Pass-The-Hash Toolkit - Docs & Info

Pass-The-Hash on Windows, a little history

Back in 2000, Ochoa wrote the paper "Modifying Windows NT Logon Credentials" (available here) describing how to implement the 'pass-the-hash' technique inWindows, back then the attack was only being implemented using a modified SAMBA client, and the functionality implemented in SAMBA at that time was more limited compared to what is available today . I was unable to public the tool and the code, but if you remember someone demonstrating the attack using a tool, that was probably my code (or not, I don't know, who cares :)).

Pass-The-Hash Toolkit

Pass-The-Hash Toolkit is comprised of three tools: IAM.EXE, WHOSTHERE.EXE and GENHASH.EXE, described next:

GENHASH.EXE

This is just a utility that uses some undocuments windows functions (the fact thaty they are undocumented doesn't mean anything more than that, only that they are undocumented :)) to generate the LM and NT hash of a password. This tool is useful to test IAM.EXE and WHOSTHERE.EXE and perhaps to do some other things. Pretty simple and small tool.

IAM.EXE

If you have ever used a modified SMB client (SAMBA, SAMBA-TNG, Impacket, some of the smb-related IMPACT modules, metasploit I think, etc) with NTLM hashes instead of the cleartext password (to avoid cracking the hashes) to access a remote machine, you know that the functionality implemented by any of these tools is useful but limited; it is very hard/time consuming/almost impossible to implement every single feature implemented by Windows that can be accessed remotely.

This includes remote registry access, remote domain administration, remote MS SQL server administration, Exchange administration, etc. Some of these APIs are implemented by free tools allowing the use of NTLM hashes directly, but others are not.

There are also lots of third-party applications with their own remotely accessible APIs that use Windows authentication and for which you are not going to find implementations allowing you to use NTLM hashes directly instead of a clear-text password.

At this point is when IAM.EXE comes into play. Using this tool you can change your current logon session information; you can change the username, domain/workgroup name, and NTLM hashes that will be used to authenticate with remote services using Windows authentication over the network. So, if you compromised a machine and used pwdump or any other tool/technique to get the hashes of the passwords of users, you don't need to bother cracking the hashes anymore because you don't need the clear-text password anymore, just use the hashes with IAM.EXE (of course, having the clear-text password is still perhaps useful, because the same password could be used in other services that do not use Windows authentication, but that's another story, I digress).

For example, instead of using a modified smbclient to list shares, or read the registry remotely, you can now use Windows instead. Just run IAM.EXE with the credentials, and then use 'net use', 'net view', Explorer, psexec (http://www.microsoft.com/technet/sysinternals/utilities/psexec.mspx) to obtain a remote shell on the target, any domain administration tool, the MS SQL Server/Exchange administration tool etc, as you would normally do if you had the clear-text password for the user you are trying to impersonate.

If you have the client for a third-party application that uses NTLM authentication, just run the tool and you are good to go. There are no limitations, you do not depend on finding an implementation allowing you to use hashes directly, simply use the regular windows client for that particular application after running IAM.EXE.

IAM.ExE is used like this:

iam.exe username domainname LMhash NThash

for example:

iam.exe administrator mydomain 01020304050607080 90A0B0C0D0E0F10 01020304050607080 90A0B0C0D0E0F10

WHOSTHERE.EXE

Another tool in the toolkit is WHOSTHERE.EXE. This tool allows you to list the logon session information that is kept by windows in memory. Running the tool gives you the username, domain/workgroup name, and NTLM hashes of the credentials used to create those logon sessions. The logon session information stored in memory that you can obtain is the one of the currently logged on users, and in some cases, of users that logged on to the machine at some point but that are not logged on right now (which is very interesting, there's bug, more on that in the future).

When you log to your desktop interactively, that creates a logon session, when you log into a machine remotely using Remote Desktop/Terminal Services, that creates a logon session, when you run the 'runas' tool, that creates a logon session, etc. There are also third-party applications that create logon sessions as well.

A practical example on how to use the tool:

Let's say you are pentesting a network with a Windows Domain. You managed to compromise a MS SQL Server joined to the domain. At this point, you can obtain user accounts stored on the local SAM database, but NOT user accounts stored in the Domain Controller. You have control over that particular machine, but not over the whole Windows domain. It would be great to have control over the Whole Windows Domain...

You obtain the username and hashes of the local SAM database (of the MS SQL Server you compromised), and use IAM.EXE on your own desktop machine to, for example, try the administrator account on ther servers on the network (including the domain controller) to see if the password is the same. but no luck..

You analize inbound network traffic for that MS SQL Server machine and you realize that the domain administrator (not necessarily the 'administrator' account, but any other user with 'Domain Admin' privileges for that matter), logs on to the MS SQL Server you compromised using Remote Desktop...that's your way in..

You upload whosthere.exe to the compromised server, run it, and you observe scrolling down the screen the hashes of the domain administrator.. Now you go to your desktop machine again, use those hashes with IAM.EXE, and connect to the Domain Controller. You have now compromised the whole domain. You can do anything you want using regular windows domains administration tools.

This is a very interesting exercise to do, you are going from the compromise of a particular server, to the compromise of the whole domain, without making any 'noise' on the network. One thing that makes the matter worse, is that as I mentioned before, the logon session information is not always ERASED from memory after a session is closed (e.g.: After a Remote Desktop connection is closed); this means that the attacker does not need to wait for the domain administrator to log into the server to steal its credentials, if the he/she logged into the compromised server at some point in time, that could be enough.

Also, next time someone asks you 'hey!, could go connect to my computer remotely using remote desktop? something is not working and I need your help', don't do it! :) if he has local admin privileges he will be able to steal your credentials.

I hope this helps you get a better idea on how the tools can be used in a real world scenario. If you are a penetration tester, this a good technique to try, if you are a network administrator, this will give you an insight of the dangers of logging remotely to a workstation, and the importance of having that workstation as secured as possible, because the compromise of that workstation can lead to the compromise of the whole domain.

IAM.EXE and Windows Server 2003

if you run IAM.EXE and it ends as expected, as If it had worked, but then you run WHOSTHERE.EXE and the credentials did not change, do the following:

-start a cmd.exe using runas, for example:

runas /user:administrator cmd.exe

and in the new console run IAM.EXE, and then WHOSTHERE.EXE to verify. And now it should work.

It seems that sometimes you need a new session different than the interactive session for LSASS.EXE to accept the modifications to the credentials in memory. If you are logging to the machine remotely using psexec/Remote Desktop etc this does not to occur (at least, this is what I observed), I had troubles like this when logging interactively to the server. Also after you run 'runas', running IAM.EXE in a regular CMD.EXE shell will start working. Don't take any of this as a precise explanation of what's going on, this is just what I observed and a way to work around it. I'll analyze what's really going on in the future.

Pass-The-Hash toolkit and LSASRV.DLL

IAM.EXE/WHOSTHERE.EXE reads at specific locations of LSASRV.DLL's address space to obtain data necessary to encrypt the credentials before changing them and other stuff. For that reason, IAM.EXE/WHOSTHERE.EXE has specific code that checks for the LSASRV.DLL version present on the system where it is run, and if it does not match with the ones I know, the program exits. Version 1.1 of the toolkit now includes the -B option (both for IAM.EXE and WHOSTHERE.EXE) that uses some 'heuristics' to try to find the addresss in runtime with the hope of covering more OS versions without having to check for every possible OS version,language, and LSASRV.DLL version.

The idea behind this is to avoid situations where you would run the tool in a system that doesn't have the correct LSASRV.DLL version most likely crashing the LSASS.EXE process and having to reboot your machine (or a remote machine if you are using WHOSTHERE.EXE, etc). not good :).

So, if you run IAM.EXE or WHOSTHERE.EXE and get something like this:
Checking LSASRV.DLL....Unknown LSASRV.DLL.
LSASRV.DLL: 00050001h. A280884h

And you also run IAM.EXE/WHOSTHERE.EXE with the -B switch and it doesn't work either, It means I don't know about your DLL version. Please send me an email with the version number you have and I'll do my best to get a hold of a copy of that exact DLL version to solve the issue. (when you are at it, also send me the text representation of the DLL version just in case , just rigth-click the DLL, properties->Version->File Version, and also the language of your windows installation, etc.)