Hashing

Hashing is a one-way cryptographic function which takes an input and produces a unique message digest as its output. Because this function is one-way, there's no way to determine the original message based on the message hash or hash digest it's outputted. The resulting message digest acts like a digital fingerprint for the original file. Another unique thing about a hash digest is that they are always the same length, regardless of how long your input is. Whether I input a file containing one word or a file containing millions of words, the output will always be the same length based on the hashing algorithm chosen. By far, the most commonly-used algorithm is **MD5**. The MD5 algorithm creates a 128-bit hash value that is unique to the input file.

Unfortunately, because the hash value output is only 128 bits long, it can create only a limited number of unique values, and this can lead to two files having the exact same resulting hash digest. When this occurs, this is known as a **collision**.

Due to the limited number of unique hash values associated with MD5, a newer algorithm called the Secure Hash Algorithm, or **SHA**, was created. SHA-1, for example, creates a 160-bit hash digest, which significantly reduces the number of collisions that occurred. **SHA-2** is a family of hash functions that contains longer hash digests. This includes the SHA-224, SHA-256, SHA-348, and SHA-512 hash functions, each of which has a digest between 224 bits up to 512 bits.

SHA-3 is the newest family of hash functions, and its hash digest can go between 224 bits and 512 bits, just like SHA-2. The major increase in security, though, with SHA-3, is that it uses 120 rounds of computations to create its message digest for each unique file.

Now, there are other hash functions available that you may come across in your daily work. These include things like RIPEMD and HMAC.

RIPEMD is the RACE Integrity Primitive Evaluation Message Digest. It comes in 160-bit, 256-bit, and 320-bit versions. But the 160-bit version is by far the most common among these. It's written as RIPEMD-160 and it's an open-source hashing algorithm created as a competitor to the SHA family, but it hasn't really gained the same level of popularity that SHA has.

Another hashing algorithm is known as the HMAC, or Hash-based Message Authentication Code. This is used to check the integrity of a message and provide some level of assurance that its authenticity is real. HMAC actually uses other

hashing algorithms to do the work, though, and it's called something like the HMAC-MD5, the HMAC-SHA1, or the HMAC-SHA256, depending on the underlying hash being used.

Digital Signature - Integrity - Non Repudiation

A digital signature is created by hashing a file and then taking that resulting hash digest and encrypting it with a private key. So, if I was going to send an email that is a couple of pages long and I wanted to digitally sign it to make sure you know that nothing was changed inside that email, I can run that email message through a hashing algorithm, like SHA-1. Then, I take that resulting 160-bit hash and I encrypt it using my private key. When I send the email to you, I'm going to attach the resulting encrypted hash with it, as well, and this is going to prove the integrity of the message and create non-repudiation. When your system receives the email, it will then decrypt the digital signature using my public key, which is going to provide you with that original 160-bit hash digest. Your system, then, takes my multiple-page email, runs it through the SHA-1 algorithm, and compares your message digest that you calculated with the one that I sent as part of my digital signature. If those two things match, then you can be assured that the email was not modified in-transit between my system and yours, and this provides us with that integrity check.

Now, since I also encrypted my SHA-1 digest with my private key, and only I have my private key, this also assures you that the person who sent the message is the only person who could have sent you the message. This provide us with the non-repudiation on the email. This non-repudiation means I can't claim that I didn't send the email to you because I'm the only one who could have because I'm the only person who has my private key.

DS-Algorithms & Code Signing

For digital signatures to be utilized, you should use either the Digital Security Algorithm, **DSA**, the Rivest-Shamir-Adleman cipher, **RSA**, or the Ecliptic Curve Cryptography version of DSA or SHA. The federal government has decided to use the Digital Security Standard, called DSS, which relies upon a 160-bit message digest created by DSA. Now, most commercial entities rely upon the RSA standard, though, because it's faster and can be used for digital signatures, encryption, and key distribution. Digital signatures have been expanded beyond just email, too. Code signing of our files relies upon the digital signature for a program or file. For example, if I created a mobile app and I wanted to put it into the app store like Google Play or the Apple App Store, the installer file would have to be digitally signed and that is

called code signed. Every developer must register with Apple or Google and they receive a private key. Just as in the email example I provided earlier, the application file is hashed and that hash is encrypted using the developer's private key. This is known as code signing and ensures that the installer hasn't been modified or corrupted since that developer published it.

Windows Password Hashing

In a Windows machine, passwords aren’t stored in cleartext, and they're not even stored in an encrypted format. No, they're actually stored as hashes. The original version of this was known as LANMAN, or the LAN Manager hash, or simply the LM hash. This was created all the way back in the late 1980s, even before Windows and T-servers roamed the Earth. This hash was based on the DES algorithm and was limited to 14 characters. Not only is this weak because it used DES, but it's even worse because Microsoft had the password broken into two seven-character chunks first and then one of those was converted to uppercase and then it was run through the encryption algorithm to create the hash. This reduced the number of possible combinations and lead to decreased security in the LM hash. Because of this, you should always disable the LM hash on your modern Windows OS and, by default, it is disabled.

To replace the LANMAN hashes, Microsoft created a replacement known as the NTLM hash, or NT LAN Manager hash. This was created to replace the LM hash once NT servers became popular in the early 1990s, and it first shipped in 1993, beginning with NT 3.1. That shows you how old this is. The NTLM used RC4 instead of DES for the way it created its hash, so again, something stronger is definitely needed.

On modern Windows machines, like the LM hash, NTLM is disabled by default. The final and newest version of password hashing for Windows is known as NTLM version two. It relies on the HMAC-MD5 hash, and is therefore, a little bit more difficult to crack. It's been around since Windows NT version four, but it's still used by any Windows machines that don't rely on Kerberos for authentication. If you're using Kerberos, such as in a domain environment, then NTLM version two is simply not used.

Hashing Attacks

Pass the Hash

Pass the Hash is a hacking technique that allows the attacker to authenticate to a remote server or service by using the underlying hash of a user's password instead of requiring the associated plaintext password as you normally would have to do. Now, if an attacker is able to sniff that hash or steal it some other way, they don't need to brute force to clear text password. Instead, they can simply reuse the hash of that arbitrary user account as they go and authenticate against remote systems and impersonate that user.
In other words, from an attacker's perspective, hashes are functionally equivalent to the original password that they generated and it doesn't mean that they need to know your actual password to use your account as if they were you. The Pass the Hash attack is very difficult to defend against because there are many possible exploits in Windows, as well as the applications that run on top of it. And any of these can by used by an attacker to elevate their permissions and then be able to pull off credential harvesting or hash harvesting that they can then use in a further attack using Pass the Hash.

Mimikatz

There are many penetration tools out there such as **Mimikatz** that give you the ability to automate this process of harvesting the hashes and conducting the attack. To prevent the Pass the Hash attack, you should ensure that only trusted operating systems are allowed to connect to your servers, that your Window's domains have their trusts set up properly, and that workstations are all patched and updated, that your multifactor authentication is being used properly in the network, and that user accounts have been set up to use the concept of least privilege.

Birthday Attack

The Birthday Attack occurs when an attacker is able to send two different messages through a hash algorithm and it results in the same identical hash digest causing a collision. This attack gets its name from something called the Birthday Paradox, which says that if you have a random group of people, the chances are that you are going to have two people in that group with the same birthday. When I teach this course in person to a group of 30 people, most of the time, two people in the class have the same birthday. At least the same month and day, if not also the year. In fact, even though there are 365 days in a year, you only need 57 people in a room to get a 99% chance of having two identical birthdays. With 23 people in a room, your

odds are 50/50. That's why in my classes of 30 students, more often than not, we have identical birthdays. In the world of hashes, two identical hash digest would result in a collision. Now, if a hacker can find two identical messages with the same hash, they can use this as an attack against your system.

Increase Hash Security

Key Stretching

Key stretching is a technique that's used to mitigate a weaker key by increasing its effectiveness and thereby increasing the time needed to crack it. When you stretch a weaker key, the weaker key is run through an algorithm to create a longer, more secure key than is normally used. And it has to be at least 128-bits long. Many systems are going to utilize key stretching to increase the security they provide. Systems like Wi-Fi Protected Access, Wi-Fi Protected Access version 2, Pretty Good Privacy, BeCrypt, and others all use key stretching.

Salting

Salting is a technique of adding random data into a one-way cryptographic hash to help protect against password cracking techniques like dictionary attacks, brute-force attacks, and rainbow tables.

Nonce

Using a nonce is another method to help secure a weaker password, where a number used once, known as a nonce, is added to the password-based authentication to help prevent an attacker from reusing your password if they're able to steal it somehow.