Attack, Defense, Forensics, Coding... Security

CAPTURING WINDOWS 7 CREDENTIALS AT LOGON USING CUSTOM CREDENTIAL PROVIDER

January 2, 2012 · *by* twrightson · *in backdoors, Programs* · *17 Comments*

For the Eternally Impatient

The quick lowdown: I wrote a DLL capable of logging the credentials entered at logon for Windows Vista, 7 and future versions which you can download at http://www.leetsys.com/programs/credentialprovider/cp.zip. The credentials are logged to a file located at c:\cplog.txt. Simply copy the dll to the system32 directory and run the included register.reg script to create the necessary registry settings.

The Detailed Technical Information

I started testing my rootkit on a windows 7 box and luckily most of it worked. The only thing that wasn't working was the ability to log credentials typed in when a user first logs in to Windows. I've had a custom GINA stub dll that's worked great for a while that I wrote years ago, it works with Windows 2000, XP and 2003. GINA is the Graphical Identification and Authentication component of Windows and handles the logon screen that we're all familiar with.

FEEDBACK ALWAYS WELCOME

If you've used something I've developed and have any comments, questions or features you'd like to see let me know. -Tyler

ARCHIVES

- January 2014 (1)
- November 2013 (1)
- June 2013 (1)
- August 2012 (1)
- July 2012 (1)
- January 2012 (3)
- December 2011 (1)
- November 2011 (1)
- October 2011 (3)
- November 2010 (1)
- June 2010 (1)
- September 2009 (3)
- August 2009 (1)

In the past you could choose to write your own GINA dll from scratch, or you could simply 'extend' the functionality of other GINA modules by creating a GINA stub dll.

Microsoft in their infinite wisdom decided to completely change the API and move away from GINA and the GINA model. Now to customize the logon experience you have to implement a Credential Provider, this is true for Windows Vista and newer (7 and 2008). Microsoft claims the reasoning behind this is to make it easier for developers to meet the demands for next generation authentication technologies (like biometrics, two factor and single sign on). Frankly in a way Credential Providers are a lot easier to work with, but in another (probably more accurate way) they're a huge pain in the ass to create our nefarious dll. From what I remember creating a GINA stub dll to log Windows credentials took me probably 3 hours. To get a credential provider to do exactly what I want took probably a good 40 hours. At this point I should probably thank my girlfriend for putting up with my obsessive programming and constant cursing.

The Credential Provider Model

I won't go over how GINA handled the authentication process, although if i get enough requests I might detail that in another post. Instead I'll focus on how the Credential Provider Model works. Credential Providers are 'In-Process COM objects', aka a DLL. Out of process COM objects would essentially be another executable that the interacting exe would interface with. COM or Component Object Model is a pretty ethereal term, it's basically a binary standard and language neutral way of implementing inter process communication, among other things. This is obviously oversimplifying it, but this isn't a discussion on COM. This was my first time working with COM programming so that definitely accounts for some of the additional time.

The default credential provider is the PasswordProvider that comes with Windows 7 and has a GUID of 6f45dc1e-5384-457a-bc13-2cd81b0d28ed. GUIDs are simply unique identifiers for our COM object. One of the most important things to understand is that Credential Providers are not responsible for actually

authenticating users. It simply gathers the necessary information, 'serializes' the data, and then hands off the credentials to the Local System Authority. Serializing the data simply involves putting the username and password (and potentially other variables) in a specific format and passing them back to logonui which then hands it off to the LSA.

A Tile is a new term for an important component of a Credential Provider. Credential providers are represented by unique tiles at the logon screen, although it's not a one to one mapping of one tile for each Credential Provider. For example the default password provider enumerates credentials or user accounts and creates a unique tile for each username. The user can then click the tile for their username, type in the password and be authenticated. Although the Credential Provider is not directly responsible for creating the graphical elements it does instruct Logonui of the fields it wishes to present to the user, this does actually make things a lot easier than programming the entire interface.

In this image you'll see that we have two credential providers co-existing. The default passwordprovider is on the left and our credential provider is on the right.

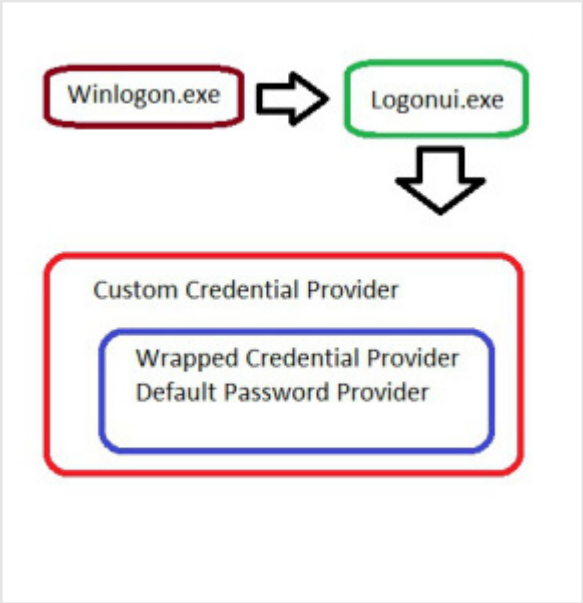
You'll notice that in this scenario there's no way for the user to distinguish between the two credential providers. So we need a way to 'force' the user into only using our credential provider. There's no way to truly 'force' a user to use a particular credential provider when multiple credential providers are presented. To accomplish our goal we can use the CredentialProviderFilter interface to filter out any other credential providers, more on this shortly.

Winlogon.exe launches Logonui.exe which then queries the credential provider and the credential provider returns information to logonui. The credential providers are defined in the registry and referenced by a GUID (a unique ID). The location

in the registry where credential providers are defined is: HKLM\Software\Microsoft\Windows\CurrentVersion \Authentication\CredentialProviders. To add a credential provider we create a new key with the GUID of the provider and copy the dll to the system32 directory. this is handled by the register.reg file included with ours.

Credential Provider Wrapper

We have the option of creating our own credential provider from scratch, filtering out any other provider and logging the credentials. This isn't optimal for a few reasons. Besides the fact that it's a considerable amount of work to create a credential provider to mimic all the existing functionality the main reason we don't want to do this is because we can't guarantee that the target system we wish to deploy our rogue credential provider on will look and feel exactly as the user expects it to. For example if the company has deployed a new credential provider to ask the user additional questions during log in and our credential provider does not include those same questions this could set off a huge red flag for anyone using that system.



Thus the best solution for us is to wrap the existing credential providers. The new process flow with a wrapped provider looks like this figure. Logonui will make the function calls into our custom credential provider, we can then choose to either handle processing for that function or hand it off to the wrapped credential provider.

We actually have many choices on how to log the username and password using our credential provider wrapper. The solution I chose was to manipulate the SetStringValue function. This function is called every time a user types a key for any field in the credential provider. Thus when a user types in a key we simply write that key to a file. This also allows us to capture ANYTHING a user types in to a field when they logon, so our provider should grab everything for next gen or non standard credential providers.

We also manipulate the GetStringValue function which gives us the string value of fields in the credential provider. This is handy because the default password provider does not prompt the user for their username. Instead the user clicks a tile with their username already filled in, thus we can't get the username from the SetStringValue function.

The implementation of my code to grab the user from the GetStringValue function demonstrates a handy feature of the way the wrapped and wrapper relationship works. In the sample wrapper from the Windows SDK logonui calls our getstringvalue function and we simply hand the variables off directly to the wrapped credential provider. The wrapped credential provider then performs the work of the getstringvalue function, everything works as it should, and all is good in the world. Since the ppwsz variable we give to the wrapped credential provider is a pointer to the PWSTR variable that will contain the string of the field specified we can access this variable when the wrapped function returns. Thus rather than implementing the getstringvalue function we simply hand off the variables as usual and print out the string from the pointer after the real function does all the work and returns.

The function for handing off the work for the GetStringValue function to our wrapped provider looks like this;

```
hr = _pWrappedCredential->GetStringValue(dwFieldID, ppwsz);
```

CredentialProviderFilter

As part of the custom credential provider I implemented the

credential provider filter. This might be one of the most helpful parts of this project/post for other developers. Microsoft has an FAQ included with their SDK samples for Credential Providers that states they will NOT release a sample credential provider filter… Thanks Microsoft for releasing a complex API and offering ZERO help, terrific.

Luckily I was able to find a broken example online, fix it, fill in some gaps and now it's fully functional. Thank you to the anonymous person who started the effort. Credential Providers are defined in the registry as well at HKLM\SOFTWARE\Microsoft \Windows\CurrentVersion\Authentication\Credential Provider Filters\. Again all you have to do is create a key with the GUID of the filter. This is included in the register.reg file included with my credential provider.

Logonui will call the Filter function for any configured credential provider filters. It will pass a BOOL variable for every credential provider configured, at which point the Filter can choose to allow or deny the specified credential provider by setting the BOOL variable to true or false.

Currently my filter only filters out the password provider, but it's incredibly easy to change it to filter out all other providers. The following evaluation checks to see if the credential provider GUID matches that of the default password provider, if so it filters it out by setting the rgbAllow BOOL to FALSE. You could change this evaluation to always be true and filter out all providers.

```
if (IsEqualGUID(rgclsidProviders[i],
CLSID_PasswordCredentialProvider))
rgbAllow[i] = FALSE;
```

Special Notes about The Credential Provider Wrapper

I originally had some issues when trying to compile on Visual Studio 2010 Express so I used Visual Studio 2008 Express and everything was fine. I have no doubt you could get this to compile with 2010, I just didn't feel like wasting time on it. In addition I had to use the _CRT_SECURE_NO_WARNINGS preprocessor

settings which are defined under the properties for the project in Visual Studio, under C/C++ -> Preprocessor Definitions. This allows us to use the fopen function rather than fopen_s, again, just laziness on my side. If Microsoft wants to force me to use a specific function it makes me want to do it less, just stubbornness from my childhood that's stuck.

I also had to use the following two defines to suppress the error 4995.

```
#define DEPRECATE_SUPPORTED
#pragma warning(disable:4995)
```

The GUID can be changed to whatever you want, just manipulate the define in the guid.h file. One of the best resources I found for helping to understand development of a credential provider was the 'Credential Provider Technical Doc.doc', I've included a copy of this doc in my full archive for the credential provider in case the link is ever broken on the interwebs. You can download the source and dll at http://www.leetsys.com/programs/credentialprovider/cp-devel.zip

Capturing Windows 2K ...
In "backdoors"

Insider Rogue Certificati...
In "General"

Social Engineering - Sc...
In "Penetration Testing"

17 Comments

dan · March 6, 2012 - 12:21 pm · *Reply→*

Good work – 6 years ago I had created a GINA stub that simply wrote the pswd out to a file and then jumped to

an external reference to the original GINA that I included. It worked for our pen tests but I saw the direction that MS was going with Cred Providers and retired my minimal C skills. Still love writing in Assembler so when I get motivated I will re-write this is ASM – where I have some skills still.

Again – good work – hope your girlfriend takes you back :-)

Trent · June 21, 2012 - 3:26 pm · *Reply→*

Unless I'm missing something, that last archive (cp-devel.zip) doesn't contain any of the provider filter code.

Trent · June 21, 2012 - 3:29 pm · *Reply→*

yep, I was missing something. Saw the empty filter.cpp and filter.h files and jumped to conclusions. You can delete both comments.

Thanks.

Rick McGee · July 31, 2012 - 11:02 am · *Reply→*

Writing a GINA stub for is not a simple process as I embarked on this journey in 2004. Capturing a user's credentials is one thing, capturing a user's password during login is another matter altogether. Hooking the NT Authority security context was easier; defeating "all" local security implementations. My project included writing data to a custom Windows Event Log among other things. I secured a federally registered copyright for this effort (See "Patriot Computer Software",

February 2006). As a former federal government employee, I lost my job over the entire matter. Reporting this vulnerability to Microsoft in 2008 via Dr. Mark Russinovich, yielded a ridiculous reply from their CERTS ~ reciting Microsoft's "10 Immutable Laws of Security -> Law #6: A computer is only as secure as the administrator is trustworthy". rick.mcgee@object-vision.com Write, as I might make it available for download and evaluation.

JW · August 26, 2012 - 11:41 pm · *Reply→*

Excellent write-up and great job on figuring out the filters implementation.

Just FYI for those who may be trying to set this up or use it as an example, with vs2012 and the windows v8.0 sdk installed, the CLSID that is exported to user space for the default provider...

CLSID_PasswordCredentialProvider

used in CoCreateInstance call to attach the wrapper to the underlying provider is no longer the correct CLSID for a stock windows 7/vista setup. It now exports the windows 8 CLSID as default.

This means that if you have already installed vs2012 and try to use the attached credential provider in this post it won't load because it will try to attach to the new provider which isn't in win7 yet.

To fix, go into the SetUsageScenerio function of the sample provider class and change the CLSID passed to CoCreateInstance to

CLSID_V1PasswordCredentialProvider

After recompile follow install directions given in the post and it should attach to your win7 cred provider as expected.

bhm · March 12, 2013 - 4:17 am · *Reply→*

Saved my day with that comment about the changed constants in VS. .)

Krzysztof Zajac · September 14, 2012 - 5:43 pm · *Reply→*

Hi, how are your changes to the Microsoft's code licensed?

Birajendu · October 25, 2012 - 2:13 am · *Reply→*

I am trying this sample to capture Username and password. in a window7 machine. Here I have only configured a local User. I am able to get the user name string through GetStringValue() fucntion. But The SetStringValue is never being called. How to capture the password string?

Paul Schwartz · March 6, 2013 - 4:01 pm · *Reply→*

Great stuff, thanks you cleared up some issues I was having with a provider I'm developing. For your next miracle, if you know any good articles on Local Security Providers I'll name my next male child after you.

Robert Insomniac · March 20, 2013 - 2:58 pm · *Reply→*

This works pretty well on x86 Windows 7 Pro. But how about to build a x64 version?

twrightson · March 23, 2013 - 11:21 am · *Reply→*

Great. I'll try to find the time to get an X64 version rolled out soon.

Jposso · May 2, 2013 - 10:02 pm · *Reply→*

Here is a very high level article…
http://technet.microsoft.com/en-us/library/ff404303(v=ws.10).aspx

Details of wrapping and API…
http://archive.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=ShellRevealed&DownloadId=7341

Happy coding…

pithhelmet8669 · June 6, 2013 - 8:24 am · *Reply→*

Tyler – and everyone -

I got the 64 bit version working….

just had to set the VS2010 to compile for 64bit

There was an error in a default config, indicating you need to change the Treat Warnings as Errors to no (/WX-)

I also added this pragma
#pragma warning(disable : 4786)

compiled and tested on both x86 windows 7, and 64bit windows 7.

Thank you!!

Robert Insomniac · July 15, 2013 - 2:38 pm · *Reply→*

Please upload x64 version to the file sharing (rghost or mediafire), so I'll be able to download it. Thanks in advance!

pithhelmet8669 · July 18, 2013 - 1:13 pm ·

Hi Robert -

I'm sorry, i do not understand your request. Do you want me to upload the IDE settings, or the IDE itself or the code? The code did not change, just a couple of configs in the IDE and your good to go.

Robert Insomniac · August 19, 2013 - 6:36 pm · *Reply→*

I mean an already built DLL file for x64 platform ...because I have no enough disk space to install Visual Studio =)

Capturing Windows 2K and XP Credentials at logon using stub GINA DLL | Tyler Wrightson's Security Blog · November 3, 2013 - 4:14 pm · *Reply→*

[...] I had grabbed the gina stub example from MSDN and simply tweaked the WlxLoggedOutSAS() function to log the credentials. Just a reminder that this will not work for Vista and later Operating Systems, as they have switched to the Credential Provider model. To accomplish the same thing in those Operating Systems you can check out my custom credential provider at http://twrightson.wordpress.com/2012/01/02/capturing-windows-7-credentials-at-logon-using-custom-cre…. [...]

Follow

Leave a Reply

Enter your comment here...

← Covert System Manipulation Tool – SimSim

Insider Rogue Certification Authority Attack →