Security review #5

Merged

scottcwang merged 4 commits into scottcwang:master from mike-arnica:security-review on Jun 24

mike-arnica commented on Jun 22 Contributor

Hi Scott!

My name is Mike Doyle and I head up security research for Arnica. I was looking at openssh-key-parser for inclusion in one of my products when I noticed your very helpful disclaimer about the library not having been security reviewed.

Security review of cryptography systems is a skill that I possess. Arnica is an Open Software Security Foundation member, and we believe in paying back to the folks who dedicate their time to building the open-source software that we depend on.

I've gone ahead and taken the time to review the code for openssh-key-parser and I've produced three commits for your consideration in this pull review.

The first changes the content of a raised exception to prevent a possible leak of key material. At present, the parser raises an EOFError when a pascal-style string's stated length doesn't agree with the number of bytes read from the keystream. While this condition would be rare under normal operating conditions, there are a few interesting ways that this could happen. One way is that an attacker who can't read the key but can somehow manipulate its contents either at rest or in transit could corrupt the value of the string length, making it a much larger number than the size of the file. This would raise your EOFError on line 239 of pascal_style_byte_stream.py, which uses the bytes read as the error message.

It's important that I mention that I didn't find anywhere in your codebase that an attacker could cause the corruption of this value. Such a condition would have to happen elsewhere in the system that uses your parser.

Anyway, the security implication of how this is implemented currently is that if such an attacker can somehow read the exception message, (for instance if the larger system logs unhandled exceptions to a place the attacker can read them,) the error message will contain the remains of the file which naturally includes the keying material in the file.

From here the attacker could use that keying material (which may or may not be passphrase protected) to produce an identical key by merely recreating the header data.

My first commit has a simple fix: it replaces the error message with the same text you use in the docstring. This prevents disclosure of the keying material.

I've checked the rest of the code for similar potential disclosures and did not find any.

My second commit adds comprehensive exception handling to PrivateKeyList.from_bytes. This method's docstring states that it will raise a ValueError under a few conditions but doesn't mention the EOFExceptions raised by other methods that from_bytes calls. I've added exception handlers which catch the EOFException and raise a corresponding ValueException to the calling code, and have updated the docstring accordingly. Also, despite there not being any other type of exceptions raised, I've added a blanket handler for any other condition which may occur. I think this is a good future-proving step.

Unhandled exceptions are typically more of an operational risk than a security risk, but they can have security implications. See CVE-2019-11177 for just one example.

I had considered just changing the docstring to declare that from_bytes might raise an EOFException, but that would create more work for the end user without providing a lot of additional value as best I can tell. The user either wants a parsed key or an error; there's really little they can do knowing that their key was somehow truncated or contains an internal size disagreement.

I've checked the rest of the code and see that you raise NotImplementedErrors in a few places. I'll let you decide if these should be caught, documented, or left as-is.

My third commit is a bit of a leave-behind. I used the Atheris fuzzer to perform some of this security review and want to contribute the scaffolding that integrates Atheris with your codebase. This can be developed and matured by myself or others in the future.

I hope this helps. Please feel free to reach out to me here on GitHub with any questions or concerns.

Mike Doyle
Head of Security Research, Arnica.io

mike-arnica added 3 commits 5 months ago

Changed an exception message to prevent possible disclosures of keyin…	···	26e0a47
Improved error handling to prevent unhandled exceptions in calling code.		d5b53b4
Added fuzzing scripts		2349be4

scottcwang commented on Jun 22 — Owner

Hello Mike,

Thank you for stopping by. I'm quite flattered by your interest, and very grateful indeed for your having taken the time to do a security review on my library.

I gladly accept your contributions and appreciate your corrections to my careless exception handling.

As you suggest, the places where I raise NotImplementedError do need attention -- I've used it to mean variously "todo", "abstract method", or "ValueError", and I will definitely need to be less haphazard.

Thanks for referring me to the Atheris fuzzer; it's new to me, so I'll accept it and put it on my todo list to understand better. In your proposed fuzzer requirements.txt, I see you pinned the openssh-key-parser version to 0.0.2, but the main branch is currently at 0.0.5. Do you see any issues with pinning it to 0.0.5 instead?

More generally, if there's anything you'd be willing to share about your plans for using this library, I'd love to hear it. I admit that this is a side project, sidelined at the moment in favour of a different shiny new toy; but I'd always welcome any general feedback and feature requests for me to prioritise once I eventually turn my attention back here. For example, it's on my todo list to add a new module containing wrappers around signing and verifying ssh signatures, and I'd be interested to know whether this would be part of your use case.

Thanks again,

Scott

❤️ 1

scottcwang reviewed on Jun 22

View changes

`tests/fuzzer/requirements.txt` Outdated

```
...  ...  @@ -0,0 +1,5 @@
     1  + bcrypt@3.2.2
     2  + cffi@1.15.0
     3  + openssh-key-parser@0.0.2
```

scottcwang on Jun 22 — Owner

Did you have a reason to pin the library under test to 0.0.2 rather than 0.0.5?

Suggested change

```
- openssh-key-parser@0.0.2
+ openssh-key-parser@0.0.5
```

mike-arnica on Jun 23 — Contributor, Author

I had a reason during research but not that I think about it, since this file is now included in openssh-key-parser, the inclusion of any version of the package in this requirements file is probably not useful. I would just delete the line entirely.

mike-arnica commented on Jun 23 Contributor Author

> Thank you for stopping by. I'm quite flattered by your interest, and very grateful indeed for your having taken the time to do a security review on my library.

You're welcome. I should be grateful that you took the time to develop and share the parser.

> I gladly accept your contributions and appreciate your corrections to my careless exception handling.
>
> As you suggest, the places where I raise NotImplementedError do need attention -- I've used it to mean variously "todo", "abstract method", or "ValueError", and I will definitely need to be less haphazard.
>
> Thanks for referring me to the Atheris fuzzer; it's new to me, so I'll accept it and put it on my todo list to understand better.

Atheris took me a little time to learn but I think it's worth it. It is based on the same LibFuzzer that openssh uses.

> In your proposed fuzzer requirements.txt, I see you pinned the openssh-key-parser version to 0.0.2, but the main branch is currently at 0.0.5. Do you see any issues with pinning it to 0.0.5 instead?

No issues, in fact now that the file belongs to the project, you probably shouldn't pin it at all. I had originally pinned it to 0.0.2 because, I believe, that's what pypi was giving me by default, but now I see pypi is looking at 0.0.5.

> More generally, if there's anything you'd be willing to share about your plans for using this library, I'd love to hear it. I admit that this is a side project, sidelined at the moment in favour of a different shiny new toy; but I'd always welcome any general feedback and feature requests for me to prioritise once I eventually turn my attention back here. For example, it's on my todo list to add a new module containing wrappers around signing and verifying ssh signatures, and I'd be interested to know whether this would be part of your use case.

Sure thing. One of the things we do at Arnica is we search customer repositories for secrets, including OpenSSH private keys. I want the ability to provide additional context to my customers, for example let them know whether their keys are passphrase protected. Python's default cryptography package doesn't give access to this sort of metadata, (as I'm sure you are aware), but I can use your parser to infer this information.

Additionally, I'll be blogging about my approach to this security review because it is illustrative of software supply chain security issues.

Thanks,

Mike Doyle
Head of Security Research, Arnica.io

Remove openssh-key-parser ae4d131

scottcwang merged commit **274447f** into `scottcwang:master` on Jun 24

---

scottcwang commented on Jun 24 — Owner

Thanks Mike! Merged.

I'd definitely be very interested in reading your blog post -- please do pingback here once it's ready.

Scott

mike-arnica commented on Jun 28 — Contributor, Author

I'll send you a link to it once it's written. By the way, you should advance the version number and maybe edit the disclaimer now ;-)

scottcwang commented on Jun 30 — Owner

Hi Mike,

Definitely. I'm doing a bit of general TLC and will release 0.0.6, and I'll credit you in the readme.

1. It turns out that the remainder of the bytes in the `EOFError` raised by `PascalStyleByteStream`'s `read_fixed_bytes` method are being used here; the parser of arbitrary-length lists of strings needs to know whether the bytestream is exhausted to conclude it has reached the end of the list. I'm making the `EOFError` include a `True` / `False` indication to show whether the bytestream is empty. I don't think an attacker would find this tiny bit of information to be particularly useful -- let me know if you disagree.
2. Just FYI, since I didn't realise this about Python either ... this PR had the catch-all exception handler written as `except e:` here. This actually isn't correct syntax, since the `e` would be parsed as the type of the error to catch, as opposed to a variable name. What we needed was `except BaseException as e:`.
3. One side-effect of having as many `NotImplementedError`s as I did is that they are ignored in the coverage report. So as I replace the `NotImplementedError`s with `ValueError`s, I'm going to need to add some more unit tests :)

Scott

mike-arnica commented on Jun 30 — Contributor, Author

1. I agree that the true/false indicator is not particularly useful.
2. You're right, BaseException is the better way to go. Python exception handling can be confusing for me.
3. Sounds right.

Since you're bumping the version it might be good etiquette to produce a security advisory for v0.0.5 and prior. Let me know if you'd like me to write it up for you.

scottcwang commented on Jul 1 — Owner

That's a good idea. I took a first stab at writing it up. How's this? I'd appreciate your advice particularly on the scoring, which I'm not familiar with.

mike-arnica commented on Jul 5 — Contributor, Author

That's a pretty good advisory write up. You provide a decent amount of detail.

Scoring this issue is difficult. CVSS is made to score vulnerabilities in complete systems, not individual libraries. For example we have to assume that an attacker can stomp on a field length, even though there's no place in *your* code where they could do that. The easiest way to pull off the attack is to modify the field length of the cipher name. It comes right after the AUTH_MAGIC which is always the same string and always Base-64s the same. If you can execute this regex substitution against the key file, or all the files in the file system:

s/^b3BlbnNzaC1rZXktdjEAA/b3BlbnNzaC1rZXktdjEA8/

you can exploit it. It's one of those things where if it isn't impossible, it's trivial. Similarly, reading the keying material out of the exception log is either impossible or trivial. For this reason, attack complexity is low, even though finding the bug took a lot of effort ;-).

The integrity and availability impacts are none. Technically, if a hypothetical attacker got a key they can impact the integrity and availability of whatever systems that key can log in to, but CVSS just looks at one link in the chain, which here are the keys being parsed. But CVSS sort of recognizes the fact that the key can be used to get to other resources with the "Scope" metric, which here we should set to "Changed" rather than "Unchanged" to accurately reflect this aspect. Since you can get all the keying material, though, the confidentiality is high rather than low.

Everything else looks good.

I've made the appropriate scoring changes to the advisory. Let me know what you think.

mike-arnica commented on Jul 5 — Contributor, Author

On further review, the attacker would, in all likelihood, need low privileges to read the exception and write to files they can't read. I've updated the score.

scottcwang commented on Jul 5 — Owner

Awesome! Your very thorough explanation makes sense and I'm very glad to have your help.

I've gone ahead and published the advisory. Thanks once again!

mike-arnica commented on Jul 6 — Contributor, Author

Thanks Scott! Here's a link to the blog post: https://www.arnica.io/blog/hacking-upstream-finding-a-0-day-in-an-openssh-key-parser-library

scottcwang commented on Jul 6 — Owner

Awesome! I'll put a link on the readme.

This has been a productive and wonderful learning experience for me, so I mustn't neglect to tell you how much I've appreciated your leadership of, and support for, my little project and the wider open-source community.

I hope you enjoy using it at Arnica -- please let me know of any new issues you come across. And when in the future I release new functionality that interests you, feel free to come back and pick it apart again.

Reviewers

scottcwang

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

---

2 participants