



Mike Pilkington

Protecting Privileged Domain Accounts: LM Hashes: The Good, the Bad, and the Ugly

February 29, 2012

[Author's Note: This is the 2nd in a multi-part series on the topic of "Protecting Privileged Domain Accounts". 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.]

I realize the LanMan (LM) hash is old-hat for many, but I've recently discovered that the LM hash is even more dangerous than I previously realized. This is due to both the ease of cracking LM hashes on today's hardware, as well as the obscure fact that there is currently no Microsoft-provided feature to remove LM hashes from memory. So even if you think you've heard it all with regard to LM hashes, I encourage you to read on to make sure you are aware of LM hashes lurking in memory.


The Bad


Although the issues with LM hashes are well-documented, let me just briefly describe how LM hashes are created and you will immediately recognize several significant issues with this algorithm. These issues highlight exactly why we need to get rid of the LM hash. In a nutshell, here's how the algorithm works, per [Wikipedia](#) [↗](#):

- The user's ASCII password is converted to uppercase
- This password is null-padded to 14 bytes

- The "fixed-length" 14-byte password is split into two 7-byte halves
- Each of the two halves is used with the DES function to create two 8-byte cipher-text values
- The two cipher-text values are concatenated to form a 16-byte value, which is the LM hash
- No salt is used in the creation of the hash


There are plenty of bad choices that were made in the design of this algorithm. Let me quickly point out a few:

- The fact that each character is an ASCII character, which [provides just 95 printable characters](#) , means that there should be only 95 possibilities per character of the password. However, it's worse than that because each lowercase character is converted to uppercase, so in fact there are only 69 possibilities per character of the password.
- The maximum number of characters in a password is 14. Anything less than that gets padded with null bytes to 14. The resulting 14-character value is then split in half. Splitting the value in half greatly reduces the search space for any password cracking software. It now just has to guess 2 values within 69^7 (~7.5 trillion) versus one value within 69^{14} (~55 septillion). This results in nearly 13 orders of magnitude difference, making it a significantly easier problem for cracking software to solve for the two 7-character halves.
- Finally, there is no salt. A salt is an extra piece of data, which typically differs per machine and/or per user, that is added to the clear-text password so that the output of the hashing algorithm will not be the same when two users have the same password. For example, without adding a salt, it means that if your password is "abc123" and my password is "abc123", our hash values will be the same. Since they are the same, it's possible to pre-compute many common (or uncommon) passwords and store them with their corresponding hash for quick and easy retrieval.

These significant issues are the reasons that pre-computed rainbow table attacks are so effective against LM hashes. The most effective pre-computed attack against LM hashes that I have seen was brought to my attention by Chad Tilbury. While teaching SANS Forensics 408, Chad pointed out to the class [this project](#)  to put the LM hash rainbow tables on solid state drives. This is an incredibly fast and inexpensive solution for cracking very complex 14-character passwords in just 5-11 seconds! What does this mean? This means that if your (or your users') LM hash is available to an attacker, **they will know the clear-text password almost instantly.**

The Ugly

So if all that isn't ugly enough, let's move on to the really ugly stuff. But before we do, I need to point out that Microsoft has long recognized the weaknesses of the LM hash and provided us with some techniques for disabling its use. Let me review these techniques and then I'll show you a glaring omission in Microsoft's attempts to protect us from the LM hash.

By default, Windows Vista and higher no longer store LM hashes on disk (in the SAM database) by enabling the "NoLMHash" Registry setting. Furthermore, [this MS article](#)  describes how to disable it for older operating system versions, as well as in Active Directory. In a nutshell, it describes how to push the NoLMHash setting via Group Policy, as shown here on my test domain controller:

"NoLMHash" Registry setting "NoLMHash" Registry setting

Another mitigation option from Microsoft provides a way to prevent Windows from authenticating with the LMv1/LMv2 challenge-response network authentication protocol. In the next few weeks I will post an article to discuss network authentication in-depth, so I won't go into all the inner workings here. Suffice it to say that an LM hash is required to complete LMv1 or LMv2 challenge-response network authentication. So if we disable the LM challenge-response protocols as provided by Microsoft, and specify that only the newer NTLMv2 protocol be used (which only requires the NT hash, not the LM hash), then Windows shouldn't need the LM hash, right? That's my thinking.

So here I have changed the network authentication level to the most secure setting, which is to only allow NTLMv2 (as well as Kerberos)-effectively disabling the LM challenge-response protocols:

NTLMv2 NTLMv2

Now I want to test the effectiveness of these changes and see if it truly disables LM hashes as I would expect. I will do this on a Windows 7 SP1 system, hoping that the latest OS will provide the necessary protection against storing LM hashes.

On my Windows 7 test machine named USER-WIN7, which is joined to the MSAD2 test domain I used in [my previous article](#), I updated Group Policy to reflect the changes made above to the domain's GPO. Here are the new settings reflected on the Windows 7 client:

Windows 7 client Windows 7 client

For good measure, I have also changed local account MIKE's password and domain account MSAD2-RESPONDER2's password and then rebooted the machine:

MSAD2-RESPONDER2 MSAD2-RESPONDER2

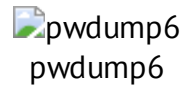
Now here's where it gets really ugly. Having implemented the policy to disable the storage of LM hashes, we should not see LM hashes stored on disk. And by implementing the policy to disable the LM challenge-response protocols over the network, there should be no need for the LM hash to be stored in memory. After all, why would it, since it is not needed for local or remote use now? Well, time to see what happens in practice.

After making the changes above to presumably prevent the use and storage of LM hashes, I've logged on at the console of the Windows 7 host as domain user MSAD2-RESPONDER2. I have also issued the *RunAs* command to logon the local user MIKE. This will allow us to see both users' hashes loaded into memory. (Remember, an account

must have a current "interactive" logon session in order for the hashes to be found in memory?see my [article on password hashes](#) for details).

Let's first look at the local hashes on disk.

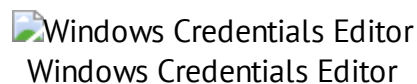
- So here I've run *pwdump6*, which will show the hashes on disk (in the SAM), for the local user accounts only:



As expected, no LM hashes are available. The first 2 accounts are disabled by default and no password was ever set for them. For user account MIKE, only his NT hash is available. So far, so good!

Now we need to look at what hashes are in memory.

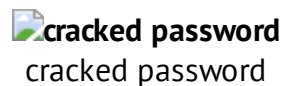
- For this task, I'll use [Windows Credentials Editor](#) [↗](#) (WCE).



This is not good at all! It turns out that Windows calculates and stores the LM hash in memory, assuming the password is less than 15 characters, **regardless of what the host or domain settings are for LM hash storage and LM challenge-response!**

Now with the LM hashes in hand, we can easily crack the passwords. Here I'm using the previously discussed [SSD implementation](#) [↗](#) of the LM rainbow tables from [Objectif Sécurité](#) [↗](#):

- Local account MIKE's cracked password:




- Domain account MSAD2-RESPONDER2's cracked password:



This is scary stuff! This means that essentially any user who has their machine comprised and hashes dumped from memory will have effectively divulged their clear-text password, since freely-available LM rainbow tables can crack the most complex 14-character passwords in just seconds.

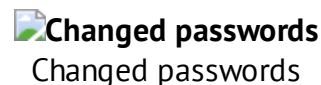
Let that sink in for a second... I discussed last time how devastating pass-the-hash attacks can be, which is absolutely the case. However, if attackers can just get the actual password, in most environments that opens up many more systems to attack from inside and often outside the network.

This issue of having LM hashes in memory was identified in [Hernan Ochoa's paper](#)  describing his tool *Windows Credentials Editor* (see page 23 of his paper). For confirmation beyond Hernan's paper and my own testing, I contacted Microsoft about this issue and the unofficial response I received was that they are aware of the problem. The general message was that even if they were to fix this problem, there are still the significant issues of pass-the-hash as well as LSA-encrypted passwords (which I'll cover next week), so they don't see enough reward versus the risk of changing a significant piece of their code base.

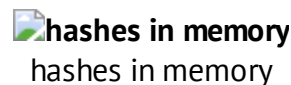
The Good

I realize this is a stretch, but here's what I will label as the one "good" thing about the LM hash?the fact that a 15-character password or greater will completely break the algorithm, and thus Windows *cannot* calculate and store the LM hash in memory (or on disk). So here's our one defense for this issue, **make your passwords 15 or more characters!**

To demonstrate this, I will change local user MIKE's password and domain user MSAD2-RESPONDER2's password to a value greater than 14 characters:



Now after a logoff/logon, let's recheck those hashes in memory:



Whew! The one redeeming value of the LM hash algorithm?the fact that we can completely break it with a sufficiently long password. A nice side effect of this is that with a password of 15 or more characters, you'll also make your NT hash very resistant to pre-computed attacks. That said, we still have to protect the NT hash to avoid pass-the-hash attacks. As discussed [in my last article](#), the way to do this is to avoid interactive logons with privileged accounts.

Conclusion

With the effectiveness of LM rainbow tables on today's hardware, we have to consider LM hashes effectively equivalent to clear-text passwords. Unfortunately, we do not have a Microsoft-provided mechanism for disabling the LM hash in memory. So for accounts you use for interactive logons, your only defense is to make sure you set your passwords to be 15 characters or more in order to break the LM hash algorithm.

This is a fairly easy solution for one-off accounts, but it gets much more difficult when trying to protect thousands of end-user accounts. While you will probably get significant push back (and maybe some laughter too), it is worth at least discussing the implementation of a 15-character password policy, or *passphrase* policy, for your organization. That, or put pressure on Microsoft to give us better option.

More to come next week, so stay tuned!

Mike Pilkington is a Sr.Security Analyst and Incident Responder for a global Fortune 500 company in Houston, TX, as well as a SANS Mentor and Community Instructor.

Tags: Digital Forensics, Incident Response & Threat Hunting

Related Content

Blog

BLOG

HUMINT and its Role within Cybersecurity

By Jon DiMaggio

Digital Forensics, Incident Response & Threat Hunting · October 4, 2024

HUMINT and its Role within Cybersecurity

This blog explores HUMINT's role in cybersecurity, detailing its implementation, benefits, and potential risks.



Jon DiMaggio



Blog



FOR528: Ransomware & Cyber Extortion Updates Implemented – What's New?

Digital Forensics, Incident Response & Threat Hunting · January 16, 2024

FOR528: Ransomware & Cyber Extortion Course Updates Implemented – What’s New?

The recent FOR528 course better addresses the differences between ransomware and cyber extortion, and provides new hands-on labs and bonus content.



Ryan Chapman



Blog

The Top 15 Summit Talks of 2023

Cybersecurity Insights, Digital Forensics, Incident Response & Threat Hunting, Cyber Defense, Cloud Security, Open-Source Intelligence (OSINT), Cybersecurity Leadership, Security Awareness, Artificial Intelligence (AI)

• December 18, 2023

Top 15 SANS Summit Talks of 2023

This year, SANS hosted 16 Summits with 209 talks. Here were the top-rated talks of the year.



Alison Kim



Subscribe to SANS Newsletters

Receive curated news, vulnerabilities, & security awareness tips



By providing this information, you agree to the processing of your personal data by SANS as described in our [Privacy Policy](#).

- ☒ SANS NewsBites
- ☒ @Risk: SecurityAlert
- ☒ OUCH! SecurityAwareness

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

Subscribe

Register to Learn	Job Tools	Focus Areas
Courses	Security Policy Project	Cyber Defense
Certifications	Posters & Cheat Sheets	Cloud Security
Degree Programs	White Papers	Cybersecurity Leadership
Cyber Ranges		Digital Forensics
		Industrial Control Systems
		Offensive Operations