How to use AI for improved vulnerability report writing

BY AYOUB · FEBRUARY 17, 2026 · LAST UPDATED ON FEBRUARY 19, 2026

Report writing is an integral part of bug bounty or any type of vulnerability assessment. In fact, sometimes, it can become the most important phase. Submitting a confusing report can often lead to misalignment and faulty interpretation of your reported vulnerability. On the contrary, a well-written submission that includes all the necessary details can help shorten the time to triage, lead to increased bounties, and avoid unnecessary communication follow-ups.

With the latest AI trends, we have internally noticed a spike in submissions that are fully written with the help of LLM Models, such as ChatGPT, Claude AI and Gemini. In this article, we want to advise security researchers and bug bounty hunters on how to effectively utilize AI to craft more compelling vulnerability reports, which can lead to higher bounties and faster triage times.

Let's dive in!

Why AI for vulnerability report writing?

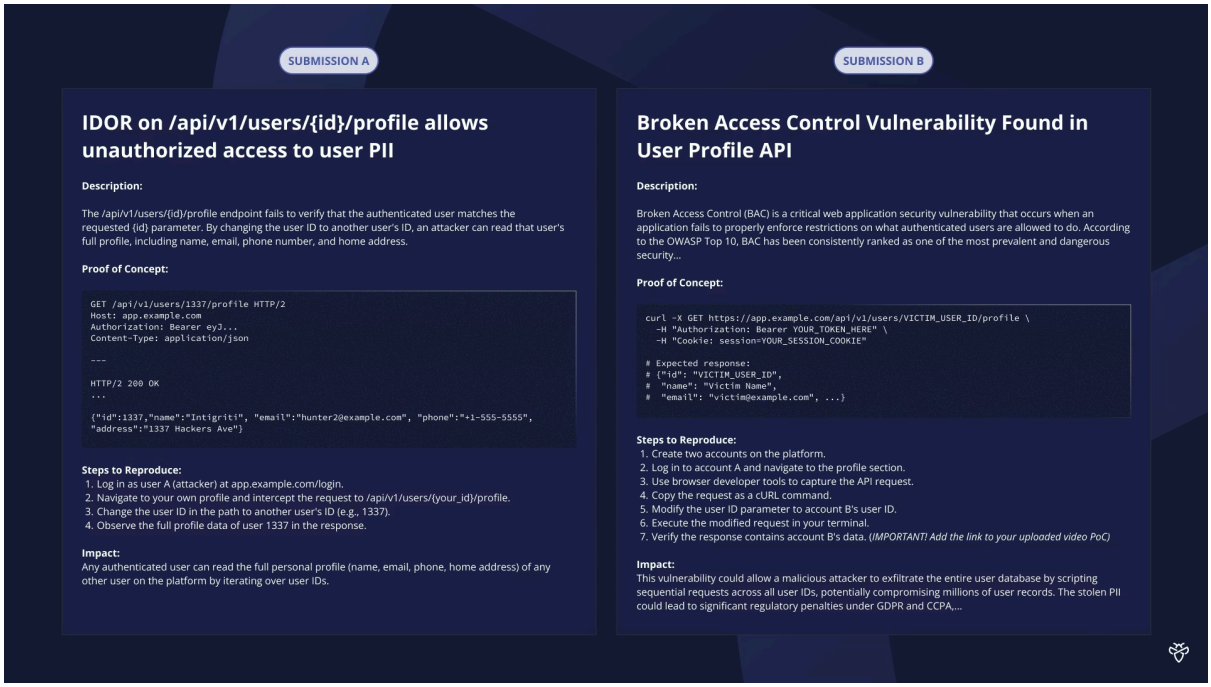
Vulnerability report writing is a tedious process that often involves clearly documenting all the behavior and steps that lead to the discovery of a security vulnerability. Luckily, with all the latest advancements in LLMs, security researchers no longer have to spend a tremendous amount of time performing this tedious task. Hence, most researchers will simply revert to letting an LLM chatbot write the entire vulnerability report for them.

However, as with many things, when used improperly, you're only shifting the problem and never truly solving it. By submitting reports that are confusing and fully LLM-generated, you're inadvertently causing longer triage times and potentially confusion among the triage team or the affected organization.

On the contrary, when considering the following 6 steps, you can shorten the time spent on writing vulnerability reports while also ensuring you're sending in compelling, well-written vulnerability submissions.

Compelling vs poor LLM-generated vulnerability reports

Before we dive into the 6 key elements, let's see if you can spot which submission performs better than the other:



Vulnerability report writing with AI

In the illustration above, we placed two submissions next to each other. Both reports describe the same broken access control (BAC) vulnerability and are generated by the exact same LLM Model (Claude AI). However, report A is simply easier to process compared to report B.

Here's a brief overview of why submission A far exceeds report B:

- Submission B includes an unnecessarily lengthy paragraph describing the vulnerability type that could have otherwise been shortened to a maximum of two sentences.
- The report includes a placeholder proof of concept that is possibly unverified.
- The steps to reproduce may be incorrect and violate the platform's Code of Conduct.
- Lastly, the impact section appears to describe attack scenarios that are speculative in nature.

Let's take a deeper dive into what to avoid whenever using AI to generate any of your reports.

1. Avoiding lengthy paragraphs of text

LLMs are quite useful in returning vast amounts of information, sometimes even too much for us, humans, to properly read and process. This includes triagers who are trained to go through the entire report. Therefore, including lengthy descriptions that could have simply been shortened or completely omitted, you'll drive up the time a triager will need to correctly process your submission. Additionally, depending on the LLM Model used, it may hallucinate and provide incorrect information, further causing confusion.

Understanding this, it's a best practice to list what is necessary and add any supporting evidence while avoiding long paragraphs of text that add no value. We also recommend reviewing the generated output to confirm its validity before including it in your submission.

2. Including a working proof of concept

As we've seen in the first practical example, it may sometimes occur to you that you include the entire LLM-generated report, including the proof of concept. However, if the LLM Model decides to alter your working proof of concept, you risk sending in a submission with an unvalidated payload.

To avoid this, human oversight must remain a key component. Like with tools, we recommend that you never assume an agentic AI or tool has validated the vulnerability you're about to report, instead always cross-check any step you add in your submission, including any payloads.

A working PoC isn't always a valid PoC

A common pitfall with AI-generated proofs-of-concept is that they may technically execute without errors (i.e., the payload fires or the code compiles), but they don't actually demonstrate the vulnerability in a production-like environment or in a product's intended configuration.

For example, an LLM might generate a PoC for a firewall product that spins up a local server but never actually enables the firewall rules. The exploit may have worked, but only because the security settings or rules were never enabled or configured as intended.

Similarly, when targeting open-source software, an LLM may help find a vulnerable code snippet and provide a proof of concept. However, the vulnerable code path may never be reachable through user-controlled input in a production-ready replica of the product.

Therefore, you must always provide a valid proof of concept that demonstrates exploitability against the target in its intended configuration. A PoC that solely works against a misconfigured replica may be deemed not applicable.

> **Rule of thumb**
>
> Always go over your documented steps once more before submitting the vulnerability. This ensures your report gets triaged faster, which leads to faster payouts.

3. Incorrectly AI-generated reproduction steps

Since applications differ vastly from one another, triagers will rely on your documented reproduction steps to navigate around the target and reproduce your identified security vulnerability. Entirely relying on an LLM to generate the full report for you without providing it with any additional context will result in an inaccurate steps to reproduce section.

This can play out in different ways. If the identified security vulnerability is indeed present, but the reproduction steps are incorrect, it may cause the triager to fail while reproducing the reported flaw. On the other hand, it will also lead to an increase in the time to triage.

To avoid such scenarios, human oversight remains mandatory. Always validate the reproduction steps before submitting any reports to avoid confusion or scenarios in which the triager fails to reproduce your identified flaw.

Additionally, keep in mind that LLMs are generally verbose, and often, won't hold off from including unnecessary steps or other verbose logs that add no value to your report. Always strive to keep your

report concise, but include the right amount of information that makes your submission more valuable and effective.

4. Following AI recommendations violating platform rules

Returning to our first practical example, we can see that within our report, the LLM appears to have recommended that we place an external link to our video proof of concept. Lacking background context, such as platform rules (e.g. Intigriti's Community Code of Conduct), LLMs will often include suggestions that may seem straightforward but are actually driving you to act against guidelines.

To avoid this, you must at all times recognize such violations or instruct the LLM to respect the platform and any applicable program rules.

> **Intigriti's Community Code of Conduct**
>
> If you're unfamiliar with our Community Code of Conduct, we generally disallow the use of external hosting and file-sharing services in an effort to limit potential leaks. If it is not possible to host the file on the platform due to technical limitations, we ask that you host the files within a password-protected ZIP file stored on a secure location, with the password included in the report.

5. Referencing attack vectors of speculative nature

As you may have experienced, LLM-based chatbots are generally great at making things work, or seem like they will work. This includes proving the validity of a non-existent or non-impactful vulnerability or by providing a speculative attack vector. However, triagers can sadly not work with assumptions and must follow the platform's triage standards (e.g., Intigriti's Triage Standards) or the outlined program rules when determining the severity of a submission.

Therefore, including the impact of speculative nature or unverified attack vectors will increase the time to triage, or worse, lead to incorrect severity assessment due to a lack of context. We always recommend that you provide sufficient evidence with each submission of how the identified vulnerability could be exploited. If you can not prove the validity of a vulnerability, it's best to gather more information before submitting a report.

6. Responding to feedback requests with AI-generated responses

When the incoming submission lacks data that a triager needs to confirm the reported weakness, they may prompt for a feedback request. In such instances, it's recommended that you follow up without using an LLM. Invoking a chatbot to respond to feedback requests may not always be the most thoughtful choice, as it may provide wrong answers, incorrect information or worse, fail to respond to the requested feedback. All of which will further drive up the time to triage, resulting in slower payouts.

Conclusion

In this article, we've summed up several tips that should help you generate well-written, AI-generated vulnerability reports that not only will get triaged and accepted faster, but will also result in faster and sometimes higher payouts. We've also learned how crucial human oversight still is. Although some LLM-based Models are progressing, we must never underestimate this important factor.

So, you've just learned something new about generating more compelling vulnerability reports with AI ... Right now, it's time to put your skills to the test! You can start by practicing on LLM playgrounds or... browse through our 70+ public bug bounty programs on Intigriti, and who knows, maybe earn a bounty on your next submission!

AUTHOR

Ayoub

Senior security content developer