Hash Length Extension Attack

First Some Review

- Hashing is a one-way function
- A file or data of arbitrary size can be hashed and the output will always be the same length
- Two identical pieces of data will always produce identical hashes
- Hash algorithms digest “blocks” of data at a time
- If the data isn’t divisible by the block length it is padded

A look at MD5

- Fixed output length of 128 bits (16 bytes)
  - “The quick brown fox jumps over the lazy dog”
  - 9e107d9d372bb6826bd81d3542a419d6
  - `9e 10 7d 9d 37 2b b6 82 6b d8 1d 35 42 a4 19 d6`
  - `1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16`
- The input is broken into chunks of 512 bits (64 bytes)
- The message is padded so the length is divisible by 512

Padding

- The padding works as follows:
  - A single bit (1) is appended to the end of the message
  - Enough 0's are appended to bring the length to 64 bits fewer than a multiple of 512
  - The last 64 bits (8 bytes) are filled with a 64 bit length of the original message, modulo 2^64

Under the Hood

- Hashing algorithm has an internal state machine
- Operates on a 128 bit (16 byte) state divided into four 32 bit (4 byte) words
  - Called A, B, C, and D
  - Initialized to certain fixed constants
    - var int a = 0x67452301
    - var int b = 0xefcdab89
    - var int c = 0x98badcfe
    - var int d = 0x10325476
- Algorithm uses each 512-bit message block to modify the state of these variables
- Final output = a + b + c + d

Example

- Registers start at:
  - 67452301efcdab8998badcfe10325476
  - 67 45 23 01 ef cd ab 89 98 ba dc fe 10 32 54 76
  - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  - | A | B | C | D |

- “The quick brown fox jumps over the lazy dog”
  - 9e107d9d372bb6826bd81d3542a419d6
  - 9e 10 7d 9d 37 2b b6 82 6b d8 1d 35 42 a4 19 d6
  - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  - | A | B | C | D |

Similar Algorithms

- MD5 is based on Merkle-Damgard construction
  - A method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions
- SHA1 and SHA2 are also based on Merkle-Damgard
- This includes SHA256 and SHA512

Implications

- So this means that a hash is the end state of the algorithm
- The hash represents the registers after processing all data and padding
- What if someone added 512 bits (1 more block) to the end of the data?
- Could you compute the new hash if you had the old one?

Hashing in Security

- De-duplication
  - Does an uploaded file exist already
- Signature based AV
  - Is an uploaded file on a blacklist
- Tamper detection
  - Has a file been altered in transit
- Audit trail
  - Transaction ID
- Authorization
  - Hashes used as a key

Length Extension Overview

- A vulnerable hashing function works by taking the input message, and using it to transform an internal state
- After all of the input has been processed, the hash digest is generated by outputting the internal state of the function
- Therefore it is possible to reconstruct the internal state from the hash digest
- This can be used to process new data
- In this way one may extend the message and compute the hash that is a valid signature for the new message

Who’s Vulnerable

- MD4
- MD5
- RIPEMD-160
- SHA0
- SHA1
- SHA256
- SHA512
- WHIRPOOL

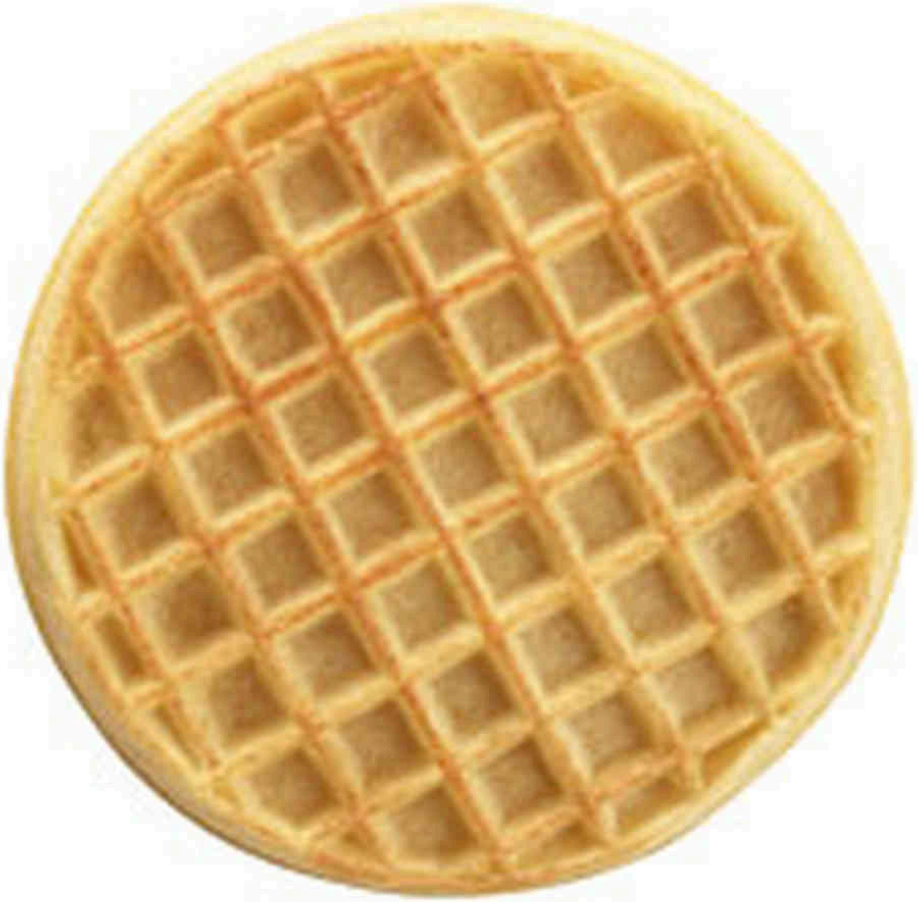
An Example - Waffles

- Imagine a service that delivers waffles to a location
- The request looks like this:
  - Original Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo
  - Original Signature: 6d5f807e23db210bc254a28be2d6759a0f5f5d99
  - Signature: SHA1(secret+data)
- The server verifies the signature before processing the request

Bring me a liege

- As an attacker you want to modify the request to order a liege waffle instead of an Eggo waffle
- Desired New Data: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo&waffle=liege
- Notice waffle is in there twice
- This is because on the backend the parameters are put into a hashmap (key:value) so the first waffle value is overwritten by the second

Because why have this



When you could have this

Upgrading

- Normally the attacker would need to know the secret to re-compute the signature
- However..
- It is possible to feed the hash (signature) into the state of your hashing function and continue where it left off
- Lets pretend we know the length of the secret (14 bytes)
  - This could be obtained through brute force, social engineering, information disclosure, etc.

Upgrading

- Recreate the padding (which is an open standard)
- Thus creating:
  - New Data:

```
count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo
\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x02(&waffle=liege


( = \x28
```

Upgrade

- The attacker knows that the state behind the hashed secret/message pair is identical to the new message up to the final &
- The attacker knows the hash digest at this point, which means they know the internal state of the hashing function
- They initialize their hashing algorithm to this point and hash the additional information
- New signature: 0e41270260895979317fff3898ab85668953aaa2

Mitigations

- You could put the secret at the end, so it is appended to whatever the message is
- Really you should use an HMAC instead of a MAC

Tools

- HashPump - https://github.com/bwall/HashPump/blob/master/README.md
- hash_extender - https://github.com/iagox86/hash_extender
- shaext.py - http://www.vnsecurity.net/2010/03/codegate_challenge15_sha1_padding_attack/

Resources

- https://en.wikipedia.org/wiki/Length_extension_attack
- https://github.com/bwall/HashPump/blob/master/README.md
- https://blog.whitehatsec.com/hash-length-extension-attacks/
- https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks
- http://www.vnsecurity.net/2010/03/codegate_challenge15_sha1_padding_attack/
- https://crypto.stackexchange.com/questions/3978/understanding-a-length-extension-attack
- http://blog.ioactive.com/2012/08/stripe-ctf-20-write-up.html#level7