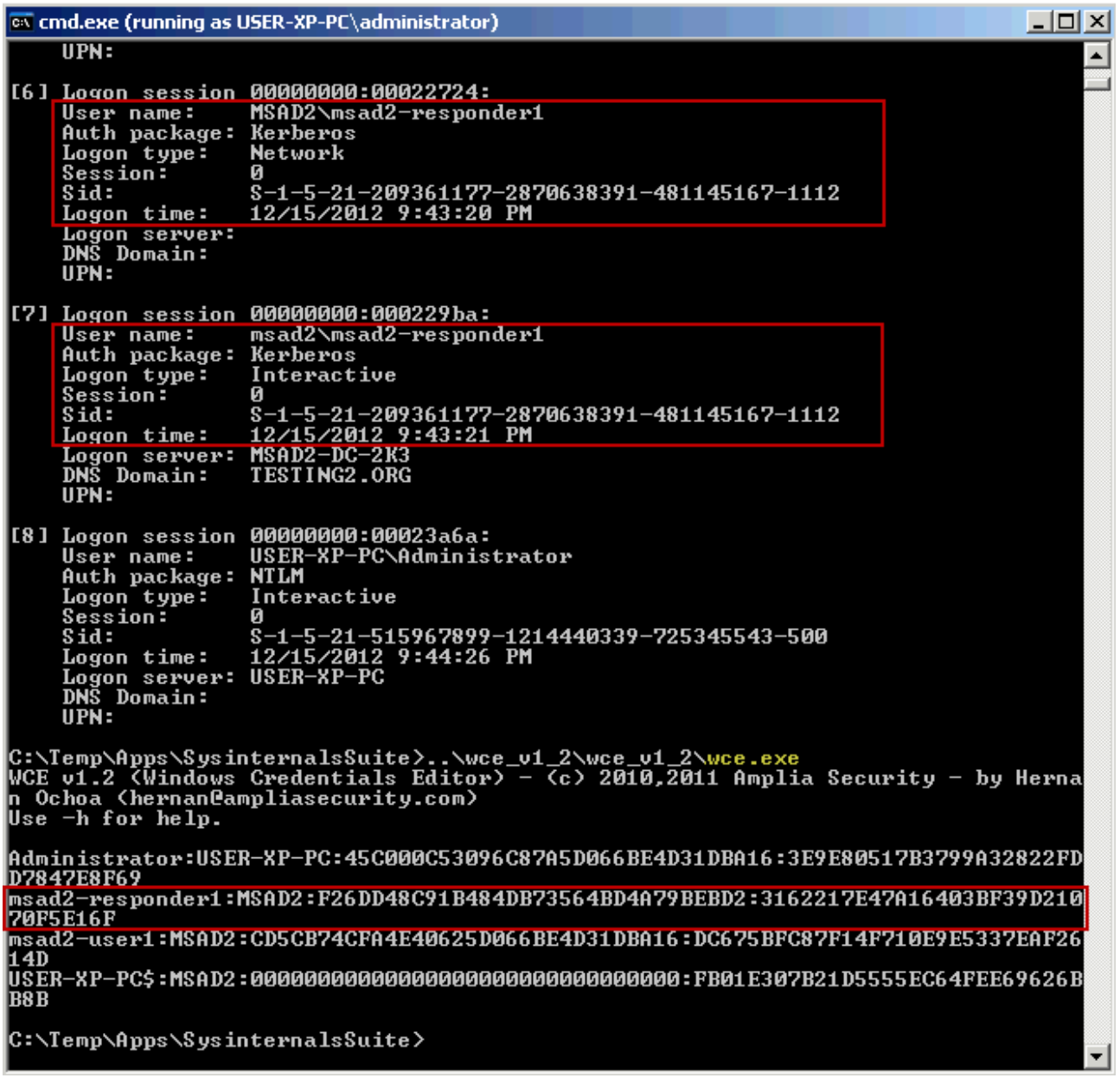


Examining the logs on the remote machine, we initially see a network logon (Type 3), but then 1 second later we see an interactive logon (Type 2) for our alternate credentials user, MSAD2-RESPONDER1.

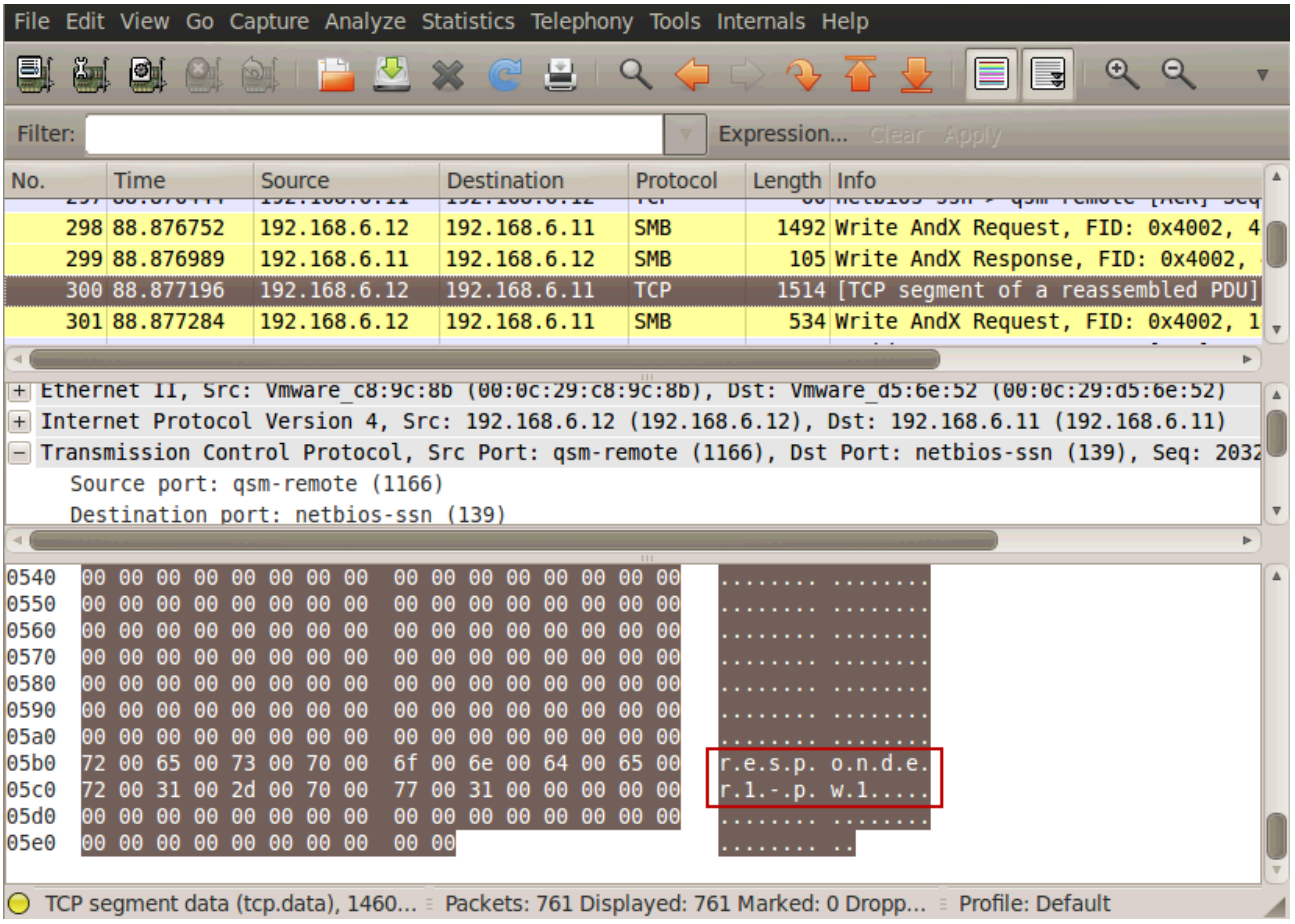

Event Viewer (Local)
Application
Security
System
Security 1,166 event(s)
Type Date Time Source Category Event User Computer
Success Audit 12/15/2012 9:44:22 PM Security Detailed Tracking 592 msad2-user1 USER-XP-PC
Success Audit 12/15/2012 9:43:39 PM Security Detailed Tracking 593 msad2-responder1 USER-XP-PC
Failure Audit 12/15/2012 9:33:59 PM Security Detailed Tracking 861 LOCAL SERVICE USER-XP-PC
Event Properties
Event
Date: 12/15/2012 Source: Security
Time: 9:43:20 PM Category: Logon/Logoff
Type: Success A Event ID: 540
User: MSAD2\msad2-responder1
Computer: USER-XP-PC
Description:
Successful Network Logon:
User Name: msad2-responder1
Domain: MSAD2
Logon ID: (0x0,0x22724)
Logon Type: 3
Logon Process: Kerberos
Authentication Package: Kerberos
Workstation Name:
Logon GUID: {aa4bd9a2-2783-34eb-84d4-0b41235f8b52}
Data: Bytes Words
OK Cancel Apply


Event Viewer (Local)
Application
Security
System
Security 1,166 event(s)
Type Date Time Source Category Event User Computer
Success Audit 12/15/2012 9:44:22 PM Security Detailed Tracking 592 msad2-user1 USER-XP-PC
Success Audit 12/15/2012 9:43:39 PM Security Detailed Tracking 593 msad2-responder1 USER-XP-PC
Failure Audit 12/15/2012 9:33:59 PM Security Detailed Tracking 861 LOCAL SERVICE USER-XP-PC
Event Properties
Event
Date: 12/15/2012 Source: Security
Time: 9:43:21 PM Category: Logon/Logoff
Type: Success A Event ID: 528
User: MSAD2\msad2-responder1
Computer: USER-XP-PC
Description:
Successful Logon:
User Name: msad2-responder1
Domain: msad2
Logon ID: (0x0,0x229BA)
Logon Type: 2
Logon Process: Advapi
Authentication Package: Negotiate
Workstation Name: USER-XP-PC
Logon GUID: {bdbec619-57da-9cc9-bbd7-adf871b20762}
Data: Bytes Words
OK Cancel Apply

This interactive logon causes the credentials of the responder account to be loaded and available for an attacker to steal. The following screenshot shows the currently logged on sessions for MSAD2-RESPONDER1 (two of them), and as expected, *WCE* shows us the password hashes for MSAD2-RESPONDER1.



To make matters worse, an additional concern with the "-u" option is that it sends the password in clear-text in order to generate the interactive logon. Microsoft Sysinternals warns us of this issue on their PsExec download page . Here is a screenshot of Wireshark showing the password being sent over the network in the clear (password = responder1-pw1):



Conclusions on *PsExec* Logon Methods, with Workarounds

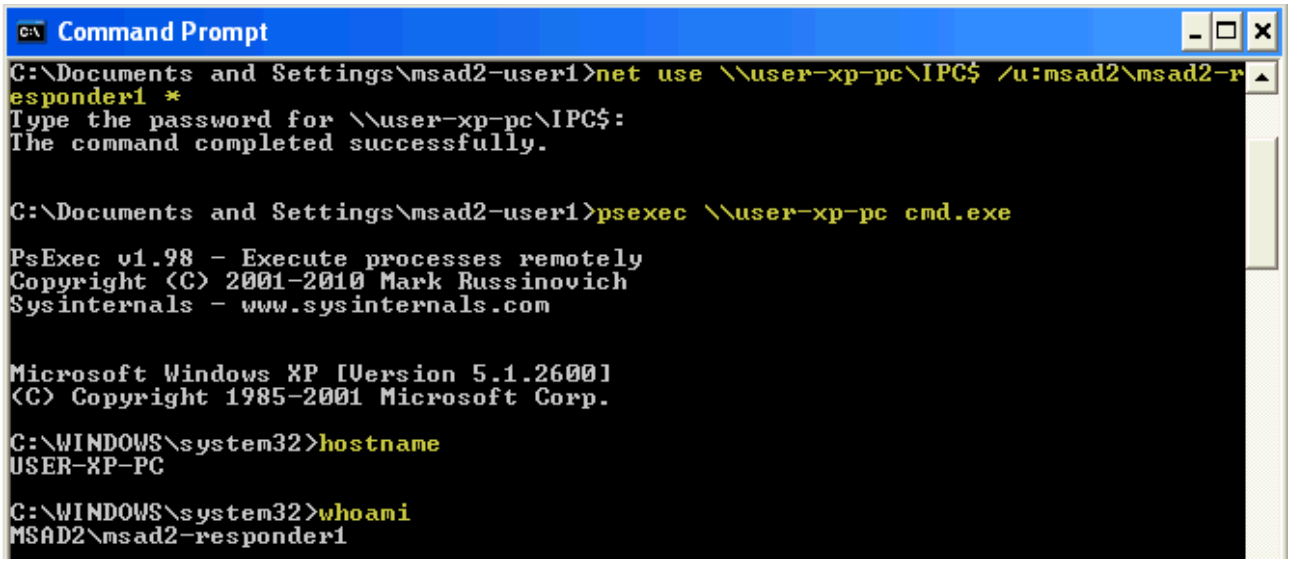
It is clearly not safe to use *PsExec's* "-u" option on an untrusted remote host. On the other hand, using *PsExec* without "-u" and therefore authenticating as the currently logged-on user is much safer and does not expose the account to theft of password hashes, the Kerberos TGT, or the plain-text password itself. As mentioned earlier, there is still the possibility of the account's delegation token being stolen. The simple fix for this risk is to enable the setting "Account is sensitive and cannot be delegated", as discussed in the article.

The ability to specify alternate credentials is a useful one, and fortunately, there are a couple of ways we can still make this work without divulging credentials on the remote host.

One possibility is the use of *RunAs*. For example, I logon to my machine as a standard user, but will often need to connect to a remote machine with a privileged account. For me, the easiest way to do this, while still being safe assuming my local machine is not compromised, is to spawn a new CMD command shell as the privileged account, using the command "**RunAs.exe /user:domain\user cmd.exe**". As I discussed in my article on hashes, using *RunAs* will still result in an interactive logon, but at least I'm doing it on my trusted workstation rather than the untrusted remote host.

RunAs is a good option in many cases, but there are situations where using *RunAs* is not feasible. For example, if the machine you are using is not a part of the privileged account's domain. It turns out there is another workaround that can allow you to specify alternate credentials but not create an interactive logon. The suggested workaround is

to first connect to the IPC$ share of the remote host using the alternate account, and then use *PsExec* without "-u". The process looks like this for my current test setup:

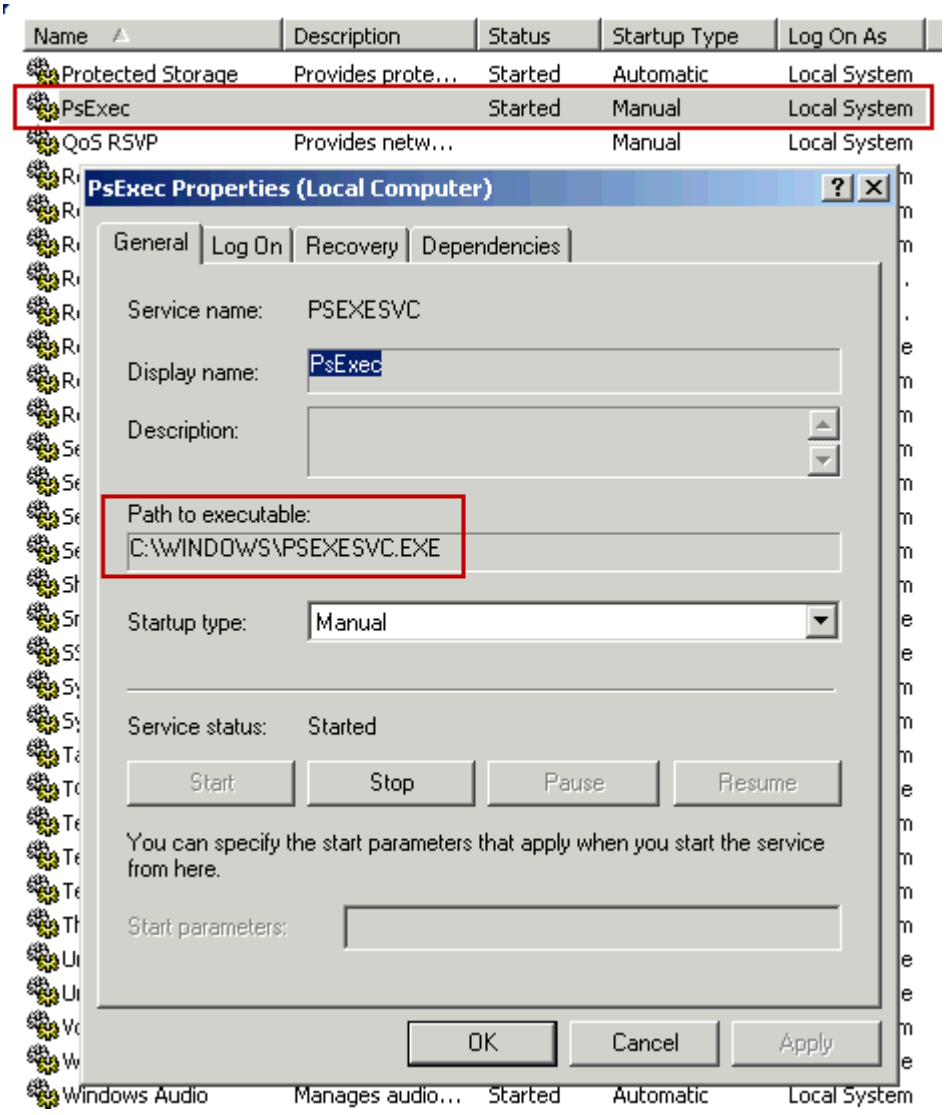


When performed this way, my tests show that only a network logon is created (no interactive logon) and also the password is not sent in clear-text (presumably since there is no interactive logon). I'd recommend you confirm this in your own environment before using it in production. The main thing to check for is that there is no Type 2 interactive logon in the logs of the remote host.

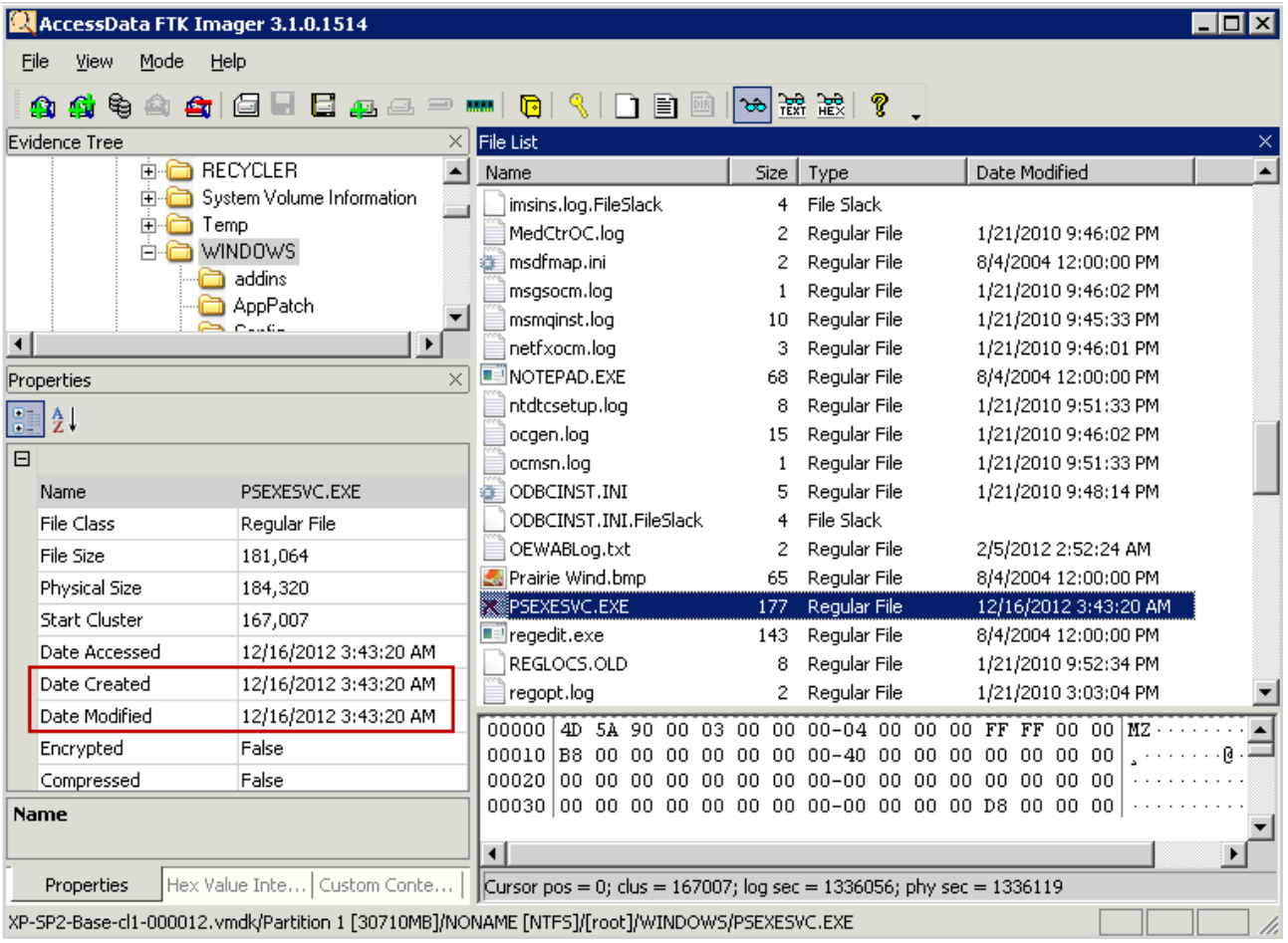
A Forensic Overview of *PsExec*

Since we sometimes see attackers use *PsExec* for lateral movement, let's take the discussion a step further and look at some artifacts that could reside from *PsExec* usage. The artifacts presented here are the direct result of using *PsExec* in my tests above. Of course there are other ways of utilizing *PsExec*, so the resulting artifacts may vary.

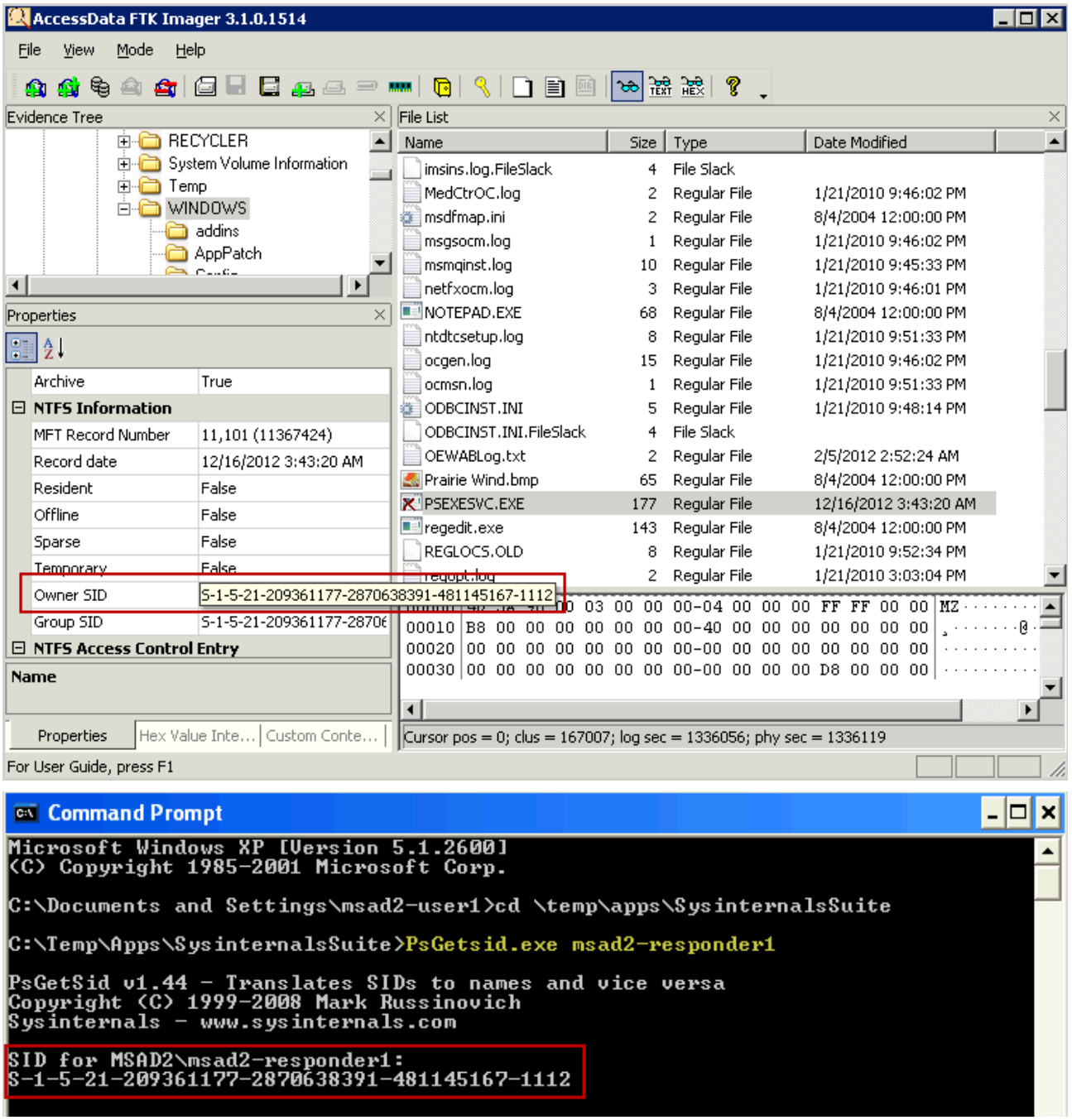
When *PsExec* executes on a remote machine, the local machine sends a service executable named PSEXESVC.EXE to the remote machine and that executable is installed as a service. Here's a look at that service running on the remote host. Notice the service executable resides in C:\WINDOWS.



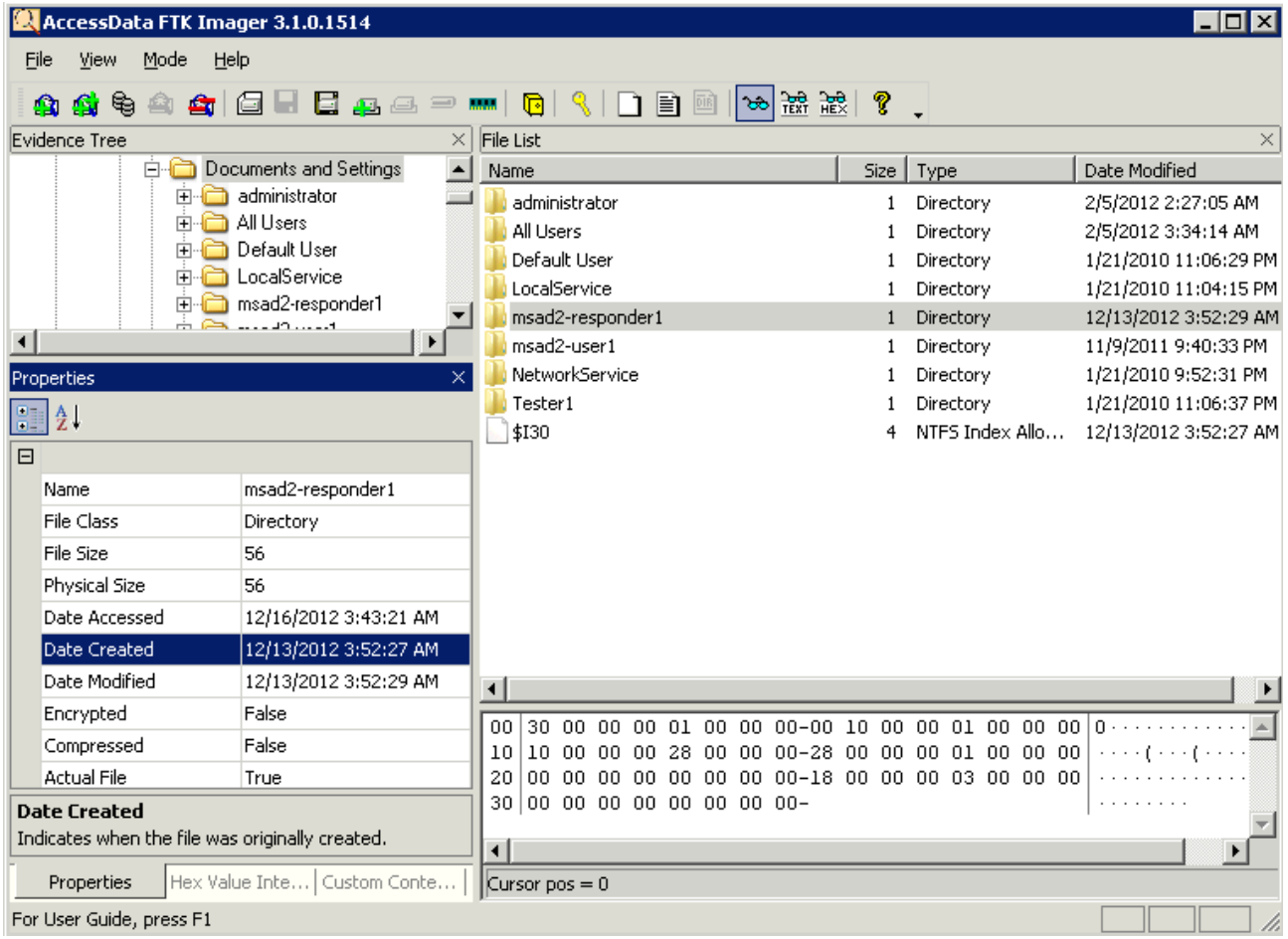
Once the user has cleanly logged off (exited) *PsExec*, the service is removed and PSEXESVC.EXE is deleted. Although *PsExec* is deleted (as indicated by the red X icon), the screenshot below shows the file and its metadata. Notice the UTC creation and modification times correspond to the second logon time in my tests above (12/15/12 9:43:20 PM Central time).



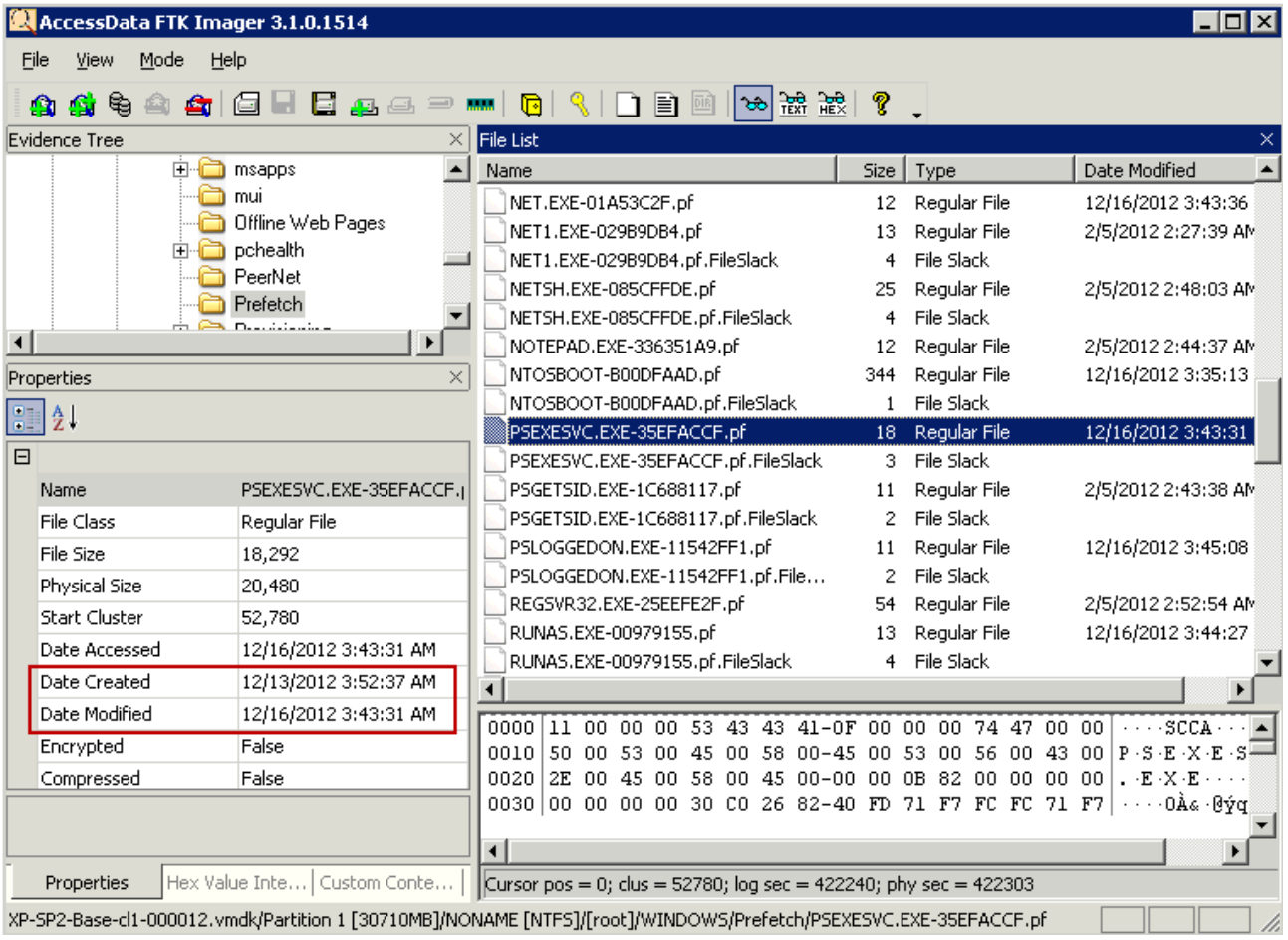
Also notice that the owner of this file is the user who connected remotely to run *PsExec*.



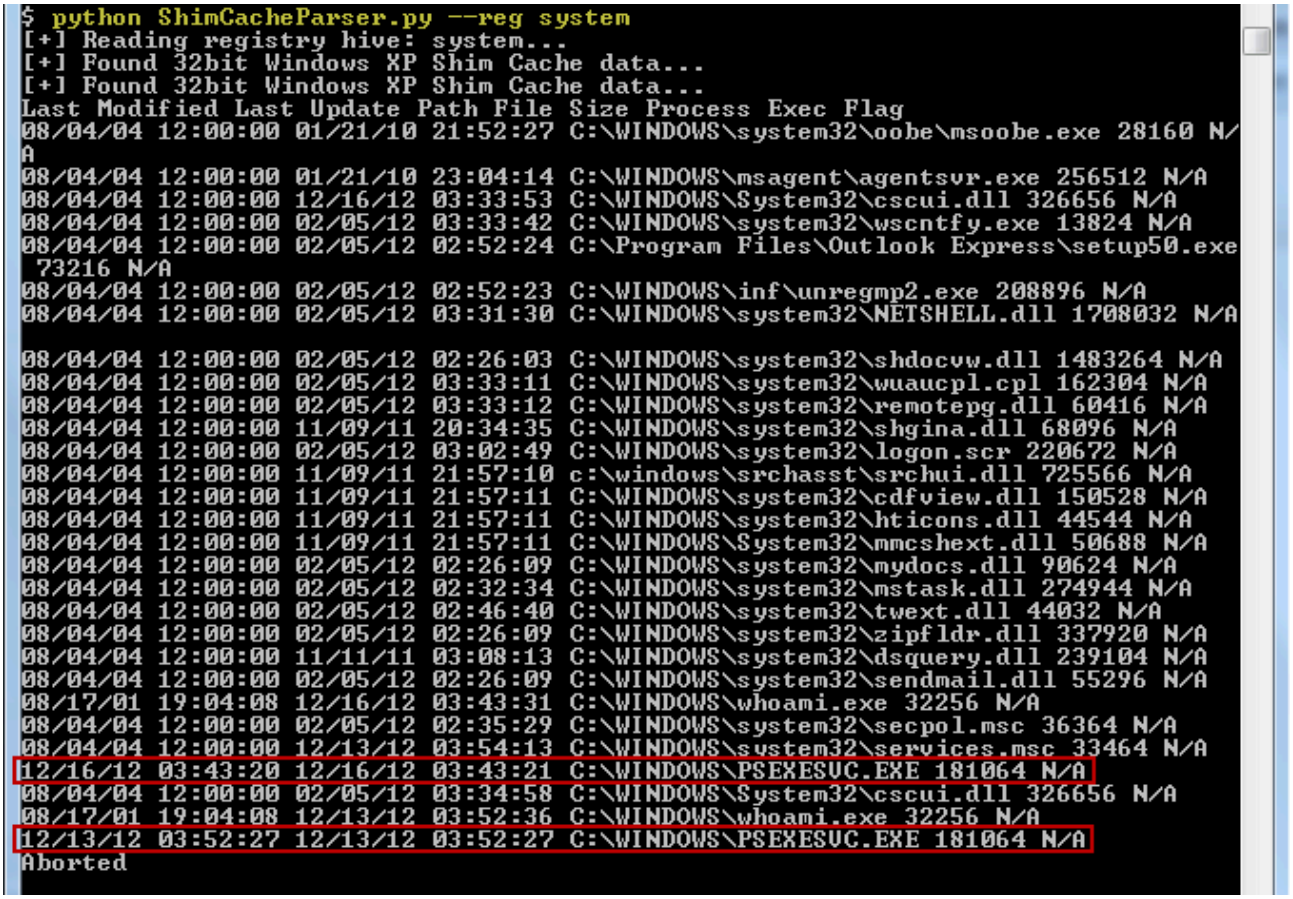
A profile is created for this user at the time of first login, if it didn't already exist. This is regardless of whether an interactive logon occurred or not. Here are the timestamps for the MSAD2-RESPONDER1 account's profile folder. Notice the UTC creation timestamp is the same time of the first logon in the tests above (12/12/12 9:52:27 PM Central time).



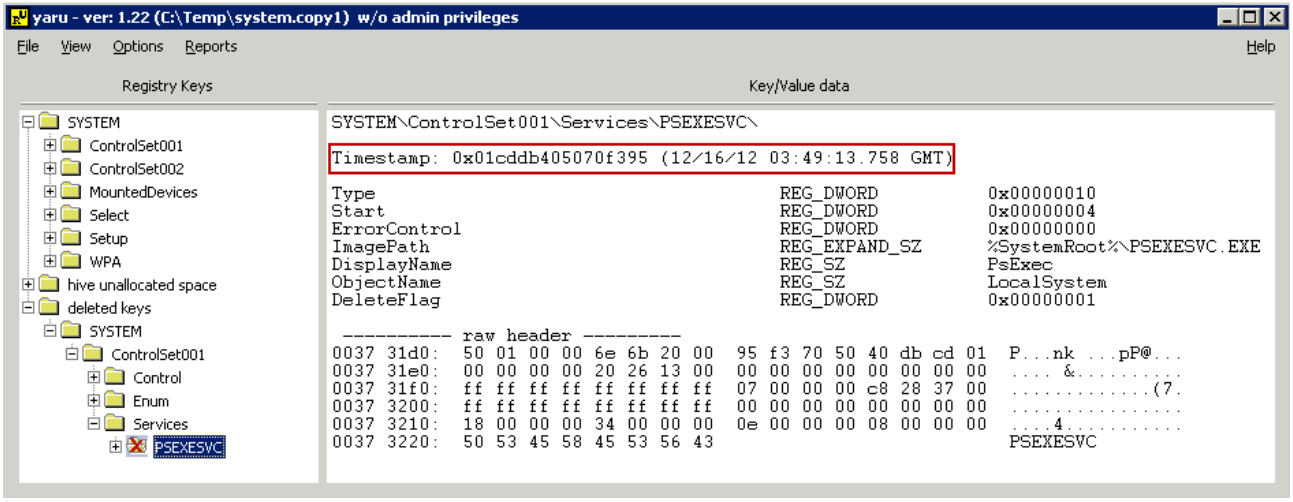
Prefetch shows both the first time the service executable was run (Prefetch file creation time minus 10 seconds) and last time run (Prefetch file modification time minus 10 seconds).



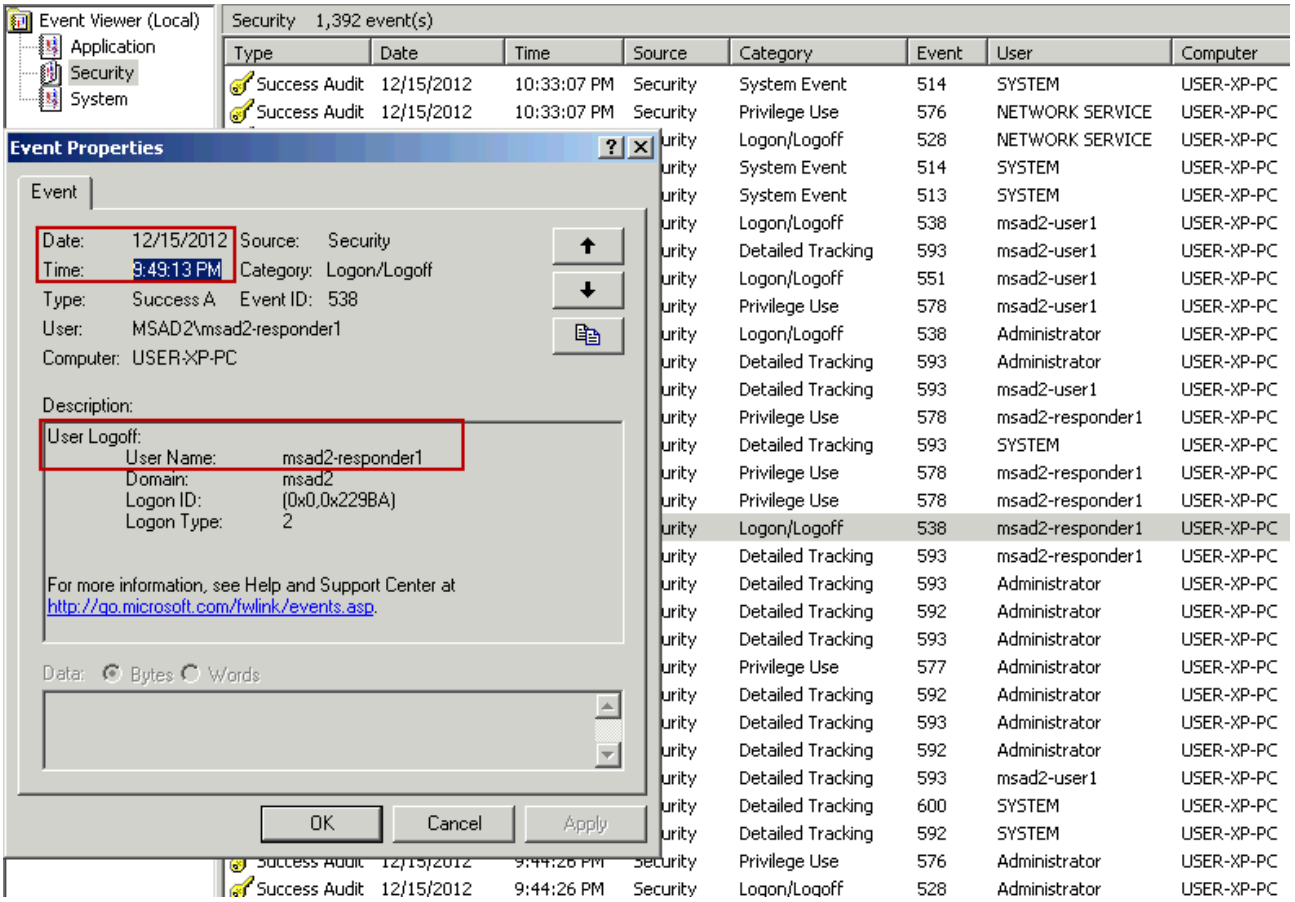
The Application Compatibility Cache shows this as well, with an entry for both times run.



Since *PsExec* is typically run remotely, these run times can be used to examine the Event Logs and look for network logons at the same time. The corresponding Event Logs for these *PsExec* executions are shown in the screenshots at the beginning of this article.



When *PsExec* is exited, the *PsExec* service is removed, but you may find the deleted service key still in the Registry. Here is the deleted key shown by YARU . Note that the last write time of the key corresponds to the time the responder account logged out and exited *PsExec* the second time (12/15/12 9:49:13 PM Central time)



Final Thoughts

I've found that there are safe ways to use *PsExec*, but unfortunately it is not obvious how to do so. Hopefully this discussion has provided a deeper understanding of the tool and how you can use it safely, as well as analyze the forensic artifacts it can leave behind.

Source: https://www.sans.org/blog/protecting-privileged-domain-accounts-psexec-deep-dive/