Rootcamp

Length extension attack

Summary

MAC: WTF?!

Set up our testing chamber

Let’s go deeper: padding

Profit !

Bonus

Conclusion

MAC: Nope!

MAC: WTF ?!

Message Authentication Code

- Authenticate
- Check integrity

$$MAC = H(Key + Msg)$$

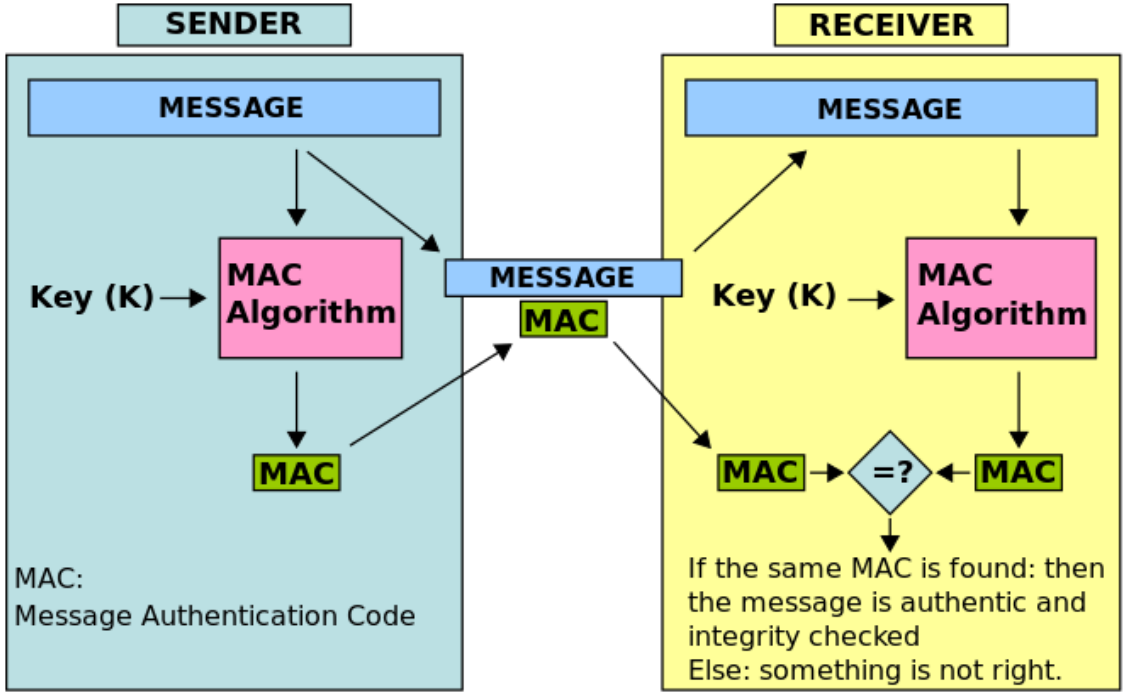
MAC: Implementation

In Python:

```
import hashlib

def mac(key, message) :
    return hashlib.md5(key + message).hexdigest()
```

MAC: And IRL ?


SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MESSAGE
MAC
MAC:
Message Authentication Code
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Testing chamber

Some piece of code ! :D

check.c → gcc -lbsd check.c → ./check

Usage: ./check <mac> # also reads user input

Let’s see known_data

Padding: How? (step 1)

Merkle–Damgård: MD5, SHA1, SHA2…

“Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512.”, RFC 1321

Padding: How? (step 2)

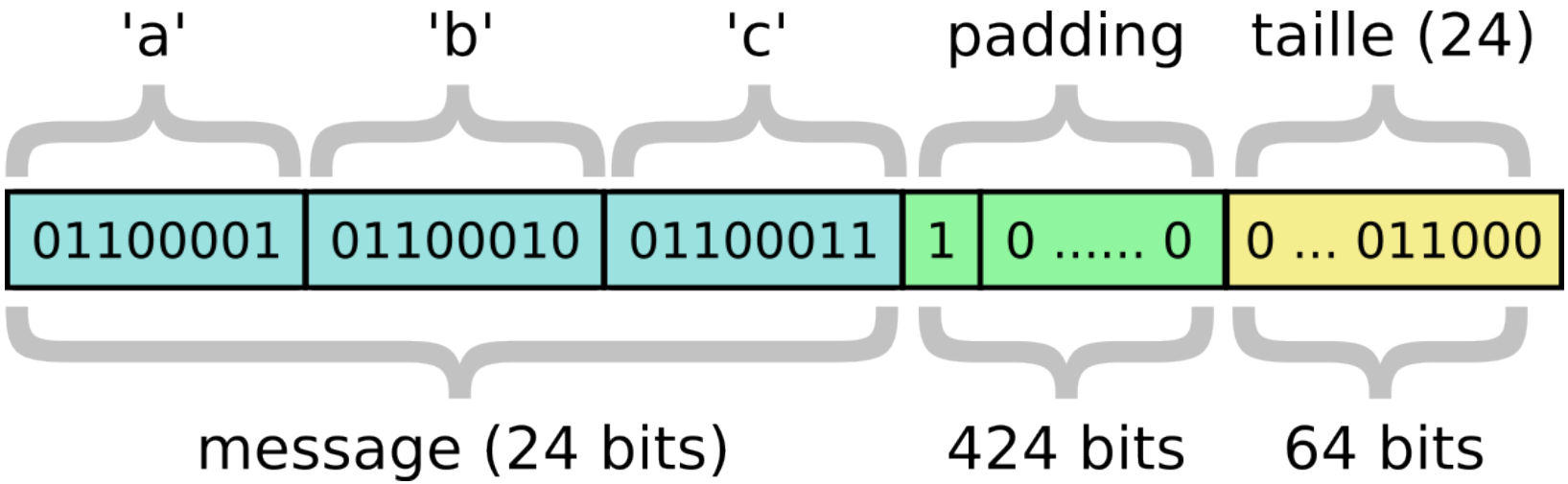
“A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step.”, RFC 1321

Finaly:

```
bloc_size = 64 bytes  (MD5)
padding_length = bloc_size - len(data) - 1 (first byte) - 8 (length on 64 bits)
hashed_msg = data + ‘\x80’ + ‘\x00’ * padding_length + len(data)
```

Padding

Summed up in a picture:



Hash computation

Computing a MD5 hash : MD5Init(), MD5Update(), MD5Final()

```
#define MD5_BLOCK_LENGTH		64

typedef struct MD5Context {
	u_int32_t state[4];			/* state */
	u_int64_t count;			/* number of bits, mod 2^64 */
	u_int8_t buffer[MD5_BLOCK_LENGTH];	/* input buffer */
} MD5_CTX;
```

A few word about the MD5Update() function

Profit, wait for it...

Step 1, compute a valid extended message.

- What do we want ?
- Add padding to our message
- Add extension
- You are ready!

rootcamp\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\xa8\x00\x00\x00\x00\x00\x00\x00flag

Profit, wait for it... (again)

Step 2, comute a valid hash for our new message.

- Prepare the context
- Go to the next block
- Change our actual state
- Add the extension
- Hash it! → 2773aa2417840404952d4e6afbd9b64e

Profit: Gotcha ! :-)

Let’s put all together!

```
$> echo -ne "$(./compute_msg.py rootcamp flag 13)" | ./check $(./mac flag)
User message = [rootcamp]
User mac = [2773aa2417840404952d4e6afbd9b64e]
[Hint] Expected mac = [2773aa2417840404952d4e6afbd9b64e]
Good!
$>
```

Bonus 1: Key size?

What to do if key size is unknown ??

Well… Bruteforce !

```
$> for i in {0..15} ; do echo -ne "$(./compute_msg.py rootcamp flag $i)" | ./check $(./mac flag) | grep -i good && echo keylen = $i ;done
```

Bonus 2: Security?

For security, use HMAC.

```
HMAC = H(key + H(key + message))
```

http://tools.ietf.org/rfc/rfc2104.txt

Bonus 3: HashPump

Super easy =)

HashPump [-h help] [-t test] [-s signature] [-d data] [-a additional] [-k keylength]

HashPump generates strings to exploit signatures vulnerable to the Hash Length Extension Attack.

-h --help Display this message.

-t --test Run tests to verify each algorithm is operating properly.

-s --signature The signature from known message.

-d --data The data from the known message.

-a --additional The information you would like to add to the known message.

-k --keylength The length in bytes of the key being used to sign the original message with.

Version 1.0.2 with MD5, SHA1, SHA256 and SHA512 support.

Bonus 4: References

http://en.wikipedia.org/wiki/Message_authentication_code

http://www.ietf.org/rfc/rfc1321.txt

http://www.opensource.apple.com/source/freeradius/freeradius-11/freeradius/src/lib/md5.c?txt

https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks

https://blog.whitehatsec.com/hash-length-extension-attacks/

http://en.wikipedia.org/wiki/HMAC

Conclusion

Stop writing your own crypto

Understand what you are doing

Understand the risks

Stay tuned for security updates!

Questions?

Contact: quentin.roland-gosselin@epitech.eu

Twitter: @Ark_444

GitHub: Ark444

IRC: Server: irc.rezosup.org

Channel: #cq-toulouse

User: Ark