On managing vulnerabilities in AI/ML systems

Jonathan M. Spring
jspringATseidotcmudotedu
CERT® Coordination Center
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA

April Galyardt
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA

Allen D. Householder
CERT® Coordination Center
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA

Nathan VanHoudnos
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

This paper explores how the current paradigm of vulnerability management might adapt to include machine learning systems through a thought experiment: what if flaws in machine learning (ML) were assigned Common Vulnerabilities and Exposures (CVE) identifiers (CVE-IDs)? We consider both ML algorithms and model objects. The hypothetical scenario is structured around exploring the changes to the six areas of vulnerability management: discovery, report intake, analysis, coordination, disclosure, and response. While algorithm flaws are well-known in academic research community, there is no apparent clear line of communication between this research community and the operational communities that deploy and manage systems that use ML. The thought experiments identify some ways in which CVE-IDs may establish some useful lines of communication between these two communities. In particular, it would start to introduce the research community to operational security concepts, which appears to be a gap left by existing efforts.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; • **Software and its engineering** → *Maintaining software*; • **Security and privacy** → **Vulnerability management**.

KEYWORDS

vulnerability management, machine learning, CVE-ID, prioritization

ACM Reference Format:
Jonathan M. Spring, April Galyardt, Allen D. Householder, and Nathan VanHoudnos. 2020. On managing vulnerabilities in AI/ML systems. In *New Security Paradigms Workshop 2020 (NSPW '20), October 26–29, 2020, Online, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3442167.3442177

NSPW '20, October 26–29, 2020, Online, USA

ACM ISBN 978-1-4503-8995-2/20/10…$15.00
https://doi.org/10.1145/3442167.3442177

1 INTRODUCTION

The topic of this paper is more "security for automated reasoning" and less "automated reasoning for security." We will introduce the questions that need to be answered in order to adapt existing vulnerability management practices to support automated reasoning systems. We suggest answers to some of the questions, but some are quite thorny questions that may require a new paradigm of either vulnerability management, development of automated reasoning systems, or both.

First, some definitions. We follow the CERT® Coordination Center (CERT/CC) definition of vulnerability: "a set of conditions or behaviors that allows the violation of an explicit or implicit security policy" [23, §1.2]. We will follow Spring et al. [51] and define ML as "a set of statistical tools that analyze data to infer relationships and patterns. Ideally, the relationships and patterns inferred by ML will lead to a useful model of the object or phenomenon that the data describes," and define artificial intelligence (AI) as "a software agent that takes actions based on its environment." To be concrete, this paper will focus on vulnerability management for just ML-enabled systems.

One practical way to think of security services for an ML system is via the set of services a Computer Security Incident Response Team (CSIRT) might provide, which is produced by Forum of Incident Response and Security Teams (FIRST) and documented by Benetis et al. [2]. A complete risk management and security perspective on ML would include more than the CSIRT services framework. However, we will work from the assertion that to manage the security of an ML-enabled system, all CSIRT services will need to be able to handle ML systems.

Specifically, of the CSIRT services, we are carving out just vulnerability management for discussion. The other services, as well as wider issues such as risk management, all have challenges as well, but we leave them as future work.

Vulnerability management includes six services [2, §7]:

- Vulnerability discovery / research
- Vulnerability report intake
- Vulnerability analysis
- Vulnerability coordination
- Vulnerability disclosure
- Vulnerability response

These areas cover a wide range. They span the interface between software developers, software users, security teams, and people who find flaws in software. There is national infrastructure in multiple countries dedicated to support these services and facilitate communication. For example, both the United States and the People's Republic of China have National Vulnerability Databases (NVDs). FIRST, a global body, provides one definition of how to score the severity of vulnerabilities, Common Vulnerability Scoring System (CVSS). The CVE scheme is designed to assist such cataloging and ranking efforts.

This paper's goal is to facilitate the creation of a trading zone between ML engineers, software architects, and security practitioners. Any trading zone requires a shared language, whether it is a physical or intellectual trading zone [15]. All participants in a trading zone want something of value they can take back to their respective communities. These items of value are often "boundary objects," which mark boundaries by being recognizable and functional in both cultures in the trading zone. Rawls and Mann [41] states that the Mitre Corporation (MITRE) produces identifiers, such as Common Vulnerabilities and Exposures (CVE) identifiers (CVE-IDs), in part for their value to trading zones as boundary objects. This background makes CVE-IDs an attractive and useful focal point for our thought experiments.

We will organize our exploration of a new paradigm for ML security around one hypothetical – what if flaws in ML systems were assigned CVE-IDs? Sections 4 and 6 do the main work on exploring the thought experiment. Before we can answer this question, we first lay some background on the current ways of identifying software vulnerabilities in Section 2. Section 3 will provide background on the current state of adversarial attacks on ML algorithms. Section 4 then steps through each of the services areas of vulnerability management to explore the impact of giving ML *algorithm* flaws CVE-IDs. Section 5 explores the ML algorithm thought experiment from the perspective of CVE Numbering Authoritys (CNAs). Section 6 steps through each of the service areas to explore the impact of giving ML *model object* flaws CVE-IDs.

2 VULNERABILITY BACKGROUND

Expanding on the definition of *vulnerability* cited in Section 1, a vulnerability is "a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes" [23, §1.2]. This definition is useful for the purposes of this paper for clarity, but CERT/CC is also a CNA, so it bears on the ensuing discussion as well.

An organization has a variety of options when responding to a vulnerability. A fix (a.k.a remediation) is usually defined as a deploying a patch that removes the vulnerable code or retiring the vulnerable system. A mitigation reduces the impact of a vulnerability without removing the vulnerable code. Example mitigations include adding network segmentation or input and traffic filtering that make it harder to exploit the vulnerability. Managing vulnerabilities in ML systems will use a combination of remediation and mitigation, just as any other sector.

There are two common axes that help distinguish vulnerabilities in modern security practice. One is within vulnerability identification. The second is level of abstraction of the vulnerable product. Sections 2.1 and 2.2, respectively, discuss these levels. Section 2.3 summarizes background the the CVE project.

2.1 Vulnerability Identification

Vulnerability identification and classification spans from scanning individual systems to organizing vulnerabilities into categories to facilitate better programming principles. A number of vulnerability identification methods exist. CVE is perhaps the most widely known, but there are others.

In increasing broadness along the identification and classification axis, we have:

- Instance of a vulnerable product
- Vulnerability in a product (e.g., CVE-ID, *VU#*)
- Category of which a vulnerability is an example (e.g., Common Weakness Enumeration (CWE), Open Web Application Security Project (OWASP))

2.1.1 Instances of Vulnerable Products. On the very specific end of this spectrum, we have instances of a vulnerable product. An *instance* is a specific computer or service that uses a vulnerable product. When an organization scans the systems it owns to perform asset management, it will find instances of a vulnerability. Instances are often tagged as the association of a host or system identifier accompanied by the ID of the vulnerability of which they are an instance.[1] Instances may also be called findings or sightings.

2.1.2 Vulnerable Products. Moving up from instances, we find vulnerable products. The practitioner community's expectation is that a product is some sort of artificial information processing system. This definition is vague because a vulnerable product is usually the thing that security practitioners say "has" the vulnerability, as defined above. Since a vulnerability may be introduced by a software defect, configuration decision, design decision, system interaction, or environmental mismatch, the "product" that has a vulnerability cannot be constrained much. Section 2.2 will discuss different categories of vulnerable products.

CVE-IDs are most closely associated with products. Section 2.3 will detail the CVE program.

CERT/CC publishes Vulnerability Notes using the *VU#* identifier. Like CVE-IDs, these are also usually at the product level. However, while *VU#* documents often describe a single CVE-ID, that is not always the case. There are VU# documents which describe multiple CVE-IDs, as well as ones that describe vulnerabilities that are out of scope for CVE entirely.

Vulnerabilities in products are the main stock and trade of vulnerability management. Such vulnerabilities often need to be triaged to prioritize actions. A popular scoring tool to communicate the technical severity of a vulnerability is CVSS. While CVSS and CVE-IDs are managed by different organizations and are officially unaffiliated with each other, they are often mentally associated due to the close relationship between uniquely identifying and triaging vulnerabilities. Section 4.2 will address CVSS in more detail.

[1] Often, this association is mediated through specific versions of software. E.g., host A has version X of software Y installed, and version X of software Y has vulnerability Z

2.1.3 Vulnerability Categories. The next broader part is the categorization of vulnerabilities. A number of frameworks exist at this level, with perhaps the best known being the CWE framework.

CWE is a "list of common software and hardware weakness types that have security ramifications" [34]. CWE is not intended to be a catalog of specific problems, but rather a collection of important design flaws that lead to "weaknesses" in software and hardware. Although MITRE is a bit inconsistent about the definition of "weakness," it is roughly equivalent to the CERT/CC definition of vulnerability. So the two main things that can be members of a CWE are a vulnerability and another CWE category. CWEs are arranged hierarchically from 10 "pillar" weaknesses which are general descriptions of all weaknesses, with intermediate and specific weakness types categorized under them.

An example CWE is "buffer overflow," and any number of CVE-IDs may be an example of this CWE. A CWE can loosely be understood as a conceptual way that someone might accidentally introduce a security weakness into some information processing system, whereas a CVE-ID identifies a concrete product version in which someone introduced a specifically identifiable security flaw.

Not all vulnerabilities associated with a CWE get a CVE-ID. This situation is common with, for example, instances (Section 2.1.1) of configuration-level vulnerabilities (see Section 2.2) in specific web servers.

The CWE specification is ambiguous whether there can be instances of software which match the description of the weakness but cannot have a security impact due to some specific circumstance, such a the code being demonstrably unreachable. Various secure coding guidance would certainly recommend avoiding such design patterns because they are fragile [48]. This guidance holds whether we name such circumstances a security weakness or not, so pragmatically we shall leave this ambiguity as it is.

OWASP is another vulnerability categorization scheme, most famous for its Top 10 document for web developers that "represents a broad consensus about the most critical security risks to web applications" [37]. Since OWASP is tailored to web applications, it is more specific than CWE. The OWASP Top 10 is also more pragmatic; the goal is to prioritize effective protective measures that a web developer should ensure during their development life cycle. OWASP focuses on secure configuration of web servers, rather than secure coding. Only one of the top 10 – "9: Using Components with Known Vulnerabilities" – overlaps with CVE-IDs; the other nine represent categories of vulnerabilities that would not normally be given a CVE-ID.

OWASP and CWE have different constituencies and reach different audiences. The categorization schemes have differing emphases that reflects their different constituencies. But both serve a similar purpose – to organize knowledge about vulnerabilities in vulnerable products.

2.2 Abstraction

The second axis, which is independent from vulnerability identification, is a description of the level of abstraction of the vulnerable product. The four levels of abstraction for vulnerable products, from most specific to most abstract, are:

- Configuration-level vulnerability
- Implementation-level vulnerability
- Protocol-level vulnerability
- Algorithm-level vulnerability

The product may be a specifically-configured instance, an implementation, a protocol, or an algorithm.

2.2.1 Configuration vulnerability. A deployed product may be vulnerable due to its configuration in situ. For example, a linux host may be vulnerable if its '/bin' directory is world-writable due to an errant sysadmin. In such a case there is nothing inherently wrong with the software, it has just been deployed in an insecure manner.

2.2.2 Implementation vulnerability. An implementation is, loosely, the source code or binary executable that is distributed as a product. Most vulnerabilities that are widely discussed are those found in implemented products, hence CVE-IDs are most closely associated with implementation vulnerabilities. The usual way of identifying the vulnerable implementation of a product is to state the versions that are vulnerable, such as "versions 3.2.9 and earlier are vulnerable."

2.2.3 Protocol vulnerability. Implementations may often be based on a protocol. The most common protocol vulnerabilities are in communications protocols – agreed ways of exchanging information between devices that devices may implement in their own, though mutually compatible, way. Examples of protocols with documented vulnerabilities include Bluetooth (e.g., CVE-2019-9506), Transport Layer Security (TLS) (e.g., CVE-2014-3566), and Server Message Block (SMB) (e.g., CVE-2020-0796). When there is a protocol vulnerability, all implementations of that protocol are, by definition, vulnerable. There may be workarounds to reduce exposure, as usual, but an implementation inherits many things from the protocol it implements, including vulnerabilities.

Vulnerability managers need not localize a vulnerability to a protocol; practically, it is every implementation of the protocol that must change. The rules for assigning CVE-IDs address this directly. A single ID is assigned to the protocol, standard, or Application Programming Interface (API) rather than multiple CVE-IDs assigned to each implementation if and only if "there is no option to use the functionality or specification [e.g., protocol] in a secure manner" [9, §7.2]. So in the case where the TLS protocol had a vulnerability, every implementation of TLS would share the same CVE-ID. A pragmatic effect of assigning CVE-IDs to protocols rather than their various implementations is that it makes clear that the protocol designer or standards body is responsible for fixing the vulnerability.

2.2.4 Algorithm vulnerability. The layer of abstraction above protocol is an algorithm vulnerability. Historically, this term has usually applied to cryptographic algorithms. For example, the cryptanalysis of Data Encryption Standard (DES) in the early 1990s [32] identified algorithmic vulnerabilities in DES that any protocol using that algorithm inherited. Any implementations of those protocols also inherited the algorithm vulnerabilities as well, as expected. Thankfully, vulnerabilities in cryptographic algorithms have become quite rare. Such vulnerabilities largely predate the current vulnerability management apparatus of CVE-IDs which has come to dominate since 2010. But there is precedent in CVE-2004-2761

for assigning CVE-IDs for cryptographic weaknesses (in this case, the MD5 algorithm's susceptibility to hash collisions).

Our placement of "algorithm" as strictly above "protocol" in the abstraction levels is an artifact of the history of networking and network security. Communications protocols arrange certain building blocks to reliably and securely exchange information. A particularly important one of those building blocks is cryptographic algorithms. Protocols infrequently but occasionally have vulnerabilities; this is usually a problem in the structure of the protocol and how information is exchanged or handled. But the protocol designers usually treated the cryptographic algorithms as special, as a sort of root of trust for the security of the protocol. However, the perspective MITRE takes with the CVE-ID rules would consider both protocols and cryptographic algorithms "products" whose functionality would be shared by other products [9, §7.2].

The specificity and abstraction descriptions are orthogonal. One can have an instance of a implementation vulnerability, a product with an implementation vulnerability, or an implementation vulnerability which is an example of a weakness type. Similarly, one can have an instance of a configuration vulnerability, a protocol (that is, a product) with a specific vulnerability, a vulnerability in an algorithm which is an example of a weakness type, etc.

This paper will discuss the hypothetical of assigning CVE-IDs to ML algorithm vulnerabilities and/or to ML model objects. This hypothetical is specifically about vulnerabilities the existing regime does not handle. The existing vulnerability management regime does not have any problem handling implementation-level vulnerabilities in ML libraries, such as buffer overflow mistakes in TensorFlow (e.g., CVE-2018-10055). Such algorithm-level vulnerabilities are well known within the ML research community, as Section 3 will discuss. However, the current vulnerability management paradigm has not had to handle many algorithm-level vulnerabilities in more traditional computing infrastructure; the last one was probably 2008 with practical collision attacks against the MD5 algorithm [11]. This mismatch is one aspect that will make our thought experiments instructive.

2.3 CVE-ID background

CVE-IDs are designed to provide unique identifiers for the purpose of tracking a vulnerability throughout vulnerability management processes, with an emphasis on enabling communication among constituents and stakeholders. The CVE program is not a stand-in for all of vulnerability management: there are relevant vulnerabilities that are never assigned CVE-IDs. Misconfigured file permissions are a common example. However, since the CVE program provides the unique identifiers that vulnerability managers use to track their main work items, it is a useful entry point that enables our thought experiments to touch, if not fully explore, all six areas of vulnerability management.

MITRE is the lead organization, but they have delegated the ability to assign CVE-IDs to about 120 CNAs [33]. The first CVE-IDs were assigned in 1999, with 1,500 vulnerabilities assigned identifiers that year – many of which had been discovered some time earlier in the decade. As of Aug 19, 2020, about 140,000 vulnerabilities have CVE entries.

CVE-IDs have power within vulnerability management. For example, the US NVD requires a CVE-ID for all entries (https://nvd.nist.gov/general/FAQ-Sections/CVE-FAQs). Because National Institute of Standards and Technology (NIST) operates the NVD and NIST produces the information security standards for the US federal civilian government, when US government security regulations say something like "patch all known vulnerabilities," the word "known" is usually understood to mean "in the NVD." Which implies that the only vulnerabilities US federal civilian government entities are required to patch are those with CVE-IDs.

In the commercial vulnerability management space, a similar scenario plays out. Asset management or vulnerability scanning products have a tendency to be based on fingerprints or signatures of device or software stacks. For example, if a scanner can determine that a web server is Apache version 2.2.31, then a simple lookup indicates it is vulnerable to CVE-2017-9788 and should be patched to a more recent version. As a consequence, vulnerability management is not driven by vulnerabilities so much as it is driven by CVE-IDs. The only community in which CVE-IDs do not entirely drive vulnerability management is website owners, where OWASP and CWE are used to label configuration-level vulnerabilities such as cross-site scripting and improper data protection configurations.

MITRE does not strictly control what counts as a vulnerability. It is worth quoting their definition at length [9, §7]:

> The CVE Program does not adhere to a strict definition of a vulnerability. For the most part, CNAs are left to their own discretion to determine whether something is a vulnerability. Root CNAs may provide additional guidance to their child CNAs. This allows the program to adapt to definitions used in different industries, legal regimes, and cultures.
> 7.1.1 If a product owner considers an issue to be a vulnerability in its product, then the issue MUST be considered a vulnerability, regardless of whether other parties (e.g., other vendors whose products share the affected code) agree.
> 7.1.2 If the CNA determines that an issue violates the security policy of a product, then the issue SHOULD be considered a vulnerability.
> 7.1.3 If a CNA receives a report about a new vulnerability that has a negative impact, then the reported vulnerability MAY be considered a vulnerability.

Section 5 will show this official definition allows space to consider flaws in ML algorithms as vulnerabilities that get CVE-IDs. Nothing in the current written guidance would need to change. However, Section 4 will also show how ML algorithms present a number of challenges to existing vulnerability management practices, including assumptions about the responsibility to fix CVE-IDs. A trained model object (see Section 3.3) is fairly clearly a product to which a CVE-ID could be assigned; Section 6 will show that choice would present a related but distinct set of challenges to existing vulnerability management practice.

3 ADVERSARIAL ML BACKGROUND

There are myriad ways in which an adversary can cause an ML algorithm to behave unexpectedly and violate either implicit or

explicit security policies. Statisticians and ML engineers rarely express such problems in vulnerability management terms. This section will introduce how the ML research, policy, and operational communities have expressed the problem.

The name of this field is adversarial machine learning (AML). Unfortunately, even here we have a terminology collision; some communities use AML to refer to generative adversarial networks, or training ML algorithms using game theory through adversarial examples. This paper exclusively uses AML to refer to attacking and defending ML algorithms. Section 3.1 summarizes the state of academic AML work via the conclusions of two recent literature reviews. Section 3.2 summarizes two recent attempts to translate the conclusions out of the AML research space to policy makers. Section 3.3 introduces modern operational considerations around engineering reliable ML systems. The understanding of these other efforts maps out empty spaces where a perspective from vulnerability management may be helpful.

3.1 Summary of academic work

Biggio and Roli [3] is a highly cited literature review within the statistical research community. Their abstract summarizes the state of affairs as "adversarial input perturbations carefully crafted either at training or at test time can easily subvert [ML systems'] predictions. The vulnerability of machine learning to such wild patterns (also referred to as adversarial examples), along with the design of suitable countermeasures, have been investigated in the research field of adversarial machine learning." While Biggio and Roli [3] does not use vulnerability management terms, their categorization is based on the basic security triad of confidentiality, integrity, and availability. The earliest published work on attacking ML algorithms documented by [3] dates to 2004.

Contemporary with Biggio and Roli [3], Papernot et al. [39] is an excellent literature review of the space, but whose target audience is the security research community. Similar to [3], Papernot et al. [39] find that "there is growing recognition that ML exposes new vulnerabilities in software systems, yet the technical community's understanding of the nature and extent of these vulnerabilities remains limited."

Both Biggio and Roli [3] and Papernot et al. [39] taxonimize attacks on ML algorithms similarly, though there are certainly differences of emphasis. The rest of this section will discuss each of the following aspects of their taxonomies in more detail.

- Both distinguish between training-time and test-time attacks.
- Both differentiate based on how much the adversary needs to know in order to perform the attack.
- They differ slightly on their implied security policies; Papernot et al. [39] identifies different kinds of attacks that violate integrity and privacy, rather than the CIA triad Biggio and Roli [3] uses.
- The papers differ in their proposed defenses.

Both distinguish between training-time and test-time attacks. If an adversary can influence the data used to train a model, different attacks are possible than if the adversary can influence the data items to be tested. The situation where an attacker can influence both test and training is often called *poisoning*, whereas the situation is called *evasion* if just test data can be influenced [3, §3.3]. Biggio and Roli [3] restrict their framework to supervised learning algorithms. Papernot et al. [39, §5] notes the training-test distinction is heavily biased towards just supervised classification algorithms, but indicates that other types of algorithms, such as unsupervised and reinforcement learning, seem to exhibit similar vulnerability even though they are much less thoroughly studied. Beyond algorithm type restrictions, this distinction covers only the model building and validation life cycle and excludes model deployment [16]. Restricting the scope to model building and validation makes sense for academic research, but our CVE-ID thought experiment will need to include deployment and environmental vulnerabilities, as discussed in Section 3.3.

Both differentiate based on how much the adversary needs to know in order to perform the attack. The terms "white box" (full-access) and "black box" (query-access) are used with similar meaning in both papers, and their meanings are similar to the way the terms are used in the fuzzing literature [31].[2] Briefly, full-access refers to the attacker having complete access to the model object, such that the attacker can load it into a controlled environment and inspect and modify its components. Query-access commonly refers to the attacker being able to provide inputs to the model and received outputs, but not be able to inspect the internals of the model object.

In broad strokes, if the adversary knows more about the model and the feature space, it is easier for them to attack the model. However, query-access attacks on models are readily feasible. More precisely, algorithms "can be threatened without any substantial knowledge of the feature space, the learning algorithm and the training data, if the attacker can query the system in a black-box manner and get feedback" [3, §3.2]. At the extreme end, in special cases an attacker can create a full-access situation from a query-access situation by recreating the model (up to machine precision!) through querying the model with carefully crafted examples [7].

Although Papernot et al. [39] and Biggio and Roli [3] differ slightly on their implied security policies, both use the term "threat model" to discuss the adversary's capabilities and goals. This is part of a security policy, but falls far short of defining a security policy for an ML system. Within the AML literature, "threat model" is used similar to the mathematical cryptography community, where a threat model is a mathematical expression of the adversary's capabilities. In AML, a threat model is a declarative statement. In operational security communities, a threat model is a the output of an investigation or observations about what adversaries can in fact do, or have done in the past, see for example Fox et al. [14]. The difference between a declarative and observational threat model is subtle, but it may cause members of the two communities to miscommunicate.

One consequence of this disconnect is that the AML community has adopted threat models that are mathematically tractable, but not likely to be observed in practice. For example, the most popular AML threat models are based on small changes to the input, usually measured with an L_p norm. These are useful for two reasons: first, they tractable to analyze, and second, they allow researchers to

[2] These terms based on color were commonly used in the literature, but we will use the more descriptive and less divisive terms "full-access" and "query-access" in their place in our discussion.

develop a principled understanding of the fundamental properties of these systems. These threat models, however, are divorced from the constraints of the real world. Specifically, permissions are often implemented on computer systems to be all or nothing; if an adversary has write access to a file, they can make arbitrary changes. If the threat model assumes the adversary has some form of write access, as implied by the ability to make changes to the input, then bounding the scope of the changes the adversary would choose to make is somewhat implausible.

Portions of the AML community consider more realistic threat models. An important class of these are modifications to the physical environment that must survive a data processing pipeline, such as a sticker that would cause an object to be misclassfied, poisoning the data that one would collect for training, or inserting trojans or backdoors into publicly released models. However, these communities are a minority within AML and have not yet received the same amount of attention as those communities that rely on mathematically tractable threat models that were the focus of the literature reviews. Whether such mathematically bounded threat models apply to these physical environments is unclear; if they do, it is likely only in the context of a wider system where such limits are either empirically motivated or enforced by other security mechanisms (such as human guards).

The literature review papers also differ in their proposed defenses. Biggio and Roli [3] recommends reactive defenses, security by design, and security by obscurity. "Reactive defenses" are similar to what a security practitioner might call "continuous monitoring and evaluation" in conjunction with an appropriate risk assessment [51, p.11]. Papernot et al. [39] presents a more conservative analysis of defenses, emphasizing that defense against most known attacks is an open area of research. Therefore, [39] prioritizes mapping out a research plan for a science of security and privacy of ML.[3] The broad recommendation is that ML algorithms will need to become resilient to distribution drifts and incorporate privacy through differential privacy methods. While this perspective essentially admits researchers do not currently know how to defend ML algorithms, that perspective should sit better with the security community than uncritically endorsing security through obscurity.

3.2 Summary of policy work

This section is not a comprehensive survey of policy work related to AML, but rather highlights two ongoing projects that are attempting to bridge the gap between ML researchers and software engineers or policy makers. The first is a private sector initiative, led by Microsoft and Harvard. The second is an effort by NIST to bridge the AML and information security policy communities.

The Microsoft/Harvard effort is perhaps centered on Kumar et al. [28], but includes other collaborative documents such as Kumar et al. [27]. Kumar et al. [26] specifically names tracking, scoring, and responding to vulnerabilities in ML systems as a gap in current practice when a ML system is under attack. This paper fleshes out what it would take to fill that gap.

One important goal stated by Kumar et al. [28] is to "equip software developers, security incident responders, lawyers, and policy makers with a common vernacular to talk about" AML through a taxonomy of ML failure modes. The taxonomy expands beyond the narrow research concerns of the academic literature, and so includes deployment and environmental failures. However, the Kumar et al. [28] taxonomy remains restricted by current research in some ways, in that attacks on supervised classification algorithms are better researched and better understood than failures of other types of algorithms; the authors attempt to overcome this gap and be as comprehensive as plausible.

Kumar et al. [28] adopt some taxonomic categories also used in the academic literature reviews. The distinction between full-access and query-access attacks (see footnote 2) is present here with similar meaning. The connection to security policy is via the CIA triad, similar to Biggio and Roli [3]. Kumar et al. [28] frames CIA as assurances for the ML system, rather than capabilities of the adversary, which aligns better with common security usage.[4]

Kumar et al. [28] identifies 11 intentionally motivated failure modes, namely:

(1) Perturbation attack
(2) Poisoning attack
(3) Model inversion
(4) Membership inference
(5) Model Stealing
(6) Reprogramming ML system
(7) Adversarial example in physical domain
(8) Malicious ML provider recovering training data
(9) Attacking the ML supply chain
(10) Backdoor ML
(11) Exploit Software Dependencies

In relation to the CVE paradigm of vulnerability management, the best way to understand these failure modes is as new candidate CWEs. The failure modes are about general kinds of things that can go wrong when designing, implementing, or using ML algorithms. Also similar to CWE, Kumar et al. [28] is clear the authors are curating a living document that will change as the community finds further failure modes.

MITRE has published one CWE related to AML: CWE-1039, "Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations." CWE-1039 tracks to "Perturbation attack" in Kumar et al. [28]. This suggests there are at least 10 further CWEs to define. CWE-1039 is more general than either of our proposed thought experiments – either a specific algorithm or a specific model could be an example of this weakness type.

NIST has a policy effort to bridge AML and information security policy in Tabassi et al. [54]. The NISTIR is a draft, and so will likely change and improve; at present, a comment period closed on Jan 30, 2020 and the authors are editing based on those comments. The version presently available is essentially just an attempt to

[3] "Science of security" is a broad term, and it is not clear which of the senses or goals identified by Spring et al. [53] is meant by [39].

[4] There are shortcomings with this common usage. One one hand, it is based on a metaphor with the physical goods, such as fortifications, which breaks down when applied to information systems [40]. On the other hand, some standards bodies, such as the Internet Engineering Task Force (IETF), list several other recommended security services in additional to just CIA, such as non-repudiation, and differentiate some parts of CIA, such as between system integrity and data integrity [49, p.274]. Failure modes in ML may put additional pressure on the basic CIA triad, and justify a change to one of these more nuanced approaches.

synthesize the academic surveys; these include [3, 39] as well as three others as primary sources [54, p.2].

Despite being a NIST document, one thing that Tabassi et al. [54] decidedly does not do is integrate or deconflict AML terminology with the terminology in the NIST Computer Security Research Center glossary, or any other set of standard information security terms [16]. Unlike Kumar et al. [28], the NISTIR draft is narrowly scoped to academic AML interests and does not account for deployment or environmental failures that would be of interest in practice [16]. We expect the next draft will improve on these shortcomings. However, for these reasons at present Tabassi et al. [54] does not bridge the NVD, CVE program, or other vulnerability management functions with the AML space.

3.3 Summary of operational work

The work on operational assurance that ML systems behave as expected covers a broad space. First, this section introduces an anatomy of an operational ML system to clearly define the parts subject to our thought experiments and contextualize them. Second, we will scope our thought experiment by summarizing valid operational concerns that are and are not security vulnerabilities through the lens of deployment context.

A deployed ML system has a broader attack surface than just the training and testing of the ML model. Our thought experiment in this paper is limited in scope to ML algorithms and model objects, but it is important to recognize that these are just some parts of an operational ML system.

Figure 1 provides a representation of the pieces involved in developing and operating an ML system.[5] While this diagram is only a rough representation of any particular system, it provides a useful tool for conceptualizing the possible vulnerabilities of any deployed ML system. Each of the components and processes in the diagram represents a different point where a vulnerability could be introduced; vulnerabilities may arise in sensors that collect the data, the data processing component, or the runtime monitoring tools, in addition to the model itself.

This perspective highlights the difference between an ML algorithm and an ML system. The ML algorithm is the particular mathematical procedure used to create an ML model object by configuring/training it on the data. The ML system includes all of the components illustrated in Figure 1. Horneman et al. [20] provides general guidance for the design and management of ML systems.

As discussed in Section 3.1, the AML research community focuses on categories of vulnerabilities introduced during the Model Building and Validation stage; particularly, categories of vulnerabilities that may be inherent in the Untrained Model or introduced by manipulation of the Training Data or Test Data 1. Kumar et al. [28], discussed in Section 3.2, broadens this focus; e.g., the category "Attacking the ML supply chain" introduces the idea that a vulnerability could be introduced to a trained model as it is being downloaded in the Model Deployment stage. However, a deployed system has yet more components which could introduce vulnerabilities. For example, if an adversary wanted you to waste time and money retraining your model, thus hurting you in a resource-constrained environment, they could attack the Benchmark Data to make your model seem as if performance was degrading for unknown reasons.

To situate ML systems in their full context, one should observe that deployed ML systems often are part of enforcing or developing policies for organizations, including governments. Such systems often embed power structures, biases, and inequity [35, 36, 58]. Both ML researchers, e.g., [13, 47], and legal scholars, e.g., [8, 12, 25, 29], have been struggling with how to seek out and eliminate these problems from ML systems. However, designing an ML system with an assured fidelity to a particular substantive policy choice is not usually considered a security question. That is, assuring a high-quality ML system generally involves assuring attributes other than security, such as fairness, equity, stability, etc. We place these other attributes out of the scope of this paper.

However, exceptions for these other attributes aside, the larger context of deployment is essential for defining what constitutes a vulnerability. Vulnerabilities live at the intersection of the software *system* and its interaction with the environment. For example, Householder [21] has argued that prior to the invention of airplanes, buildings did not have airplane related vulnerabilities. Something about the world changed with the invention of airplanes, and now the interaction between buildings and their world contains a new class of vulnerable states, damage caused by impacts with airplanes. In the same way, before the advent of self-driving cars, ML systems did not have vulnerabilities involving stickers on stop signs [18]. Since the deployment context has changed to include self-driving cars, the interaction with the environment now includes such vulnerabilities. Accounting for these diverse environments is both a challenge and a motivation for our thought experiments.

4 THOUGHT EXPERIMENT 1: ALGORITHMS

This section explores one hypothetical situation – what if flaws in ML algorithms were assigned CVE-IDs? We explore the consequences for vulnerability management by analyzing each of the six vulnerability management service areas [2]. CERT/CC has published one vulnerability note about an ML algorithm vulnerability, but it is tracked only with the internal identifier VU#425163 and does not list a CVE-ID [22]. The note describes how ML classifiers trained via a gradient descent algorithm are vulnerable to arbitrary misclassification attacks. Although the focus of our thought experiment will tend to be generic, in cases where a concrete example is helpful we will use the CERT/CC vul note.

As Section 3.3 discussed, there are many parts that go into an operational ML system, and the algorithms are just one part. The thought experiment here is just working through what would happen if CVE-IDs were assigned to algorithms. Of course, systems that use the algorithm should likely inherit the CVE-ID; although there is guidance for defending ML systems from vulnerabilities in their algorithms, the guidance is evolving rapidly because most guidance has been quickly shown to be inadequate [6].

Each of the following subsections examines our thought experiment through one of the CSIRT services within vulnerability management, as introduced in Section 1.

[5] Note that this diagram can represent both supervised and unsupervised ML systems; however for reinforcement learning, the general pipeline is the same, but the model building and validation stage will require modification.

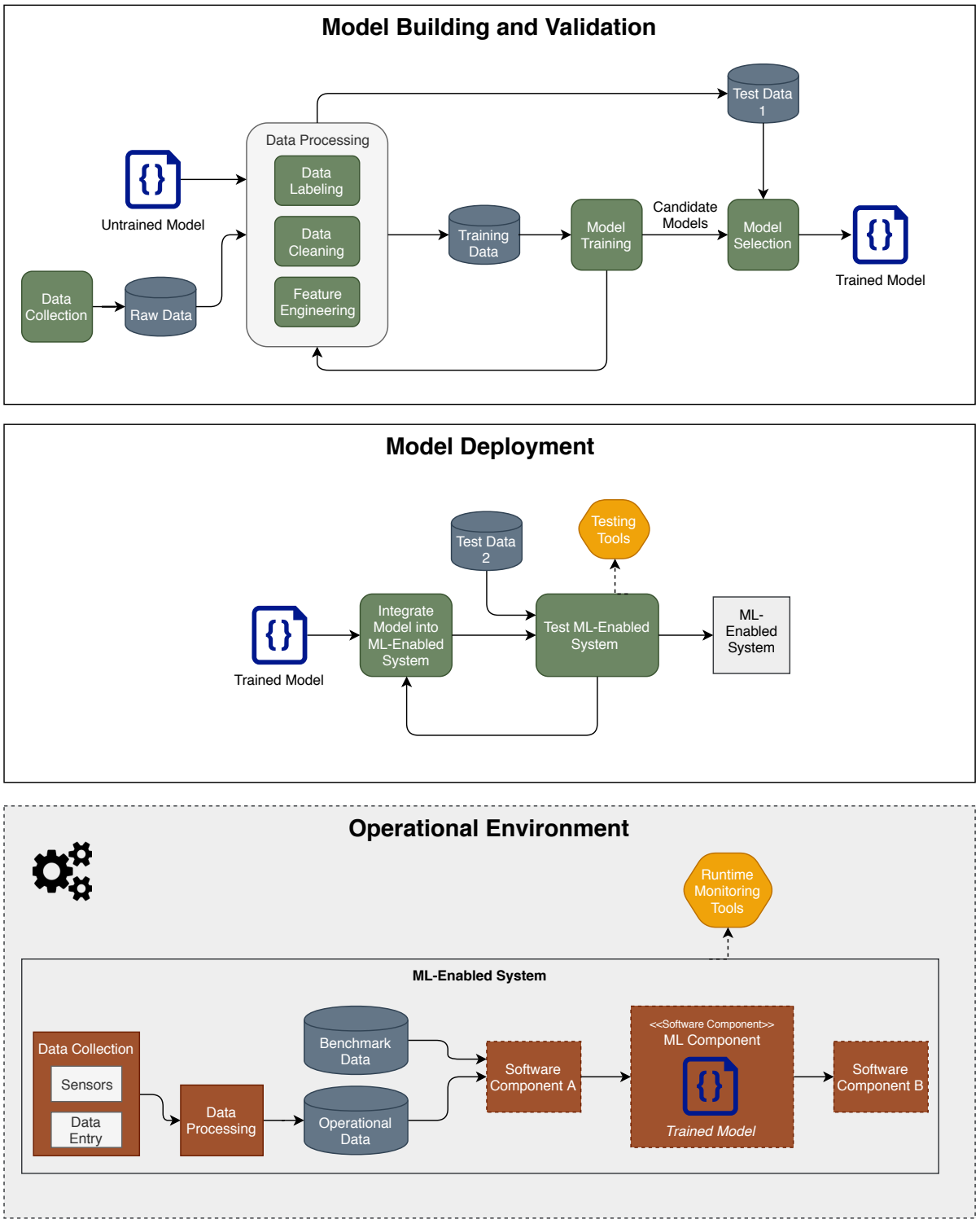

Figure 1: Each of the green boxes represents a process, often with a human directly involved. Each of the burnt-orange boxes represents a software component in the operational environment. In an operational ML system, the model needs frequent updating, and this process could be represented by adding a loop to this diagram.

4.1 Vulnerability discovery / research

In our scenario, vulnerability discovery will not change much. All current vulnerability discovery work can continue as before. If we start assigning CVE-IDs to ML algorithm vulnerabilities the number of people who should be counted as doing vulnerability discovery will jump higher overnight, as AML folks will be added to the ranks. Although it is difficult to estimate the size of the AML community, over 1,900 papers have been published since 2014 according to a widely followed tracker [5], suggesting a substantial influx.

As in the current paradigm, in our thought experiment not all ML algorithm vulnerabilities will end up with an assigned CVE-ID. However, the reasons for this will remain the same; there are at least three possible reasons why a CVE-ID may not be assigned:

- The vulnerability research is internal to product development, and will be fixed without being made public
- The vulnerability is not independently fixable from another vulnerability
- The vulnerability researcher plans to sell the vulnerability or otherwise use it in attacks

The first two are expressly forbidden to have CVE-IDs in the CVE specification [9, §7]. Those in the third category may be assigned a CVE-ID eventually, once defenders detect attacks using it.

This first reason will not unduly stress our thought experiment. Most mature software engineering projects conduct internal testing before release. Tests may include fuzzing, program verification, code review, or unit tests. In general, if code pre-release fails these tests, it is fixed without being assigned a CVE-ID because it does not need to be discussed publicly. Such testing regimes for ML systems are one recommendation in both the academic and policy literature.

The second reason a vulnerability may not receive a CVE-ID is more slippery in relation to ML algorithms. Given that how to prevent these attacks on ML algorithms is not known in principle, it may be considered an area of active research. However, we can say something about what should get different CVE-IDs. If the algorithms are distinct ML algorithms, then they should get distinct CVE-IDs, even if they exhibit the same flaw type (as documented by [28], for example). This practice would be analogous to the way in which two web browsers get separate CVE-IDs even if they both have a buffer overflow, even though buffer overflow is the same kind of flaw in both. Similarly, if there are distinct versions of the same algorithm, they will only get separate CVE-IDs if the fixes are independent. This practice is analogous to the way that multiple versions of a web browser may be affected by a single CVE-ID today.

The abstraction level hierarchy presented in Section 2.2 presents some further trouble for "independently fixable" that is not so easily resolved. The hierarchy implies that all protocols or models that use a vulnerable algorithm and implementations of those protocols or models will share the same CVE-ID. This sharing is implied because the implementations are not independently fixable from each other; the algorithm may need to be fixed and then this change propagated down the abstraction levels. However, as is the case currently where every instance of a product is not always susceptible to vulnerabilities in the product, due to configuration changes or other workarounds in place, it may be possible to use a vulnerable algorithm in a way that that does not expose every algorithm vulnerability. This, too, is an area of active research, and a topic we will return to in Sections 4.3 and 4.6.

Metric Name	Group	Options
Attack Vector	Exploitability	Physical; Local; Adjacent; Network
Attack Complexity	Exploitability	Low; High
Privileges Required	Exploitability	None; Low; High
User Interaction	Exploitability	None; Required
Scope	Impact	Changed; Unchanged
Confidentiality	Impact	None; Low; High
Integrity	Impact	None; Low; High
Availability	Impact	None; Low; High

Table 1: Summary of CVSS v3.1 base metrics [10].

Finally, vulnerabilities discovered for the purpose of attacking others would mostly have the same properties in a world where ML algorithms get CVE-IDs. It may seem that disambiguating different attacks is harder with ML algorithms. But given the current prevailing exploit-as-a-service [17] and crimeware-as-a-service [50] situations, it is already exceedingly difficult to know which vulnerability an adversary exploited. And if ML algorithms get CVE-IDs, it may slowly become easier to identify and catalog which attacks an ML system is expected to be vulnerable to or resist.

4.2 Vulnerability report intake

In addition to receiving reports, this service area is where any analysts "review, categorize, prioritize, and process" a report [2]. There are various human processes that will need to adjust if ML algorithms receive CVE-IDs and become part of this process. We will focus on prioritization. Review, categorization, and processing will all require analysts to acquire new skills and expertise to understand ML algorithms, but this may be handled in the medium term by creating a CSIRT dedicated to the open-source ML community. Such specialized CSIRTs exist for industrial control systems, for example. And for broader security awareness beyond vulnerability management, Information Sharing and Analysis Centers (ISACs) serve a similar function. Prioritization cannot be so easily handed off to a specialized workforce.

Prioritization implies giving a CVSS score to the vulnerability [10]. Although a CVSS base score is explicitly not to be used on its own as a prioritization score, the technical severity as measured by CVSS is often considered an important factor in vulnerability prioritization for many organizations today. The CVSS base score is intended to reflect the "reasonable worst case impact", and encapsulates the relative ease of exploitation (exploitability) and the direct consequences of a successful exploit (impact). The dimensions of exploitability include attack vector, attack complexity, required privileges, and user interaction; while impact considers scope and the confidentiality-integrity-availability triad [10, §1]. Table 1 summarizes the options for CVSS base metrics.

What would happen if we try to give a CVSS score to VU#425163? The first thing we notice is that the guidance of evaluating the "reasonable worst case impact" has limited utility when applied to ML algorithms. The meaning of "reasonable" in the context of an algorithm is difficult to bound, as an algorithm could be used in so many

different contexts. When considering exploitability, there are plausible scenarios where the misclassification can be forced remotely (attack vector: network), the attack can be automated [38] (attack complexity: low), the adversary can submit test cases without prior authentication (privileges required: none), and the user does not have to do anything (user interaction: none). For impact, the first metric is scope – can the compromise of the vulnerable component be used by the adversary to compromise other parts of the information system of which it is a part to increase the scope of the attack. For ML algorithms, attacking the behavior of the ML algorithm can change the behavior of the car, robot, or system in which the trained ML model is embedded (scope: changed). The last three questions are about the CIA triad, but since scope is changed it is the worse case of the algorithm itself or the wider scope of what is the adversary can access. Kumar et al. [28] would characterize this kind of misclassification attack as a failure of algorithm integrity, so we can safely set integrity to high. The same CVE-ID will apply to forcing a Tesla autopilot to change lanes, which, in a reasonable worst case, could cause the car to crash, which sets availability to high as well. Even if we ignore confidentiality and set it to none, the CVSS score is maxed out at 10.0.

This is probably not a reasonable CVSS score, or at least it is out of sync with scores for other vulnerabilities. It is unclear whether this indicates our ML algorithm vulnerability is particularly bad, or whether CVSS just is not suited to analyze such vulnerabilities. Given that CVSS has a host of problems, including complaints it does not apply properly to domains such as industrial control systems, medical devices, and robots, we hypothesize the problem lies with CVSS in our thought experiment as well. Spring et al. [52] proposes an alternative, stakeholder-specific vulnerability prioritization scheme that would be more easily adapted to ML algorithm vulnerabilities.

4.3 Vulnerability analysis

The purpose of vulnerability analysis is to triage incoming reports, understand the root cause of the vulnerability, and develop countermeasures to fix or mitigate the vulnerability [2].

Triage is heavily dependent on prioritization, and so mirrors the discussion in Section 4.2. The ingredient triage adds is asset management. For example, if the vulnerable component identified by the CVE-ID is not installed anywhere the security team is responsible for, then the priority does not matter. Asset management of ML algorithms is a challenging problem. The modern cybersecurity paradigm already struggles with vulnerabilities in libraries that may be used in diverse products; these challenges are well-documented by the work on Software Bill of Materials (SBOM) [24]. Tracking vulnerable ML algorithms will only make this problem more urgent. Such tracking may meet initial resistance from the system vendors, under the argument that CVE-IDs may reveal the algorithm they use and that choice may be intellectual property or some such. Whether this legal argument carries will likely be jurisdiction specific; in places with a strong right to explanation (such as in the GDPR), such information seems less likely to be protected.

Root cause analysis is affected in similar ways as vulnerability discovery. The AML academic research field is currently engaged in root cause analysis for all the vulnerable algorithms. What assigning CVE-IDs would change is to drive conversation between the AML research community and system engineers and system owners, who will likely gain increased awareness that their ML systems have flaws. As Kumar et al. [28] identifies, there is a gap here in that the two communities lack a shared language in which to communicate. So far the main efforts seem to have been to teach system owners the language from AML. Assigning CVE-IDs to algorithm vulnerabilities would start the work of teaching the AML researchers language from operational security.

Developing countermeasures for vulnerabilities in ML algorithms will continue to be a hard problem. While there is 15 years of research within the AML community, we are not aware of any public discussion on how to build an ML system with appropriate mitigations or workarounds in place to isolate the algorithm from adversary interference. Security professionals use mitigations and workarounds for vulnerabilities regularly, when a fix is not available or not practical. For example, one recommended mitigation for any SMB service is to only expose it to a trusted local network, never the internet. This workaround does not fix SMB vulnerabilities, but it does mitigate them. Biggio and Roli [3, §5.1] summarizes six papers under the heading "reactive defenses" that likely map to workarounds and mitigations. CVE-IDs may help system owners and developers track which workarounds are necessary for their systems. But the system engineering practices for ML-enabled systems are still a work in progress as well [20]. The need is especially dire if such systems have a cybersecurity task such as malware identification, which must be exposed to adversary-crafted input [51].

4.4 Vulnerability coordination

Coordinated Vulnerability Disclosure (CVD) is "the process of gathering information from vulnerability finders, coordinating the sharing of that information between relevant stakeholders, and disclosing the existence of software vulnerabilities and their mitigations to various stakeholders, including the public" [23, §1.2]. The previous sections have highlighted the knowledge gap between security operations and AML researchers. CVD is probably the place where those knowledge gaps will evidence as barriers. All three steps – gathering information, sharing that information, and disclosing vulnerabilities – are dependent on shared knowledge.

For example, consider VU#425163. If that vulnerability were to get a CVE-ID, its finders in the AML community would likely react with confusion. The issue has been known for years; they would have some questions about why assign an identifier now. Next, who do we share information about the vulnerability with – it is already public, after all. And there is no easy way, at present, to notify specific vendors because there is no listing of which products use which algorithms. Despite these facts, the statements from the engineering and policy community are that the information about these flaws is not getting where it needs to; that is the whole premise of the efforts by [28]. Communication and coordination seem to be failing somewhere, likely at multiple points. Assigning VU#425163 a CVE-ID would not solve any of these problems. But it might start some conversations that may build some trust and shared understanding that form the beginnings of coordination.

4.5 Vulnerability disclosure

Disclosure is closely linked to coordination. Two topics that would be affected here are announcements and timelines. Vulnerability announcements are affected because they include the results of several things discussed above, such as severity scores, identifiers, recommended fixes or mitigations, and a description using shared terminology. The relevant constituents may also be different from current vulnerability announcements, as Section 4.4 indicated. While these are a lot of changes to announcements, none of them require further discussion; timelines do.

In security operations, vulnerability disclosure timelines are already a contentious topic. The consensus position is often described as "coordinated," as in CVD, where a vendor is told about vulnerabilities in their software before the public or attackers so that the vendor has a chance to develop and deploy a fix. The AML community works under what might be called a zero notification paradigm – results tend to be published with no prior warning to those who develop or use the algorithm. There are security practitioners who advocate and practice this for vulnerabilities in traditional systems, but it is not the norm in most communities. These two differing sets of norms would come into conflict if a widely used and distributed product were tagged with a CVE-ID because overnight an ML algorithm vulnerability is discovered, posted to arXiv, and assigned a CVE-ID. How this conflict would be resolved depends on various political, operational, and technical factors. While we cannot venture a prediction as to how the conflict will resolve, the fact that there will be a conflict between these two norms is almost certain.

4.6 Vulnerability response

Vulnerability response is where operations folks do something to prevent vulnerabilities from being exploited. There are two basic steps: detecting which systems an organization manages are vulnerable to which flaws and applying fixes or mitigations to those systems. For any system, traditional IT or ML, some of the vulnerable systems an organization manages will not have CVE-IDs; web server misconfigurations are a common example. We focus on those vulnerabilities with CVE-IDs. Both detection and response would need to adapt if ML algorithms are assigned CVE-IDs, because in practice many detection and response workflows are based on CVE-IDs as the primary unit of work.

The change to detection would depend on how much work can be done during analysis and coordination to link a vulnerable algorithm to deployment in specific product versions. This connection gets easier if the vulnerability is associated with certain published models that use the algorithm, but may get harder if the vulnerability depends on features of the training or test data. However, proprietary products rarely reveal their model or algorithm, which, in the short term at least, will make detection challenging.

It is likely that there will not be simple vulnerability scans to detect ML algorithm vulnerabilities, which upends the current detection paradigm that relies heavily on operations like Nessus scans. If detection does not adapt, one possible outcome of our thought experiment is that while ML algorithms have CVE-IDs, there is no operational impact because those CVE-IDs are never identified on deployed systems. For at least the medium term, detection of systems with vulnerable ML algorithms would be manual or mostly manual based on annotation of asset management databases. Whether or not this is acceptable will depend on the volume of vulnerable systems and the volume of attacks against systems without fixes or mitigations in place.

As Section 4.3 discussed, developing fixes and mitigations for these algorithm vulnerabilities will continue to be a hard problem. Response has a dependency on there being a fix or mitigation for operations folks to apply. This alludes to another possible fate of the thought experiment: there is raised awareness of exactly which systems are vulnerable to which kinds of attacks, but nothing for anyone to do about it. This statement is a bit overly dramatic, of course. System owners can either make a risk management decision that the system provides enough value to be worth the risk, or not. And usual system security principles such as least access and least privilege should still apply.

A more measured possible fate of our thought experiment is that *system owners become aware that they have taken on more vulnerable systems than they expected or understood, and re-evaluate either their need or deployed protections for those systems.* Until the AML community can provide more comprehensive fixes, this may be the best response available. And, if we establish the connection to between AML research and operational security sooner, then it should reduce the time to communicate and deploy those fixes when they become available.

5 BY THE LETTER OF THE RULES

Section 4 explored the impact of assigning CVE-IDs to ML algorithm vulnerabilities in general. This section explores the details of the CNA rules on assigning CVE-IDs to specifically ask whether CERT/CC could be justified in assigning a CVE-ID to the vulnerability note VU#425163 identifying gradient descent as vulnerable to misclassification attacks [22]. The CNA rules [9] have four aspects to consider:

(1) What is a Vulnerability?
(2) How many Vulnerabilities?
(3) CNA Scope
(4) Requirements for Assigning a CVE ID

We consider each of these in detail below.

5.1 What is a Vulnerability?

CVE-ID assignment rules [9] allow for a degree of latitude for CNA judgement, but do provide some specific guidance. Each of these criteria can be applied to VU#425163, though it takes a bit more work than with a traditional implementation-level vulnerability in a product.

Rule 7.1.1 says if the vendor recognizes the report as a vulnerability, then the report must be considered a vulnerability. In the case of most protocols and some algorithms, there is often a standards body responsible for maintaining the specification. That standards body is seen as the vendor for the purposes of CVE-ID assignment. For example, the cryptographic algorithm MD5 is specified in IETF RFC-1321 [45] and has at least one assigned CVE-ID (CVE-2004-2761). But unlike MD5, the gradient descent algorithm is not "owned" by a standards body. There are multiple variants of gradient descent [46], but the basic stochastic algorithm for training neural

networks dates to the late 1980s [1]. Therefore it is difficult to pin down a specific "vendor" who would authoritatively judge 7.1.1. The authors of the paper are the closest thing to the role of "vendor", but this seems like a poor fit for assigning responsibility for maintenance. Vulnerabilities can still be identified in abandonware though, so this should not be a big problem.

Rule 7.1.2 says the report should be considered a vulnerability if a product security policy is violated. In the case of a stochastic algorithm such as a classification algorithm, it seems that acceptable false positive or false negative rates might constitute such a policy. If an attacker can present the system with inputs that would otherwise be rare, it seems that a policy based on FPR or FNR would be violated. One might make a similar argument about fit quality for a regression needing to meet some specified tolerance. But stochastic policies can be difficult for security analysts to reason about.

The question of negative impact posed in Rule 7.1.3 is somewhat harder. The assertion of VU#425163 is that all systems in which gradient descent is used are susceptible to exploitation by adversarial input. So even though some implementations might not have a security relevant impact were exploitation attempted, it seems likely that *some* security relevant impact will occur in *some* implementations. So far, it seems our strongest arguments in favor of treating VU#425163 as a vulnerability per CVE-ID assignment rules are 7.1.2 and 7.1.3 coupled with CNA discretion.

5.2 How many Vulnerabilities?

Proceeding through the four questions, we reach section 7.2 of the CNA rules, which focuses on the concept of independent fixes. Rule 7.2.1 simply prohibits duplicate assignments. Rule 7.2.2 prohibits assignment when a dependency on another vulnerability exists, which is not the case for our example. Rule 7.2.3 suggests resolving uncertainty about independence by assigning a single CVE-ID. In effect, splitting is taken to be easier than merging should revisions be needed.

Rule 7.2.4 addresses what to do when "multiple products are affected by the same independently fixable vulnerability" arising from shared code. But both protocols and algorithms can have multiple implementations and therefore may not share code even though they share a vulnerability, making this rule inapplicable.

Rule 7.2.5 resolves the situation when the vulnerability originates in functionality or specification, as is the case for both protocol and algorithm vulnerabilities. If there is a way to implement the functionality securely, then each implementation that fails to do so must get its own CVE-ID according to rule 7.2.5.a. Conversely, a single CVE-ID is required by rule 7.2.5.b when there is no way to implement the specification or functionality securely. Rule 7.2.5.c resolves ambiguity in favor of multiple assignments.

In understanding 7.2.5, compare the vulnerability in this gradient descent algorithm to known errors in floating point handling algorithms. There is not a vulnerability in floating point handling per se because it is possible to handle floating points properly. Floating point algorithms have known errors that can lead to vulnerabilities, for example, CVE-2006-6499. This situation is an example of 7.2.5.a. The C Secure Coding standard discusses these floating point algorithm issues under FLP00-C through FLP07-C [48]. However, there is no known secure method for training a model with gradient descent (see Section 3.1). So it does not seem comparable to algorithms like floating point handling – which have known problems but also have known secure methods for use.

VU#425163 describes a problem with every system that uses gradient descent in training models, so rule 7.2.5.b seems most relevant here, thereby requiring a single CVE-ID assignment rather than one per affected product. This is consistent with assigning CVE-IDs to cryptographic algorithms and protocol specifications as previously noted.

5.3 Scope and Requirements

The remainder of the CVE-ID assignment rules are easier to get through: Rule 7.3 verifies that the vulnerability is in scope for the CNA making the assignment. CERT/CC's scope as a CNA covers assignment related to its vulnerability coordination role, so this falls within our scope.

Rules 7.4.1, and 7.4.2 verify that the report is intended to be public, which is true because VU#425163 exists. Rule 7.4.3 prohibits duplicate assignment to previously assigned CVE-IDs. Rules 7.4.4, 7.4.5, and 7.4.6 address the differences between vulnerabilities for which someone other than the CNA must take action to resolve. The only relevant one for our case is rule 7.4.6, which allows assignment for cases where the affected product(s) or service(s) are not owned by the CNA but are customer controlled. CERT/CC does not own the gradient descent algorithm, and it is used in customer controlled systems.

Rule 7.4.7 requires assignments not be made for products that are neither licensable nor publicly available. For VU#425163, although the gradient descent algorithm itself is not licensed, it is publicly available, so rule 7.4.7 does not impede us.

Finally, rule 7.4.8 requires CNAs to consider *only* these rules when making assignment decisions. Therefore per the CNA assignment rules, it seems that VU#425163 deserves a CVE ID.

5.4 Assignment conclusions

In a discussion among the authors and the analyst staff at CERT/CC, several of us hold the view that VU#425163 and issues like it might be better suited as a category of vulnerability – such as CWE entries or an OWASP item to avoid – rather than CVE-IDs. The lack of vendor ownership of an algorithm (or family of algorithms) was one recurring concern. This question of ownership is open around ML systems generally; for example, what exactly can be patented (and therefore owned) is unclear. But in general, a specific model for a specific purpose can be patented but an algorithm like logistic regression cannot. If an object is patented, it definitely has an owner; but an unpatentable item may still have an "owner" in the CVE sense if, for example, an algorithm is specified by an open standards body. We have not been able to resolve the relevant algorithm ownership question to our satisfaction.

Another concern centered on the fact that not all trained models are exposed to attacker-controlled input to the same degree, so the fact that gradient descent was used to train a model embedded in the system may not imply that an attacker can exploit it. Finally, from a vulnerability management operational perspective, many organizations have policies that require known vulnerabilities (that is, those with CVE-IDs assigned) to be fixed in a timely manner.

Because the only known fix for VU#425163 basically boils down to defense-in-depth, it was easier for many analysts to consider this as a weakness – and therefore deserving of a CWE entry – rather than a CVE-ID.

This situation does not resolve a final recommendation for or against assigning a CVE-ID to VU#425163. On the one hand, the CNA rules recommend assigning a CVE-ID. On the other hand, at least some professional vulnerability analysts think of it as a weakness and not a vulnerability. These tensions should be addressed in either outcome; it is unclear whether the CNA rules should change or the professional community intuitions should adapt.

6 THOUGHT EXPERIMENT 2: MODEL OBJECTS

This section explores a second hypothetical situation, if algorithm vulnerabilities such as VU#425163 are not assigned CVE-IDs, what if the the trained model objects themselves were assigned CVE-IDs?

Specifically, as discussed in Section 3.3, the trained model object within the ML system can enter a vulnerable state when the ML system interacts with its environment. Since the trained model object is a component of the software system that has a defined version, can persist for long periods of time, and can enter a vulnerable state, we explore the consequences of assigning trained model objects CVE-IDs. It is possible this scenario has happened; CVE-2019-8760 identifies a vulnerability in Apple's Face ID software that was fixed by "improving Face ID machine learning models." However, our thought experiment will cover ML models generally, not just models with a security function. Furthermore, Apple has not released details about its response to CVE-2019-8760, therefore to achieve a useful level of detail in the discussion we will exercise another thought experiment about known flaws in a popular model object.

As the specific example, consider Xu et al. [57] which generated an adversarial t-shirt pattern that successfully evaded the person detection capability of two COCO [30] trained object detectors, Faster R-CNN [44] and YOLO v2 [43]. Although Xu et al. [57] do not release their code nor specify precisely which trained model objects they used, there is sufficient detail in their paper that, at a minimum, the torchvision implementation of Faster R-CNN, available in a version pre-trained on COCO [55], and the darknet implementation of YOLOv2, available in a version pre-trained on COCO [42], are both vulnerable to the adversarial t-shirt pattern. For brevity, we refer to this candidate CVE-ID as CVE-tee.

The hypothetical CVE-tee will focus on these pre-trained model objects. However, this example makes space for further questions about to what a CVE-ID should be assigned. YOLOv2 is a framework for training a neural network on image data sets. In the terms of Figure 1, it is a model building and validation framework. Some aspects of YOLOv2 are fixed – the ML algorithm it trains is a single neural network. Many are configurable. Some aspects, such as the training data set, are configurable. The CVE-tee example focuses on the COCO data set. There are examples of problems localizable to the training data set – for example, Buolamwini and Gebru [4]. Whether a CVE-ID might be more productively assigned to the model building framework or the training data set are open questions for future work. This paper is focused on just two of the more extreme points in the possible space of where a CVE-ID might be assigned – the ML algorithm (Section 4) and a specific trained model object (this section).

Each of the following subsections examines our thought experiment through one of the CSIRT services withing vulnerability management, following our thought experiment in Section 4.

6.1 Vulnerability discovery / research

As in the prior thought experiment, vulnerability discovery will not change much beyond the substantial increase in the number of persons who should be counted as doing vulnerability discovery. For example, there are a growing collection of academic papers that already take as a starting place a publicly available model object and publish exploits that force those model objects into undesirable states, such as our motivating example for CVE-tee.

As in the current paradigm, not all vulnerable model objects will be assigned a CVE-ID, for reasons much the same as in the prior thought experiment. A particular concern for assigning trained model objects CVE-IDs is persistence of the the trained model. For example, some versioned models might be so short-lived as to not warrant CVE-ID assignment, such as during training itself or during internal development. The CVE rules do not specifically address a minimum time of existence to assign an ID, although there is mention of the need for public notification, which implies mass human-scale response times. That said, there are many trained model objects that have a persistence measured in years, notably the models released by popular deep learning frameworks, such as in torchvision and keras.io. The YOLOv2 exemplar models for CVE-tee, for example, has been available since 2016.

A second concern is that, as of this writing, all trained models are vulnerable to an attacker who is aware of the defense strategies used to defend the model [6, 56]. This implies that every machine learning system that is released would have open CVE-IDs associated with its trained model. Such mass assignment is not necessarily a negative thing, because it makes clear to both the machine learning community and the security community that using these models in situations that impact a given security policy is delicate matter that requires careful thought.

6.2 Vulnerability report intake

In the prior thought experiment, the focus was on assigning algorithms CVE-IDs. Section 4 has an underlying assumption that there are relatively few algorithms to study. In this thought experiment, we consider assigning CVE-IDs to trained model objects; this suggests several orders of magnitude more CVE-IDs be assigned because one widely used algorithm can produce many such vulnerable model objects.

For example, we identified two candidate models for CVE-tee, the official torchvision and darknet implementations of Faster R-CNN and YOLOv2, respectively. We chose those models out of familiarity. However, there are many additional model files available for download that have been pre-trained on COCO. For example, Faster R-CNN was published at Neurips in 2015, and their initial implementation was in MATLAB (https://github.com/ShaoqingRen/faster_rcnn). The authors later released a version in python (https://github.com/rbgirshick/py-faster-rcnn), and the success of their approach has led to multiple implementations in

pytorch, tensorflow, keras, etc. These model object versions may be considered similar enough in some important sense to receive the same CVE-ID. However, tracking all of the Faster R-CNN and YOLOv2 model objects trained on COCO and their derivatives is a non-trivial task.

An additional concern, as in the prior thought experiment, is prioritization of an ML vulnerability. Although the trained model is a more concrete product than an algorithm, there is still a wide variety of contexts in which the trained models might be used. The "worst case" reasoning of CVSS will lead to us giving this vulnerability, and presumably most CVE-IDs in trained models, a "critical" CVSS score (above 9.0). While this may be justified, it will cause friction during report intake if there is a large influx of high priority vulnerabilities.

6.3 Vulnerability analysis

As discussed in the prior thought experiment, root cause analysis and the development of mitigations are open research questions in the AML community.

The processes of understanding the root cause and developing countermeasures may be aided by having a more concrete model object and vulnerable state to focus on. For example, in the case of CVE-tee, the vulnerability report is a demonstration that two popular object detectors are fooled by a person wearing a particular pattern on their clothing, specifically a t-shirt. This leads to various mitigation strategies that may be more or less appropriate given the broader context of why an object detector is used to detect the presence of persons in a frame of video. These could range from social interventions, such as requiring persons to wear a specific uniform and enforcing that requirement without machine learning, to technical ones. A particular technical approach might be to move away from standard cameras and object detectors trained on COCO to infra-red cameras and object detectors trained on IR data, such as https://www.flir.com/oem/adas/adas-dataset-form/. Since the engineering involved in creating a thermally active adversarial pattern is harder than printing a t-shirt, this may be a useful mitigation.

6.4 Vulnerability coordination

As in the prior thought experiment, the AML community will likely react with confusion if a CVE-ID is assigned to a set of trained model objects, for much the same reasons as the confusion that would result from an algorithm being assigned a CVE-ID. Again, the general issue has been known for years, the vulnerabilities are already public, and the notification of vendors is difficult because there is no list of which vendors use which model objects (or their derivatives) for their products. Assigning CVE-tee a CVE-ID would not solve any of theses problems, but it might start some conversations that may build the trust and shared understanding necessary for coordination to begin.

Assigning CVE-IDs may also have a chilling effect on the willingness of researchers to share trained models with the public. If, for example, the assignment of a CVE-ID is perceived by the researcher as a negative mark upon their work, then this may make a given researcher less likely to release the code and trained models that make it easier for others to continue to move the field forward or put the results of research to innovative uses. Similarly, since the current state of AML is unable to isolate the root causes of a given vulnerability – that is, we do not know what lines of code to change or different algorithm to use – researchers may perceive the assignment of CVE-IDs as unnecessary, which may erode rather than build trust.

Risks of chilling effects notwithstanding, there is reason to believe that the AML and security communities can build the necessary relationships. First, the CSIRT community has been here before. For example, the medical device community is on a path to more mature vulnerability management. Roughly, this has meant building relationships between the CSIRT and medical device communities. That journey has included Food and Drug Administration (FDA) regulations on pre-market and post-market handling of vulnerabilities and the Health ISAC creating a community in which vulnerability management skills can be cultivated and encouraged [19]. Such relationship building has had fits and starts, but it provides historical lessons that could facilitate improved outcomes for connecting the AML and CSIRT communities. Second, there are high-profile examples within the AML community that attempt to perform CVD, such as OpenAI staging the release of its GPT-2 model over a period of six months (https://openai.com/blog/gpt-2-6-month-follow-up/). This community building will be hard, and regulation is not the right solution for every community, but ML is not going to become less important and so it is probably wise to start this hard work.

6.5 Vulnerability disclosure

As in the prior thought experiment, there is likely to be a conflict between the zero notification paradigm commonly practiced by the AML community and the CVD paradigm adopted by other security communities. This is not ameliorated by changing the level at which the CVE is assigned. It remains to be seen how this conflict will resolve. Specifically, it remains unclear if academic researchers, who are under significant pressure to share their results as quickly and widely as a possible, would be willing to wait to publish their exploits.

6.6 Vulnerability response

The response portion is where operations takes some action to prevent vulnerabilities from being exploited. As in the prior thought experiment, the steps are similar: identify which systems are vulnerable to which flaws, and then respond by mitigating those vulnerabilities as necessary. The modifications necessary to these processes when ML algorithms are assigned CVE-IDs are broadly similar to when model objects are assigned CVE-IDs. In both cases, it will be challenging to identify which systems have which vuls, either because the system was trained with a vulnerable algorithm or contains a model object that has a known vulnerability.

We believe the key difference will be in developing mitigations. For example, a CVE-ID assigned to VU#425163 could give only general advice. For example, Figure 1 indicates the adversarial pattern could be dealt with at a variety of levels. An operator could modify the environment so that the sensor is unlikely to encounter the adversarial pattern, or modify the sensor itself to make the pattern more difficult to produce, or add a run-time monitoring tool focusing on detecting such patterns, or modify the software components upstream from the trained model to filter out such patterns,

or modify the software components downstream of the trained model to ameliorate the effects of fooled inputs. In contrast, if a CVE-ID were assigned to a model object – and a particular threat to it, such as CVE-tee – the advice given could be more specific. For example, suggested mitigations could include modifying the social environment to enforce clothing norms that preclude such a pattern, investing in infrared sensors so that an attack would need to produce thermal patterns – which is much harder – to fool the sensor, etc. Such increased focus and reduced scope may lead to more productive conversations between the relevant stakeholders.

7 CONCLUSION

These changes to vulnerability management are simultaneously minor and revolutionary. Although there are important practical differences between assigning CVE-ID to algorithms versus model objects, the two thought experiments result in similar changes to the current vulnerability management paradigm.

From the following perspectives, the changes are minor.

- MITRE's guidance for CNAs need not change.
- The number of ML algorithm vulnerabilities would only be a small percentage increase over the 20,000 CVE-IDs assigned annually (in 2019); model object vulnerability assignments would create more CVE-IDs than algorithms but still not more than several hundred per year.
- The presence of these vulnerabilities in existing ML algorithms and model objects is well-known and repeatedly demonstrated.
- Within vulnerability management and security management more generally, it is normal for sector-specific groups (such as ISACs or Information Sharing and Analysis Organizations (ISAOs)) to form to handle sector-specific issues.
- Challenges in vulnerability management due to supply chain and asset management may be emphasized, but are not new.
- Current CVSS scoring norms struggle to adapt to various existing stakeholder communities, and it is not clear that AML is worse than other areas such as medical devices.

On the other hand, the following changes indicate a paradigm shift for either AML, vulnerability management, or both.

- Although cryptographic algorithms have had vulnerabilities in the past, algorithm-level vulnerabilities have been essentially unknown in the current vulnerability management regime.
- How to fix the vulnerable algorithms or defend model objects is not known, so any assigned CVE-IDs would be open issues for an unknown length of time.
- Asset owners and security policy folks who are fluent in vulnerability management do not currently speak a shared language with academic AML researchers.
- Engineering guidance for ML systems is nascent [20] and it is not clear how to handle supply chain documentation or asset management, including whether algorithms should be identified as assets; model objects are often not identified as assets in practice, though it would be easier to identify them than algorithms.
- Current CVSS scoring norms probably are not suitable for either algorithm or model-object vulnerabilities.
- AML publication timelines and CVD norms are in conflict.

Many of these changes are in tension; sometimes one aspect of a change is minor while from another perspective the same change is revolutionary. The thought experiment does not provide a strong recommendation for or against; like many problems in vulnerability management, it is nuanced and complicated. However, many of the major changes are probably things that would benefit both communities. So while they may be hard, they are desirable. From this perspective, the ML engineering, vulnerability management, and AML communities probably should build the appropriate bridges and communication infrastructure. Then the question is whether we can make the time.

ACKNOWLEDGMENTS

Copyright 2020 ACM. This material is based upon work funded and supported by the Department of Defense 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. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. CERT Coordination Center®is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

REFERENCES

[1] S Becker and Yann Lecun. 1989. Improving the convergence of back-propagation learning with second-order methods. In *Proceedings of the 1988 Connectionist Models Summer School, San Mateo*. Morgan Kaufmann, 29–37.

[2] Vilius Benetis, Olivier Caleff, Cristine Hoepers, Angela Horneman, Allen Householder, Klaus-Peter Kossakowski, Art Manion, Amanda Mullens, Samuel Perl, Daniel Roethlisberger, Sigitas Rokas, Mary Rossell, Robin M. Ruefle, D'esir'ee Sacher, Krassimir T. Tzvetanov, and Mark Zajicek. 2019. *Computer Security Incident Response Team (CSIRT) Services Framework*. Technical Report ver. 2. FIRST, Cary, NC, USA.

[3] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317–331.

[4] Joy Buolamwini and Timnit Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*. 77–91.

[5] Nicholas Carlini. 2020. A Complete List of All Adversarial Example Papers. https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html

[6] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *arXiv preprint arXiv:1902.06705* (2019).

[7] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. 2020. Cryptanalytic Extraction of Neural Network Models. *arXiv:2003.04884 [cs]* (July 2020). http://arxiv.org/abs/2003.04884 arXiv: 2003.04884.

[8] Danielle Keats Citron and Frank A. Pasquale. 2014. The Scored Society: Due Process for Automated Predictions. *Washington Law Review* 89, 8 (2014). https://ssrn.com/abstract=2376209

[9] CVE Board. 2020. *CVE Numbering Authority (CNA) rules*. Technical Report ver. 3.0. MITRE, Bedford, MA. https://cve.mitre.org/cve/cna/rules.html

[10] CVSS SIG. 2019. *Common Vulnerability Scoring System*. Technical Report version 3.1 r1. Forum of Incident Response and Security Teams, Cary, NC, USA. https://www.first.org/cvss/v3.1/specification-document

[11] Chad R Dougherty. 2008. VU#836068: MD5 vulnerable to collision attacks. https://kb.cert.org/vuls/id/836068 Accessed 2020-08-10.

[12] Vera Eidelman. 2018. The First Amendment Case for Public Access to Secret Algorithms Used in Criminal Trials. *Georgia State University Law Review* 34, 4 (August 2018).

[13] Golnoosh Farnadi, Behrouz Babaki, and Lise Getoor. 2018. Fairness in Relational Domains. In *AIES '18: Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, Jason Furman, Gary Marchant, Huw Price, and Francesca Rossi (Eds.). New Orleans, LA, USA. https://doi.org/10.1145/3278721.3278733

[14] David Fox, Eric Arnoth, Clement Skorupka, Catherine McCollum, and Deborah Bodeau. 2018. *Enhanced Cyber Threat Model for Financial Services Sector (FSS) Institutions.* Technical Report 18-1725. Homeland Security Systems Engineering and Development Institute, McLean, VA.

[15] Peter Galison. 1999. Trading zone: Coordinating action and belief. *The Science Studies Reader* (1999), 137–160.

[16] April Galyardt, Nathan M. VanHoudnos, and Jonathan M. Spring. 2020. *Comments on NISTIR 8269 (A Taxonomy and Terminology of Adversarial Machine Learning).* Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=637327

[17] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. 2012. Manufacturing Compromise: The Emergence of Exploit-as-a-service. In *Conference on Computer and Communications Security*. ACM, Raleigh, North Carolina, USA, 821–832.

[18] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv:1708.06733 [cs]* (March 2019). http://arxiv.org/abs/1708.06733 arXiv: 1708.06733.

[19] Health ISAC. 2019. Medical Device Security Media Education Materials. https://h-isac.org/cvd-media-kit/ Accessed Aug 18, 2020.

[20] Angela Horneman, Andrew Mellinger, and Ipek Ozkaya. 2019. *AI Engineering: 11 Foundational Practices.* Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[21] Allen Householder. 2015. Systemic Vulnerabilities: An Allegorical Tale of Steampunk Vulnerability to Aero-Physical Threats. https://www.youtube.com/watch?v=4AHpL3kVHw4

[22] Allen Householder, Jonathan M. Spring, Nathan VanHoudnos, and Oren Wright. 2020. Machine learning classifiers trained via gradient descent are vulnerable to arbitrary misclassification attack. https://kb.cert.org/vuls/id/425163/

[23] Allen D. Householder, Garret Wassermann, Art Manion, and Christopher King. 2020. *The CERT® Guide to Coordinated Vulnerability Disclosure.* Technical Report CMU/SEI-2017-TR-022. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. https://vuls.cert.org/confluence/display/CVD

[24] Michelle Jump and Art Manion. 2019. *Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM).* Technical Report. National Telecommunications and Information Administration, Washington, DC.

[25] Joshua A. Kroll, Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu. 2017. Accountable Algorithms. *U. PA. Law Review* (2017), 633–. https://scholarship.law.upenn.edu/penn_law_review/vol165/iss3/3

[26] Ram Shankar Siva Kumar, Magnus NystrÃ¶m, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. 2020. Adversarial Machine Learning – Industry Perspectives. arXiv:cs.CY/2002.05646

[27] Ram Shankar Siva Kumar, David R. O'Brien, Kendra Albert, and SalomÃ© Viljoen. 2018. Law and Adversarial Machine Learning. arXiv:cs.LG/1810.10731

[28] Ram Shankar Siva Kumar, David R. O'Brien, Kendra Albert, SalomÃ© Viljoen, and Jeffrey Snover. 2019. Failure Modes in Machine Learning Systems. *arXiv preprint* 1911.11034 (2019).

[29] David Lehr and Paul Ohm. 2017. Playing with the Data: What Legal Scholars Should Learn About Machine Learning. *UCDL Rev.* 51 (2017), 653.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr DollÃ¡r. 2015. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]* (Feb. 2015). http://arxiv.org/abs/1405.0312 arXiv: 1405.0312.

[31] Valentin Jean Marie Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. 2019. The art, science, and engineering of fuzzing: A survey. *IEEE Transactions on Software Engineering* (2019).

[32] Mitsuru Matsui. 1993. Linear Cryptanalysis Method for DES Cipher. In *Advances in Cryptology — EUROCRYPT (LNCS 765)*, Tor Helleseth (Ed.). Springer, Lofthus, Norway, 386–397.

[33] MITRE Corporation. 2010. Common Vulnerabilities and Exposures. http://cve.mitre.org. last access May 2, 2020.

[34] MITRE Corporation. 2014. Common Weakness Enumeration: A Community-Developed Dictionary of Software Weakness Types. http://cwe.mitre.org.

[35] Safiya Umoja Noble. 2018. *Algorithms of oppression: How search engines reinforce racism.* NYU Press, New York, NY.

[36] Cathy O'Neil. 2016. *Weapons of math destruction: How big data increases inequality and threatens democracy.* Broadway Books, New York, NY.

[37] OWASP Foundation. 2017. OWASP Top Ten. https://owasp.org/www-project-top-ten/ Accessed 2020-08-10.

[38] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, Rujun Long, and Patrick McDaniel. 2016. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv* 1610.00768 (2016).

[39] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. 2018. SoK: Security and privacy in machine learning. In *European Symposium on Security and Privacy*. IEEE, London, UK, 399–414.

[40] Wolter Pieters. 2011. The (social) construction of information security. *The Information Society* 27, 5 (2011), 326–335.

[41] Anne W Rawls and David Mann. 2010. *The Thing is What is Our 'What': An Ethnographic Study of a Design Team's Discussion of 'Object' Clarity as a Problem in Designing an Information System to Facilitate System Interoperability.* Technical Report 10-2594. MITRE Corp, McLean, VA, United States.

[42] Joseph Redmon. 2016. yolov2.weights. https://pjreddie.com/media/files/yolov2.weights Accessed Aug 10, 2020.

[43] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242* (2016).

[44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]* (Jan. 2016). http://arxiv.org/abs/1506.01497 arXiv: 1506.01497.

[45] R. Rivest. 1992. The MD5 Message-Digest Algorithm. RFC 1321 (Informational). , 21 pages. https://www.rfc-editor.org/rfc/rfc1321.txt Updated by RFC 6151.

[46] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[47] Stuart Russell, Daniel Dewey, and Max Tegmark. 2015. Research Priorities for Robust and Beneficiall Artificial Inteligence. *AI Magazine* 36, 4 (2015).

[48] Robert C Seacord. 2005. *Secure Coding in C and C++.* Pearson Education, Upper Saddle Ridge, NJ.

[49] R. Shirey. 2007. Internet Security Glossary, Version 2. RFC 4949 (Informational).

[50] Aditya K Sood and Richard J Enbody. 2013. Crimeware-as-a-service: A survey of commoditized crimeware in the underground market. *International Journal of Critical Infrastructure Protection* 6, 1 (2013), 28–38.

[51] Jonathan M. Spring, Joshua Fallon, April Galyardt, Angela Horneman, Leigh Metcalf, and Ed Stoner. 2019. *Machine Learning in Cybersecurity: A Guide.* Technical Report CMU/SEI-2019-TR-005. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=633583

[52] Jonathan M Spring, Eric Hatleback, Allen D. Householder, Art Manion, and Deana Shick. 2020. Prioritizing vulnerability response: A stakeholder-specific vulnerability categorization. In *Workshop on the Economics of Information Security.* Brussels, Belgium.

[53] Jonathan M Spring, Tyler Moore, and David Pym. 2017. Practicing a Science of Security: A philosophy of science perspective. In *New Security Paradigms Workshop.* ACM, Santa Cruz, CA, USA.

[54] Elham Tabassi, Kevin Burns, Michael Hadjimichael, Andres Molina-Markham, and Julian Sexton. 2019. *A Taxonomy and Terminology of Adversarial Machine Learning.* Technical Report Draft NISTIR 8269. National Institute of Standards and Technology, Gathersburg, MD, USA. https://csrc.nist.gov/publications/detail/nistir/8269/draft

[55] Torchvision. 2017. fasterrcnn_resnet50_fpn_coco. https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth Accessed Aug 10, 2020.

[56] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. 2020. On Adaptive Attacks to Adversarial Example Defenses. *arXiv:2002.08347 [cs, stat]* (Feb. 2020). http://arxiv.org/abs/2002.08347 arXiv: 2002.08347 version: 1.

[57] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. 2019. Adversarial T-shirt! Evading Person Detectors in A Physical World. *arXiv:1910.11099 [cs]* (Nov. 2019). http://arxiv.org/abs/1910.11099 arXiv: 1910.11099.

[58] James Zou and Londa Schiebinger. 2018. AI can be sexist and racistâ€"itâ€™s time to make it fair. *Nature* 559 (2018), 324–326.