Yet Another Padding Oracle in OpenSSL CBC Ciphersuites

presented by Raphael Finkel

based on `https://blog.cloudflare.com/yet-another-padding-oracle-in-openssl-cbc-ciphersuites/`

Keeping Current, September 5, 2018

The Cryptographic Doom Principle

- reference: `https://moxie.org/blog/the-cryptographic-doom-principle/` by Moxie Marlinspike
- If you have to perform any cryptographic operation before verifying the MAC (message authentication code) on a message you've received, it will *somehow* inevitably lead to doom.
- MAC is a cryptographic digest based on a secret key shared by Alice and Bob.
- Proper use of MAC: Encrypt Then Authenticate (Encrypt-then-MAC, EtA; used in IPsec)
  - Alice sends to Bob: E(P) || MAC(E(P))
    - Detail: E(P) also includes such information as the initialization vector and the encryption algorithm; both are then covered by MAC().
  - Bob first verifies MAC(E(P)), satisfying the principle. If that test passes, Bob decrypts P.
  - Good: Verifies integrity of E(P), therefore it also verifies integrity of P.
  - Good: MAC(E(P)) provides no information about P.

Authenticate and encrypt (Encrypt-and-MAC, E&A; used in SSH)

- Alice sends to Bob: E(P) || MAC(P)
- Bob must first decrypt E(P) to get P, then confirm MAC(P), violating the principle.
- Good: verifies integrity of P.
- Not good:
  - May theoretically reveal information about P in MAC(P).
  - No integrity check on E(P).
- Bad: vulnerable to chosen-ciphertext attacks on E.
  - Man-in-the-middle Morton can try various versions of E(P), noting whether Bob gets as far as trying to verify MAC(P).
  - SSH is also vulnerable to a plaintext-recovery attack.

Authenticate then encrypt (MAC-then-encrypt, AtE; used in SSL/TLS)

- Alice sends to Bob: E(P || MAC(P)).
- Bob must first decrypt E(P || MAC(P)) to get P and MAC(P), then confirm MAC(P), violating the principle.
- Good: verifies integrity of P.
- Not good: does not verify integrity of E(P || MAC(P)).
- Bad: vulnerable to Vaudenay's man-in-the-middle attack.

CBC (Cipher-block chaining) decryption

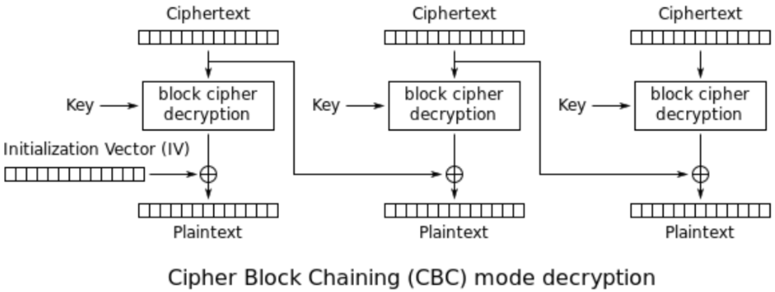

Figure 1: From Wikipedia

Vaudenay's attack

- A *padding-oracle* attack, based on CBC decryption. The message to be encrypted is padded to a multiple of the block size.
- The pad is N > 1 bytes, each containing the value N.
- Bob decrypts, then looks at the last byte, seeing N. Bob then verifies that the last N bytes all have that same value. If not, padding error. Otherwise Bob verifies MAC(P).
- M (Morton, the man in the middle) arbitrarily modifies R, the last byte in the penultimate ciphertext block, which has a deterministic effect on the last byte L of the last deciphered block, which Bob uses to determine padding.
- Either Bob gets a padding error or a MAC error. M can distinguish these outcomes by a timing analysis.
- By trying at most 255 possible modifications of R, M can eventually trigger a MAC error, because L is now 1, which is a valid padding value. M has decrypted the last byte of the plaintext: $R \oplus 1$.
- M can now force L to be 2 and repeat the technique on the previous byte, and so on, until M knows the entire plaintext.

Circumventing Vaudenay's attack

- Bob can circumvent Vaudenay's attack by running the padding check plus the MAC check in constant time, never distinguishing to M which check has failed.
  - Store the result of each test in a success variable via AND operations, doing each test regardless of the current success status; then return the status.
  - The padding value can be any value between 1 and blocksize-1; to ignore non-padding bytes that B need not check, it checks them anyway but ignores the (most likely bad) result with a mask.
  - This code was introduced in TLS.

May 2016: Yet Another Padding Oracle in OpenSSL CBC Ciphersuites

- CVE-2016-2107: reported to OpenSSL on 4/13/2016 by Juraj Somorovsky.
- You can test any host via `https://filippo.io/CVE-2016-2107/`. SSLLabs also tests.
- Our server, `https://www.cs.uky.edu` is not vulnerable. However, `https://elar.soas.ac.uk` is vulnerable.
- Alice uses AtE: she sends E(P || MAC(P) || padding || padding length) in CBC mode. Bob first decrypts, then verifies the padding, then verifies MAC(P), then accepts P.
- This algorithm violates the Cryptographic Doom Principle.
- Coding error: The constant-time code did not check for a padding length that is higher than legitimately possible.
  - B's mask can be fooled to check no bytes of the padding at all.
  - So M can discover if a message is made entirely of bytes with value n $\geq$ maxpad + 20.

Some details

- Assume block length is 32B, and MAC length is 20B. Then the padding length p is constrained: $1 \leq p \leq 32 - 1 - 20$, that is, *maxpad* is 11B.
- If the padding length is marked as 31, the mask for checking the MAC is entirely out of range, so the MAC is not checked at all, although the padding is checked.
- So M can discover if a message is made entirely of bytes with value $n \geq$ *maxpad*.
- If M can POST to B in an unauthenticated way, M can invoke B's padding and MAC check on its own plaintext.
- M aligns an unknown byte R (trying all possibilities) at the start of two blocks containing 31 that it POSTs to B. When the MAC check passes, the target byte is $31 \oplus R$.
- Sorry, but the details are pretty murky.

The fix

- Paraphrasing the blog: Under TLS-1.1, if the connection uses AES-CBC and B supports AES-NI, M can recover at least 16B of anything it can get the client to send repeatedly (by adding some JavaScript that does exactly that, such as HTTP cookies) just before M-controlled data.
- The public report, including the bug and the fix,released 5/3/2016.
  - The fix was developed by Kurt Roeckx of the OpenSSL development team.
  - B should add a constant-time check that $p \leq$ *maxpad*.

And now for something completely different

- Look for https://map.what3words.com/lunch.hops.liked
- what3words: The world is divided into 3m $\times$ 3m squares.
- Each square has a 3-word address.
- The address is in English or any of about 20 languages.
- Populated areas use shorter words.
- Useful for giving a precise location, even if there is no nearby street address
  - Meeting someone at a large gathering
  - Specifying where a drone should deliver your packet.
- There are apps for Android and iOS.

Original idea

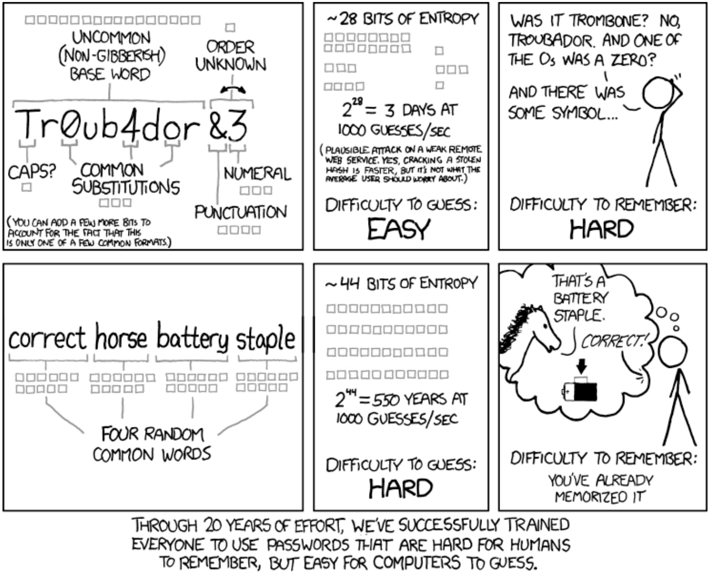
UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Figure 2: xkcd.com/936/