New issue

Jump to bottom

CVE-2020-25658 - Bleichenbacher-style timing oracle in PKCS#1 v1.5 decryption code #165

Closed **tomato42** opened this issue on Oct 26, 2020 · 28 comments

tomato42 commented on Oct 26, 2020 • edited

Current PKCS#1 v1.5 decryption code:

python-rsa/rsa/pkcs1.py
Lines 246 to 267 in 6f59ff0

```
246     blocksize = common.byte_size(priv_key.n)
247     encrypted = transform.bytes2int(crypto)
248     decrypted = priv_key.blinded_decrypt(encrypted)
249     cleartext = transform.int2bytes(decrypted, blocksize)
250
251     # Detect leading zeroes in the crypto. These are not reflected in the
252     # encrypted value (as leading zeroes do not influence the value of an
253     # integer). This fixes CVE-2020-13757.
254     if len(crypto) > blocksize:
255         raise DecryptionError('Decryption failed')
256
257     # If we can't find the cleartext marker, decryption failed.
```

performs the checks on the decrypted value in turn, aborting as soon as first error is found, it also raises an exception in case of errors. This likely provides enough of a timing side channel to mount a Bleichenbacher style attack.

While it's unlikely that a completely side-channel free implementation is possible (see https://securitypitfalls.wordpress.com/2018/08/03/constant-time-compare-in-python/), it should be possible to minimise the side-channel by making at least the execution path the same irrespective of previous checks and by providing an API that returns a randomly generated secret in case of error (instead of leaking the timing side-channel by rising an exception) for uses that decrypted value directly as an input to a hash or use it as a symmetric key.

tomato42 commented on Nov 10, 2020 Author

The code is basically unchanged for at least 10 years:

python-rsa/rsa/pkcs1.py
Lines 120 to 135 in 3f8c551

```
120     blocksize = common.byte_size(priv_key['n'])
121     encrypted = transform.bytes2int(crypto)
122     decrypted = core.decrypt_int(encrypted, priv_key['d'], priv_key['n'])
123     cleartext = transform.int2bytes(decrypted, blocksize)
124
125     # If we can't find the cleartext marker, decryption failed.
126     if cleartext[0:2] != '\x00\x02':
127         raise ValueError('Decryption failed')
128
129     # Find the 00 separator between the padding and the message
130     try:
131         sep_idx = cleartext.index('\x00', 2)
```

so we can be reasonably sure that all released versions since 2.1 (the first version that included RSA decryption API) are vulnerable

ran-isenberg commented on Nov 11, 2020

Hey guys, this seems like a critical security issue. We've been working with python-jose which depends on this repo.
Our security scanner, Snyk, tagged this repo with a high level risk security issue.
Is somebody working on this?
Thanks in advance

👍 1

tomato42 commented on Nov 11, 2020 Author

1. it's not critical, it's medium severity, though we would need to calculate CVSS to be sure. That being said, I haven't seen any Bleichenbacher CVEs scored high, let alone critical
2. python-jose depends on python-rsa, but it will not use it if better libraries are available, you should use python-jose with pyca/cryptography, then python-rsa code will be unused and unexploitable
3. nobody is working on this; we've decided with Sybren to make it public specifically so that somebody else could start the work on this

ran-isenberg commented on Nov 11, 2020

Thanks for the quick reply.

ran-isenberg mentioned this issue on Nov 11, 2020

CVE-2020-25658 - Bleichenbacher-style timing oracle in PKCS#1 v1.5 decryption code mpdavis/python-jose#195

Open

avishayil commented on Nov 11, 2020

Hi **@tomato42**

> 1. it's not critical, it's medium severity, though we would need to calculate CVSS to be sure. That being said, I haven't seen any Bleichenbacher CVEs scored high, let alone critical
> 2. python-jose depends on python-rsa, but it will not use it if better libraries are available, you should use python-jose with pyca/cryptography, then python-rsa code will be unused and unexploitable
> 3. nobody is working on this; we've decided with Sybren to make it public specifically so that somebody else could start the work on this

Snyk apparently calculated the score here, see here: https://app.snyk.io/vuln/SNYK-PYTHON-RSA-1038401

tomato42 commented on Nov 11, 2020 Author

For one, as it's timing based, so it's harder to exploit than issues like https://nvd.nist.gov/vuln/detail/CVE-2017-13098 (https://snyk.io/vuln/SNYK-JAVA-ORGBOUNCYCASTLE-32031) or any of the other issues listed on https://robotattack.org/ page. It also allows exactly the same kind of attack as ROBOT, so impact to confidentiality or integrity should be similar as those issues.

So I don't think that rating is consistent with other similar issues.

sybrenstuvel commented on Nov 11, 2020 • edited Owner

Thanks for the report & the interest, people, it's much appreciated.

How about this approach, would that be a proper fix for this issue?

```
def decrypt(crypto: bytes, priv_key: key.PrivateKey) -> bytes:
    blocksize = common.byte_size(priv_key.n)
    encrypted = transform.bytes2int(crypto)
    decrypted = priv_key.blinded_decrypt(encrypted)
    cleartext = transform.int2bytes(decrypted, blocksize)

    # Detect leading zeroes in the crypto. These are not reflected in the
    # encrypted value (as leading zeroes do not influence the value of an
    # integer). This fixes CVE-2020-13757.
    crypto_len_bad = len(crypto) > blocksize

    # If we can't find the cleartext marker, decryption failed.
    cleartext_marker_bad = not compare_digest(cleartext[:2], b'\x00\x02')

    # Find the 00 separator between the padding and the message
    try:
        sep_idx = cleartext.index(b'\x00', 2)
    except ValueError:
        sep_idx = -1
    sep_idx_bad = sep_idx < 0

    anything_bad = crypto_len_bad | cleartext_marker_bad | sep_idx_bad
    if anything_bad:
        raise DecryptionError('Decryption failed')

    return cleartext[sep_idx + 1:]
```

The weakest spot here I think is the call to `cleartext.index(b'\x00', 2)`. I'm open to any suggestions as to how to get rid of it and replace it with something constant-time.

sybrenstuvel mentioned this issue on Nov 11, 2020

The decryption code is PKCS#1 v1.5 non-compliant - no padding length check #164

Closed

tomato42 commented on Nov 11, 2020 Author

> Thanks for the report & the interest, people, it's much appreciated.
>
> How about this approach, would that be a proper fix for this issue?
>
> ```
> def decrypt(crypto: bytes, priv_key: key.PrivateKey) -> bytes:
>     blocksize = common.byte_size(priv_key.n)
>     encrypted = transform.bytes2int(crypto)
>     decrypted = priv_key.blinded_decrypt(encrypted)
>     cleartext = transform.int2bytes(decrypted, blocksize)
>
>     # Detect leading zeroes in the crypto. These are not reflected in the
>     # encrypted value (as leading zeroes do not influence the value of an
>     # integer). This fixes CVE-2020-13757.
>     crypto_len_bad = len(crypto) > blocksize
>
>     # If we can't find the cleartext marker, decryption failed.
>     cleartext_marker_bad = not compare_digest(cleartext[:2], b'\x00\x02')
>
>     # Find the 00 separator between the padding and the message
>     try:
>         sep_idx = cleartext.index(b'\x00', 2)
>     except ValueError:
>         sep_idx = -1
>     sep_idx_bad = sep_idx < 0
>
>     anything_bad = crypto_len_bad | cleartext_marker_bad | sep_idx_bad
>     if anything_bad:
>         # raise DecryptionError('Decryption failed')
>         return cleartext[sep_idx + 1:]
>
>     return cleartext[sep_idx + 1:]
> ```
>
> The weakest spot here I think is the call to `cleartext.index(b'\x00', 2)`. I'm open to any suggestions as to how to get rid of it and replace it with something constant-time.

the fact that `cleartext.index()` raises an exception will likely leak enough information to still mount an attack, using `cleartext.find()` would be better, but realistically, we need to traverse the whole `cleartext` reading each byte, as I'm quite sure `cleartext.find()` will exit the faster the earlier it finds the null byte

second, in case of failure we definitely can't return part of `cleartext`

sybrenstuvel commented on Nov 11, 2020 • edited Owner

How about using `before, separator, after = cleartext[:2].partition(b'\x00')` ? We can then inspect `len(before)` to test for correctness, and if correct, return `after` instead of `cleartext[sep_idx+1:]` . This should cause a copy of each byte, either into `before` or `after` , and the following correctness test is just a single integer comparison.

> second, in case of failure we definitely can't return part of cleartext

Oh geesh, that shouldn't have been in here. The exception was just disabled to make it possible to do a timing test, and certainly won't be in the final code ;-)

tomato42 commented on Nov 11, 2020 Author

> How about using `before, separator, after = cleartext[:2].partition(b'\x00')` ? We can then inspect `len(before)` to test for correctness, and if correct, return `after` instead of `cleartext[sep_idx+1:]` . This should cause a copy of each byte, either into `before` or `after` , and the following correctness test is just a single integer comparison.

yes, that does make for nice and clean code, but again, I don't expect it to have any better timing characteristic than `cleartext.find()`

for example of code that attempts side-channel free behaviour see here: https://github.com/openssl/openssl/blob/d8701e25239dc3d0c9d871e53873f592420f71d0/crypto/rsa/rsa_pk1.c#L170-L278

sybrenstuvel commented on Nov 11, 2020 Owner

Yeah, I made something similar, but looping over each byte and doing some operations there is much slower than what we have now.

tomato42 commented on Nov 11, 2020 Author

from my previous work on this, I've noticed that subscripting strings and arrays is very expensive (like `cleartext[i]`), it's much faster to do `for i, b in enumerate(cleartext)` ; but yes, it won't be near as fast as native code

sybrenstuvel closed this as completed in `dae8ce0` on Nov 15, 2020

---

tomato42 commented on Nov 15, 2020 Author

I'm quite sure that the code in `dae8ce0` does not fix it.

piyushpyoaknorth commented on Nov 16, 2020

Hi,
We have the same issue as highlighted by Snyk. What should be the right solution for it?

tomato42 commented on Nov 16, 2020 Author

> Hi,
> We have the same issue as highlighted by Snyk. What should be the right solution for it?

1. audit your code for use of rsa decryption—if you don't use it you're not vulnerable; it's a false positive
2. propose fixes to this library to fix it—though that's unlikely to be successful in the end, as I wrote above
3. modify code you depend on so that it uses libraries that do provide side-channel free behaviour for RSA decryption

piyushpyoaknorth commented on Nov 16, 2020

Thanks **@tomato42**

1. I checked our code, and we don't use the `rsa` library directly. But it is used indirectly from `tls` and `jwt` libraries. We use Django
2. I am not too experienced in python, to fix/monkey patch the library. But will give it a shot :)
3. I think this is what I can try and do.

How did you end up resolving this issue for you?

tomato42 commented on Nov 16, 2020 Author

> Thanks **@tomato42**
>
> 1. I checked our code, and we don't use the `rsa` library directly. But it is used indirectly from `tls` and `jwt` libraries. We use Django

`tls` ? You mean this https://pypi.org/project/tls/#description ??

I don't see `jwt` using python-rsa, unless we're not talking about https://github.com/GehirnInc/python-jwt

> 2. I am not too experienced in python, to fix/monkey patch the library. But will give it a shot :)

I would advise against monkey-patching this code, writing side-channel secure cryptographic code is hard, you really should have any changes around it audited by a cryptographer

> How did you end up resolving this issue for you?

I'm not using python-rsa, I just noticed that it's vulnerable and widely used

piyushpyoaknorth commented on Nov 16, 2020 • edited

> Thanks **@tomato42**
>
> 1. I checked our code, and we don't use the `rsa` library directly. But it is used indirectly from `tls` and `jwt` libraries. We use Django
>
> `tls` ? You mean this https://pypi.org/project/tls/#description ??

Yes, by https in Django for the servers

> I don't see `jwt` using python-rsa, unless we're not talking about https://github.com/GehirnInc/python-jwt

Let me check once again.

> 1. I am not too experienced in python, to fix/monkey patch the library. But will give it a shot :)
>
> I would advise against monkey-patching this code, writing side-channel secure cryptographic code is hard, you really should have any changes around it audited by a cryptographer

Okay. Will do that for sure.

> How did you end up resolving this issue for you?
>
> I'm not using python-rsa, I just noticed that it's vulnerable and widely used

Ah. I see.

---

attila123 commented on Nov 27, 2020

@sybrenstuvel Thanks for the fix! Could you please release a new version?
For me it was also picked up by a vulnerability scan (Anchore). Many people would benefit/save time by not having to deal with this vulnerability (even to research what it is). So it would be quite appreciated, I think. :) Thanks in advance :)

---

tomato42 commented on Nov 27, 2020 Author

@attila123 #165 (comment)

---

attila123 commented on Nov 27, 2020

@tomato42 thanks. Then it is quite misleading to have this issue closed.

 **attila123** mentioned this issue on Nov 30, 2020

Used python rsa library brings PKCS#1 decryption vulnerability, is google-auth affected? googleapis/google-auth-library-python#646

Closed

NicoleG25 commented on Dec 1, 2020

Hi,
Could someone please clarify then if this issue was addressed ? :)

Thanks in advance !

 **tomato42** mentioned this issue on Dec 1, 2020

Delegating to pyca/cryptography operations we can't secure #169

Open

---

attila123 commented on Dec 10, 2020 • edited

@NicoleG25 This is definitely not addressed. Why: the supposed fix `dae8ce0` was committed on Nov 15, 2020, but the latest release of this library is 4.6 released at Jun 12, 2020 (see https://pypi.org/project/rsa/#history).

I am not a security expert, but as **@tomato42** 's #165 (comment) above: "I'm quite sure that the code in `dae8ce0` does not fix it."

Good news is that at least google-auth does not use this problematic decrypt method (see above), so it can be considered false-positive in my use-case.

---

sybrenstuvel commented on Jan 10, 2021 Owner

Version 4.7 has just been released to pypi.

🚀 3

---

dalesit commented on Jan 15, 2021

Issue still being flagged in scanning for version 4.7

👍 1

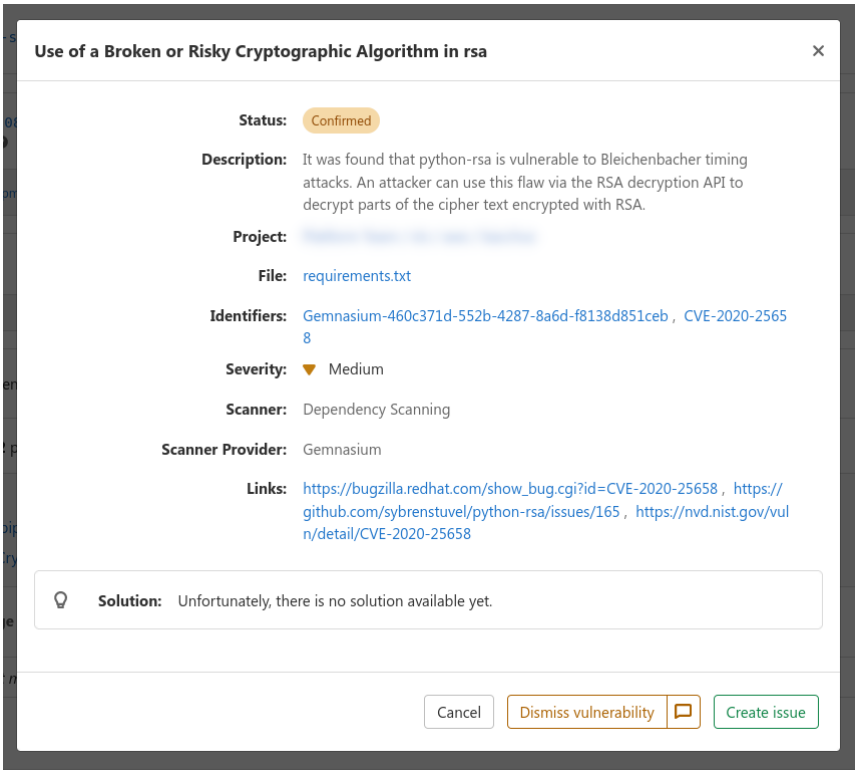
---

sybrenstuvel commented on Jan 15, 2021 Owner

What does that mean **@dalesit** ?

batisteo commented on Mar 4, 2021

I can't say if **@dalesit** mean the same thing, but here's what I'm getting from Gitlab Dependency Scanning (`gemnasium-python`)



vavkamil commented on Jul 28, 2021 • edited

Info about the fixed version was never added into the gemnasium vulnerability database https://gitlab.com/gitlab-org/security-products/gemnasium-db/-/blob/master/pypi/rsa/CVE-2020-25658.yml#L11

snarfed mentioned this issue on Nov 1, 2021

Upgrade to google-cloud-datastore 2.x? googleapis/python-ndb#733

Closed

ktdreyer mentioned this issue on Jan 12

require cryptography in packaging metadata (and remove rsa) googleapis/google-auth-library-python#941

Open

mtremer pushed a commit to ipfire/ipfire-2.x that referenced this issue on Feb 14

`python3-rsa: Update to version 4.8 and python-3.10` ··· `0f721b0`

ken-duck mentioned this issue on Aug 23

Incorrect vulnerability details: CVE-2020-25658 python-rsa OSSIndex/vulns#301

Closed

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

10 participants