Carnegie Mellon University
Software Engineering Institute

# LESSONS LEARNED IN COORDINATED DISCLOSURE FOR ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING SYSTEMS

*Allen Householder, Vijay Sarvepalli, Jeff Havrilla, Matthew Churilla, Lena Pons, Shing-hon Lau, Nathan VanHoudnos, Andrew Kompanek, and Lauren McIlvenny*

August 2024

DOI: 10.1184/R1/26867038.V1



## Table of Contents

# Introduction

In this paper, we, as researchers at Carnegie Mellon University's (CMU's) Software Engineering Institute (SEI), incorporate several lessons learned from the coordination of artificial intelligence (AI) and machine learning (ML)[1] vulnerabilities at the SEI's CERT Coordination Center (CERT/CC). We also include our observations of public discussions of AI vulnerability coordination cases.

Risk management within the context of AI systems is a rapidly evolving and substantial space. Even when restricted to cybersecurity risk management, AI systems require comprehensive security, such as what the National Institute of Standards and Technology (NIST) describes in *The NIST Cybersecurity Framework (CSF)*.[2]

In this paper, we focus on one part of cybersecurity risk management for AI systems: the CERT/CC's lessons learned from applying the Coordinated Vulnerability Disclosure (CVD)[3] process to reported "vulnerabilities" in AI and ML systems. Therefore, we analyze the *process of coordination* rather than the details of *what is being coordinated*. However, some case details matter to the process, and we will highlight the relevant facets as we go. Because we focus on the coordination process, we do not address taxonomies of adversarial AI techniques,[4, 5] AI vulnerabilities,[6, 7] AI risk management,[8] or AI incident analysis.[9, 10]

---

1 In this paper, we discuss both AI and ML; however, from this point forward, we use AI to refer to both AI and ML since ML is a subset of AI. If we specifically discuss ML, we use that abbreviation.

2 *The NIST Cybersecurity Framework (CSF) 2.0* (https://www.nist.gov/cyberframework)

3 *The CERT Guide to Coordinated Vulnerability Disclosure* (https://certcc.github.io/CERT-Guide-to-CVD/)

4 *MITRE Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS)* (https://atlas.mitre.org)

5 *NIST AI 100-2 E2023: Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations* (https://csrc.nist.gov/pubs/ai/100/2/e2023/final)

6 *MITRE Common Weakness Enumeration (CWE)* (https://cwe.mitre.org)

7 *NIST Vulntology* (https://github.com/usnistgov/vulntology)

8 *NIST AI Risk Management Framework (RMF)* (https://www.nist.gov/itl/ai-risk-management-framework)

9 OECD *AI Incidents Monitor (AIM)* (https://oecd.ai/en/incidents-methodology)

10 *AI Incident Database (AIID)* (https://incidentdatabase.ai)

## Methodology

In the context of AI vulnerabilities, our analysis of the CVD process roughly follows the structure of the failure mode effects analysis (FMEA) process, although we do not go into as much detail as a full FMEA process normally would.

According to D. H. Stamatis, the FMEA process is an eight-step method:[11]

1. *Select the team and brainstorm*
2. *Functional block diagram and/or process flowchart*
3. *Prioritize*
4. *Data collection*
5. *Analysis*
6. *Results*
7. *Confirm/evaluate/measure*
8. *Do it all over again*

Because we are reporting preliminary findings, we do not attempt to complete all eight steps of the FMEA process. Instead, we focus on the first five steps. We summarize our methodology, which is based on the FMEA process, in Table 1.

[11] *Failure Mode and Effect Analysis - FMEA from Theory to Execution* (2nd Edition Revised and Expanded) (https://asq.org/quality-press/display-item?item=H1188). *Section 2.1.5 The Process of Conducting an FMEA.* (pp. 36). American Society for Quality (ASQ) (https://app.knovel.com/hotlink/pdf/id:kt00AOASI1/failure-mode-effect-analysis/process-conducting-an)

*Table 1: Process Failure Modes and Effects Analysis Summary*

| FMEA Process Step(s) | Comment |
|---|---|
| 1. Select the team and brainstorm | We selected the vulnerability analysis team at the CERT/CC, which is part of the SEI at CMU. |
| 2. Functional block diagram and/or process flowchart | We provided these in the *Process Model* section. |
| 3. Prioritize | We framed the question as, "What are the failure modes of the CVD process for issues involving AI systems?" We prioritize issues that are significant to the coordination process for AI systems, although in some cases those issues are like those encountered in other technology niches. |
| 4. Data collection | Our data collection was opportunistic based on reports we received at the CERT/CC through April 2024 as well as public media reports of other cases in the same time frame. Our collection process was informal, and we focused on problem coverage more than their frequency. |
| 5. Analysis<br>6. Results | We document our preliminary findings in this paper. We focus on qualitative collection and coverage of problems instead of quantification of problem prevalence. |
| 7. Confirm/evaluate/measure<br>8. Do it all over again | We omitted these steps because this is a preliminary paper. |

# Background: Coordinated Vulnerability Disclosure

Coordinated Vulnerability Disclosure (CVD)[12] is a process that begins with the discovery of a vulnerability in an information system. This discovery immediately divides the world into two sets of people: those who know about the vulnerability and those who do not. The goal of CVD is to keep adversaries in the latter set for as long as possible while the relevant stakeholders work to at least mitigate, if not remediate, the vulnerability. CVD participants often ask the following questions:

1. What actions should I take in response to this knowledge?
2. Who else needs to know what and when?

The CVD process continues until the answer to the first question is *nothing* and the answer to the second question is *nobody*.

## What Is a Vulnerability?

We take our working definition of a *vulnerability* from the *CERT Guide to Coordinated Vulnerability Disclosure* (i.e., CVD Guide):[13]

> *A vulnerability is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy.*

Note that this definition is deliberately broad:

- It does not hinge on the specific impact of the exploitation. We are not beholden to the confidentiality, integrity, availability (CIA) triad in our definition. We can accommodate other impacts, such as the Parkerian Hexad, which extends CIA to include authenticity, possession/control, and usefulness.[14] Going even further, our definition allows us to include various harms (undesirable outcomes) that can arise in the context of AI systems and societal-scale systems.[15]
- It does not depend on an intelligent actor as an adversary (hypothetical or otherwise). We can quickly explain this in physical terms: A leaking foundation in an earthen dam is a hazard that needs to be addressed regardless of whether malicious actors are involved or not. Yes, a malicious actor could exploit these conditions to cause harm, but the harm can still be realized in the absence of any malice. Ultimately, it is necessary to respond regardless of the adversarial threat.
- It does not require that the conditions or behaviors are unintended or surprising. The features of the system that allow the potential for harm to exist may be intentional, deliberate, and expected.

---

12 *CERT® Guide to Coordinated Vulnerability Disclosure: Coordinated Vulnerability Disclosure is a Process, Not an Event* (https://certcc.github.io/CERT-Guide-to-CVD/tutorials/cvd_is_a_process/)

13 *CERT® Guide to Coordinated Vulnerability Disclosure: Vulnerability* (https://certcc.github.io/CERT-Guide-to-CVD/tutorials/terms/vulnerability/)

14 *Parkerian Hexad* (https://www.sciencedirect.com/topics/computer-science/parkerian-hexad)

15 See, for example, the types of harm listed in Microsoft's *Types of harm* page (https://learn.microsoft.com/en-us/azure/architecture/guide/responsible-innovation/harms-modeling/type-of-harm).

What is usually unexpected is the scope or scale of the harmful outcomes. Unintended consequences of desired features can be vulnerabilities. In AI systems, *model cards* have been proposed as a way "to clarify the intended use cases of machine learning models and minimize their usage in contexts for which they are not well suited."[16]

- Finally, it does not require that the security policy be explicit. Explicit security policies are often developed in an *empirical policy discovery process*[17] by which some event occurs, a harm is realized, and then what had been an *implicit policy* is made explicit after consideration.

One way to reduce the vagueness inherent in acknowledging an *implicit policy* is to adopt a generalized catch-all statement such as the following:

- What is not expressly permitted is prohibited.
- What is not expressly prohibited is permitted.

However, this approach only works in environments where policy writers are certain that they know exactly what they want and have already articulated it. Otherwise, this approach just sets up a situation where a policy violation occurs, but it is determined to be a problem with the policy being underspecified, so the policy is updated, and we return to the *empirical policy discovery process*.

We propose a more values-based approach that makes the intent of the policy clear without necessarily listing all the permitted or prohibited things. For example, a policy might include a statement such as "the system shall not allow harms to occur."[18] That statement is not specific enough to dictate what the system should *do* in any specific circumstance, but it at least explicitly acknowledges the intent of the system's creators. This can make it easier to recognize policy violations when they are identified. For example, a specific set of conditions that allows a specific harm to occur would be immediately recognized as a policy violation even though the details were not laid out in advance.

---

16 *Model Cards for Model Reporting* (https://arxiv.org/pdf/1810.03993)

17 *Empirical policy discovery* is a fancy way of saying, "We did not know what we did not want until it happened, and we did not want it, so now we are writing down that we do not want it to happen again."

18 This is conspicuously close to Asimov's First Law of Robotics, which states, "A robot may not injure a human being or, through inaction, allow a human being to come to harm." While there are logical issues that arise with Asimov's formulations of those laws (many of which he explored extensively in his own fictional stories), this "law" may serve as a reasonable approximation of a default implicit policy that many would agree with and is closely aligned to the principle of non-maleficence embodied in *primum non nocere*, "First, do no harm." Implicit policies are certainly influenced by the surrounding culture, and there is little doubt that Asimov's ideas about the Laws of Robotics have influenced the zeitgeist.

## How Does a Process Apply?

A different way to arrive at a definition of *vulnerability* that is perhaps more relevant to our study of AI cases is to identify the unit of work as a *reported problem* rather than requiring strict adherence to the definition of a *vulnerability*.[19] Given a report, we can examine the question, "Does this reported set of facts about the conditions and behavior of a system require coordinated effort to remediate?" In other words, "Should we apply the CVD process to this report?"

A brief horticultural analogy may illustrate the point. Consider the following two questions: "Is this plant a weed?" and "Should we remove this plant?" The first question is ontological, while the second is procedural. Right away we can see the challenge if we were to ask, "Is corn a weed?" or "Is grass a weed?" We do not have enough information to answer these questions without additional information about the intended use of a particular plot of land. Now imagine two contexts for these questions:

- On a golf course, corn plants are considered a weed, and grass plants are not.
- In a farm field, grass is considered a weed, and corn plants are not

We might struggle to answer the ontological question with an adequate definition of *weed* because the definition is inherently based in a *policy* rather than in the properties of the plant itself. In fact, a common answer for "What is a weed?" is "any plant that grows where you do not want it to." The categorical definition hinges on the existence of a policy, whether implicit or explicit.

However, given an understanding of the policy (i.e., intended land use), we can interpret the answer to the procedural question in either context. We should remove corn plants from golf courses and remove grass from farm fields. We did not need to inquire about the detailed features of the plants; it is sufficient to know what belongs and does not, which the policy is sufficient to inform.

*Coordination* is the most important concept when discussing CVD. When one individual, organization, or entity knows about a problem, and another individual, organization, or entity can fix that problem, there is a need to coordinate. Knowledge of a problem *must* precede resolution. Unless the actors who can implement the solution are aware of the problem, they cannot fix it. So, there must be a way for those who know but are unable to act to convey that information to those who can act if only they knew. That is the part of *Disclosure* that is inherent to coordination. There is no coordination without at least this form of disclosure.

Another relevant aspect of *Disclosure* in CVD is that informed consumers make better choices. In the simplest sense, informed users deploy fixes. Mere *fix availability* is necessary, but it is not a sufficient condition for *fix deployment*, which represents the actual remediation of vulnerable systems.[20] Going beyond their role in the critical path of remediation, when consumers of products and services do not have adequate information about the features and risks of the products and services they depend on, they cannot make informed choices. *Disclosure*—from those who find problems to those who can fix

---

[19] *On Managing Vulnerabilities in AI/ML Systems* (https://doi.org/10.1145/3442167.3442177)

[20] The distinction between *Fix Ready* and *Fix Deployed* is elaborated in the *Vultron CVD Protocol: CVD Case State Model Introduction* (https://certcc.github.io/Vultron/topics/process_models/cs/).

them, and from those who know about the problem and the fixes to those who use the affected systems—provides a public service that allows consumers to make more informed choices about the products and services they use and improves their ability to manage their own risk profile. *Disclosure* improves processes for everyone by allowing (1) information to be collected and aggregated and (2) trends in that information to be identified.

In our opinion, *Vulnerability* is the least important part of CVD. It does not matter as much whether the problem meets any given definition of vulnerability. There are plenty of definitions, and while they have many similarities, there are always edge cases where one definition is met while another is not. Good definitions are always useful; however, we are more concerned with the fact that there are known states of the world that allow harm to be realized and that those states are resolved in a way that minimizes the resulting harm that occurs, which usually involves coordination and disclosure.

## Where Does a Vulnerability Reside?

An AI system is a software system that incorporates an AI component. An AI component can be thought of as the portion of the system that comprises the trained model, the data used to train the model, model configurations, and any data transformations required for the model to ingest data or contextualize output.[21] An AI system almost always has other software that is outside the scope of the AI component, and vulnerabilities can affect any part of the whole system. One challenge that software engineers face in understanding and controlling risks in AI systems is that, by design, these systems break some principles of isolation and encapsulation. This tight coupling leads to software components that are harder to analyze, maintain, and document.

As we illustrate in Figure 1, vulnerabilities in AI systems can exist in the following:[22]

- the model itself
- the training data
- the other system components
- any combination of the model, the operational data, interfaces between the model, and the rest of the software system

How we detect vulnerabilities, which stakeholders are affected (for coordination), and what is involved in mitigating the vulnerabilities will vary based on where the vulnerability resides.

---

[21] *Component Mismatches Are a Critical Bottleneck to Fielding AI-Enabled Systems in the Public Sector* (https://arxiv.org/pdf/1910.06136)

[22] In other words, traditional software vulnerabilities remain a concern in AI systems with additional concerns brought by the interaction of AI components.

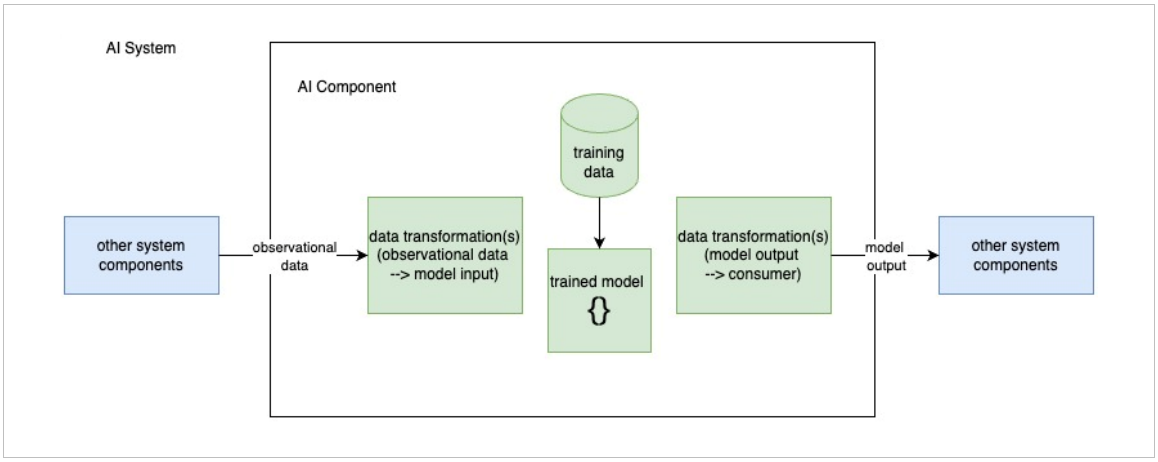


*Figure 1: AI System View*

## Process Model

We assume the following process model for CVD. This model is based on the CVD Guide[23] and expanded based on *The Vultron Coordinated Vulnerability Disclosure Protocol* (i.e., Vultron Protocol).[24] The CVD Guide focuses on the *Reporting* through *Publication* steps, but the broader process shown in Figure 2 is consistent with both the *CVD Guide* and the technical process models defined as part of the Vultron Protocol.

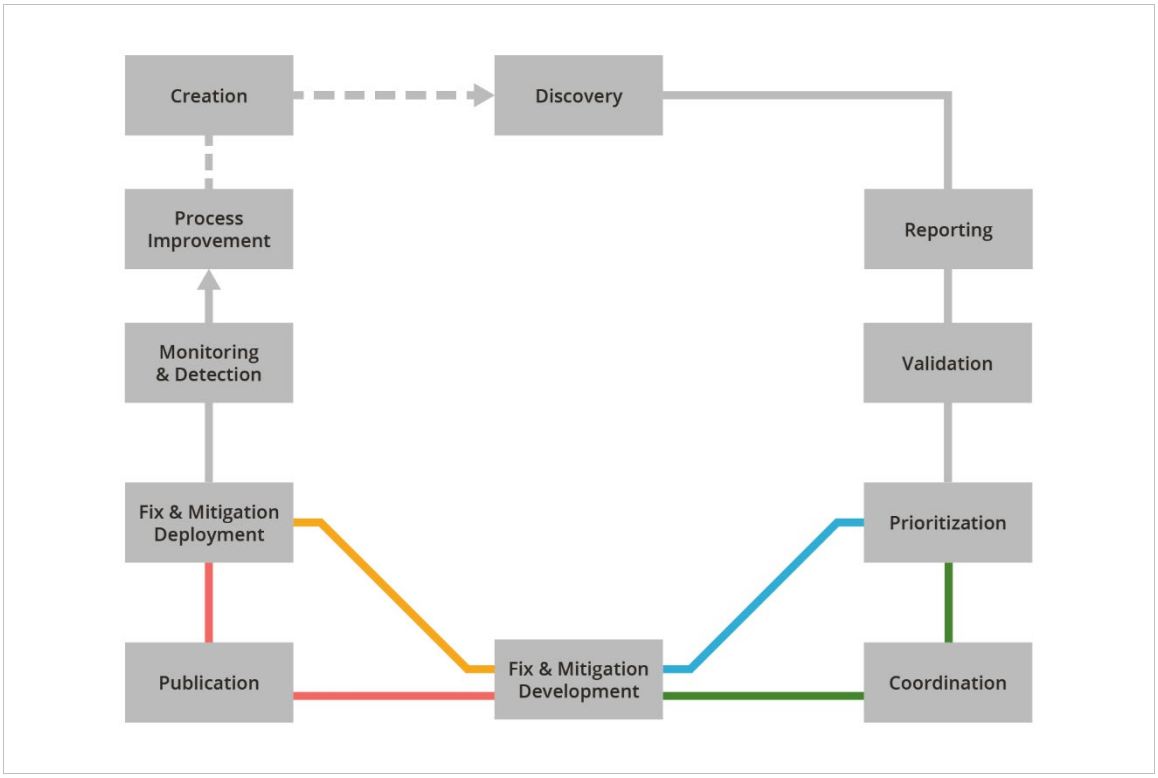


*Figure 2: A Process Model for Coordinated Vulnerability Disclosure*

[23] *The CERT® Guide to Coordinated Vulnerability Disclosure* (https://certcc.github.io/CERT-Guide-to-CVD/)

[24] *Vultron CVD Protocol: The Vultron Coordinated Vulnerability Disclosure Protocol* (https://certcc.github.io/Vultron/)

## The Process Is Not Strictly Linear

Although we drew Figure 2 as if the process is a continuous cycle, where each step completes before the next one begins, we acknowledge that the process often includes cycles and feedback loops. For example, the information provided in *Reporting* might be insufficient to complete the *Validation* step, or it might take a few rounds of *Reporting*, *Validation*, *Prioritization*, and *Coordination* before *Fix & Mitigation Development* begins for all affected parties.

Also, not all steps occur in every scenario. In some cases, such as a vulnerability reported in a software-as-a-service (SaaS) system, there might be only one party involved in the vendor/deployer role, so the *Coordination* or *Publication* steps might be omitted. Likewise, while both are considered good practices for CVD practitioners, *Monitoring & Detection* and *Process Improvement* are not always as integrated into the CVD process as we might prefer to see.

## Success and Failure Occur in CVD Process Steps

Each step in the process can succeed or fail. To claim that CVD is *working as intended*, the success criteria for each step from *Discovery* through *Process Improvement* should be met. Although it would be nice if *Creation* always succeeded or if no vulnerabilities were ever produced because then the CVD process itself would be unnecessary. While that is a worthy goal, we expect it is one that can only be approached rather than an achieved 100% of the time.

In Table 2, we describe the success and failure criteria for each step. In the *Process Steps and Their Failure Modes* section, we then expand the sketches of success and failure modes from Table 2.

*Table 2: Broad Success and Failure Criteria for CVD Process Steps*

| Process Step | Success | Failure |
|---|---|---|
| Discovery | Finders discover existing vulnerabilities. | Finders do not discover existing vulnerabilities. |
| Reporting | Finders/reporters identify vulnerabilities and report them to parties who can take further action. | Finders/reporters do not identify and report vulnerabilities, or no one who can act on them receives the vulnerability reports. |
| Validation | Vendors and/or coordinators validate (i.e., confirm, reproduce) reported vulnerabilities. | Vendors and/or coordinators do not validate (i.e., confirm, reproduce) reported vulnerabilities. |
| Prioritization | Vendors and/or coordinators prioritize valid vulnerabilities appropriately. | Vendors and/or coordinators do not prioritize valid vulnerabilities appropriately. |
| Coordination | Vendors and/or coordinators coordinate prioritized vulnerabilities with affected parties. | Vendors and/or coordinators do not coordinate prioritized vulnerabilities with affected parties.[25] |
| Mitigation & Fix Development | Vendors develop fixes for prioritized vulnerabilities. | Vendors do not develop fixes for prioritized vulnerabilities. |
| Publication | Vendors, coordinators, and/or finders/reporters disseminate information about fixes or mitigations to affected system owners. | Vendors, coordinators, and/or finders/reporters do not disseminate information about fixes or mitigations to affected system owners. |
| Fix & Mitigation Deployment | System owners widely deploy fixes or mitigations. | System owners do not widely deploy fixes or mitigations. |
| Monitoring & Detection | All parties increase their vigilance in detecting exploit tools and attacks to recognize adversarial activity. | Not all parties are vigilant in detecting exploit tools and attacks to recognize adversarial activity. |
| Process Improvement | Vendors communicate the results of their root cause analyses of known vulnerabilities to inform the development process and decrease the number of new vulnerabilities. | Vendors do not conduct root cause analyses of known vulnerabilities, or they do conduct root cause analyses, but they do not communicate the results, failing to inform the development process and decrease the number of future vulnerabilities. |
| Creation (of the Next Vulnerability) | Vendors ensure that known vulnerability types do not recur. | Vendors do not prevent known vulnerability types from recuring. |

[25] We provide an extensive list of potential coordination failure modes in the *CERT® Guide to Coordinated Vulnerability Disclosure: Troubleshooting Coordinated Vulnerability Disclosure* (https://certcc.github.io/CERT-Guide-to-CVD/howto/coordination/cvd_recipes/).

# Process Steps and Their Failure Modes

In this section, we describe the 11 steps of the process and the failure modes that we have observed in each step. Some of these failure modes are specific to AI products, services, and their vendors, whereas others are more general and can apply to any novice vendor or sector. Over the years, we have observed similar CVD capability evolution in network infrastructure, traditional computing, mobile computing, operations technology, consumer Internet of Things (IoT), and embedded computing. In that sense, AI-centric organizations may just be at a different point in their learning curve due to their relative newness.

The various failure modes we describe in this section are based on our participation in and observation of the CVD process for vulnerabilities in AI systems. In places where we can cite public reports of coordination-related problems, we do so. However, CERT/CC policy prohibits us from discussing the coordination details of any particular case beyond the technical details we publish in our vulnerability notes. The intent of this policy is to maintain trust with the participants of the CERT/CC coordination process and avoid drawing unnecessary attention to the parties involved. We focus on describing the problems. While we do cite a few specific vulnerabilities in the following sections, we do so to illustrate a relevant concept rather than to imply wrongdoing.

## 1. Discovery

We begin our process model at the vulnerability *Discovery* step because without vulnerabilities to be discovered, the CVD process is not needed. This step succeeds when an existing vulnerability is found. It *fails* when no existing vulnerabilities are found.

In the following sections, we describe the possible failure modes encountered during the *Discovery* step.

### The SaaS Model Inhibits Independent Security Testing

Security testing may violate the terms of service (ToS). This concern is shared with any SaaS and many websites and other online applications that limit (by terms of service and acceptable use policies) what actions are permissible by users.[26] This concern can be mitigated using clear vulnerability disclosure programs, where the vendor or service provider makes clear what kind of testing is permitted, thus protecting security analysts who might otherwise commit ToS violations. Wiz.io researchers

[26] *Overview of Risks, Threats, and Vulnerabilities Faced in Moving to the Cloud* (https://insights.sei.cmu.edu/library/overview-of-risks-threats-and-vulnerabilities-faced-in-moving-to-the-cloud/)

found a vulnerability in Replicate.com's service that highlights the value of vulnerability reporting even if their findings could violate the ToS.[27, 28, 29]

### Architectures Are Unfamiliar to Many

Graphics processing unit (GPU) architectures have grown rapidly in importance, yet their impact on system security is less well understood. Expertise in specialized hardware is a problem common to any specialized computing environment (e.g., embedded, field-programmable gate array [FPGA], application-specific integrated circuits [ASICs], operational technology [OT], IoT), but it is notable in the AI space simply because of its rapid growth and scale.

VU#446598: *GPU kernel implementations susceptible to memory leak*[30] (aka LeftoverLocals[31]) is a vulnerability that highlights differences in mindset. Traditional CPU-based[32] computing has long recognized the need to isolate memory between processes, whereas the LeftoverLocals vulnerability demonstrated that some GPU-based computing failed to isolate local memory between distinct processes.

### The System Under Test Has Limited Instrumentation

Introspection and instrumentation of AI components is an area of open research. It is often quite challenging (even for the developers) to understand the behavior of the system in specific instances. Whereas most traditional software can be observed using debuggers to evaluate the state of a system through each step of its execution, interpreting the influence of internal model parameters on system behavior dramatically increases the complexity for understanding what an AI system is doing.

Use of AI-as-a-Service (AIaaS) can exacerbate these challenges, since even the system owner might have to treat the AI components as an opaque box under the service providers' control.

### Security Analysis Tooling Is Lacking

Software security testing and analysis tends to focus on finding specific categories of problems. Many of these problem categories have existed for quite some time and are well understood. For example, both memory protection errors and Structured Query Language (SQL) injection vulnerabilities have multiple tool sets available that analysts can use to find and evaluate them. In the AI space, the

---

27 *The Risk in Malicious AI Models: Wiz Research Discovers Critical Vulnerability in AI-as-a-Service Provider, Replicate* (https://www.wiz.io/blog/wiz-research-discovers-critical-vulnerability-in-replicate)

28 *Shared Network Vulnerability Disclosure* (https://replicate.com/blog/shared-network-vulnerability-disclosure)

29 *Replicate: Terms of Service* (https://replicate.com/terms)

30 CERT Coordination Center: VU#446598: *GPU Kernel Implementations Susceptible to Memory Leak* (https://www.kb.cert.org/vuls/id/446598)

31 *LeftoverLocals: Listening to LLM Responses Through Leaked GPU Local Memory* (https://arxiv.org/pdf/2401.16603)

32 *CPU* stands for central processing unit.

technology itself is changing rapidly, as are the toolkits available to security analysts. That is not to discount tools such as the Python Risk Identification Tool for generative AI (PyRIT),[33] the Adversarial Robustness Toolbox (ART),[34] or CleverHans.[35] Our point is that a significant gap remains between the tools available for traditional software security analysis and those available for AI software security analysis. The availability of tools is partially limited by the limited capability to examine the models we just described.

A separate CMU study on the use of hazard analysis in AI systems found similar gaps. The researchers described it in their 2022 paper:[36]

> *Regarding tools, we have found that prior work in responsible AI focuses on data and model development and the fairness and biases of AI models. While this is an essential component of developing safe AI systems, our participants noted that it is only one part of a larger product system. There is a need to support practitioners in thinking more broadly about entire product systems and the people who are using these systems. Future tools and processes should support engineers in thinking about how end-users may think about and behave with AI-based systems.*

### Testing the Infrastructure Has Capital Costs

Some models are very expensive to construct (e.g., large language models [LLMs] with training costs above $100 million).[37] The type of organizations that can construct these models is limited to very capital-intense ventures with "deep pockets." Some systems are effectively single-instance systems, making it difficult for individuals or organizations with fewer resources to adequately reproduce a test environment independent of the production system.

There is also a human capital cost to test for issues and interpret results. Analysts who understand both the security and AI aspects of a system are rare at present, so there might not be enough of these analysts available to look for vulnerabilities.

## 2. Reporting

Vulnerability *Reporting* succeeds when found vulnerabilities are reported to some individual, organization, or entity that is at least one step closer to being able to fix them than the reporter. In most cases, reporting directly to the vendor of the affected product or service is preferable. Situations where that is not the case often involve third-party coordinators who have the capability and capacity to

---

33 *Azure: PyRIT* (https://github.com/Azure/PyRIT)

34 *Trusted-AI: Adversarial Robustness Toolbox* (https://github.com/Trusted-AI/adversarial-robustness-toolbox)

35 *CleverHans Lab: CleverHans* (https://github.com/cleverhans-lab/cleverhans)

36 *Exploring Opportunities in Usable Hazard Analysis Processes for AI Engineering* (https://insights.sei.cmu.edu/library/exploring-opportunities-in-usable-hazard-analysis-processes-for-ai-engineering/)

37 *Measuring Trends in AI* (https://aiindex.stanford.edu/report/)

coordinate multiple parties when necessary to ensure that fixes and mitigations are propagated to all affected products and services and their respective vendors and service providers.

This step fails when found vulnerabilities are not provided to those on the path to fixing the problem or when the information becomes public prior to those parties being aware of the problem.

In the following sections, we describe the possible failure modes encountered during the *Reporting* step.

### AI Community Members Are Unaware of Existing Coordination Practices, Processes, and Norms

The AI community has expanded rapidly, transforming readily available components into comprehensive solutions like chatbots, image detectors, and virtual assistants. Despite the growing demand for these applications, the community lags in addressing vulnerabilities and adopting established software maintenance processes such as CVD.

Recent attempts, like the Coordinated Flaw Disclosure (CFD)[38] process, merely introduce new terminology that reiterates the CVD process for AI-based software. This underscores the AI software development community's lack of mechanisms to coordinate issues, regardless of whether they are labeled as vulnerabilities or flaws. Creating a duplicate process obscures the ways that vulnerability coordination for AI software is simply an extension of the existing process and fails to benefit from decades of experience in conventional software.

We are not implying that the stakeholders are identical for traditional cybersecurity vulnerabilities and AI flaws. The impact of vulnerabilities in AI components may have more far-reaching implications to an organization's risk exposure than traditional product vulnerabilities. Organizational risk management has a broader scope than just adversarial exploitation, including potential losses or costs resulting from abusive content creation, models discriminating against protected classes, or other harms. This shift in scope is a driver of our emphasis on the process-as-category definition of vulnerability at the beginning of this paper.

Coordination remains a crucial aspect of the CVD process, yet it appears to be neither widely adopted nor recognized within the growing AI software development community. In contrast, recent software development communities are well versed in the CVD process and have been implementing it with varying degrees of success.[39] Our position is that the CVD process—with appropriate adaptations to accommodate the expanded harms and stakeholder-specific concerns—remains well poised to address these issues.

---

38 *Coordinated Disclosure for AI: Beyond Security Vulnerabilities* (https://arxiv.org/html/2402.07039v2)

39 *Coordinated Vulnerability Disclosure Programme Effectiveness: Issues and Recommendations* (https://www.sciencedirect.com/science/article/pii/S0167404822003285)

### Affected Products or Services Cannot Be Identified

Identifying affected software when disclosing vulnerabilities is a well-known challenge, which is exacerbated in AI due to the multiple software components and their interactions as well as the absence of software composition data like a software bill of materials (SBOM) for these complex systems. In particular, the tight coupling of data and software results in a fluidity of identity within subcomponents of the systems. The level of abstraction about the system an SBOM is designed to describe fails to capture many of these subcomponents.

AI software, often built from conceptual algorithms, makes it difficult to document or identify vulnerabilities in these inherent capabilities. For instance, in 2020, the CERT/CC published VU#425163: *Machine learning classifiers trained via gradient descent are vulnerable to arbitrary misclassification attack*, which states the following:[40]

> *Machine learning models trained using gradient descent can be forced to make arbitrary misclassifications by an attacker that can influence the items to be classified. The impact of a misclassification varies widely depending on the ML model's purpose and of what systems it is a part.*

At the time, it was challenging to determine which systems, products, or services were affected. This challenge was partly due to a lack of transparency in the use of gradient descent as a core algorithm in the software stack implementations of ML classifiers. Some components of AI software, such as models, could also be affected by a vulnerability. These models, which function as both data input and software, are often not identified as software or documented with components and versions in detail. SBOMs might eventually help, but their effective use for AI-unique components remains to be seen.

A class of adversarial ML vulnerabilities also exists where the vulnerability is introduced in the training data and metadata of the ML model. These attacks are commonly referred to as *data poisoning attacks*.[41] Mitigating the insertion of data poisoning attacks is a data provenance issue that existing tools (e.g., datasheets for data sets and data-set versioning) are ill equipped to solve. For the purposes of coordination, being able to trace data provenance is important to (1) ensure that data owners are included in the coordination effort where appropriate and (2) help expose attack vectors from data when an owner cannot be identified or is otherwise questionable.

Finally, identifying affected versions presents a subproblem. It is not always straightforward to determine the version of an AI system under test, especially for services where the concept of discrete versions is elusive. Continuous deployment practices, while beneficial for other reasons, can complicate identifying versions because it is often unclear what version a particular endpoint had at the time of testing. This lack of certainty seems to be primarily a problem of incomplete control over the testing environment. For some AI systems, the challenge of continuous deployment is amplified because models are being autonomously trained and deployed within a component of a larger software system.

---

[40] CERT Coordination Center VU#425163: *Machine Learning Classifiers Trained via Gradient Descent Are Vulnerable to Arbitrary Misclassification Attack* (https://kb.cert.org/vuls/id/425163)

[41] *MITRE ATLAS: Poison* Training *Data* (https://atlas.mitre.org/techniques/AML.T0020)

These model updates can be pushed much faster than vulnerabilities can be coordinated, and the development process relies on having sound tests in place to have confidence that newly deployed models are not introducing problems.

### Vendor Identification Is Challenging

Further down the supply chain, even when affected products (e.g., a vulnerable open source library) can be identified, it is not always straightforward to pinpoint a specific vendor or determine the impact on downstream products, services, and vendors. As larger vendors absorb software projects due to popularity or usage, the original vendor may change or become unapproachable for the CVD process.

An SBOM can potentially address this issue, but its use is not as widespread as it could be, and its coverage of potential vulnerabilities is unclear. One significant challenge any bill of materials (BOM) effort faces is assigning identities to artifacts (i.e., naming things).[42] The question remains whether an SBOM can trace every artifact to its source, similar to how food labels often leave "spices and natural flavorings" unspecified.

The analogous concept of an AI BOM (AIBOM) has also been proposed,[43] but it is even less far along than the work on SBOMs. An AIBOM also introduces a number of its own concerns.[44]

### The Vendor Is Unprepared to Receive Reports

Some vendors are unprepared to receive and process vulnerability reports. This deficiency is particularly evident in the coordination of AI-related issues. We identified two prevalent scenarios that hinder coordination efforts:

1. The vendor is inexperienced and lacks a developed CVD intake process.
2. The vendor has an established CVD process, but it is not integrated with the AI development process. Even for vendors with an existing experienced Product Security Incident Response Team (PSIRT),[45] we have found that the PSIRT is often not well connected to the AI developer community within the organization.

---

42 *CISA: Software Identification Ecosystem Option Analysis* (https://www.cisa.gov/resources-tools/resources/software-identification-ecosystem-option-analysis)

43 *Army Issues RFI for Project Linchpin AI Bill of Materials* (https://executivegov.com/2023/11/army-issues-rfi-for-project-linchpin-ai-bill-of-materials/)

44 *Our Response to the US Army's RFI on Developing AIBOM Tools* (https://blog.trailofbits.com/2024/02/28/our-response-to-the-us-armys-rfi-on-developing-aibom-tools-2/)

45 *FIRST: PSIRT Services Framework* (https://www.first.org/standards/frameworks/psirts/psirt_services_framework_v1.1)

### The Vendor Reacts Unconstructively to Reports

A common issue among many novice vendors is the belief that nobody should be talking openly about product flaws, and this belief can sometimes dominate a vendor's initial response to a vulnerability report. Sometimes this is easy to overcome by further explaining the CVD process to improve the vendor's understanding, but not always.

Unconstructive behavior from vendors can include the following:

- ignoring the reporter because they are not a paying customer
- treating the reporter with hostility (e.g., threatening legal action against them)
- treating vulnerability reports as trade secrets

In addition to the common problems in CVD reporting identified in the CVD Guide,[46] we have observed that AI software vendors also tend to treat a vulnerability report as irrelevant unless it has something to do with the core algorithms or data input into the AI system. Vendors sometimes view flaws in other software components inherent in the system as somebody else's problem.

## 3. Validation

The *Validation* step succeeds when the recipient acknowledges the reported issue as a genuine problem. This step fails when the reported issue is not recognized as valid due to reasons such as an insufficient description, non-reproducibility of claims, or other factors.

In recent AI CVD disclosures, vendors have either dismissed reports for not meeting the current vulnerability definition[47] or deemed the software documentation as adequate to address the reported problem.[48]

This step presents technical challenges for the vendors of AI software and coordinators of AI vulnerabilities. Issues such as testing infrastructure costs, identifying affected versions, rapid development cycles, and unfamiliar environments can make it difficult for the reporter to provide a clear and reproducible problem description. Consequently, it can be challenging for the report recipient to reproduce the claims sufficiently to validate the report.

In CVD, the purpose of a report is to provide adequate information to replicate the contained claims. However, it is not always clear what constitutes a good report for an AI system. Questions arise regarding the necessary amount of environmental or contextual detail; version identification; and the complexity of different versions of data, architecture, training regime, trained model, and surrounding

---

46 *CERT® Guide to Coordinated Vulnerability Disclosure: Reduce Friction in the Reporting Process* (https://certcc.github.io/CERT-Guide-to-CVD/howto/initiation/reduce_reporting_friction/)

47 *I Stumbled Upon LLM Kryptonite – and No One Wants to Fix this Model-Breaking Bug* (https://www.theregister.com/2024/05/23/ai_untested_unstable/)

48 CERT Coordination Center VU#253266: *Keras 2 Lambda Layers Allow Arbitrary Code Injection in TensorFlow Models* (https://kb.cert.org/vuls/id/253266)

software. These complexities can make it difficult to replicate results in even a standard deployment environment. The details of where the problem resides in the system—in the model versus the training data, for example—will determine what information is required for a good report.

In the following sections, we describe the possible failure modes encountered during the *Validation* step.

### Vendors Claim That a Vulnerability Does Not Meet the Current Definition or Requirements

This failure mode is somewhat related to the lack of maturity of the vendor's (or coordinator's) operations in handling AI-related vulnerabilities (discussed in the *Reporting* section). While the PSIRT may have a clear definition of traditional hardware and software vulnerabilities, it may not be able to entirely understand or validate a report of AI-related vulnerabilities using the same methods. A coordinator may also lack the tools or capabilities (including personnel) to validate the report, which may include multiple steps or technical assignments that are not readily available using their current procedures/processes. Coordinators need to understand how to tease out the nuances of what undesired behavior constitutes a vulnerability. For example, it may be necessary to develop mechanisms to differentiate undesired model output that results from data drift from a problem in the software operation resulting from an attack.

### Vendor Documentation Has a Limited Effect on Vulnerability Determination

A challenge arises when users have expectations about the capabilities and limitations of components since their expectations may not always align with the vendor's documentation. Vendors sometimes use their documentation as a shield by asserting that a common use case is not a vulnerability simply because the documentation advises against it. One example of this phenomenon is CVE-2023-48022, in which the vendor asserts, "This report is irrelevant because […], as stated in [the product's] documentation, [it] is not intended for use outside of a strictly controlled network environment."[49]

Our perspective is that if the default behavior of a product or service allows insecure use that leads to security-relevant impacts when exploited, despite documentation that provides guidance for correct use, it is considered a vulnerability. Conversely, if the default behavior of a product or service is acceptable and the documentation outlines how to alter the configuration to allow a potentially insecure use with appropriate warnings, it is likely not a vulnerability. This perspective underscores the importance of intuitive and secure default behaviors in product design.

For AI systems, relying on documentation to avoid vulnerable deployed instances raises the problem of explainability, performance expectations, and non-deterministic behaviors. It may not be possible to adequately document "secure" default behaviors of AI systems because these behaviors cannot be known before the system is operational. For example, an AI system may behave differently depending on the input data or other deployment contexts used to fine-tune the system. AI systems that perform well (i.e., in an acceptably secure fashion) in one context might demonstrate undesired behavior in

---

[49] *NIST National Vulnerability Database: CVE-2023-48022* (https://nvd.nist.gov/vuln/detail/CVE-2023-48022)

other contexts. In these cases, documentation can be useful to warn end users about the possible risks of using the system, but documentation alone does not remediate the vulnerability.

## 4. Prioritization

Report *Prioritization* is when stakeholders prioritize valid reports for follow-up action.[50] In this step, we interpret *Prioritization* as not merely comparing one report to another but also comparing fix development to new feature development.[51] This step succeeds when valid reports of problems are regularly blended into the tasking priorities of new feature development in affected vendor products. This step fails when new features always take precedence over problem fixes or when use of inadequate metrics perpetually delays reports from reaching those who can fix the problems.[52] We are not saying that fixes always supersede new features; rather, we are saying that it is a red flag if features are always given more attention than reported problems. The AI community may be uniquely vulnerable to the incentives of always chasing bleeding-edge features given the intense competition underway in the emerging generative AI industrial complex.[53]

In the following sections, we describe the possible failure modes encountered during the *Prioritization* step.

### Business Incentives Can Cause Short-Term Myopia

Technical debt,[54] akin to financial debt, accrues over time. The pressure to deliver features promptly often leads to the postponement of addressing potential issues. The risk of postponing a fix or mitigation—much like unnoticed cracks in a dam—accumulates silently alongside technical debt. By the time it is recognized as a problem, it has often escalated into a significant issue. Significant financial incentives can prompt individuals to overlook potential risks in the pursuit of immediate gains, which leads to misaligned incentives.[55] This short-term focus can result in long-term vulnerabilities, which seems poised to become a larger problem given the high rate of investment in AI systems. It is crucial to balance immediate business incentives with the long-term health and security of the system.

---

50 *SSVC: Stakeholder-Specific Vulnerability Categorization: Enumerating Stakeholders* (https://certcc.github.io/SSVC/topics/enumerating_stakeholders)

51 We touch on the prioritization of the deployment process in the *Deployment of Fixes and/or & Mitigations* section.

52 *CERT® Guide to Coordinated Vulnerability Disclosure: Prioritization Heuristics* (https://certcc.github.io/CERT-Guide-to-CVD/howto/coordination/prioritization/)

53 *Measuring Trends in AI* (https://aiindex.stanford.edu/report/)

54 *10 Years of Research in Technical Debt and an Agenda for the Future* (https://insights.sei.cmu.edu/blog/10-years-of-research-in-technical-debt-and-an-agenda-for-the-future/)

55 *A Series on Four Overarching Themes Across Acquisition Programs: First Theme, Misaligned Ince*ntives (https://insights.sei.cmu.edu/blog/a-series-on-four-overarching-themes-across-acquisition-programs-first-theme-misaligned-incentives/)

Even organizations that have processes to manage technical debt might not know about the new ways an AI system can accrue technical debt. AI systems are more data dependent, so they can develop feedback loops, experience model drift, and have problems that are difficult to reproduce.[56] Some of these topics may be familiar to ML engineers, but teams that focus on prioritization and tasking may not be aware of the added responsibility needed to properly manage and maintain AI systems. The state of the practice regarding constraining technical debt is also related to the maturity of model development. Products with a high degree of volatility in their data and data pipelines tend to have more difficulty constraining technical debt.[57]

### The Norms of Expected Behavior Are Not Well Known

Software engineering has a long history of recognizing and dealing with the risks associated with research-quality code that makes it into production.[58] AI culture does not share that history of computing and cybersecurity practices. It is highly academically focused. Open collaboration through draft papers outstrips the peer review process; however, we are not complaining about the world moving too fast. We are simply acknowledging that traditional computing already had a problem with communicating secure development practices to students and early career learners, and the AI community has even less exposure to those concepts.

Reporting, prioritizing, and responding to discovered problems in software is not new to AI systems. We do not necessarily need to build completely new processes. We do need to adapt existing processes to different operational tempos and stakeholder expectations.[59] The CERT/CC's experience with introducing new stakeholder communities to the CVD process over the years has led us to be skeptical of claims that "We are so different that we are special and need special treatment." The false uniqueness bias is a known phenomenon in psychology.[60]

The speed at which AI software products have burst into existence has meant that these organizations have not had much opportunity to understand their responsibilities within the existing cybersecurity frameworks much less understand how these frameworks need to change to account for additional weaknesses introduced by non-traditional software in the stack. While we understand that solving challenging domain-specific problems is paramount to these organizations, the field needs to mature and understand that providing safe and secure software products is important.

---

[56] *Hidden Technical Debt in Machine Learning Systems* (https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf)

[57] *Engineering AI Systems: A Research Agenda* (https://arxiv.org/abs/2001.07522)

[58] *The Evolution of Science Projects, Third in a Four-Part Series Exploring Themes Across Acquisition Programs* (https://insights.sei.cmu.edu/blog/the-evolution-of-science-projects-third-in-a-four-part-series-exploring-themes-across-acquisition-programs/)

[59] Coordinated Disclosure for AI: Beyond Security Vulnerabilities (https://arxiv.org/html/2402.07039v2)

[60] *Explaining False Uniqueness: Why We are Both Better and Worse Than Others* (https://compass.onlinelibrary.wiley.com/doi/full/10.1111/j.1751-9004.2008.00076.x)

Cybersecurity practices and procedures have matured over decades. Today we have standards bodies at the national and international levels that are committed to understanding and helping practitioners mitigate cybersecurity risks. These bodies should be utilized when attempting to mitigate cybersecurity problems in AI software. At the same time, these bodies and organizations need to work to develop and extend existing standards related to AI because the ML components of an AI system are vulnerable in new and unique ways that are unfamiliar to the traditional cybersecurity community.

Finally, some AI community responses to very real concerns about the negative social impacts (i.e., AI harms) of the features of their software (whether intentional or not) are reminiscent of Dan Akroyd's Saturday Night Live character Irwin Mainway, who, when confronted about his company's toy for children, Bag O'Glass, being unsafe, retorted, "It has a big sticker that says, 'Kid: Be Careful. Broken glass.'" Warning labels and disclaimers are fine for what they are, but software must do better than that.

## 5. Coordination

*Coordination* is the process of engaging all the parties affected by a problem to produce and deploy a fix or mitigation for the benefit of the public. This step succeeds when valid reports affecting multiple parties are successfully communicated, received, and facilitated to communicate next steps. Successful *Coordination* should lead to a measurable reduction in harm.[61] This step fails when valid reports affecting multiple parties are not coordinated, which can lead to larger windows of vulnerability among end users. For the AI ecosystem, we have found great disparity in expectations concerning both the process that needs to be followed to coordinate vulnerability reports as well as the desired outcomes of that process.

In the following sections, we describe the possible failure modes encountered during the *Coordination* step.

### Vendors Fail to Cooperate with Others

Multiparty CVD cases are increasingly common, especially with AI-related vulnerabilities. The involvement of multiple parties is a direct result of the software supply chain where AI components are included in other products and services, which can then be layered even further (e.g., data from one vendor leading to models trained by another, leading to others fine-tuning models in further applications). Coordination across these parties can become discordant. This issue is not specific to AI vendors, but when a large case involves a significant number of novice vendors, it can be tricky to sort out all the issues that can arise.

---

[61] "The ideal scenario occurs when everyone coordinates and cooperates to protect the public." from *CERT® Guide to Coordinated Vulnerability Disclosure: Why Coordinate Vulnerability Disclosures?* (https://certcc.github.io/CERT-Guide-to-CVD/howto/preparation/why_coordinate/). For a deeper analysis of what success means in CVD, see the CERT/CC GitHub site (https://certcc.github.io/Vultron/topics/background/cvd_success/).

### Vendor Tempo Is Mismatched

The deeper a vulnerability is embedded in a product or service, the more coordination time is required to ensure that all end products and services are fixed. For example, when a traditional software library component has a problem, it is necessary to ensure that all downstream-dependent vendors incorporate a root-level fix into their products as well. This is a general problem with coordinating multiple organizations with different operating tempos based on network effects. AI-based systems can add to the challenge when vendors work at vastly different paces because of their different systems engineering maturity levels and different business drivers.

The failure to coordinate accounting for the differences in vendors' operating tempos can result in the premature release of sensitive business and security information to public forums. Different organizations release bug fixes using differing paradigms (e.g., closed versus open source systems). Novice vendors may not have established CVD policies that lead them to conduct an unplanned and reactive disclosure process. Likewise, information about the vulnerability being coordinated might be intercepted (due to lax security practices) by bad actors who want the information to be made public immediately. Information leaks like these that occur before agreed disclosure time frames undermine the trust in the CVD process and make it harder to engage affected parties in the future.

### Vendors Insist That Problems Must Be Reported by Customers

Novice vendors sometimes assume that customers are the only people who could possibly want to contact them about problems with their products and services. Or perhaps they feel that they need to pay attention to only paying customers because they are paying customers. However, CVD often happens because somebody notices something and wants it to be fixed out of a sense of public service, regardless of whether they are a customer of the affected vendors. Problems reported by non-customers is a common experience for coordinators like the CERT/CC, where we often receive reports of vulnerabilities in products and services offered by organizations that we have no prior relationship with. This is not unique to AI organizations, but it is pervasive among AI organizations because they are commonly new vendors.

### Vendors Insist That Reporters Sign NDAs to Interact

This problem is an extension of the previous one. Not only is it incorrect to assume that the only reports a vendor might receive are from paying customers, but it is also incorrect to assume that reporters will be able to or willing to sign a non-disclosure agreement (NDA) to interact with the vendor about a problem report. Those reporting a problem already have the information, and they can do with it what they please. If the vendor wants the reporter to share information with them, the vendor should not make it onerous to do so. This may be more common in the AI community when intellectual property protection concerns, driven by competitive forces, lead to overly restrictive communication that can prevent open dialog with outsiders, let alone with competitors.

We are not discouraging vendors from offering bug bounties with an NDA as part of the agreement. However, the NDA should be limited to forfeiting the bounty if the information is otherwise disclosed. "Catch-and-kill" tactics to deal with vulnerability reports are possible, but they are ethically dubious,

just as they are in journalism. There is a genuine risk of coupling bug bounties with NDAs to buy silence from reporters and allow the vendor to continue offering vulnerable products and services to the public who remains unaware that information has been kept from them. This approach may violate the concept of informed consent when it comes to license agreements and acceptable use policies,[62] not to mention transparency requirements placed by securities regulations and business reporting requirements.[63]

In the past, we have also encountered vendors that expressed concerns about their ability to coordinate vulnerability response with their competitors due to regulatory considerations like antitrust laws. While we are not in a position to provide legal advice in that regard, we must acknowledge that these concerns may also arise with current or future AI vendors. The U.S. Federal Trade Commission appears to support CVD practices,[64] so there is cause for optimism that this legal worry might not be as big of a problem as it first appears. Vendors have communicated to us privately that the involvement of a neutral third-party coordinator can help relieve their concerns about antitrust policies.

### Vendors Are Unfamiliar with Existing CVD Processes

Again, this failure mode is not unique to AI, but vendors being unfamiliar with existing CVD processes is common due to the sheer number of novice vendor organizations involved. There is always a learning curve for vendors new to CVD practices.

Misconceptions and misunderstandings about the CVD process can also pose a problem. Many of the concepts and recommendations we write about in the CVD Guide apply here, including assumptions about customer relationships, NDAs, and problems like vendor hostility toward reporters.

As we mentioned in a previous section (*The Vendor Is Unprepared to Receive Reports*), an organization's existing PSIRT capability may not be adequately connected to its AI development capability. However, even if the communication path exists, the potential for a cultural divide between cybersecurity and AI engineering staff remains.

### Traceability of Components Across the Supply Chain Is Lacking

Ultimately, coordination requires the vendor of the affected product(s) to be informed about the problem so it can take action to assess its risk and mitigate or remediate it for its end users. An important question in coordination is, "Does product or service Y contain or otherwise depend on product or service X?" The answer to this question may be necessary to determine who else needs to know about a

---

62 *Contextualizing End-User Needs: How to Measure the Trustworthiness of an AI System* (https://insights.sei.cmu.edu/blog/contextualizing-end-user-needs-how-to-measure-the-trustworthiness-of-an-ai-system/)

63 *OpenAI Illegally Barred Staff from Airing Safety Risks, Whistleblowers Say* (https://www.washingtonpost.com/technology/2024/07/13/openai-safety-risks-whistleblower-sec/)

64 *The NIST Cybersecurity Framework and the FTC* (https://www.ftc.gov/business-guidance/blog/2016/08/nist-cybersecurity-framework-and-ftc)

problem in product or service X. However, often only the vendor of product Y knows whether such a dependency exists, which can lead to a catch-22 when determining who needs to know about X's problem.

Traceability across products is already challenging in traditional software systems, where there might be a hierarchy of libraries, configurations, and other components that make up the software, and many products are incorporated into other products as part of the supply chain. We can illustrate this challenge with a few specific questions:

- What products and services are affected by a vulnerability in Log4J?[65]
- What products and services are potentially affected by the insufficient randomness in Transmission Control Protocol (TCP) sequence numbers?[66]

Although we answered these questions at the time for the specific vulnerabilities we referenced, a new report today would require more effort to determine which vendors' products and services would be affected.

The supply chain for an AI system can be more opaque than traditional software systems, and they produce a wider variety of artifacts than traditional software systems. As we described earlier, AI systems usually include traditional software artifacts as well as AI components whose supply chain includes architectures, algorithms, data sets, models, etc. Each of these components might have a distinct development process and version control. It is not sufficient to know who the library vendors are in an AI system; the provenance and reliability of the data used to train the models, the tuning done along the way, sources of guardrails used to make the system safer, and other information must also be considered.[67]

### Vendors Disagree on How to Reduce Societal Harm

The AI ecosystem is engaged in a broader debate among vendors about how much to be concerned about avoiding potential harms caused by the construction and use of their products. Because there are so many different stakeholders in this debate, it can be hard to disentangle the pressures and agendas that drive conversations about problems that can lead to harm in today's global zeitgeist. It is not surprising that disagreements occur over the best response to these problems, leading to potential gridlock and failure during a coordination event. Trying to harmonize these disparate perspectives about problems of global significance might be the most difficult challenge as evidence and opinions about the value of CVD evolves over time.

---

65 CERT Coordination Center VU#930724*: Apache Log4j Allows Insecure JNDI Lookups* (https://www.kb.cert.org/vuls/id/930724)

66 CERT Coordination Center VU#498440: *Multiple TCP/IP Implementations May Use Statistically Predictable Initial Sequence Numbers* (https://www.kb.cert.org/vuls/id/498440)

67 *CISA Software Must Be Secure by Design, and Artificial Intelligence Is No Exception* (https://www.cisa.gov/news-events/news/software-must-be-secure-design-and-artificial-intelligence-no-exception)

## 6. Fix & Mitigation Development

The *Fix & Mitigation Development* step is necessary because it is not always possible to fully remediate every identified problem.[68] Fixes are always preferred, of course, but when a problem cannot be remediated, a mitigation may have to suffice. This step fails when a problem has been reported, it is valid, it has been prioritized, it is being coordinated, and yet it is either not fully fixed or its mitigations can be trivially bypassed.

In the following sections, we describe the possible failure modes encountered during the *Fix & Mitigation Development* step.

### The Root Cause of a Problem Cannot Be Isolated or Localized in Code or Data

Problems in AI systems can occur in the algorithm or logical construct, in a protocol or specification, in an implementation (e.g., training method, training data, model production process), or in its configuration. It is not always easy to "draw a circle around" a specific component and say, "The root cause of the problem is inside this circle." If you cannot isolate the root cause of a problem, it can be even more difficult to fix.

Failure of a fix to reach the root cause of a problem may lead to harmful regressions as new attack paths and exploits are discovered that trigger an unresolved problem. These "band-aid" mitigations or fixes lead not only to ongoing risk and potential harm but also reduced trust in the vendor's ability to reliably deal with similar problems in the future.

The AI vulnerability landscape is so young that we expect that many more years of research and operational experience will be necessary to gain better insight into how to measure progress in the field of patch reliability. (Currently, measuring progress is mostly driven by the field of automated program repair.[69, 70])

---

[68] Here we follow the definitions given in *DoD Instruction 8531.01 DoD Vulnerability Management* (https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/853101p.pdf): "Remediation occurs when the vulnerability is eliminated or removed. Mitigation occurs when the impact of the vulnerability is decreased without reducing or eliminating the vulnerability."

[69] *Automated Program Repair* (https://dl.acm.org/doi/abs/10.1145/3318162)

[70] *Automated Program Repair, What Is It Good For? Not Absolutely Nothing*! (https://dl.acm.org/doi/abs/10.1145/3597503.3639095)

### Stochastic Behavior Conflicts with Binary Policies

In many AI systems, the expected performance is inherently probabilistic. For example, in an image recognition system, a certain threshold of false positives or false negatives may be acceptable. However, traditional cybersecurity policies tend to frame compliance as binary: The system state either complies, or it does not. The code allows the SQL injection to succeed, or it does not.

When an AI system can only achieve compliance to a binary policy with some probability based on its stochastic behavior, we describe this as *stochastic compliance to a binary policy.*[71] A reasonable response is to adjust the policy to be defined in terms of acceptable thresholds of system behavior, which we refer to as a *stochastic policy*. It can be challenging to determine whether a problem has been fixed when, in fact, the behavior has just been tuned to keep the undesired behavior within some threshold.

We are skeptical about claims of stochastic compliance with stochastic policies in the presence of malicious actors. Most stochastic compliance assumes some reasonably well-understood distribution across the input data that the system is likely to encounter. While this assumption may be reasonable for an otherwise isolated system that is subject only to measurable environmental inputs, any adversary has some leeway to choose the parameters of their attack, and they are not always constrained to stay within reasonable bounds.

### Non-Regression Is Not Ensured

Over time, the field of software engineering has developed methodologies to ensure that software has not regressed to a previously known bad state. Methods such as unit testing, regression testing, and code coverage analysis ensure that, upon release, software does not break its existing functionality or regress to a known bad state. These methods are still applicable for the software portions of an AI-based system.

However, there are no such methodologies in deep neural networks. Even given the same code, data, and algorithm, the stochastic nature of ML will produce a different model during training each time. Also, the ability to handle every possible input means that, as the dimensionality of the input grows, the ability to test all possible inputs for system faults becomes non-computable.

Imagine there is a case we want to test that is exceedingly difficult due to the complexity of ML input types. When approaching this problem, the ML field will try to ensure that all inputs within a certain distance are handled correctly. However, as with all thresholding, there is no way to know if the inputs just outside of this range produce a bad state. So, for testing, the larger the distance becomes, the more difficult it is to ensure correctness.

---

[71] *Stochastic compliance* means that the system is considered to be in compliance if its behavior is consistent with the policy some fraction of the time. There is also the possibility that the policy itself is specified as a performance threshold, which we would characterize as *stochastic compliance to a stochastic policy*.

### A Problem Might Have to Be Mitigated vs. Remediated

It is not always possible to remove a problem entirely. In those cases, a workaround or mitigation may be necessary. A characteristic of an AI system is that the performance of the model measured against its validation data is reproducible only for that validation data set and will not be exactly replicated for a different data set. Model behavior in deployment can change as the world changes over time, so a problem may be introduced or reintroduced completely outside the control of the vendor or user. Therefore, any mitigations should inherently be viewed as fragile.

However, deploying models with so-called additional software that allows for introspection, monitoring, or warning about undesired behavior might backfire by increasing the attack surface. Developing a test suite for these mitigations requires careful construction and can be difficult to achieve in practice because high-fidelity examples of failure cases are not always available. The open-endedness of the adversarial space for these software products also poses a challenge for developing a test suite that searches enough to detect attack modes that have not been previously observed. There is active research into constraining the attack surface of an AI model as well as its training data and operational data. However, these techniques do not apply universally among AI model types, and they cannot always be implemented due to constraints from the domain.[72]

There is a separate question about the modularity and interchangeability of AI models in a software system. A remediation could replace one class of AI model with another that has better observability or introspection. However, the interchangeability of AI models is constrained by factors related to the larger context of the problem, the data, the domain, and other software constraints. A generative language model is not an exchangeable solution for an image recognition problem, for example.

### Solution Sufficiency Is Not Agreed To

The kinds of problems in AI systems that are likely to require coordinated response typically extend well beyond the usual CIA impacts of traditional cybersecurity vulnerability response. This is not exclusively an AI problem; it is more driven by recognizing that the impacts of software behaviors that violate expectations can reach far beyond the control flow of a program in a CPU. The challenge is that the expectations that exist are unclear, and what a sufficient "mitigation" or "remediation" looks like is also unclear.

We have already observed vendors that mitigate reported AI issues by retraining models to avoid the behavior observed given the specific reported inputs. Too often, it is trivial to adapt the adversarial example to reproduce the undesired behavior after the mitigation is applied. This approach is equivalent to "patching the proof of concept" instead of fixing the root cause of a problem, which is a known

---

[72] *NIST AI 100-2e2023: Adversarial Machine Learning* (https://doi.org/10.6028/NIST.AI.100-2e2023)

issue for some vendors in the CVD process.[73] The result is, at best, an unstable arms race where every mitigation is countered by a new demonstration of the problem.

## 7. Publication

The optional *Publication* step brings awareness of the problem to the global community. We can segment this community into a few relevant stakeholder groups:

1. current and potential future customers (often including deployers and owners/operators) of the product(s) and/or service(s) affected by the problem
2. consumers who are known **not** to be affected but who also need to know for certain that they do not need to take any special action to be protected
3. security product and service providers who develop tools and methods for detection and incident response
4. data aggregators who collect and provide data sets that enable longitudinal research of trends in system security problems over time
5. governmental bodies responsible for protecting critical infrastructures not under their direct control
6. other vendors that were not made aware of problems that may affect them or issues in prior steps of the CVD process

This step succeeds when information about problems and their well-tested mitigations and fixes are known to these stakeholders. It fails when this information is not made available to stakeholders in a usable form and in a timely fashion.

In the following sections, we describe the possible failure modes encountered during the *Publication* step.

### A CVE ID Is Not Assigned

From the system deployer (i.e., owner, user) perspective, many vulnerability response processes depend on a CVE identifier (ID) assignment as the triggering event to initiate their response process. Numerous organizations have a vulnerability response process that is effectively unaware of any problem that does not have a CVE ID assigned to it.

The CVE assignment process relies on the CVE Numbering Authorities (CNAs) that are tied as closely as possible to the vendor or parties responsible for fixing a vulnerability when it is identified. However, for AI systems, there may be multiple overlapping CNAs that contribute to the data used to

[73] *CERT® Guide to Coordinated Vulnerability Disclosure: Response Process* (https://certcc.github.io/CERT-Guide-to-CVD/tutorials/response_process/vendor/#fix-the-problem)

train and test the models that AI systems rely on. This reliance on overlapping CNAs could mean that even a problem with a single model could quickly turn into a multiparty coordination scenario.

Not every AI system problem that requires coordinated disclosure will be appropriate or in scope for CVE ID assignment.[74, 75] AI issue identifiers are not limited to CVE IDs. All that is needed are tracking numbers to recognize task units of work. When CVE IDs are not assigned, it is impossible to tell whether a problem has or has not been addressed because it cannot be tracked. However, Vulnerability Disclosure Program (VDP) report IDs are another way to track problems.

### NDAs Impede Transparency

In our discussion of *Coordination* failure modes, we mentioned how NDAs can be used and misused. However, NDAs are an issue that affects publication as well. NDAs can limit the participation of finders, coordinators, vendors, or other participants in the CVD process. If these participants are unable to fully explain problems to their stakeholders (including the public), then the public's ability to make informed choices about the privacy, safety, and security of AI-based products and services can be blocked or limited. Likewise, being able to recognize and analyze trends in problem resolutions can be similarly impeded or left incomplete.

### Components Are Hidden Inside Products and Services

As we described in the *Reporting* step, it can be difficult to tell who the responsible parties are for a particular problem due to the opacity of the supply chain. This issue arises again in the *Publication* step because it is not always obvious to a stakeholder using an AI-enabled product that it is affected by a vulnerability in one of its subcomponents. SBOMs might enable this connection to be detected, but the human communication aspects of recognizing the relevance of a reported problem in a component to the use of a not-obviously-related product has more to do with messaging than technical data.

### Publishing Failures in AI Systems Is Viewed as a Knowledge-Building Exercise

There is a case to be made for publishing AI system failures to provide information for future threats and vulnerabilities that extend beyond the immediate operational imperatives driven by current risks and threats. It has been our experience that it is valuable to write about all the different ways an emerging technology can fail and be misused by attackers if not properly mitigated or fixed. This information builds a foundation of knowledge that the software engineering community can learn from when building the next generation of products and services. This information can also help developers (1) shift security-minded design, implementation, and testing earlier in the development cycle; (2) learn from the mistakes of others; (3) prevent vulnerabilities in AI-engineered systems; and (4) benefit from concrete examples of how they are failing today.

---

74 *On Managing Vulnerabilities in AI/ML Systems* (https://dl.acm.org/doi/10.1145/3442167.3442177)

75 *CVE and AI-Related Vulnerabilities* (https://medium.com/@cve_program/cve-and-ai-related-vulnerabilities-3ae6ad8ae81b)

## 8. Fix & Mitigation Deployment

Regardless of whether the *Publication* step occurs, the next step in our process model is *Fix & Mitigation Deployment*. This step succeeds when fixes or adequate mitigations exist and are deployed. It fails when fixes or adequate mitigations have been created and are available yet are not deployed to the affected systems.

In the following sections, we describe the possible failure modes encountered during the *Fix & Mitigation Deployment* step.

### The Deployer Is Unaware of the Problem

If the deployer does not know about the problem or the availability of a fix, it cannot remediate the systems it is responsible for. A deployer might be unaware of this information for the following reasons:

1. They might not be paying adequate attention to security information about the products and services they depend on.
2. The *Publication* step resulted in information shared in channels that the deployer does not monitor.

Communication involves at least two parties: a sender and a receiver. In this case, the deployer must actually *receive* the message. The vendor, coordinator, or reporter *sending* the message is a necessary but not sufficient condition.

### The Deployer Does Not Prioritize Deployment

The deployer may be aware of a fix or mitigation, but it might not prioritize the deployment of that fix or mitigation. Commonly used cybersecurity prioritization tools (e.g., Common Vulnerability Scoring System [CVSS][76]) are often insufficient for assessing the impact of problems in AI systems, which can be more diffuse than traditional cybersecurity vulnerabilities. For example, CVSS focuses on the traditional CIA impacts, but as we already described, these impacts are not the only relevant concerns for AI systems. More broadly, these impacts are not the only concerns in traditional software either, which is increasingly evident with the emergence of societal-scale systems.[77]

---

[76] *FIRST: Common Vulnerability Scoring System SIG* (https://www.first.org/cvss/)

[77] *5 Issues to Consider When Engineering Societal-Scale Systems* (https://insights.sei.cmu.edu/blog/5-issues-to-consider-when-engineering-societal-scale-systems/)

### Affected Versions and Fixed Versions Are Not Identified

While the software in an AI system can be tracked, typically by using existing package management and versioning mechanisms, this tracking rarely transfers to the model and data the system might use. The ML community has not defined and adopted best practices for model versions, checksums, or cryptographic signatures that allow analysts to determine where their impact started and where it was remediated. The absence of best practices also makes it difficult for a consumer of a model to determine if they are affected when they might only have the date the model file was downloaded and no additional logging information or proof of what model was acquired, from where, and what its version was.

Similarly, version identity is a problem for data. For example, when ImageNet[78] updates its popular dataset for ML, how do stakeholders know what version of the data set their model uses? It gets even more complicated in the world of LLMs, where data sets like LAION-400M,[79] which are popularly used to train these models, are completely distributed and only an index is maintained. Both the data sets and the distributed systems they rely on can change without the data set maintainers knowing. How do stakeholders know which version of each of these components their model was trained on? Knowing these versions is especially important since a model can be poisoned by altering a small number of sites a data set relies on.[80]

Given the challenges already described about identifying and isolating affected components, this challenge will likely be a recurring issue for deployers of AI systems for some time. How deployers version their models, or combinations of models and data, is inconsistent and still developing. In systems with highly volatile models, versions might be deployed for only a short time; thus, they can be hard to track or capture. In the worst cases, versions of models and data are not documented or reproducible at all.

---

78 *ImageNet* (https://www.image-net.org)

79 *LAION-400-Million Open Dataset* (https://laion.ai/blog/laion-400-open-dataset/)

80 *Poisoning Web-Scale Training Datasets is Practical* (https://arxiv.org/pdf/2302.10149)

### The Update Process Is Insecure

Deployment should not expose the deployer to additional risk. In many cases, the update process for a model is to download a new version from a model aggregator (e.g., Hugging Face).[81] This download can be done as part of a build process, the installation process, or even at runtime. While this method of providing updates is not much different from dynamic package management or mechanisms used by frameworks such as Python's *pip* or Node's *npm*, we have observed many AI systems that do not incorporate attestation mechanisms (e.g., cryptographic signature verification) prior to loading the downloaded models, data, or code.[82]

Furthermore, deserialization issues, such as those described in VU#253266: *Keras 2 Lambda Layers Allow Arbitrary Code Injection in TensorFlow Models,*[83] have been observed in numerous ML frameworks, and they seem likely to happen again.

## 9. Monitoring & Detection

The *Monitoring & Detection* step succeeds when the coordinating parties are keeping watch and can notice when problems arise after fix availability, publication, and deployment. Problem examples might include incomplete or inadequate mitigations, exploit publication, attack observations, and the like. This step succeeds when there are sufficient processes in place to identify similar events when they occur. This step fails when those events pass unnoticed.

In the following sections, we describe the possible failure modes encountered during the *Monitoring & Detection* step.

### No Monitoring Occurs

The simplest failure mode occurs when no follow-up monitoring occurs. Stakeholders cannot see what they are not looking for. In an AI system, follow-up monitoring might require refactoring the software to support introspection into the model or its data.[84] For a vulnerability detected when the model is implemented, this refactoring may be more or less context specific to a system that depends on that model implementation. So, coordinating and communicating about what is required to fix the problem may include stakeholders that are not typically involved in this process.

---

81 Hugging Face (https://huggingface.co)

82 While we are here, we will also throw a bombastic side eye at `curl | bash` and its variations.

83 CERT Coordination Center VU#253266: *Keras 2 Lambda Layers Allow Arbitrary Code Injection in TensorFlow Models* (https://kb.cert.org/vuls/id/253266)

84 As we have noted previously, introspection and sensemaking of AI components at runtime is an area of active research, so the near-term practicality of such a refactoring is questionable.

### Scanning Tool Bias Exists

Vulnerabilities that have CVE IDs show up in scanners, but those without CVE IDs show up less often. CVE coverage of AI systems is quite limited, so scanning tools may not find known vulnerabilities. See the *Publication* step to learn more about ID assignment.

### Vulnerability Management Does Not Handle Mitigation Well

CVD feeds into vulnerability management (VM), which, in turn, expects patches. These patches are typically viewed in binary terms, either applied or not, and are usually considered to be remediations (i.e., once applied, the problem is removed). Patches are not generally seen as mitigations (i.e., risk is reduced, but the problem is not completely removed).

Many AI "patches" serve as mitigations rather than remediations. As a result, we lack effective ways to discuss these patches within the context of VM metrics, such as the "percentage of patches applied." Because VM metrics cannot be used in the same way as they can in traditional vulnerabilities, any metrics used can lead to a false sense of accomplishment since people may believe they are performing well when they are measuring with the wrong metrics.

### Reports of Inadequate Fixes or Mitigations Are Not Resolved

Sometimes stakeholders think that vulnerabilities are resolved, but it turns out that a fix was incomplete or otherwise inadequate. When this occurs, it is important that the *Coordination* step continues until the new issues are resolved. If the *Coordination* step does not continue, the *Monitoring* step will fail to achieve the goal of ensuring that fixes are adequate and sufficient.

### The Publication of an Exploit Goes Unnoticed

During the *Coordination* step, it is always possible that some other party has also discovered the problem. An exploit can be released that is completely independent of an ongoing CVD process. When this occurs, the urgency of the CVD case usually increases. Exploits that go unnoticed can hinder the *Coordination* step.

### Attacks Go Unnoticed

Similarly, attacks may occur during or after a CVD case is coordinated. These attacks can go unnoticed if signatures or other detection methods are not developed. Undeveloped detection methods can be of particular concern for AI systems where the output of the system is difficult to validate. In these cases, it is important to have adequate means to inspect model behavior to detect evidence of problems; unfortunately, monitoring models' behavior in deployment is inconsistent.

## 10. Process Improvement

The *Process Improvement* step is successful when insights from the execution of the process are used to enhance future development and coordination practices. These insights can prevent future vulnerabilities or help manage existing ones. Feedback can take the form of root cause analysis that leads to enhanced testing protocols, additional procedural checkpoints, or improved threat models. This step fails if the feedback loop is not established.

In the following sections, we describe the possible failure modes encountered during the *Process Improvement* step.

### Root Cause Analysis Is Not Conducted

Understanding the origin of a problem is crucial to rectify it. Identifying the specific system feature where the problem occurred is a key part of root cause analysis. However, identifying the flaw is just the beginning of adapting the process to prevent similar future issues. A vulnerability often indicates a process that inherently allows its creation. As Charles Perrow argues in his book *Normal Accidents: Living with High Risk Technologies*,[85] organizational and management factors significantly contribute to technological disasters. Processes that are susceptible to common errors and system failures require improvement.

### Root Cause Analysis Does Not Lead to Process Changes

A root cause analysis can pinpoint the specifics that led to a vulnerability and suggest process improvements to mitigate similar future issues. However, if these insights are not integrated into the process, there is no chance of improvement.

### Root Cause Analysis Leads to Inadequate Process Changes

Knowing the root cause and making changes is not enough. It is essential to verify that the changes made had the desired effect. Superficial changes that fail to address underlying issues are ineffective.

---

[85] *Normal Accidents: Living with High Risk Technologies (Updated Edition)* (https://press.princeton.edu/books/paperback/9780691004129/normal-accidents)

### AI Development Ignores Good Software Engineering Practices

Software engineering practices have adapted over time through by adopting new practices and lessons from past failures. Secure development practices[86] and DevSecOps,[87] for example, emerged to address the need for enhanced security, faster development cycles, automation, and cross-functional collaboration across the software lifecycle. However, in some organizations, AI development operates separately from software engineering, either due to a lack of internal expertise or disconnected efforts. This disconnect can lead to repeating past mistakes when lessons learned from previous approaches are not applied effectively.

### Software Engineering and Cybersecurity Ignore AI Development

Software engineering and cybersecurity experts sometimes underestimate how their expertise applies to AI systems development. Gaps often emerge due to misaligned goals, different development paces, and a lack of shared concepts and vocabulary. However, the solution is not simply a matter of applying current practices to AI systems. Software engineering and cybersecurity practices and methods must also adapt to the specific needs of AI systems engineering.[88]

## 11. Creation (of the Next Vulnerability)

We began our process journey with the *Discovery* step, observing that the existence of vulnerabilities leads to the need for the CVD process. Success in this step occurs if no further vulnerabilities are created. In fact, we maintain that there is *always another* vulnerability,[89] so the best process improvement we can hope for is to reduce how often new vulnerabilities are introduced by avoiding past mistakes. As a result of this assumption, we invoke Adam Savage's maxim: "Failure is always an option."

In the following sections, we describe the possible failure modes encountered during the *Creation* step.

### Threat Models Are Naïve

Rarely are those who develop software systems aware of the creativity and ingenuity that a motivated adversary will use to find a vulnerability in a system. Threat models are an important part of understanding the threats a system should be secured against. However, when threat models are found in research, they tend to be very narrowly defined and do not represent threats found in the real word.

---

86 *Secure Development* (https://www.sei.cmu.edu/our-work/secure-development/)

87 *DevSecOps* (https://www.sei.cmu.edu/our-work/devsecops/)

88 *Creating Transformative and Trustworthy AI Systems Requires a Community Effort* (https://insights.sei.cmu.edu/blog/creating-transformative-and-trustworthy-ai-systems-requires-a-community-effort/)

89 *An Analysis of How Many Undiscovered Vulnerabilities Remain in Information Systems* (https://insights.sei.cmu.edu/library/an-analysis-of-how-many-undiscovered-vulnerabilities-remain-in-information-systems/)

Another problem is that when people perform security evaluations on AI systems, these evaluations might be too simplistic. As stated in the paper *On Evaluating Adversarial Robustness*,[90]

> *After a specific threat model has been defined, the remainder of the evaluation focuses on adaptive adversaries which are adapted to the specific details of the defense and attempt to invalidate the robustness claims that are made.*

With this in mind, we need to ensure that systems have quality threat models and evaluations are performed to ensure that vulnerabilities in the system are limited.

### The Security Policy Is Unclear

Implicit policies are based on individual expectations and societal norms. However, with new and rapidly developing technology, we do not know what is possible, impossible, or reasonable to expect. So, our expectations are often improperly formed and underspecified. Nobody writes security policies to prohibit things they do not realize are even possible. (This is not the same as saying they are *impossible*.) The result is that we often must "get burned" by something before we even realize it was a problem.

### The Use of Libraries Is Naïve

Dependency security is a critical part of understanding software, particularly in environments like AI software, where the software is generated dynamically, and the complexity exists both in developing the AI software and its operation in an environment. For example, the YOLOv5 software that is used for AI-based object detection downloads about 88 additional libraries and another 10 or more libraries when it is operationally used with models and additional capabilities. Trail of Bits found 10 vulnerabilities in YOLOv7 that were largely based on the insecure use of dependencies.[91] This style of development and operations introduces a new dynamic and exposes our lack of understanding of the software composition of operational AI software. Current software composition analysis (SCA) tools cannot sufficiently find libraries that are vulnerable and how their use in certain scenarios can introduce a vulnerability.

Apart from unknown or undetected software dependencies, the insecure use of libraries is common in AI software. For example, the secure use of Python Pickle[92] is not followed in many AI-based software development efforts.[93]

---

90 *On Evaluating Adversarial Robustness* (https://arxiv.org/pdf/1902.06705)

91 *Assessing the Security Posture of a Widely Used Vision Model: YOLOv7* (https://blog.trailofbits.com/2023/11/15/assessing-the-security-posture-of-a-widely-used-vision-model-yolov7/)

92 *Understanding Python Pickling and How to Use It Securely* (https://www.synopsys.com/blogs/software-security/python-pickling.html)

93 *Pickle Serialization in Data Science: A Ticking Time Bomb* (https://www.robustintelligence.com/blog-posts/pickle-serialization-in-data-science-a-ticking-time-bomb)

### The Use of Data Is Naïve

The separation of data and code has been one of the important security concerns in computing.[94] The principle is quite simple; code (i.e., instructions for some processor) should be kept distinct from data to prevent untrusted code from being executed when masked as data. This is a caution that is not typically used in the build and operations of AI software. AI software depends on the data and further parts of the data embedded in models (discussed next) to create solutions and services.

In traditional computing, data is needed to test and verify algorithms but not to create or modify the algorithm itself. However, today's AI software uses data in ways that move beyond this traditional approach, putting data in the critical path and potential path for vulnerabilities. The idea of training data is common in AI software; this training data is frequently inadequately documented and provided in its raw form to many consumers and users. This data requires proper assessment as part of software composition analysis. The information gathered should also be tracked in more detail than it is today, including versions and capabilities, which are currently not being gathered and tracked for AI software data. How data documentation integrates into versioning is also an area of active research and inconsistent implementation.

### The Use of Models Is Naïve

IBM defines an AI model as "a program that has been trained on a set of data to recognize certain patterns or make certain decisions without further human intervention."[95] This idea that an AI model is software is not well understood by the AI software community. Similar to data, models remain unaccounted for and unknown in many cases, which allows vulnerabilities to either exist or be introduced without clarity or awareness of the AI software developer and operator. In some cases, AI software developers allow models to perform arbitrary tasks (e.g., execute arbitrary code that may not be relevant to the current process). Currently, there are no reverse engineering tools that provide a way to audit the models and provide visibility into the model's capability. Therefore, a need exists to both develop tools and determine what techniques exist to provide that visibility, including gaps in current capabilities. Consumers and end users may lack tools to restrict the model to perform only the specific tasks relevant to the current problem being worked on by using AI software.

### Computing Architectures Lack Security Features

GPUs were traditionally used for displaying and performing high-speed operations for controlling the screen. However, the exceptional general-purpose parallel processing of GPUs has made them integral to modern computing, especially in AI-based software. This paradigm of moving GPUs into a massive parallel-computing engine has exploded in the past few years. It is an unfortunate theme in

---

94 *Separation of Data and Code* (https://wiki.c2.com/?SeparationOfDataAndCode)

95 *IBM: What Is an AI Model*? (https://www.ibm.com/topics/ai-model)

cybersecurity that security is often only considered after the fact. Much of CPU security that has evolved in the past few years has not been applied to GPUs.

For example, CPUs have been maturing in providing memory isolation that allows processes and users to access distinct memory regions and maintain the separation of data. GPUs do not have any such native ability to isolate memory regions, and problems (e.g., side-channel problem) are only made worse with the vast amounts of data that GPUs can access in memory[96] without performing any routine data cleanup. A recent vulnerability dubbed LeftoverLocals demonstrated another issue that disclosed the "lack of memory isolation in GPU kernels"[97] (i.e., small snippets of GPU code). As the AI stack of software evolves, the problems are likely to become visible in some of the essential hardware parts of AI solutions.

### The Supply Chain Is Complex (and Possibly Immature)

All the earlier failure modes relate to huge supply-chain issues since the software stack is deep. The supply chain starts with the hardware vendors that provide hardware capabilities and application programming interface (API) libraries and is followed by multiple levels of software solutions that embed components like a Matryoshka doll with hidden layers of unaccounted software. Some software from the supply chain is also dynamically downloaded and used without the user's knowledge or interaction. As these libraries, models, and data are downloaded dynamically, the traditional problems (e.g., domain squatting, download hijacking) can further complicate proper AI software vulnerability assessments, reporting, and coordination.

---

96 *Modern GPUs Vulnerable to New GPU.zip Side-Channel Attack* (https://www.bleepingcomputer.com/news/security/modern-gpus-vulnerable-to-new-gpuzip-side-channel-attack/)

97 CERT Coordination Center VU#446598: *GPU Kernel Implementations Susceptible to Memory Leak* (https://kb.cert.org/vuls/id/446598)

# Conclusion

We in the CERT/CC have been coordinating the disclosure and response of software vulnerabilities since 1988. In that time, many IT changes have come and gone, and the coordination has continued to evolve and adapt. The emergence of AI systems represents an expanded need for coordination to resolve system problems that might otherwise lead to harm and loss in the face of adversarial use.

In this paper, we highlighted a variety of ways that the CVD process and its participants and stakeholders will need to adapt and address the concerns that come with expanding AI systems and their dependencies.

## Four Key Takeaways

We conclude this paper with four key takeaways:

- AI is software.
- Software engineering matters, even in AI systems.
- Coordination and disclosure are important parts of CVD.
- The focus should be on fixing problems rather than arguing over definitions.

### AI Is Software

Yes, AI is a different way of thinking about software. Collecting and cleaning data and training models does not resemble the software development that came before it. However, AI systems are built using software that runs on hardware, and there is no escaping that the problems we expect to encounter as more AI components are incorporated into information processing systems will be *in addition to*, not *in lieu of* the cybersecurity concerns that are rampant in the traditional software that preceded it.

### Software Engineering Matters, Even in AI Systems

A great deal of prior work in software engineering has been invested into ensuring that certain quality attributes are present in both the *products* of the development effort as well as the *process* that produces those products. These quality attributes—reliability, robustness, scalability, performance, maintainability, adaptability, testability, debuggability, security, privacy, safety, fairness, ethics, and transparency—are no less important in the context of AI systems. We are increasingly aware that, as the reach and influence of software grow, so does the responsibility to ensure that it does not expose those who depend on it to unnecessary risk of harm.

### Coordination and Disclosure Are Important Parts of CVD

*Coordination* is the most important part of CVD. When one individual, organization, or entity knows about a problem and another individual, organization, or entity can fix that problem, there is a need to coordinate.

*Disclosure* is a close second. Informed consumers make better choices. Ultimately, if a stakeholder is unaware of the flaws in a system they own or are considering acquiring, they cannot make informed decisions about their continued use or acquisition of it. Publishing information about problems and ensuring the transparency of associated risks and their potential mitigations are important foundational processes that allow stakeholders to balance their own risks.

*Vulnerability* is the least important part of CVD. Asking, "Is *this* an AI vulnerability?" is not the same question as, "Do we need to do something about *this* undesired behavior in *this* AI system?" It does not matter as much whether the problem meets any given definition of vulnerability. With AI systems, our broad definition of "vulnerability" is sufficient to capture the range of potential problems that can arise. What matters is whether known problems are fixed before they become incidents. What is still developing is a process by which those who know can coordinate with those who can do something about it to make the world safer for everyone.

### The Focus Should Be on Fixing Problems Rather Than Arguing Over Definitions

In 1976, a riverboat pilot on the Ohio River going under the I-79 bridge on Neville Island spotted a crack in a welded beam. He subsequently reported it to the authorities, and a potential disaster was averted.[98] Could an adversary have exploited this flaw to cause significant damage? Yes, of course. Was that likely at the time? Probably not. However, does the absence of such an adversarial exploitation scenario imply that the problem was not worth remediating? Hardly. A problem worth fixing is still a problem worth fixing, and coordination is necessary when those who know about the problem are not the ones who can fix it.

## Final Thoughts

In this paper, we described several challenges faced by those who are trying to resolve problems in AI systems: finders, reporters, vendors, coordinators, and deployers alike. The CVD process must continue to evolve to meet the changing landscape of stakeholders, their needs and expectations, and the systems and technologies they rely on.

---

[98] *The Award-Winning Bridge That Failed* (http://pghbridges.com/articles/pressroto_nevilleisland_i79/index.htm)

# Legal Markings



This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.



DM24-1040

# Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

**Phone**: 412/268.5800 | 888.201.4479
**Web**: www.sei.cmu.edu
**Email**: info@sei.cmu.edu