OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Securing Open Source in the Age of AI

A Practical Guide for Maintainers, Security Engineers, Researchers, and the Communities That Depend on Them

Chris Aniszczyk
CNCF

Christopher "CRob" Robinson
OpenSSF

David A. Wheeler
OpenSSF

Version: 1.0 | May 2026

Executive Summary

AI tools are transforming how software is built, secured, and attacked, and open source is at the center of that shift. Large Language Models (LLMs) can now find vulnerabilities, generate patches, and review code at a speed and scale no human team can match. That is both an enormous opportunity and a mounting operational challenge for the open source ecosystem.

This eBook, produced by the **Open Source Security Foundation (OpenSSF**) and the **Cloud Native Computing Foundation (CNCF)**, distills real-world experience from maintainers, security engineers, and downstream consumers into concrete, actionable guidance. It is organized around three audiences:

- **Maintainers** who need to manage AI-generated contributions and vulnerability reports without being buried by them
- **Security Engineers and Researchers** using AI tools to find, disclose, and remediate vulnerabilities responsibly
- **The broader community** working to keep the open source supply chain secure as AI accelerates both offense and defense

What you will find in this eBook:

- How to set your project up to handle AI-assisted contributions and reports at scale, including security policies, reporting guidelines, and threat models
- How to use AI tools effectively to improve your project's security posture, with practical prompting techniques and a curated toolkit
- What responsible AI-assisted vulnerability disclosure looks like, and why providing patches and proof-of-concept code matters more than ever
- The real limitations and risks of AI tools, including hallucinations, slopsquatting, cost, and over-inflated severity scores, and how to guard against them

The bottom line: AI does not change the fundamentals of open source security. Least privilege, minimal attack surfaces, coordinated vulnerability disclosure, and proactive security engineering still win. What AI changes is the velocity of attacks, of reports, of fixes, and of the expectations placed on maintainers and security engineers alike. The communities and projects that learn to work with these tools intentionally will be better positioned than those that ignore them or are overwhelmed by them. Once the slop is cleaned we can then use these tools proactively as part of our steady-state of operations, but there will be bumps along that path.

This is math, not magic. And with the right practices, it is manageable.

Introduction

The past year has seen a rapid increase in the use of AI tools to create software and find vulnerabilities in existing codebases. While initially the robotic flood was predominantly "**AI Slop**", a few months into 2026 we started to see a **marked improvement** in the quality of reports sent to open source projects and commercial entities. We are on the cusp of realizing the dream of having helpful assistants that can augment our work by leveraging domain subject matter expertise that may not exist within a team. This is math, not magic, and it is not foolproof, however leveraging AI tools for development, security analysis, and remediation planning has reached acceptable quality levels to accelerate security outcomes for constrained maintainers and projects.

Recent LLM advances, where these tools have found previously-undiscovered vulnerabilities, have led some to observe that **cybersecurity looks more like proof of work**; the **Security Theater** of it all distracts from actually gaining practical utility from these newer tools. Using LLMs or AI-powered tools is becoming a vital and very useful toolset in helping create and review software. However, the robots are not going to solve all of the problems. The rapid rise of the utility of AI systems is the latest step in a long line of cyber-tooling being applied to open source software development. Fuzzers and static code analyzers are more recent examples from recent history of the application of both free and open as well as commercial tooling being used to assist in securing software. The interesting promise of LLMs is that it truly can be an "expert system" and augment a developer with specific application and cybersecurity expertise that they normally may not have access to.

In collaboration with open source project maintainers, security engineers, and other stake-holders that have been directly impacted by this deluge of data, we've collected a series of best practices informed by real experiences for anyone operating or contributing to this space to greatly ease the burden on developers and defenders alike.

Contents

The Realities of LLMs

LLMs are a snapshot in time and a reflection of the data and training that shaped them. Like any process, they can be tampered with, but that's not the focus here. These tools have evolved to the point where they are very good at finding both discovered and undiscovered bugs, on top of reviewing security issues themselves. AI has the advantage of patience, velocity, and scalability that an unaugmented human can no longer match. Agentic technologies allow the models to divide up the analysis and coding work up into discrete, digestible batches accelerating creators, defenders, and attackers.

Frozen in Time

It is that "frozen in time" aspect that all stakeholders in the ecosystem need to understand. The models are trained with a static dataset that exists within a specific time and space. The output they produce largely comes from pattern-matching. Typically, these systems do not automatically get updates and stay abreast of current events, and extra steps need to be taken to source newer external data for the model to be able to take these new data points into account. New models provide updates, just like new versions of software provide new features. To access more current information, you must provide additional context and commands.

The Duplicate Report Problem

A common problem with using AI to scan and report vulnerabilities is that other people are doing the same thing at the same time with the same tools. Anyone with access to a chatbot prompt can now claim to be a "security researcher". We have a whole community of active researchers today (each with varying levels of skills and experiences), plus we have project users reporting bugs, we have community members finding and fixing problems, and we have downstream manufacturers attempting to do the right thing by the projects they leverage as part of their product portfolios. With the hype behind using AI to discover vulnerabilities, we now truly have the "multiple eyes" that Linus' Law always spoke of, but when you have a dozen people all finding the exact same problem at around the same time and reporting it all to the project, it makes it hard on the maintainers to triage all the work and separate new findings from duplicates.

These new capabilities have been unleashed up on the ecosystem without first seeking to update these decades-old processes and infrastructure. The last 25 years of Coordinated Vulnerability Disclosure (CVD) has been intrinsically a HUMAN problem, one that worked on human-time and abilities. The new frontier models are not constrained by puny humans' need to sleep or only mentally be able to hold a certain cognitive load. Projects are seeing higher frequency of duplicate reports, which as related **previously**, takes precious time away from working on the project's goals while they must be triaged. Interestingly, this is an area where these new technologies show the most promise in helping relieve burdens off of maintainer's plates by possibly being enlisted to assist in triage and deduplication. Just being better at finding problems does not ultimately make security better. Substantial moves need to be made to empower maintainers to have access to these tools, develop methods to help confirm and triage (and deduplicate) the reports, and vet provided patches. Developers need ACCESS to the tools, time to LEARN how best to use and integrate them, and then time to USE, EVALUATE, and REACT to the findings.

The Robots Can't Rate YOUR Risk for You (and Will Over-Inflate Severity).

This problem has existed long before machine-assisted intelligence entered the conversation. NO third party can tell YOU what YOUR risks or risk appetites are. External people (and robots) don't have insights into what controls, processes, or policies you may or may not have in place. They also do not know precisely how you deployed, configured, and are using that software, nor what data and systems are adjacent to it and could be touched. Risk should ALWAYS be viewed from the eyes of the RISK HOLDER. Most external analysis of security vulnerabilities falls into one of two categories: an honest but inaccurate assessment of the risk (not having insights about you) or hyperbole and cleaving to the most drastic and dramatic outcomes, again with no context into the journey that software took to your system. Always take external assessments of risk with a full jar of salt (not just one puny little grain!). Review each report within your context and provide the assessment from that perspective.

Only as Good as the Data (and the Prompt)

In software engineering there long has been the maxim "Garbage in, Garbage out". This has never been so clear than in AI technologies. The AI models have all been trained on freely available open source software, exactly because there were no barriers to entry to do so. The examples that taught the robot about coding may not have been created by skilled software engineers nor may not have ever been intended for downstream commercial use. The AI does not know that and ingests the good code along with the **not-so-good**. Much like its creators, AI also makes mistakes and can write bad code. The use of AI also matters greatly here, requiring more expertise to use effectively, not less. Someone using these tools without understanding the problem at hand will create worse outputs. We can now extend this maxim to Garbage in, Garbage prompt, Garbage out. Remember that that free puppy isn't always free and may contain fleas and other creepy-crawlies! While that puppy is eager to please and remarkably fast at fetching, if you ask it to get some food don't be surprised when it brings you a dead squirrel instead of kibble.

"We can now extend this maxim to Garbage in, Garbage prompt, Garbage out."

Prompt Engineering: Quality In, Quality Out

While often portrayed as simply "talking to the computer," writing efficient and effective prompts is a skill unto itself. Writing a good prompt helps focus the model more precisely, it can more effectively break down the work into smaller, more focused tasks, and helps manage the resource budget available to the prompter. Like a human, an AI only holds so much knowledge at once. Proper management of the context is crucial. As newer, more advanced models become more broadly available, the engineers that understand how best to engage with these tools shepherd the robot will see the best outcomes. Prompt quality can be paired with evaluation, repeatable test prompts, regression testing, and measuring the security outcomes being worked towards, such as the reduction of false-positive findings and helping reduce the time-to-triage.

The Costs

The financial cost of using AI is rarely discussed. These tools are powerful and require equally powerful resources to produce all of these results. Like most things in life, all of this: subscriptions, tokens, and infrastructure to use these tools, is not free.

Beyond direct financial cost, the environmental footprint of AI tooling is worth factoring into your decisions. Training large models is enormously energy-intensive; inference is less so but adds up at scale. Choosing a smaller, focused model for a well-scoped security task is not just cheaper, it is also the lower-carbon choice. The right model for the job is good engineering and good stewardship. An SLM, a small language model geared to coding is going to have far less room to stray than a model trained on "The Kardashians" with significant operating expense reduction.

Sometimes the costs are funded by employers, sponsors, grants, or donations; other times not. There is a very real digital divide between those developers and organizations that can afford to fund this use and those that can not. These tools tend to be very "DIY" without a lot of instructions. It is very easy for even experienced users to get stuck going down a rabbit-hole and blow through a token or API budget, incurring unplanned costs (personal, project, or corporate). Much like with the migration from physical data centers to cloud and all the rigor and analysis that has developed to truly measure those services, so too will these systems improve, but today there are few guardrails in place to stop you from quickly spending your whole budget on a single task.

Thinking at a societal level, it is costly to build the facilities that house all the expensive gear to create software, find and fix vulnerabilities, and generate videos of cartoon cats eating tacos to the electronic beats of an AI-generated song pumping in the background. There are costs to power and cool these machines. When all you have is a hammer, everything tends to start looking like nails. LLM, open models, agents, etc. all have their time and place, and sometimes choosing to NOT use them is a better outcome for the specific task at hand. The impact to developers' jobs and livelihoods is also seldom brought up in all the hype; this dynamic needs to change as more and more technologists hit cognitive load caps and burn out.

Maintainers Guide: Integrating AI Safely

Setting Your Project(s) Up For Success

There are several things that open source maintainers and project members should **consider** to make their projects AI-hardened and AI-fluent. The reality is any contributor may use AI for any part of their contribution and there is no absolute guarantee someone can recognize the difference. We don't mean that AI should never be used nor contributions that use AI should be refused, but rather set your project up for success to provide LLMs and external contributors guardrails to safely use those tools in a manner that is acceptable to the project's maintainers and community at scale. Here are several things projects should consider to help ease the workload being put upon them by massive use of these automated tools:

Security Policy

The project should publicly share how to report security defects and how those reports are handled. These are things like a SECURITY.md or SECURITY.txt or other publicly-accessible document. Tell your community, your consumers, and external parties that want to interact with you how you want to be contacted. Consider setting up a private or protected way for researchers and reporters to confidentially contact you and share this information before it leaks to the general public.

[SEE POLICY **EXAMPLES**]

Reporting Guidelines

The project should state what their stance on AI-contributed or derived content is. Establish rules so that external contributors know how to share patches or reports effectively. Also document what data is required to be an acceptable report. Consider asking to privately have shared proofs of concept/vulnerability, focus on the specific section of code that is impacted, what could be done to exploit it, and consider requiring a proposed patch for the project to review. It is your opportunity to declare your expectations so they can be met. [SEE GUIDELINE **EXAMPLES**]

Consider an AI Policy

The **OpenSSF**, **CNCF**, and other foundations are working on documenting a specific policy to govern AI contributions. Documenting the desires of the project community in a discoverable location helps ensure that outsiders can comply. This also can state the anti-patterns of what types of AI-content the project will not accept.

[SEE AI POLICY **EXAMPLE** FROM KUBERNETES]

Documented Threat Model & Real Use Case Patterns and Antipatterns

AI (and external contributors) are more successful if the project shares how they desire the software to be used, acceptable scenarios to be deployed into, and what problems the project is aware of that could go wrong. Threat modeling can be as simple as answering four questions (What are we building? How can it be broken? What are we going to do about that? And How well did we do?), you could follow a formal methodology like STRIDE, **read this free book on it,** or even use tooling to create attack surface analysis and data-flow diagrams. The important thing is to state where the project is concerned and what controls of mutations are in place to protect against it. Also, stating scenarios or configurations that are EXPLICITLY forbidden, or that are not of concern are equally as useful. Many times tools will report an "amazing" finding, but that exploit requires root access or some bizarre and improbable configuration that users are advised NEVER to use. Telling the robots and researchers ahead of time helps focus reports on issues that are likely and that matter — it doesn't guarantee they will, but documenting this gives you something to point to

when you politely say "No, thanks". [SEE THREAT MODEL **EXAMPLES** AND PYTHON **"THREAT MODEL AS CODE"**]

Using AI to Improve Your Security

Using AI tooling to improve your project's security can be a game-changing experience.... both for good and not-so-good if not appropriately managed; even to the point of frustration. A security-focused mindset should be adopted when using AI tools to assist in writing code and reviewing the security of a project. Starting off by reading the **Security-Focused Guide for AI Code Assistants** is a great way to set the approach context for us human readers. Here are some additional techniques and tools for leveraging AI-technologies to improve the security of your project:

Context is for Closers

The single most important thing to understand about working with AI tools, be it LLMs, GenAI, Agentic, etc. is helping the robot help you by providing the proper context to execute tasks. We'll talk about two specific techniques (Skills and "Enhanced Prompting"), but it is critical to understand, just like people, giving the robot access to specific, focused data makes it better and less prone to error. Broadly, the technique is called Retrieval-Augmented Generation (**RAG**), and depending on the tool you're using it comes by several names: Projects, Context Cache, Collections, or Notebooks. Newer techniques such as **the Model Context Protocol (MCP)** is an additional method to provide your digital helper with guardrails to assist you. MCP and RAG are not tools that can be enabled as guardrails by default.

"As a Security Engineer..." Prompting and Beyond

LLMs function better when given the appropriate context (a lot like us humans!). Adjusting how you approach the tools can drastically make the experience more secure. Asking the model to take on a persona can help tell the model to pull in specific learnings and assets, and leverage those as it is determining how best to fulfill your requests. However, only telling it to take on the persona is often not enough. Past studies have shown that if this was the only thing you did, the results were actually less secure (see [**catherinetony2024**] and [**connordilgren2025**]). This might not be as true for the newest models, but clearly this is sub-optimal. There's no need for this to be the only thing you do. So combine good prompting with our next point to super-charge embedding crucial cyber and application security learnings into your daily work. Understanding how to **craft effective prompts** helps the system help you get to the outcomes you're seeking more quickly. Reach out to others and learn what works for them and adapt it to your needs. A poorly written prompt could output nonsense, requiring fine-tuning of the phrasing. It might also get stuck in a loop, quickly exhausting your token and compute budget, leaving you frustrated and without a solution.

> Threat modeling can be as simple as answering four questions (What are we building? How can it be broken? What are we going to do about that? And How well did we do?), you could follow a formal methodology like STRIDE, **read this free book on it**, or even use tooling to create attack surface analysis and data-flow diagrams.

This is where the Linux Foundation and other open source communities can work together to create and maintain better persona files (see the **OpenSSF Personas** as an example) and **skills** to assist those that are not as aware or technically able to integrate secure development practices by design and by default and allow your robot buddy to be a good code reviewer that has the appropriate security mindset.

Application Security (AppSec) AI Skills

Context is Signal for AI. Adding skills to your LLM are the "easy button" to integrate specific artifacts and resources into your development and security analysis. Bringing in unique subject matter expertise, like the **OWASP Top 10** list or the **MITRE ATT&CK framework**, help laser-focus the model's work. Adding the OpenSSF's **Concise Guides** and other guidance to help explicitly steer the LLM into industry-accepted supply chain security good practices. Provide the AI assistant with specific focused information and prompts relevant to security, and the AI will be more effective.

Available Tools

Creating and maintaining the most amazing software anywhere is a hard job! There is a wealth of tools and learnings that developers and other engineers are enabled to use today to make their interactions with AI coding and LLM-enabled analysis even more useful!

MLSecOps Practical Guide: To start off, when talking about AI-related technologies (AI. ML, LLM, GenAI, Agentic, MCP, etc), understand that there are a whole host of people involved outside of the traditional software or cybersecurity engineers that help build and secure our software today. The **MLSecOps Practical Guide** speaks to some of the new personas that are involved in creating and training AI-based solutions, and how best to weave in both traditional application security practices, but also highlights where these new technologies require us to craft new ways of securing the systems and data these solutions work with.

Consider Open/Local & Small Models vs. Public LLMs: As developers, we are gifted with a near-infinite number of tools and technique combinations. This "gift" extends into the AI space with a multitude of commercial and open models, large and small, that are available for free or a fee. No one model is "best" at every scenario, and developers should consider the pros and cons for each model and budget accordingly as new projects integrate AI tools. The Frontier Models get a lot of press, and tend to be at the bleeding-edge of technological innovation. Consider the current public models stacked up against the true next-generation tools that are only available privately at the moment: The current public models are very effective at assisting in generating and reviewing code, scanning code bases and binaries, and are ready for use today. While they can not do some of the more complex tasking and stringing together a large series of analysis, they absolutely can serve developers and consumers today and provide very good results. A drawback of the public large models is that they are shared environments, where your data and requests can mingle with other users of the model and even are used to train models, which you may not wish. While there are safeguards in place, an intelligent attacker can infer and possibly exfiltrate data against your wishes simply due to the shared nature of the infrastructure.... just like anything in the cloud today if not properly protected.

An alternative is running open models on local or isolated hardware (a technique that many commercial enterprises, in fact, use today). This approach allows you to ensure that your sensitive data never leaves your trust boundary and allows you to tune the model to your specific needs, as opposed to a general purpose model that must serve multiple use-cases. Also think about Small Language Models (SLMs) that are fast, efficient, and focused. Depending on the problem you are trying to solve, these may offer performance and data security benefits over the public large language models (with the trade-off of missing out on advanced features, which you may or may not need for the particular task at hand).

Cyber Reasoning Systems: You don't have to have access to the most advanced models on the planet to reap the benefits of these new technologies. The Open Source Software Cyber Reasoning Systems (**OSS-CRS**) are the results of a collaboration between the OpenSSF and the **AIxCC competition**. The AIxCC projects looked at how best to use LLMs in combination with traditional scanning tools and other techniques to find and fix previously unseen vulnerabilities. These projects now have an **open community** where these systems are continuing to be refined and improved and are available for anyone to pick up and use within their work today.

Free Tools for Qualifying Projects: Security is at the forefront of many of our communities' thinking. So much so that our **LF**, CNCF and OpenSSF foundations have made arrangements with several of our member organizations to provide access to commercial-grade tooling for qualifying open source projects and maintainers. **Kusari Inspector** is one such **tool freely available to our communities**. Built by our community members, Inspector blends cybersecurity knowledge and practices with the power of machine learning to simplify many common security tasks and directives a project may want to implement. We hope to add many other free tools in the future so OSS project maintainers can take advantage of them and OSS users can reap their benefits.

Good Prompt + Good Context (code and various security scans) + Good Model = Good Results

If any one of these things is missing your work can go down a dark, sad path. As a reminder, good engineering practices contribute to good security outcomes. In the age of AI, this is critical.

But, the Future is So Bright… We are at the middle section of our AI/ML/LLM/GenAI/Agentic/etc. journey. As the ecosystem builds consensus on secure standards and tools, we should

Good Prompt + Good Context (code and various security scans) + Good Model = Good Results

expect even more capabilities available to developers. With the increase in velocity of reports upstream and to manufacturers, LLMs can be leveraged to assist in triaging and analyzing this multitude of defect reports. As veteran and new researchers alike take to their keyboards with AI assistance, it is expected that multiple Finders will discover the same/similar defects and report them. Again, here is where LLMs can assist in review and deduping of reports.

"A fool with a tool is still a fool."

Things to Watch Out For

Dr. David A. Wheeler at the Linux Foundation is known to proclaim "A fool with a tool is still a fool." This absolutely applies to the use of AI technologies and techniques. Having the most advanced "intelligence" at your side does not guarantee perfect code nor perfect security–indeed the only perfect security is a computer in concrete unplugged. There is a blended whole series of old and new threats that are unique to the components and methodologies involved. The tools are not infallible, and they are well documented to provide false data in a problem described as "hallucinations". The OpenSSF's **Secure Agentic Framework (SAF)** helps illustrate common threats and patterns that exist with using agentic tools, as well as methods to mitigate these risks. For every good person using AI to make software and organizations more secure, there are equal numbers of those lurking in the

shadows seeking to exploit unknown or unpatched vulnerabilities. Tools like the SAF help defenders understand where they may be at risk.

Common problems that are seen within LLM use include hallucinations in the writing of code (e.g. writing of imperfect and flawed code with full assertion of its veracity). Another problem with the models occurs when dependencies are falsely offered. Sometimes this will take the form of recommendations to install old, vulnerable dependencies as the "secure solution," (remember how when they were trained!) and at other times, the models can consistently recommend imaginary dependencies (be sure to double check these). This latter scenario has caught the interest of malicious actors due to the surprising consistency in the suggestions across different model providers. With these consistent imaginary package names in hand, black hats will preemptively register malicious packages with these bad package names in an attack now called "slopsquatting". This mirrors the traditional "typosquatting" attacks where the bad guys will register intentionally mistyped urls in the hopes of capturing inattentive users not checking and blindly clicking links that seem good enough. What is old is new again, just with a different name.

Much like with the other model peculiarities, the old cybersecurity maxim "Trust, but verify" must be top of mind when evaluating the model's output before merging. Having that healthy dose of skepticism and taking time to double check the output will pay dividends in the future and ideally avoids many common security missteps.

We also expect an uptick in project forking downstream. Legislation like the EU's Cyber Resilience Act put steep fines on manufacturers and very tight timelines for reporting and repairing. If upstream is not able or interested in reviewing and ingesting all of these reports on this new time-horizon, downstream will start to take steps to protect themselves from legal and reputation risk. We could see numerous forks of the same upstream package proliferate throughout repositories. This not only creates more digital cruft, but will cause confusion for downstream users as to which project is the appropriate one to ingest and use (e.g., because it has security updates). Once a project is forked, how will these downstream developers keep parity with features and functions of the main upstream project and how much time, effort, and resources will they commit to this new technical debt they now fully own without a community of collaborators? Consider the additional technical debt and workload burden you will take on prior to rushing to fork upstreams' work. Think about how your access to these new tools can assist the existing project and community prior to taking the drastic step of "going it alone".

Another technique that adversaries will use is the seeding of intentionally malicious packages and models. Understanding where the model or package came from and how it was curated until you ingested it is critical to staying cyber safe and sound. This is where the classic solutions of packaging signing (now with 100% MOAR AI!!! E.g. **Model Signing**) help developers consuming these items understand who created the artifact and that it was untampered with down to the point of ingesting it into your project. Signed artifacts help assure consumers of all types that these artifacts came from trusted sources and have not been tampered with–but only when verified. Signing does not solve all problems and make things instantly secure, but just like with current application development using signed artifacts helps lay down a solid foundation to enhance with your new development work. A signed model helps consumers validate how that artifact or model was built. It is one technique that can work with CI/CD systems to enforce policy decisions and can be paired with other attestations like SLSA to get a fuller picture of the pedigree and provenance of the packages you are working with.

One last tool available to the community is our LF Education courses. These classes cover a broad range of topics that can

help arm developers and operators with the skills and understanding they need to complete their jobs. Most recently the OpenSSF created **Secure AI/ML-Driven Software Development (LFEL1012)**, a free course that covers many of these and other topics. An AI assistant is going to be more or less effective depending on the prompting of its user, so if you develop software, take a course on how to develop secure software such as our free course **Developing Secure Software (LFD121)**.

Phoning a Friend

Lastly, we've learned from decades of open source collaboration that we are all stronger together. All across the Linux Foundation ecosystem of projects and communities there are a wide variety of deep subject matter expertise that we can all lean into and rely upon. Groups like the CNCF **TAG-Security & Compliance** or the **OpenSSF** are dedicated to helping improve the ecosystem for everyone.

Researchers' Guide: Coordinated AI-Assisted Disclosure

Software engineers aren't the only ones recognizing the power and utility of this new class of tools. Many professional security researchers and hobbyists are jumping onboard and using the Frontier and Open Models to find vulnerabilities. As we saw with the advent of the wide use of fuzzers over a decade ago, the research community has enthusiastically embraced these new capabilities, oftentimes to the detriment of upstream open source maintainer and security teams globally.

Finding a bug is great. Reporting that bug to the project and maintainers is at the core of how open source software grows and continually improves. Sadly the speed at which these new tools work, the overwhelming verbosity of the output, and lack of context into the project that is being probed will many times bury these volunteers in insurmountable mountains of data they have to sift through. Each vulnerability report can take a developer between 1-8 hours to review and evaluate. These tools tend not to find just one defect at a time, oftentimes they'll pattern-match their way around a code base and dump tens, dozens, hundreds of reports onto a project at once with minimal context. Here are some examples that those brave souls that want to find the "next big vulnerability" can follow to ensure the best experience for everyone involved in that interaction.

An excellent place to start is with the **Guidance for Security Researchers to Coordinate Vulnerability Disclosures with Open Source Software Projects**. This lays out the foundation for successful interactions with open source projects and their developers. Building upon that work, consider the following:

Respect the Reporting Guidelines of the Project

It is critical to approach a project where they are and within acceptable mechanisms under which their community operates. Random "drive-bys", especially at the scale "enhanced" by AI tools, is not welcome. To maximize the impact of the time you spent documenting a defect, ensure you take several minutes to learn about the community to identify the right people and formats for reporting; this last step will help make the report reception more successful.

Privately Provide the Project a Proof of Concept (POC) and/or Proof of Vulnerability (POV)

One of the core tenets of Coordinated Vulnerability Disclosure (CVD) is ultimately ensuring that the end-consumer of a project or product has the ability to quickly deploy fixes and reduce the public window between disclosure and patching as greatly as possible. Oftentimes vulnerabilities reported are difficult to reproduce or hit very corner-case scenarios within a program and can take time to understand the problem and then think about how to solve it.

This is why projects are encouraged to offer a private, secure means to share confidential information back and forth between a Finder and other parties that the maintainer may require to assist in triage, patch creation, testing, and staging updates so they are available for downstream at Public Disclosure (PD). Providing a Proof of Concept (POC) or Proof of Vulnerability (POV) can save a large amount of time during the private phase of the

disclosure. This helps focus the maintainer precisely onto what behaviours are seen, and can help kickstart the patch creation process in a more focused manner.

These POC/POV are valuable to the defenders working to correct the vulnerability, but sadly they are equally useful to bad actors. It is critical that such things are protected and not shared publicly until after fixes are available to the public. Even before the advent of AI-assisted research, POCs can quickly be weaponized and redistributed as attacks against vulnerable software users. Frontier Models have reached a stage in their existence where they can be very good at writing a POC. If you are using some form of AI assistant in your research, consider having the system craft the POC to share with the maintainer through their secure communication channel to accelerate their defence work.

Wherever Possible, Provide a Proposed Patch to Address Your Discovery

Fixes are more important than findings. Full stop.

The reality of upstream maintainership is a constant deluge of unsolicited reports, data, PRs, and requests. You may feel that you are "helping" by reporting a high volume of potential vulnerabilities, but doing so without **POC/POV** and a **PROPOSED PATCH** creates large volumes of unbudgeted work and quite frankly removes value and adds additional work to everyone involved.

Providing the developer with a proposed patch, whether or not that code is exactly what gets merged, helps the developer see precisely where the vulnerability is exposed and starts them thinking about solutions for it. This can save HOURS of work on their part by giving them a baseline to start from rather than a blank terminal window. LLMs have improved in this regard, and can be a helpful ally for both defenders and researchers in creating that first draft of a fix that can be then quickly iterated over. You have the privilege to use these tools, whereas the majority of developers upstream lack access to use that privilege responsibly. Exercise that privilege by providing proposed patches instead of simply generating more noise and work for the maintainer.

Provide Test Cases to Avoid Future Regressions

Much like the request to provide patches alongside the report, writing test cases are not only informative about how the bug manifests itself, but also can be integrated into the existing test harnesses. Much like the patch-writing, providing pre-written test cases accelerates getting the remediations out to the end-users of the product and helps ensure that this defect never returns again. The greater the test coverage of a project, the less likely it is to see repeat defects returning like a shambling zombie shuffling back from the grave looking for a tasty snack.

Be Aware, the Code is Open, Someone May Have Already Found What You Did

This will be reinforced at the close of this discussion where we highlight the real-world reality of AI usage. Open Source is amazing. It is the engine of innovation and an epic value-creator for the global economy. The true strength of OSS is that anyone, at any time, from anywhere can look at the code and understand what's going on, and based on the license, can use it for their own purposes. How awesome is it to potentially have thousands of contributors all working towards common goals?

OSS is SO awesome, that is it is a key dataset that LLMs are trained upon, ...ALL "the open source", the good, the bad, and the ugly. This means at any given moment it is quite likely that others are looking at the exact same code you are investigating. This could be the bad guys, it could be other researchers, it absolutely is contributors to the project and the maintainer, it is also downstream

consumers of the code, and a whole host of students and academics. Someone might be faster at submitting the PR than you are, or as oftentimes in the case, the vulnerability may have been privately disclosed to the project and they already have patches in the pipeline for release. Be aware of this, especially when it comes time for acknowledgements in the public advisory, that you may be sharing the glory with other curious souls and circuits.

Be Clear and Up-Front if and How AI Tools Were Used in the Report

Everyone is at different stages of their AI journeys. Some have been in the thick of it already and are overwhelmed and simply don't want to deal with more unsolicited "helpers". AI reports tend to be overly verbose and are not always clear, concise, and to the point as to what was discovered, how it was discovered, and what the possible impacts could be. Provide a clear description of what you've discovered.

Understand that some projects may not wish to accept code that was generated through these technologies and not accept security analysis for them. While it may not feel satisfactory for the reporter, for creators of software that typically has "no warranties or expectations of support" this is a perfectly legitimate response; choosing how best they wish to spend their time and energy volunteering. If the project has an AI-use policy, review and understand how they want to interact with these types of engagements. Respecting their rules and wishes goes a long way towards building trust and eventual acceptance of the work and collaboration to solve the problems you are sharing.

Where We All Go From Here

"It's been a busy year this week in AI."

This whole treatise was inspired by the recent events going on in and around the ecosystem. The recent moves by the **Frontier Model providers** as well as the announcement of **Project Glasswing** highlight the energy and interest in this space to honestly try and use tools to make software objectively better. Another interesting phrase used within the industry, "it's been a busy year this week in AI." Things, without exaggeration, are changing every month: new techniques, new tools, new models; on and on it goes.

These new tools do risk bifurcating the ecosystem into those that have the means and support to access and use these tools, and those that do not. We must all collectively work to ensure that these capabilities are open and available for upstream maintainers of critical projects. This is yet another set of useful things to be integrated into our daily operations.

We're very excited to get the advanced capabilities of the new Frontier Models into the hands of maintainers through efforts like Project Glasswing. There are real, tangible benefits in helping them improve and secure the software that is the digital unpinnings of our world. As the initial cohort of developers starts to use and learn from these new tools, we'll be excited to share those learning with the broader ecosystem. We're confident that many of the techniques are absolutely transferable to the publicly available models and tools, and finding the good practices with this focused group can help accelerate others' journeys into exploring using these tools more. We'll also be working with everyone involved to communicate what was found and fixed, and collectively share what we've all learned.

There are groups within the ecosystem that are considering these issues and are working together to help solve these challenges today. Industry, the frontier model providers, and the community are already engaging and collaborating together to leverage LLMs and other tools to help improve the quality and security of software. Downstream is already starting to see the results of efforts like **Project Glasswing** in the form of PRs and patches coming into the pipeline. Additionally, teams that participated in the DARPA AIxCC competition were required to **donate their work as open source software**. Some of these teams are converting to neutral governance under the Linux Foundation, and there will be opportunities to extend the usability of these tools for upstream open source maintainers. Using various combinations of these types of industry efforts, at the Linux Foundation, we are taking this new reality very seriously and are working in multiple streams to address the different concerns of all the stakeholders involved. We are working to get these advanced tools and tokens into the hands of upstream maintainers and helping develop tools and training so that these software engineers don't need to be AI experts. Instead, they can continue being experts on the amazing software they create and curate every day (with a little help from some robot-buddies!). We're working with industry to help them understand the new patch and disclosure patterns that they will need to adapt to to serve their businesses and customers. We're also working in an assortment of working groups and communities to develop tools, skills, and best practices for the ecosystem.

There was already a lot of low hanging fruit in security vulnerabilities and the industry as a whole was doing a mediocre job fixing some of them. Very few companies were actually doing a great job fixing all of them. This reinforces the actual, real, and continued imperative to double down in an ongoing fashion and rapidly adopt proactive security engineering practices; least privilege, minimal images, rebuild and redeploy, default deny, etc.

The current way industry is deploying applications, treating them as fragile infrastructure—pets even—is unsustainable. Instead, maintainers of global infrastructure need to think about technology as a replaceable good. They should focus on enabling people, understanding intent, driving the outcome, and redesigning processes, rather the approaches we experience today. The security practices remain unchanged, when they are applied and how that is done is the only change.

Even with all the AI tooling to patch and fix these as soon as they come in, business and organizations aren't fast enough to update.

We're curious to get feedback from the community. The OpenSSF **Vulnerability Disclosure Working Group** is working on codifying the current best practices from the community and sharing how projects and researchers can responsibly use AI tools to improve security. You can take their **survey** today to share your perspective!

We're also working with Linux Foundation Research to explore AI security practices. We'd love to get your **feedback** for that study that will be published later this year.

Remember what Uncle Ben said, "With great power comes great responsibility." Security is a team sport. It takes players that fill different roles out on the field working together toward the common goal of improving security for everyone. Come join us, we're always eager to strengthen our ranks to help claim that win for the community and all consumers of open source!

About the Authors

Chris Aniszczyk is CTO of Cloud and Infrastructure at Linux Foundation and CTO of Cloud Native Computing Foundation (CNCF). He has over 20 years of experience as an open source developer and a passion for building a better world through open collaboration.

Christopher Robinson (aka CRob) is the Chief Technical Officer and Chief Security Architect for the Open Source Software Foundation (OpenSSF). With over 25 years of experience in engineering and leadership, he has worked with Fortune 500 companies in industries like finance, healthcare, and manufacturing, and spent six years as Program Architect for Red Hat's Product Security team.

Dr. David A. Wheeler is an expert on developing secure software and on open source software. He created the Open Source Security Foundation (OpenSSF) courses "Developing Secure Software" (LFD121), "Understanding the EU Cyber Resilience Act (CRA)" (LFEL1001), and "Secure AI/ML-Driven Software Development" (LFEL1012). His other contributions include "Fully Countering Trusting Trust through Diverse Double-Compiling (DDC)". He is the Director of Open Source Supply Chain Security at the Linux Foundation and teaches a graduate course in developing secure software at George Mason University (GMU).

Acknowledgements

The following community members helped contribute to and review this article:

Chris Tanski Cloud Security Architecture — Elevance Health
Emily Fox Portfolio Security Architect — Red Hat
Faseela K CNCF Technical Oversight Committee (TOC)
Greg Kroah-Hartman Linux Kernel Maintainer — The Linux Foundation
Madison (Oliver) Ficorilli Staff Manager — GitHub
Michael Lieberman CTO & Founder — Kusari
Nicole Schwartz Staff Product Manager — Anaconda
Sarah Evans Distinguished Engineer — Dell

OpenSSF

OPEN SOURCE SECURITY FOUNDATION

openssf.org