Episode 1 - Smokin' the Hash

1. **Defintion**

   From: http://www.irongeek.com/i.php?page=videos/cryptographic-hash-md5

A hash takes an arbitrary block of data and returns a **fixed-size bit string**, the *cryptographic hash value*, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value. The data to be encoded are often called the *message*, and the hash value is sometimes called the*message digest* or simply *digest*.

2. **History of the Hash**

   From: *http://www.cosic.esat.kuleuven.be/publications/article-1532.pdf*

"In their 1976 seminal paper on public- key cryptography [31], (Dr. Whitfield) Diffie and (Martin) Hellman identified the need for a one-way hash function as a building block of a digital signature scheme. The first definitions, analysis and constructions for cryptographic hash functions can be found in the work of Rabin [74], Yuval [99], and Merkle [60] of the late 1970s. Rabin proposed a design with a 64-bit result based on the block cipher DES [37], Yu- val showed how to find collisions for an n-bit hash function in time 2n/2 with the birthday paradox, and Merkle's work introduced the requirements of collision resistance, second preimage resistance, and preimage resistance."

3. **Some uses for hashes**

   From: *http://www.irongeek.com/i.php?page=videos/cryptographic-hash-md5*

   a. detecting data changes
   b. storing or generating passwords
   c. making unique keys in databases
      virus scanner uses hashes of known viruses
   d. ensuring message integrity
   e. non-repudiation
      1. digital signatures

4. **What makes a good hash?**

   From: *http://en.wikipedia.org/wiki/Cryptographic_hash_function*

   The ideal cryptographic hash function has four main properties:

   a. it is easy to compute the hash value for any given message
   b. it is infeasible to generate a message that has a given hash
   c. it is infeasible to modify a message without changing the hash
   d. it is infeasible to find two different messages with the same hash.

5. **Hash families**

   a. MD* (MD2, MD4, MD5) - Merkle–Damgård hash function
      1. MD5 - 128 bits
         a. has been found to suffer from hash collisions (birthday attacks)

   b. SHA* (SHA0, SHA1, SHA2, SHA3) - "Secure Hash Algorithm"

From: http://en.wikipedia.org/wiki/SHA-1

2. Created by NIST and the NSA in 1993 (SHA0)
3. SHA1 = 160 bits (20 bytes), 40 digit long value
4. SHA2
    a. Multiple bit strengths = (224, 256, 384, 512 bit lengths)
5. SHA3
    b. **Keccek** was awarded the SHA3 as part of NIST's hash creation competition

c. Other hashes
    1. NT/NTLM - stored in Windows SAM database
    2. Radius - Enterprise auth for network infrastructure, etc
    3. WPA/WPA2 - wireless keys for auth to a wireless network
    4. RIPEMD-160
    5. HAVAL-128

6. Strengthening hashes

a. Salting hashes

    From: https://crackstation.net/hashing-security.htm

    From: http://scientopia.org/blogs/goodmath/2013/03/02/passwords-hashing-and-salt/

    1. Stored alongside the password hash
    2. Random salts for each, makes hashes more resistant
        a. makes rainbow tables, lookup tables, and reverse lookup tables more difficult to create or reference

b. Stretching hashes

    From: https://crackstation.net/hashing-security.htm

    1. Processor intensive hash function
    2. Make it fast to generate, but slow to reverse
    3. Downside is that web applications requiring auth will take longer to run.
    4. Examples:
        1. Scrypt: http://www.tarsnap.com/scrypt.html
        2. Bcrypt: https://www.usenix.org/legacy/events/usenix99/provos.html
        3. PBKDF2: http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

c. HMAC

    From: http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

    From: http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

    1. Adds a secret key to the hash so that only someone who knows the key can use the hash to validate a password
    2. Key must be stored in a separate secure location (e.g. not in the same DB as the hash and salt)

d. Encrypting the hash
    1. Using a decent scheme, like AES will further secure the hash.

7. How to crack a hash

From: http://www.codinghorror.com/blog/2012/04/speed-hashing.html

a. Dictionary attack
    1. Trying hundreds to millions of possible words to find a hash match

2. low chance of success, unless using targeted wordlists
3. Can be used in finding collisions.

b. Brute force
1. Trying every possible combination
2. Takes a LONG time
3. GPU/hardware acceleration definitely needed
4. Can also actively attempt password cracking on a server's service (e.g. SSH)
    a. Can attempt logins one at a time, which is nearly impossible

c. Rainbow tables
From: https://www.freerainbowtables.com/tables/
1. Creates a complete list of values to test and query against.
2. very large tables
    a. MD5 = 24GB
    b. SHA1= 24GB
    c. Faster than Brute Force, but salting makes these less useful, and must be regenerated if the salts are random.

8. Programs used to crack a hash

a. oclHashcat
1. GPU enhanced
2. CPU enabled
3. Runs on both Windows and LInux
4. Can be used to specify certain password filters
    a. upper, lower, number, special
    b. other complexity reqs, like length, no number is first character, etc

b. Cain and Abel
From: http://www.oxid.it/cain.html
1. cracks password hashes, like NT/NTLM
2. Windows RDP passwords
3. Cisco IOS
4. VNC Passwords
5. RADIUS Hashes

c. John The Ripper
From: http://www.openwall.com/john/
1. Linux Shadow passwords
2. NT/NTLM Password hashes

d. Aircrack-ng
From: http://www.aircrack-ng.org/
1. Cracks WPA/WPA2 using a wordlist

Additional Information:
http://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/