pad session values to 1025 bytes by default #1791

New issue

Merged **mergify** merged 1 commit into `main` from `pad_session` on Jun 2, 2021

Conversation 15 · Commits 1 · Checks 0 · Files changed 2 · +40 −2

jberger commented on Jun 2, 2021 Member

Pad the session value to at least 1025 bytes. This should prevent a small value from being used as part of a brute-force attack to decode the session secret.

👍 1

Grinnz commented on Jun 2, 2021 Contributor

If the padding byte doesn't matter I'd prefer to use something more indicative like null bytes maybe?

`pad session values to 1025 bytes by default` c99dee4

jberger force-pushed the `pad_session` branch from **04984ca** to **c99dee4** 4 years ago Compare

jberger commented on Jun 2, 2021 Member Author

> If the padding byte doesn't matter I'd prefer to use something more indicative like null bytes maybe?

I suppose that could work since it is base64 encoded. Originally I did think about null, but I chose a text character because I was worried about how it would interact with the HTTP protocol etc, but on reflection that doesn't matter once base64 encoded. Anyway, I don't care what the padding is. In theory it could be random as long as it isn't `}` (by this implementation).

Reviewers

kraih ✓

jhthorsen ✓

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

9 participants

Grinnz commented on Jun 2, 2021 • edited Contributor

The other option is to do this padding after the serialization and base64 so it happens regardless of the serializer. It would have to be some bytes that are not valid in base64 but are valid in cookie values (punctuation other than `",;\/+=` would work).

kraih requested a review from **a team** 4 years ago

kraih approved these changes on Jun 2, 2021 View reviewed changes

jhthorsen approved these changes on Jun 2, 2021 View reviewed changes

jhthorsen left a comment Member

I think this is a very clever solution 👍

mergify bot merged commit **`2ac681a`** into `main` on Jun 2, 2021

jhthorsen deleted the `pad_session` branch 4 years ago

stigtsp mentioned this pull request on Jun 4, 2021

perlPackages.Mojolicious: 9.17 -> 9.19 Merged
NixOS/nixpkgs#125618
9 tasks

latk commented on Oct 30, 2022

I am trying to understand the rationale of this change (prompted by [a Stack Overflow question](#)). The stated purpose of this change is to make brute-force key recovery attacks more difficult, by padding the cookie payload until it is at least 1025 bytes long. The length seems to have been chosen arbitrarily. The padded cookie value is later used in Controller.pm as input for a HMAC:

```
    my $sum = Digest::SHA::hmac_sha256_hex("$name=$ e"
    return $self->cookie($name, "$value--$sum", $op   s)
```

I think the change in this PR fails to achieve its stated purpose:

1. The difficulty of a brute-force attack depends on the entropy of the input, not directly on its length. Adding deterministic padding does not add entropy, and therefore does not make brute-force attacks more difficult. At most, increasing the input length produces a linear slowdown.

2. As far as I understand HMACs, the difficulty of a key recovery attack depends only on the strength of the secret key, up to the limit imposed by the hash function's internal size. The difficulty for key recovery does not depend on the size of the plaintext message, which is known to the attacker. For the same reason, using bytes from a CSPRNG for padding the message as a kind of salt would not help.

If my understanding happens to be correct, it could make sense to revert this change. **The primary effect of this PR seems to be increased bandwidth usage for users, not increased security.** If this padding is still desired, it could be added just before the HMAC calculation, without carrying it around in the cookie.

Towards the goal of defending against brute-force attacks, other mechanisms might be more helpful – such as extending the documentation for Mojolicious' secrets to explain how to generate secrets with sufficiently high entropy (e.g. `head -c 32 /dev/urandom | base64` or `openssl rand -base64 32`). NIST recommends key lengths of at least 112 bits for HMACs. I would also mention that secrets would ideally be provided by the deployment environment, and never written into the application's source code.

(Since the HMAC maintains its security properties regardless of the changes in this PR, this is not a vulnerability. I therefore commented here instead of going through the vulnerability disclosure process.)

👍 3

jberger commented on Nov 1, 2022 Member Author

It was a very specific brute force attack. I don't know if we're mentioning which openly but a common tool could be used if the encrypted message was short enough. Meanwhile you have to be able to strip it out easily and not cost too much performance since this happens a lot

👀 1

CarterSScott commented on Feb 1, 2023

I ran into an issue where I was trying to access the cookie that the session generates directly through the signed_cookie method. The padded Z's were still there and I had to write some regex to remove them. Should there be some sort of check in place for the signed_cookie method to remove these padded Z's?

stigtsp commented on Mar 19, 2023

> It was a very specific brute force attack. [..]

Hi! Would you mind sharing some details about this attack?

jkramarz commented on Sep 26, 2024

Hi all! I'm quite surprised by existence of an elegant Perl web framework and, this project's maturity and good documentation.
It's my first time here, I'm a security researcher who encountered it during one of our missions, just like **@cervoise**, who inspired this merge request with his article.

As described by **@jberger** in his blog post, the default secret for the application is both predictable and constant for particular application. It definitely eases work while in development and facilitate test deployments on multiple server instances behind a load balancer.

In the same time, "Your secret passphrase needs to be changed" in application log is the only sign that somebody overlooked configuring a proper secret in the production environment. As described by **@latk** in previous comment, in the current implementation the signature is as secure as the chosen secret.

Unfortunately, it seems that changes introduced in c99dee4 were specifically targeted on breaking input validation of *hashcat* cracking module for bare HMAC signature. It is now more difficult to validate that application was configured properly, unless you decide that the cookie is actually a *bit malformed HMAC-SHA256 signed JWT* and crack it as such with a bit of code fiddling. As it effectively created a new format of signature, not more and not less secure, but one that just requires slightly different handling, I'm letting you know before submitting a merge request to *hashcat*.

❤️ 3

s1037989 commented on Sep 26, 2024 via email Contributor

Hi! Thanks for reaching out to discuss this! I'm just a community member
myself, but appreciate the topic. Are you saying that you're warning the
Mojolicious team that their security measure is about to have a cracking
tool released to the world? And, regardless, do you have a suggestion for
what could be done to improve the framework? Is the reported issue only the
simple default password, or is there another problem that should be solved
with some superior technique?

…

jkramarz commented on Sep 26, 2024 • edited

@s1037989, more like notifying that strapping together parts of existing modules with duct tape and bubble gum works just fine for cracking the signatures in efficient way.

Not sure when the token signatures switched from HMAC-SHA1 to HMAC-SHA256 (it predates this merge request), but since then it were already fine. Solution implemented here, while clever in what it achieves, in my opinion does not directly increase security of the tokens. It is equal to HMAC-SHA256 signed JWT, without usual JWT-specific attacks.

If the idea of environments (e.g. like in Ruby on Rails, `RAILS_ENV=production` , with separate configuration) is present here, so there is some determinant sufficient to decide that the app is no longer in development, and it would be possible to prevent running application without secret configured - I would call it a day.

Grinnz commented on Sep 26, 2024 Contributor

There is a production mode, which can be set manually or by running the application with the hypnotoad application server. It may be reasonable to refuse to start in production mode without a configured secret, but that deserves its own issue/discussion.

kraih commented on Sep 27, 2024 Member

Maybe also worth mentioning that the app generator shipping with Mojolicious does generate a config file with secret:

mojo/lib/Mojolicious/Command/Author/generate/app.pm
Line 202 in `ecb44cf`

```
202        - <%= sha1_sum $$ . steady_time . rand  %>
```

Should those be longer by default too?

jkramarz commented on Sep 27, 2024 • edited

@kraih, it is definitely less predictable, let's say good enough to not call it a "default value" and ditch dictionary attack attempts.

But back to the question: I don’t have formal education in cryptography, so let's try to refer to some more respected sources:

In RFC 2104 (HMAC: Keyed-Hashing for Message Authentication) section 3 says that:

> The key for HMAC can be of any length (keys longer than B bytes are
> first hashed using H). However, less than L bytes is strongly
> discouraged as it would decrease the security strength of the
> function. Keys longer than L bytes are acceptable but the extra
> length would not significantly increase the function strength. (A
> longer key may be advisable if the randomness of the key is
> considered weak.)

Output length L of HMAC-SHA256 is 256 bits, so it seems that 256-bit long secret is desirable.

The same stands in NIST Special Publication 800-107 (Recommendation for Applications Using Approved Hash Algorithms)

> For example, if the desired security
> strength of the HMAC application is 256 bits, the HMAC key K shall be generated with a
> security strength of at least 256 bits, and an approved hash function with the message
> digest length of at least 256/2 (128) bits shall be used.

But I'd rather say, that the biggest problem would be not in length, but in key generation algortihm itself.

SHA1 gives you 160 bits of output, but I believe that it gets something like ~75 bits of randomness on input (8 digits of Unix epoch seconds over the last few years + some fractional number of 15 digits).
Furthermore, according to perldoc, rand *is not cryptographically secure and you should not rely on it in security-sensitive situations.* In some rare cases, this generator may even have its seed tied to current time, that is your second source of randomness. Not sure if it's worth trying to analyze it deeper, let's just say that this key generation (or more like derivation?) method is not widely used, "recommended" or "approved" :-)

Instead of trying to proof correctness of that one, I'd rather pack some random octets from Crypt::Random, that is actually a cryptographically secure random number generator (NIST SP 800-133 asks for one for this use) and call it a day.

👍 3

stigtsp mentioned this pull request on Sep 28, 2024

Secure by default sessions #2200 Open

stigtsp commented on Sep 28, 2024 • edited

I've done some work on a proposal to fix this that seems to be in line with what **@jkramarz** (and **@latk**) is raising:

- **Secure by default sessions** #2200

jkramarz mentioned this pull request on Oct 11, 2024

Add support for Mojolicious session cookies hashcat/hashcat#4090 Open