Carnegie Mellon University
Software Engineering Institute

PRIORITIZING VULNERABILITY RESPONSE: A STAKEHOLDER SPECIFIC VULNERABILITY CATEGORIZATION

Jonathan M Spring, Eric Hatleback, Allen Householder, Art Manion, Deana Shick[†]

November 2019

Introduction

This report is the second part of a research agenda about prioritizing actions during vulnerability management. Many organizations use the Common Vulnerability Scoring System (CVSS) for this purpose today. For problems with CVSS as it is, see the first part of our research agenda: *Towards Improving CVSS*.[1] This report presents a testable Stakeholder-Specific Vulnerability Categorization (SSVC) that avoids some problems with CVSS. Our informed hypothesis takes the form of decision trees for different vulnerability management communities. We welcome others to test and improve it.

This report proposes a functional system to make our proposal concrete, as well as preliminary tests of its usefulness. However, our proposal is a detailed hypothesis to test, or a conversation starter, not a final proposal. In so far as is practical, we aim to avoid one-size-fits-all solutions. The stakeholders in vulnerability management are diverse, and that diversity needs to be accommodated in the main functionality, rather than squeezed into hard-to-use optional features.

We will improve vulnerability management by framing decisions better. The modeling framework determines what output types are possible, what the inputs are, what aspects of vulnerability management are in scope, what aspects of context are incorporated, how the model handles context and different roles, and what those roles should be. As such, the modeling framework is both important and difficult to pin down. We approach this problem as a satisficing process. We are not seeking optimal formalisms; we are seeking an adequate formalism. Others may have different satisfactory models, and that is okay.

The organizing concept of our decision-making procedure will be decision trees. A decision tree represents important elements of a decision and their possible values along with the decision that results from those values. We suggest decision trees as an adequate formalism for practical, wide-spread advice on vulnerability prioritization. We do not claim it is the only viable option. We will also suggest decision trees for use by specific vulnerability management stakeholder communities. These suggestions are hypotheses for a viable replacement for CVSS in those communities, but the hypotheses require empirical testing before they can be justifiably called fit for use. We propose a methodology for such testing.

[†] We thank Will Dormann, Madison Oliver, Vijay Sarvepalli, and Laurie Tyzenhaus for helpful comments on prior drafts.

[1] Jonathan Spring, Eric Hatleback, Allen D. Householder, Art Manion, Deana Shick. Towards Improving CVSS. Software Engineering Institute, Pittsburgh, PA. 2018. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=538368

Representing information for decisions about vulnerabilities

We take the fundamental concept in need of representation to be the *decision*. CVSS avoids talking about decisions and instead takes *technical severity* as its fundamental concept. We take severity's role to be to inform decision making about vulnerability management. The CVSS standard indicates vulnerability management decisions, and only those decisions, as what they expect CVSS scores to inform,[2] yet the standard does not provide clear advice on how CVSS scores might be used to inform decisions. Since some organizations are converting CVSSv3.0 base scores directly into decisions anyway, we propose that decisions, rather than severity, are a more useful output. Our requirements for an adequate decision-making process is that it should clearly define whose decisions are involved, properly use evidentiary categories, be based on reliably available evidence, be transparent, and be explainable.

When we consider decisions about managing the vulnerability, rather than just technical severity, we must be clear about *whose* decisions are involved. Organizations that produce patches and fix software clearly have different decisions to make than those that deploy patches or other security mitigations. Furthermore, organizations in the aviation industry may have different priorities than organizations that make word processors. We take this to indicate a requirement: any formalism must be able to capture adequately the different decisions and priorities exhibited by different stakeholder groups. And as a usability requirement, the number of stakeholder groups needs to be small enough to be manageable, both by those issuing scores and those seeking them.

Decisions are not numbers. Decisions are qualitative actions that an organization can take. In many cases, numerical values can be directly converted to qualitative decisions. For example, if your child's temperature is 105°F (40.5°C), you decide to go to the hospital. Conversion from numerical to qualitative values can be complicated by measurement uncertainty and design of the metrics. For example, CVSS scores were designed to be accurate with +/- 0.5 points of the given score.[3] If we take the recommended dividing line between high and critical – 9.0 – then it is unclear how to convert a CVSSv3.0 score of 8.9. For example, under a Gaussian error distribution, 8.9 is really 60% high, 40% critical. We want our decisions to be distinct and crisp, and statistical overlaps of scores within 1.0 units, for example, would muddy decision recommendations.

We propose to avoid numerical representations and consider only qualitative data as inputs and outputs for any vulnerability management decision process. Quantified metrics are more useful when (1) data for decision-making is available and (2) the stakeholders agree on how to measure. Vulnerability management does not yet meet either criterion. Furthermore, it is not clear to what extent measurements about a vulnerability can be informative about other vulnerabilities. Each vulnerability has a potentially unique relationship to the socio-technical system in which it exists, including the internet. The context of the vulnerability, and the systems it impacts, is inextricably linked to managing it. Temporal and environmental considerations should be primary, not optional.

---

[2] "CVSS provides a way to capture the principal characteristics of a vulnerability … reflecting its severity ... to help organizations properly assess and prioritize their vulnerability management processes." See https://www.first.org/cvss/.

[3] https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf.

We will make our deliberation process as clear as practical. Therefore, we risk belaboring some points to ensure our assumptions and reasoning are explicit. Transparency should improve trust in the results.

Finally, any result of a decision-making process should be *explainable*. Explainable is used with its common meaning. An explanation should make the process intelligible to an interested, competent, non-expert person. This meaning is not the same as "explainable" as used in the research area of explainable artificial intelligence. There are at least two reasons common explainability is important; first, for troubleshooting and error correction; and secondly, for justification of proposed decisions to others.

Summarizing, our conceptual requirements for informing the vulnerability management process are:

- Outputs are *decisions*
- Pluralistic recommendations among a manageable number of stakeholder groups
- Qualitative inputs
- Qualitative outputs, with no (unjustified) shifts to quantitative calculations
- Transparent justification of process
- Explainable results

Formalization options

This section will briefly survey the available formalization options against the six requirements described above. Table 1 summarizes the result. This survey is opportunistic, and it is based on conversations with several experts and our professional experience as authors. The search process certainly leaves open the possibility of missing a better option. However, at the moment, we are searching for a satisfactory formalism, rather than an optimal formalism. We only need to search until a satisfactory option is found. Thus, we focus on highlighting why some common options or suggestions do not meet the above criteria. We will argue that decision trees are a satisfactory formalism.

Many quantitative options are ruled out, such as anything involving statistical regression techniques or Bayesian belief propagation. Most machine learning (ML) algorithms are also not suitable, because they are both unexplainable in our sense and quantitative. Random forest algorithms may appear in scope, as each individual decision tree at least can be traced and the decisions explained.[4] However, how the available decision trees are created or mutated – and why a certain set of them work better – is not transparent enough. In any case, random forests are only necessary when decision trees get too complicated for humans to manage, which we believe we demonstrate below need not be the case for vulnerability management.

---

[4] Russell, Stuart J. & Norvig, Peter. Artificial Intelligence: A Modern Approach. 3rd edition. Prentice Hall. 2010. ISBN 9780136042594

Logics are generally better suited for capturing qualitative decisions. Boolean first-order logic is the "usual" logic with material implication (if/then), negation, existential quantification, and predicates. For example, SAT solvers and SMT solvers are used in program verification to automate decisions on when some condition holds, or whether software contains a certain kind of flaw. But while the explanations provided by logical tools are accessible to experts, non-experts may still struggle. We note, however, that under special conditions, logical formulae representing decisions about categorization based on exclusive-or conditions can be more compactly and intelligibly represented as a decision tree.

Table 1: Comparison of formalization options for vulnerability prioritization decisions

	Outputs are decisions	*Pluralistic suggestions*	*Qualitative inputs*	*Qualitative outputs*	*Transparent*	*Explainable*
Parametric regression	×	×	✓	×	×	✓
CVSS v3.0	×	×	✓	×	×	×
Bayesian belief networks	×	Maybe	×	×	✓	✓
Neural networks	×	×	×	×	×	×
Random forest	✓	✓	✓	Maybe	×	Maybe
Other machine learning	×	Maybe	×	×	×	×
Boolean 1st order logics	Maybe	Maybe	✓	✓	✓	Maybe
Decision trees (small)	✓	✓	✓	✓	✓	✓

Decision Trees

A decision tree is an acyclic, flowchart-like structure where nodes represent aspects of the decision or relevant properties and branches represent possible options for each aspect or property. Figure 1 is an example of a decision tree, where the decision is whether or not to eat cake. Each decision point can have more than two options, or have different options from other decision points, even though this example involves all homogeneous binary decisions.

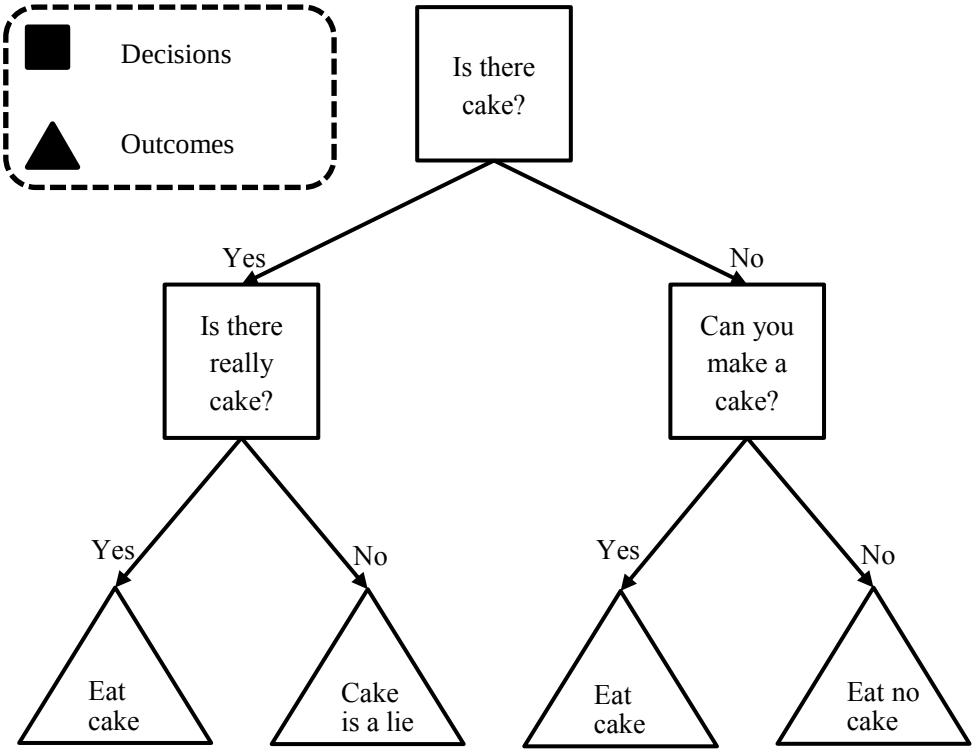

Figure 1: Example decision tree about whether to eat cake. Traditionally, squares represent decisions to be made and triangles are outcomes. Decision trees also always carry some assumptions, for example here one is that if cake is available, you will eat it. As with all decision trees, assumptions many need to be revisited occasionally and, if contested, moved to explicit decisions in the tree.

Decision trees can be used to meet all of the desired criteria described above. The two less-obvious criteria met by decision trees are plural recommendations and transparent tree-construction processes. Decision trees support plural recommendations simply because a separate tree can represent each stakeholder group. The opportunity for transparency surfaces immediately: any deviation among the decision trees for different stakeholder groups should have a documented reason, supported by public evidence when possible, for the deviation. Transparency may be difficult to achieve, since each node in the tree and each of the values needs to be explained and justified, but this cost is paid infrequently.

There has been limited but positive use of decision trees within vulnerability management. For example, Vulnerability Response Decision Assistance (VRDA) studies how to make decisions about how to respond to vulnerability reports.[5] This report continues roughly in the vein of such work to construct multiple decision trees for prioritization within the vulnerability management process.

[5] Burch H, Manion A, Ito Y, Vulnerability Response Decision Assistance (VRDA). Software Engineering Institute, Carnegie Mellon University. Jun 2007. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=51036

Decision trees for vulnerability management

Viable decision guidance for vulnerability management should, at a minimum, consider the stakeholder groups, their potential decision outcomes, and the data needed for relevant decision points. The following sections will address each of these parts in turn. These should be taken as instructive examples. While we strive to make the examples realistic, we suggest further community engagement and empirical assessment to test our examples. The following construction should be treated as an informed hypothesis rather than a conclusion.

Enumerating stakeholders

Stakeholders in vulnerability management can be identified within multiple independent axes. For example, by their responsibility: whether the organization *develops* patches, *applies* patches, or *coordinates* patches. Organizations may also be distinguished by type of industry sector. While it might be useful to enumerate all the sectors of the economy, we propose to draft decision points that include those from multiple important sectors. For example, we have safety-related questions in the decision path for all developers and appliers, so whether the stakeholder is in a safety-critical sector or not, the decision will be addressed.

This choice not to segregate the decisions by sector is reinforced by the fact that any given software system might be used by different sectors. It is less likely that one organization has multiple responsibilities within the vulnerability management process. Even if there is overlap within an organization, the two responsibilities are often located in distinct business units with distinct decision-making processes. We can treat the responsibilities as non-overlapping, and therefore we can build two decision trees – one for each of the "patch development" and "patch deployment" responsibilities in the vulnerability management process. We leave coordination as future work. Each of these trees will have different decision points that they take in to arrive at a decision about a given unit of work. The next two sections describe the decision space and all the relevant decision points, respectively, that we propose for these two responsibilities within the vulnerability management process.

Our target audience is professional staff responsible for making decisions about information systems. This audience includes a broad class of professionals, and includes developers, system maintainers, and administrators of many types. It also includes risk managers, technical managers, incident responders, and so on. Although every layperson who owns a computing device makes decisions about managing it, this is not our target audience. The following decision system may help such laypeople, but we are not intending it to be used by that audience. Relatedly, C-level executives and public policy often make, shape or incentivize decisions about managing information systems; however, this is not our target audience either. To the extent that decision trees for vulnerability management help higher-level policy decisions, we believe it will be by making the technical decisions more transparent and explainable to policy makers. While this policy-making audience may see indirect benefit, it is not our primary target and we are not designing it to be used by policy-makers directly.

Enumerating decisions

Stakeholders with different responsibilities in vulnerability management have largely different decisions to make. This section focuses only on the differences among organizations based on their vulnerability management responsibilities. Some decision makers may have different responsibilities in relation to different software. For example, an Android app developer is a developer of the app, but is a patch applier for any changes to the Android OS API. This situation is true for libraries in general. A web browser developer makes decisions about applying patches to DNS lookup libraries and TLS libraries. A video game developer makes decisions about applying patches released to the Unreal engine. A medical device developer makes decisions about applying patches for the Linux kernel. And so on. Alternatively, one might view applying patches as just de facto including some development and distribution of the updated product. Or one might take the converse view, that development de facto includes updating libraries. Either way, in each of these examples (mobile device apps, web browsers, video games, medical devices), we recommend that the professionals making the genuine decisions be explicit about what those decisions are, what they view their role(s) to be, and in relation to what software projects. If those decisions are explicit, then they can use whichever advice or structure from this document is relevant to them.

Developing patches — At a basic level, the decision at a software development organization is whether to issue a work order and what resources to expend to fix the vulnerability in the organization's software. Prioritization is required because, at least in the current history of software engineering, the effort to patch all known vulnerabilities will exceed available resources. The organization will consider several other factors to build the patch – refactoring a large portion of the code base may be necessary for some patches, while others require relatively small changes. We focus here just on the priority of building the patch, in four categories as displayed in Table 2.

Table 2: Proposed meaning for developer priority outcomes.

Developer Priority	Description
Defer	Do not work on the patch at present.
Scheduled	Develop a fix within regularly scheduled maintenance using developer resources as normal.
Out-of-band	Develop a fix out-of-band, taking resources away from other projects and releasing the fix as a security patch when it is ready
Immediate	Develop and release a fix as quickly as possible, drawing on all available resources, potentially including drawing on or coordinating resources from other parts of the organization.

Applying patches — Whether or not to apply an available patch is a binary decision. However, additional defensive mitigations may be necessary sooner than patches are available and may be advisable after patches have been applied. We recognize that vulnerability management actions are different when a patch is available and when it is not available. When a patch is available, the action usually is just to apply the patch. When there is not yet a patch the action space is more diverse, but should be some kind of mitigation of the vulnerability (i.e., shutting down services, additional security controls, etc.) or accepting the risk of not mitigating the vulnerability. In this report, we model the decision of "with what

priority should the organization take action on a given vulnerability management work unit," agnostic to whether a patch is available or not. A unit of work would mean either applying a patch or deploying a mitigation. Both patches and mitigations often remediate multiple identified vulnerabilities. The patch applier should answer the suggested questions for whatever unit of work they are considering as a whole, single unit. There is not necessarily a reliable function to aggregate a recommendation about a patch out of its constituent vulnerabilities. For the sake of simplicity of examples, we treat a patch as a patch of one vulnerability, and comment on any difficulty in generalizing our advice to a more complex patch where appropriate. Table 3 displays the action priorities for the patch applier, which are similar to the patch developer case.

Table 3: Proposed meaning for applier priority outcomes.

Applier Priority	Description
Defer	Do not act at present.
Scheduled	Act during regularly scheduled maintenance time.
Out-of-band	Act more quickly than usual to apply the fix out-of-band, during the next available opportunity, working overtime if necessary.
Immediate	Act immediately, focus all resources on applying the fix as quickly as possible, including, if necessary, pausing regular organization operations.

Coordinating patches — In coordinated vulnerability disclosure (CVD), the available decision is whether or not to coordinate a vulnerability report. VRDA provides a starting point for a decision tree for this situation.[6] VRDA is likely adequate for national-level CSIRTs doing general CVD, but other CSIRT types may have different needs than national-level teams. Future work may elicit those types and make a few different decision options. Specialized coordination organizations exist, for example ICS-CERT conducts CVD for safety-critical systems. We do not develop a coordination tree in this work, but future work could use our principles and design techniques to refine and evaluate VRDA or some other decision tree for coordinated vulnerability disclosure. The CERT guide to CVD provides something similar for those deciding how to report and disclose vulnerabilities they have discovered.[7]

Within each setting, the decisions are a kind of equivalence class for priority. That is, if an organization has three vulnerabilities to deploy patches for, and they are all "scheduled" priority, then the organization can decide which to do first however it likes. The priority is equivalent. This may feel uncomfortable, as CVSS gives the appearance of a finer-grained priority. CVSS appears to say not just 4.0 to 6.9 is "medium" severity, but that 4.6 is more severe than 4.5. However, as we discussed previously (see

[6] Burch H, Manion A, Ito Y, Vulnerability Response Decision Assistance (VRDA). Software Engineering Institute, Carnegie Mellon University. Jun 2007. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=51036

[7] Allen D. Householder, Garret Wassermann, Art Manion, Chris King. The CERT® Guide to Coordinated Vulnerability Disclosure. Section 6.10, https://vuls.cert.org/confluence/display/CVD/6.10+Troubleshooting+Coordinated+Vulnerability+Disclosure+Table.

page 2), CVSS is only designed to be accurate within +/- 0.5, and in practice is scored with errors of around +/- 1.5 to 2.5.[8] An error of this magnitude is enough to make all of the "normal" range from 4.0 to 6.9 equivalent, because 5.5 +/- 1.5 is the range 4.0 to 7.0. Our proposal is an improvement over this situation. CVSS errors often cross decision boundaries – that is, the error range often includes the transition between "high" and "critical" or "medium." Since our approach keeps the decisions qualitatively defined, this fuzziness does not creep in to our errors. Returning to the example of an organization with three vulnerabilities to patch of "normal" priority, in the system we propose they can be patched in any order. This is an improvement over CVSS, since based on the scoring errors CVSS was essentially just giving random fine-grained priorities within qualitative categories anyway. With our system, organizations can be more deliberate about conveniently organizing work that is of equivalent priority.

Scope

One important variable in the answers to all the below decision points is scope. There are at least two aspects to scope. One is how the boundaries of the affected system are set. A second is how far forward in time or causal steps one reasons about effects and harms. We put forward recommendations for both of these below. However, users of the decision process may want to define different scopes. Users may define a different scope as long as the scope is consistent across decisions, plausible, explicit, and accessible to all relevant decision makers.

Boundaries of the affected system

One distinction here will be whether the system of interest is software per se or a cyber-physical system. One problem is that in every practical case both are involved. Software is what has vulnerabilities, and what vulnerability management is focused on patching. Multiple pieces of software run on any given computer system. In order to consider software vulnerabilities at a useful scope, we follow prior work and say, for all intents and purposes, a vulnerability affects "the set of software instructions that executes in an environment with a coherent function and set of permissions".[9] This definition is useful because it lets us keep to common usage and intuition and call the Linux kernel, at least a specific version of it, "one piece" of software. But decision points about safety and mission impact are not about the software per se, they are about the cyber-physical system of which the software is a part. The term "physical" in cyber-physical system should be interpreted broadly; selling stocks or manipulating press outlet content are both best understood as affecting human social institutions. These social institutions do not have much of a corporeal instantiation, but they are physical in the sense that they are not merely software, and so part of cyber-physical systems.

---

[8] Allodi L, Cremonini M, Massacci F, Shim W. The Effect of Security Education and Expertise on Security Assessments: the Case of Software Vulnerabilities. In: WEIS 2018. See figure 1. The more accurate half of professionals estimated CVSS scores in ranges such as [+2,0] (that is, between overestimating by 2 to being correct), [+2,-2], and [0,-2].

[9] Spring J, Kern S, Summers A. Global adversarial capability modeling. APWG Symposium on Electronic Crime Research (eCrime) 2015 May 26. IEEE.

The hard part of delineating the boundaries of the affected system is specifying what it means for one system to be a part of another. Just because a computer is bolted to a wall does not mean the computer is part of the wall's purpose of separating physical spaces. At the same time, an off-premise DNS server may be part of the site security assurance system if the security cameras on-premises rely on the DNS server to connect to the displays at the guard's desk. We suggest that computer software is part of a cyber-physical system if the two systems are mutually manipulable, meaning that changes in the part (software) will, at least often, make detectable and relevant changes in the whole (cyber-physical system), and changes in the whole will, often, make relevant and detectable changes in the part.[10]

When reasoning about a vulnerability, we assign the vulnerability to the nearest, relevant, yet more abstract, discrete component. For example, this choice is particularly important when assessing technical impact on a component. This description bears some clarification, via each of the adjectives:

- *More abstract* means it's at least one level of abstraction above the specific location of the vulnerability. For example, if the vulnerability is localized to a single line of code in a function, then the function, the module, the library, the application, the product, and the system it belongs to are all *more abstract*.
- *Relevant* implies that the impacted component must be in the chain of abstraction moving upward from the location of the flaw.
- *Discrete* means there is an identifiable thing that can be fixed (e.g., it's the unit of patching).
- *Nearest* is relative to the abstraction at which the vulnerability exists.

Products, libraries, and applications tend to be appropriate objects of focus when seeking the right level to analyze the impact of a vulnerability. Hence when reasoning about the technical impact of a vulnerability localized to a function in a module in an open source library, the nearest relevant discrete abstraction is the library, because the patches are made available at the library level. Similarly, analysis of a vulnerability in closed source database software that supports an enterprise resource management system would consider the technical impact to the database, not to the ERM system.

Reasoning steps forward

This aspect of scope is about both immediacy, prevalence, and causal importance. Immediacy is about how soon after the decision point adverse effects should occur to be considered relevant. Prevalence is about how common adverse effects should be to be considered relevant. Causal importance is about how much relative contribution that exploitation of the software in the cyber-physical system should make to adverse effects to be considered relevant. Our recommendation is to walk a pragmatic middle path on all three aspects. Effects are not relevant if they are merely possible, too infrequent, far distant,

[10] Spring JM, Illari P. Building general knowledge of mechanisms in information security. Philosophy & Technology. 2018.

or unchanged by the vulnerability. But effects are relevant long before they are absolutely certain, ubiquitous, or occurring presently. Overall, we summarize this aspect of scope as *consider plausible effects based on known use-cases of the software system as a part of cyber-physical systems.*

Likely decision points and relevant data

We propose decision points and their values which should factor in to decisions in vulnerability prioritization. Each decision point is tagged with which stakeholder it is relevant to: patch appliers, patch developers, or both. We emphasize that these descriptions are hypotheses to be tested and validated. We have made every effort to put forward informed and useful decision frameworks with wide applicability, but the goal of this document is more to solicit feedback than a declaration. We welcome questions, constructive criticism, refuting evidence, or supporting evidence about any aspect of this proposal.

One important omission from the values for each category is an "unknown" option. Instead, we recommend explicitly identifying an option that is a reasonable assumption based on prior events. Such an option requires reliable historical evidence for what tends to be the case, and of course future events may require changes to these assumptions over time. Therefore, our assumptions both require evidence and are open to debate in light of new evidence. Different risk tolerance or risk discounting postures are not addressed in the current work, and accommodating such tolerance or discounting explicitly is an area for future work. This flexibility fits in to our overall goal of supplying a decision-making framework that is both transparent and fits different community's needs. Resisting an "unknown" option discourages the modeler from silently embedding these assumptions in their choices for how the decision tree flows below the selection of an "unknown" option, were we to have one.

Decisions might take in to account the following factors (in no particular order):

Exploitation (Developer, Applier)

Evidence of active exploitation of vulnerability.

The intent of this measure is the present state of exploitation of the vulnerability. Our intent is not to predict future exploitation, only to acknowledge the current state of affairs. Predictive systems such as EPSS could be used to augment this decision or to notify stakeholders of likely changes.[11]

Table 4: Exploitation decision values

None	No evidence of active exploitation, no public proof of concept (PoC) of how to exploit the vulnerability.
PoC (Proof of Concept)	Private evidence not shared, hearsay, or typical public PoC in places such as Metasploit or ExploitDB. The PoC condition also obtains if the vulnerability has a well-known method of exploitation. For example, open-source web prox-

[11] Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker. Improving Vulnerability Remediation Through Better Exploit Prediction. WEIS. Boston, MA. June 3, 2019.

	ies are the PoC code for how to exploit any vulnerability in the vein of improper validation of TLS certificates. As another example, Wireshark serves as a PoC for packet replay at-tacks on ethernet or WiFi networks.
Active	Shared, observable, reliable evidence of exploit being used in the wild by real attackers; credible public reporting.

Technical Impact (Developer)

Technical impact of exploiting the vulnerability.

When evaluating technical impact, recall the scope definition above. Total control is relative to the affected component in which the vulnerability resides. If a vulnerability discloses authentication or authorization credentials to the system, this information disclosure should also be scored as "total" if those credentials give an adversary total control of the component.

Table 5: Technical Impact decision values

Partial	Exploit gives adversary *limited* control over, or information exposure about, behavior of the software which contains the vulnerability. Or the exploit gives the adversary an importantly low stochastic opportunity for total control. In this context, "low" means that the attacker cannot reasonably make enough attempts to overcome the low chance of each attempt not working. Denial of service is a form of limited control over the behavior of the vulnerable component.
Total	Exploit gives adversary *total* control over behavior of the software, or total disclosure of all information on the system, which contains the vulnerability

Utility (Developer, Applier*)

How useful the exploit is to the adversary.

Heuristically, we base utility on a combination of value density of vulnerable components and virulence of potential exploitation. This framing has the benefit that often one can analytically derive these categories from a description of the vulnerability and the affected component. The meaning of virulence (slow or rapid) and value density (diffuse or concentrated) are defined below. Roughly, the utility is a combination of the value of each exploitation event with the ease and speed with which the adversary can cause exploitation events. So that overall, we define utility as:

Table 6: Utility decision values

Laborious	Slow virulence and diffuse value

* Appliers only use this feature as a suggested constraint on the values for *mission impact*.

Efficient	{rapid virulence and diffuse value} OR {slow virulence and concentrated value}
Super Effective	Rapid virulence and concentrated value

Virulence

- Slow: Steps 1-4 of the kill chain[12] cannot be reliably automated for this vulnerability for some reason. These steps are reconnaissance, weaponization, delivery, and exploitation. Example reasons why a step may not be reliably automatable include that the vulnerable component is not searchable or enumerable on the network, weaponization may require human direction for each target, delivery may require channels that widely-deployed network security configurations block, and exploitation may be frustrated by adequate exploit prevention techniques enabled by default (ASLR is an example exploit prevention tool).
- Rapid: Steps 1-4 of the of the kill chain can be reliably automated. If the vulnerability allows remote code execution or command injection, the default response should be rapid.

Value Density

- Diffuse: The system which contains the vulnerable component has limited resources. That is, the resources which the adversary will gain control over with a single exploitation event are relatively small. Examples of systems with diffuse value are email accounts, most consumer online banking accounts, common cell phones, and most personal computing resources owned and maintained by users. A "user" is anyone whose professional task is something other than the maintenance of the system or component. As with *safety impact*, a "system operator" is anyone who is professionally responsible for the proper operation or maintenance of a system.
- Concentrated: The system which contains the vulnerable component is rich in resources. Heuristically, such systems are often the direct responsibility of "system operators" rather than users. Database systems, Kerberos servers, web servers hosting log-in pages, and cloud service providers are examples of concentrated value. However, usefulness and uniqueness of the resources on the vulnerable system also inform value density. For example, encrypted mobile messaging platforms may have concentrated value not because each phone's messaging history has a particularly large amount of data, but because it is uniquely valuable to law enforcement.

The output for the *utility* decision point can also be visualized as Table 7.

Table 7: Utility to the adversary, as a combination of virulence and value density

Virulence	Rapid	Efficient	Super effective

[12] Hutchins EM, Cloppert MJ, Amin RM. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research. 2011;1(1):80.

Slow	Laborious	Efficient
	Diffuse	Concentrated
	Value Density	

Alternative heuristics for proxying adversary utility are plausible. One such example is the value the vulnerability would have were it sold on the open market. Some firms, such as Zerodium,[13] make such pricing structures public. The valuable exploits track our virulence and value density heuristics for the most part. Within a single system, whether it is Apache, Windows, iOS or WhatsApp, more kill chain steps automated leads to higher exploit value. Remote code execution with sandbox escape and without user interaction are the most valuable exploits, and those features describe automation of the relevant kill chain steps. How equivalently virulent exploits for different systems are priced relative to each other is more idiosyncratic. It does not only track value density of the system, but presumably also the existing supply of exploits and the installation distribution among the targets of Zerodium's customers. Currently, we simplify the analysis and ignore these factors. However, future work should track, look for, and prevent large mismatches between the outputs of the *utility* decision point and exploit markets.

Safety impact (Developer, Applier)

Safety impacts of affected system compromise.

We take an expansive view of safety, in which a safety violation is a violation of what the Centers for Disease Control (CDC) calls "well-being."[14] Physical well-being violations are common safety violations, but we also include economic, social, emotional, and psychological well-being as important. Weighing fine differences among these categories is probably not possible, so we will not try. Each decision option lists examples of the effects that qualify for that value/answer in the various types of violations of well-being. These examples should not be considered comprehensive or exhaustive, but rather as suggestive.

The stakeholder should consider the safety impact on the operators and users of the software they provide. If software is repackaged and resold by a stakeholder to further downstream entities who will then sell a product, the initial stakeholder can only reasonably consider so many links in that supply chain. But a stakeholder should know its immediate consumers one step away in the supply chain. Those consumers may repackage or build on the software and then provide that product further on. We expect that a stakeholder should be aware of common usage of their software about two steps in the supply chain away. This expectation holds in both open-source and proprietary contexts. Further steps along the supply chain are probably not reasonable for the stakeholder to consider consistently; however, this is not license to be willfully ignorant of common downstream uses of the stakeholder's software. If the stakeholder is contractually or legally responsible for safe operation of the software or cyber-physical system

[13] https://zerodium.com/program.html

[14] See CDC, "How is well-being defined?" https://www.cdc.gov/hrqol/wellbeing.htm#three, within the Health-related quality of life (HRQOL) part of the CDC. Aug 13, 2019.

of which it is part, that also supersedes our rough supply-chain depth considerations here. For software used in a wide variety of sectors and deployments, the stakeholder may need to estimate some sort of aggregate safety impact. Aggregation suggests that the stakeholder's response to this decision point cannot be less than the most severe plausible safety impact, but we leave the specific aggregation method or function as a domain-specific extension of our work here. When we say "system operator" we mean those who are professionally responsible for the proper operation of the cyber-physical system, as the term is used in the safety analysis literature.

Table 8: Safety Impact decision values

Safety Impact[15]	**Type of harm**	**Description**
None	all	This response does not mean literally no impact, just that the effect is below the threshold for all aspects described in Minor.
Minor (any one or more of these conditions hold)	Physical harm	Physical discomfort for users (not operators) of the system.
	Operator resiliency	Requires action by system operator to maintain safe system state as a result of exploitation of the vulnerability where operator actions would be well within expected operator abilities; OR causes a minor occupational safety hazard.
	System resiliency	Small reduction in built-in system safety margins; OR small reduction in system functional capabilities that support safe operation.
	Environment	Minor externalities (property damage, environmental damage, etc.) pushed on other parties.
	Financial	Financial losses, which are not readily absorbable, to multiple persons.
	Psychological	Emotional or psychological harm, sufficient to be cause for counselling or therapy, to multiple persons.
Major (any one or more of these conditions hold)	Physical harm	Physical distress and injuries for users (not operators) of the system
	Operator resiliency	Requires action by system operator to maintain safe system state as a result of exploitation of the vulnerability where operator actions would be within their capabilities but requires their full attention and effort; OR significant distraction or discomfort to operators; OR causes significant occupational safety hazard.
	System resiliency	System safety margin effectively eliminated but no actual harm; OR failure of system functional capabilities that support safe operation.

[15] These categories are based on hazard categories for aircraft software (see DO-187C, https://en.wikipedia.org/wiki/DO-178C, and also Section 3.3.2 of the FAA System Safety Handbook, Dec 2000, https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/media/Chap3_1200.pdf.

	Environment	Major externalities (property damage, environmental damage, etc.) pushed on other parties.
	Financial	Financial losses likely to lead to bankruptcy of multiple persons.
	Psychological	Widespread emotional or psychological harm, sufficient to be cause for counselling or therapy, to populations of people.
Hazardous (any one or more of these conditions hold)	Physical harm	Serious or fatal injuries, where fatalities are plausibly preventable via emergency services or other measures.
	Operator resiliency	Actions that would keep the system in a safe state are beyond system operator capabilities, resulting in adverse conditions; OR great physical distress to system operators such that they cannot be expected to operate the system properly.
	System resiliency	Parts of the cyber-physical system break, but the system's ability to recover lost functionality remains intact.
	Environment	Serious externalities (threat to life as well as property, widespread environmental damage, measurable public health risks, etc.) pushed on other parties.
	Financial	Socio-technical system (elections, financial grid, etc.) of which the affected component is a part is actively destabilized and enters unsafe state.
	Psychological	N/A
Catastrophic (any one or more of these conditions hold)	Physical harm	Multiple immediate fatalities (emergency response probably cannot save the victims).
	Operator resiliency	Operator incapacitated (includes fatality or otherwise incapacitated).
	System resiliency	Total loss of whole cyber-physical system of which the software is a part.
	Environment	Extreme externalities (immediate public health threat, environmental damage leading to small ecosystem collapse, etc.) pushed on other parties
	Financial	Social systems (elections, financial grid, etc.) supported by the software collapse.
	Psychological	N/A

Advice for gathering information to answer the safety impact question

The factors that influence the safety impact level are diverse. This document generally leaves how a stakeholder should answer a question as future work. At a minimum, understanding safety impact should include gathering information about survivability of the vulnerable component, available operator actions to compensate for the vulnerable component, relevant insurance, and viability of existing back-up measures. Each of these information items depends heavily on domain-specific knowledge, and so it is out of scope to give a general-purpose strategy for how they should be

included. For example, viable manual backup mechanisms are likely important in assessing the safety impact of an industrial control system in a sewage plant, but in banking the insurance structures that prevent bankruptcies are more important.

Exposure (Applier)

What is the accessible attack surface of the affected system or service?

Measuring attack surface precisely is difficult, and we do not propose to perfectly delineate between small and controlled access. If a vulnerability cannot be patched, other mitigations may be used. The effect of these mitigations is usually to reduce exposure of the vulnerable component. Therefore, an applier's response to Exposure may change if such mitigations are put in place. If a mitigation changes exposure and thereby reduces the priority of a vulnerability, that mitigation can be considered a success. Whether that mitigation allows the applier to defer further action will vary case by case.

Table 9: Exposure decision values

Small	Local service or program; highly-controlled network.
Controlled	Networked service with some access restrictions already in place (whether locally or on the network). A successful mitigation must reliably interrupt the adversary's attack, which requires the attack is detectable both reliably and quickly enough to respond. *Controlled* covers the situation in which a vulnerability can be exploited through chaining it with other vulnerabilities. The assumption is that the number of steps in the attack path is relatively low; if the path is long enough that it is implausible for an adversary to reliably execute it, then *exposure* should be *small*.
Unavoidable	Internet or another widely-accessible network where access cannot plausibly be restricted or controlled. For example, DNS servers, web servers, VOIP servers, email servers, etc.

Mission impact (Applier)

Impact on the mission essential functions[16] of the organization.

A mission essential function (MEF) is a function "directly related to accomplishing the organization's mission as set forth in its statutory or executive charter" (FEMA pg A-1). Identifying MEFs is part of business continuity planning or crisis planning. The difference between MEFs and non-essential functions is, roughly, that an organization "must perform a [MEF] during a disruption to normal operations"

[16] For information about identification of mission essential functions, see U.S. Department of Homeland Security, Federal Emergency Management Agency. Federal Continuity Directive 2: Federal Executive Branch Mission Essential Functions and Candidate Primary Mission Essential Functions Identification and Submission Process. June 13, 2017. https://www.fema.gov/media-library-data/1499702987348-c8eb5e5746bfc5a7a3cb954039df7fc2/FCD-2June132017.pdf

(FEMA pg B-2). The mission is the reason an organization exists, and MEFs are how that mission is effected. Non-essential functions do not directly support the mission per se; however, non-essential functions support the smooth delivery or success of MEFs. Financial losses, especially to publicly-traded for-profit corporations, would be covered here as a (legally mandated) mission of such corporations is financial performance.

None	Little to no impact.
Non-essential degraded	Degradation of non-essential functions, where chronic degradation would eventually harm essential functions.
MEF support crippled	Activities that directly support the essential functions are crippled, but essential functions can continue for a time.
MEF failure	Any one mission essential function fails for a period of time longer than acceptable. The overall mission of the organization is degraded but can still be accomplished for a time.
Mission failure	Multiple or all mission essential functions fail and ability to recover those functions degraded. The organization's ability to deliver its overall mission fails.

Advice for gathering information to answer the mission impact question

The factors that influence the mission impact level are diverse. This document generally leaves how a stakeholder should answer a question aside. At a minimum, understanding mission impact should include gathering information about the critical paths that involve vulnerable components, viability of contingency measures, and resiliency of the systems that support the mission. There are various sources of guidance on how to gather this information; see for example the FEMA guidance in Continuity Directive 2 or OCTAVE FORTE.[17] This is part of risk management more broadly. It should require the vulnerability management team to interact with more senior management to understand mission priorities and other aspects of risk mitigation.

As a heuristic, we suggest using the Utility (Developer, Applier) question to constrain *mission impact*. If *utility* is super-effective, then mission impact is at least "MEF support crippled." If *utility* is efficient, then mission impact is at least "Non-essential degraded."

On the cost of patching

Our method is for prioritizing vulnerabilities based on the risk stemming from exploitation. Costs, the other side of the risk equation, are out of our scope. There are at least three aspects of cost that are out of scope. First and most obvious is the transaction cost of applying the patch. System administrators are paid for their time doing the work to develop or apply the patch, and this may require other transactional costs such as downtime for updates. Second is the cost, represented as a risk of failure, that the patch introduces some new error or vulnerability. Regression testing is part of what developers pay related to

[17] Brett Tucker. OCTAVE® FORTE and FAIR Connect Cyber Risk Practitioners with the Boardroom. 6/21/2018. https://insights.sei.cmu.edu/insider-threat/2018/06/octave-forte-and-fair-connect-cyber-risk-practitioners-with-the-boardroom.html

this type of cost. Finally, there may be an operational cost of applying a patch, representing an ongoing change of functionality or increased overhead. Cost may be one way a decision maker orders work within one priority class (scheduled, out-of-band, etc.).

Once you fix all the high priority vulnerabilities, you fix the mediums with the same effort you were spending on the highs.

Patch developer tree

Figure 2 displays our proposed prioritization decision tree for the patch developer. Both developer and applier trees use the above decision point definitions. Each tree is a compact way of expressing assertions or hypotheses about the relative priority of different situations. Each tree organizes how we propose a stakeholder should treat these situations. Rectangles are decision points, and triangles represent outcomes. The values for each decision point are different, as described above. Outcomes are priority decisions (defer, scheduled, out-of-band, immediate). Outcome triangles are color coded.

- Defer = gray with green outline
- Scheduled = yellow
- Out-out-band = orange
- Immediate = red with black outline

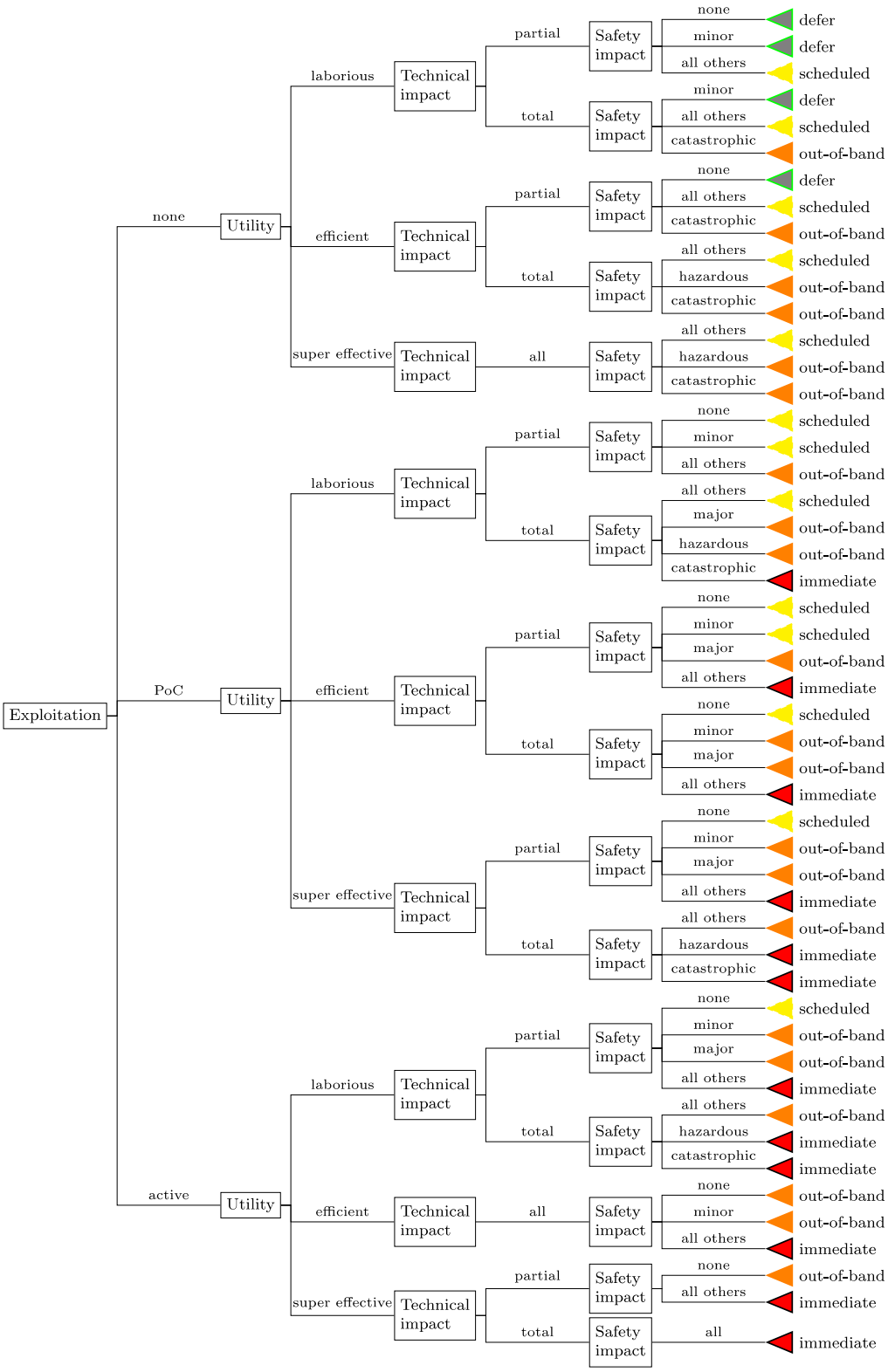

Figure 2: Proposed vulnerability prioritization decision tree for patch developers.

Patch applier tree

Our proposed patch applier tree spans Figure 3, Figure 4, and Figure 5. The state of *exploitation* is the first decision point, but in an effort to make the tree legible we have split the tree into three sub-trees over three pages. Make the decision about *exploitation* as usual, and then go to the correct subtree.

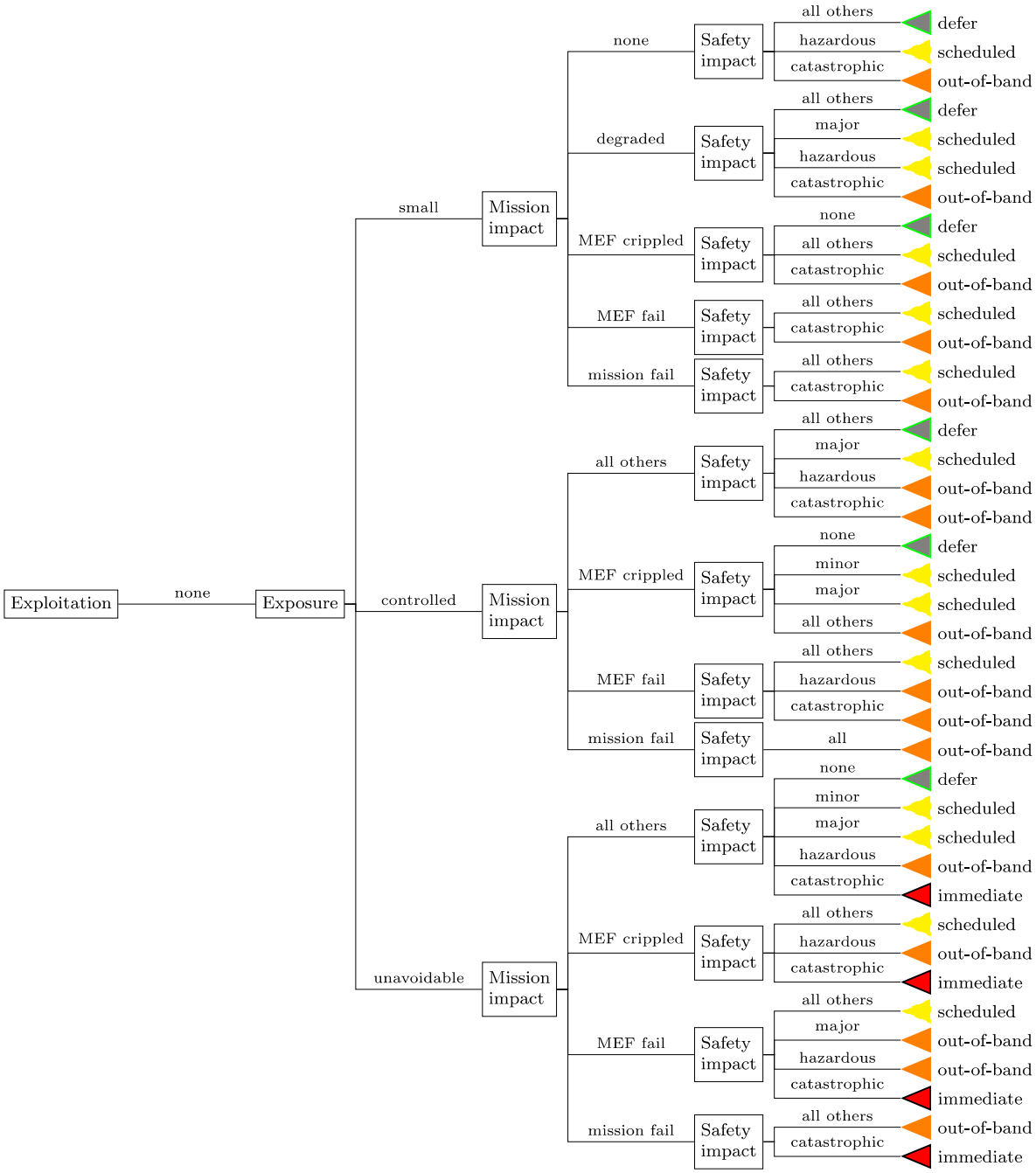

Figure 3: Proposed vulnerability prioritization decision tree for patch appliers (continued in Figure 4 and Figure 5).

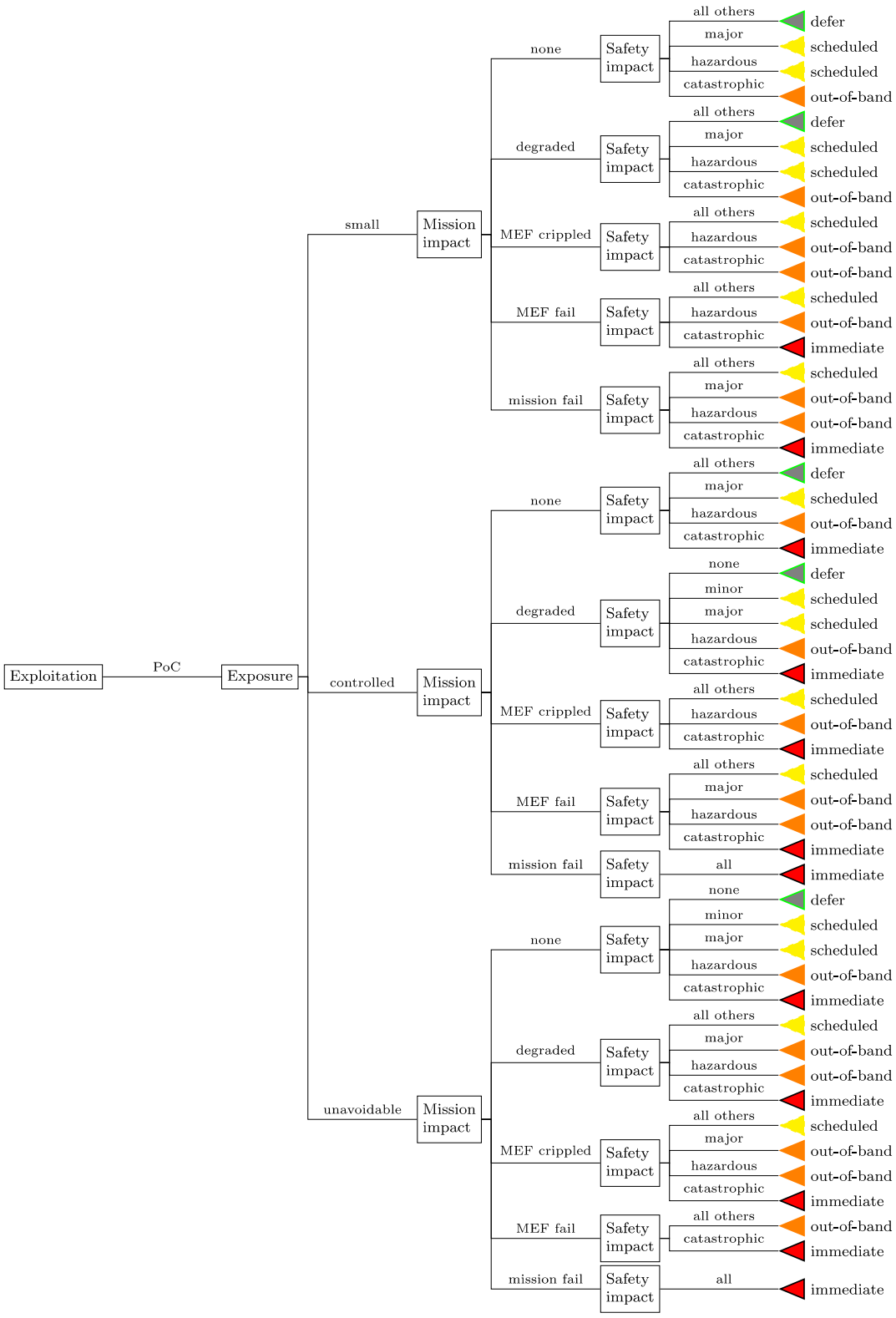

Figure 4: Proposed vulnerability prioritization decision tree for patch appliers (continued in Figure 3 and Figure 5).

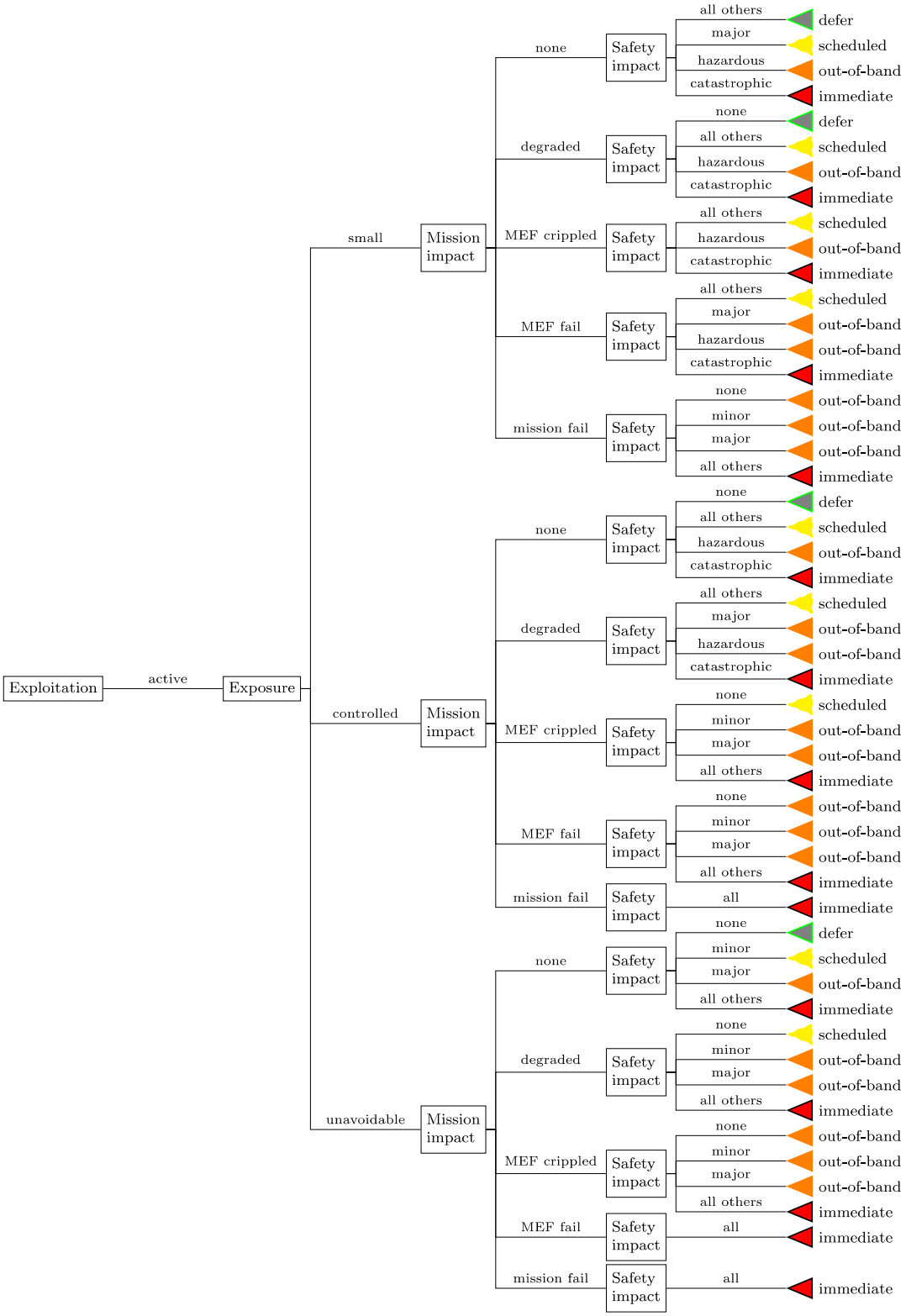

Figure 5: Proposed vulnerability prioritization decision tree for patch appliers (continued in Figure 3 and Figure 4).

Evidence gathering guidance

A patch developer or patch applier should, as much as possible, have a repeatable evidence collection and evaluation process to answer each of these decision points. However, we are proposing decisions for humans to make, so the evidence collection and evaluation is not totally automatable. That caveat notwithstanding, some automation is possible.

For example, whether exploitation modules are available in ExploitDB, Metasploit, or other sources is straightforward. We hypothesize that searching Github and Pastebin for exploit code should be automatable. A developer or applier could then define "exploit PoC available" to be positive search results for a set of inputs derived from the CVE entry in at least one of these venues. At least, for those vulnerabilities that are not "automatically" PoC-ready, such as TLS middleperson attacks or network replays.

Some of the decision points require some substantial upfront analysis effort to gather risk assessment or organizational data. However, once gathered this information can be efficiently reused across many vulnerabilities and only refreshed occasionally. An obvious example of this is the mission impact decision point. To answer this, a patch applier will have to have done an analysis of their essential functions, how they interrelate, and how they are supported. Exposure is similar; answering that decision point requires an asset inventory, adequate understanding of the network topology, and a view of the enforced security controls. Independently operated scans, such as Shodan or Shadowserver, may have a role to play in evaluating exposure, but the whole exposure question cannot be pushed to a binary question of whether an organization's assets appear in those databases. Once the applier has the situational awareness to understand mission essential functions or exposure, selecting the answer for each individual vulnerability is usually straightforward.

Stakeholders who use the prioritization method should consider releasing the priority with which they handled the vulnerability. This disclosure has various benefits. For example, if the developer publishes a priority ranking, then appliers could consider that in their decision-making process. One reasonable way to include it is to break ties for the applier. If an applier has three "scheduled" vulnerabilities to patch, they may address them in any order. If two were produced by the developer as "scheduled" patches, and one "out-of-band," then the applier may want to use that information to favor the latter.

In the case where no information is available or the organization has not yet matured its initial situational analysis, we can suggest something like defaults for some decision points. If the applier does not know their exposure, that means they do not know where the devices are or how they are controlled, and so they should assume exposure is unavoidable. If the decision maker really knows nothing about the environment in which the device is used, we suggest assuming a major safety impact. This position is conservative, but software is thoroughly embedded in daily life now so we suggest that the decision maker needs to provide evidence no one's well-being will suffer. The reach of software exploits is no longer limited to a research network. Mission impact, similarly, the applier should assume that the software is in use at the organization for a reason and that it supports essential functions unless they have evidence otherwise. With a total lack of information, assume MEF support crippled as a default. Exploitation needs no special default, if adequate searches are made for exploit code and none is found, the answer is none. The decision set {none, unavoidable, MEF crippled, major} results in a scheduled patch application.

Development methodology

Our initial starting point for the decision trees was rather different than what is presented here. We initially hypothesized different trees for different sectors (safety-critical, highly-regulated, and everyone else), for example. The initial trees also included additional decision points, such as developer's patch distribution channels and the financial loss to the applier of the vulnerability being exploited. We conducted informal evaluations of these trees by selecting a past CVE and discussing how each of the authors would evaluate the priority of that vulnerability. This method quickly surfaced some problems, and we iterated this tabletop exercise until broad-scope problems stopped blocking our informal evaluations. We quickly reorganized the trees for different sectors into just one tree per role, for example, but with the new trees always asking the safety impact question. We also elaborated assumptions about scope and what safety and mission impact mean during this process. During this process we also decided to focus on decision trees for the patch developer and patch applier, and leave the coordination decision as future work.

For this tabletop refinement, we could not select a mathematically representative set of CVEs. Our goal was to select a handful that would cover diverse types of vulnerabilities. The CVEs that we used for our tabletop exercises are CVE-2017-8083, CVE-2019-2712, CVE-2014-5570, and CVE-2017-5753. We discussed each of these from the perspective of patch developer and patch applier. We evaluated CVE-2017-8083 twice, because our understanding and descriptions had changed materially after the first three CVEs (six evaluation exercises). After we were happy that the decision trees were clearly defined and captured our intentions, we began the formal evaluation of the draft trees described in the next section.

Evaluation of the draft trees

We conducted a pilot test on the adequacy of our hypothesized decision trees. The method of the pilot test is described in the Pilot methodology subsection. The study resulted in several changes to our hypothesized trees. We capture those changes and the reason for each of them in the Pilot results subsection. The trees in sections Patch developer tree and Patch applier tree include the improvements reported in the section Improvements instigated by the pilot.

Pilot methodology

The pilot study tested inter-rater agreement of decisions reached. Each author played the role of an analyst in both stakeholder groups (developing patching and applying patches) for nine vulnerabilities. We selected these nine vulnerabilities based on expert analysis, with the goal that the nine cases cover a useful series of vulnerabilities of interest. Specifically, we selected three vulnerabilities to represent safety-critical cases, three to represent regulated-systems cases, and three to represent general computing cases.

During the pilot, we had not formed guidance on how to assess the values of the decision points. However, we did standardize the set of evidence that was taken to be true for the point in time representing the evaluation. Given this fixed information sheet, we did not synchronize an evaluation process for

how to decide whether *exploitation*, for example, should take the value of none, PoC, or active. We had agreed on the descriptions of the decision points and the descriptions of their values in (a prior version of) section ***Error! Reference source not found.***. The goal of the pilot was to test the adequacy of those descriptions by evaluating whether the analysts agreed. We improved the decision point descriptions based on the results of the pilot. Our changes are documented in the section Improvements instigated by the pilot.

In the design of the pilot we held constant the information available about the vulnerability. This strategy restricted the analyst to decisions based on our framework given that information. That is, it controlled for differences in information search procedure among the analysts. The information search procedure should be controlled because this pilot was about the framework content, not how people answer questions based on that content. A separate study with an aim to advise on how analysts should answer the questions in the framework should be devised after the framework is more stable. Our basis for this assessment that information search will be an important aspect in using the evaluation framework is based on our experience while developing the framework. During informal testing, often disagreements about a result involved a disagreement about what the scenario actually was; different information search methods and prior experiences led to different understandings of the scenario. This pilot methodology holds the information and scenario constant to test agreement about the descriptions themselves. This strategy makes constructing a prioritization system more tractable by separating problems in how people search for information from problems in how people make decisions. This report focuses only on the structure of decision making. Advice on how to search for information about a vulnerability is a separate project we leave for future work. In some domains, namely exploit availability, we have started that work in parallel.

The structure of the pilot test is as follows. Table 10 is an example of the information provided to each analyst. The developer portfolio details are struck out because this decision item was removed following the pilot. The decision procedure for each case is as follows: for each analyst, for each vulnerability, for each stakeholder group, do the following. Start at the root note of the relevant decision tree (patch applier or patch developer). Document the decision branch you think matches the vulnerability for this stakeholder context. Document the evidence that supports that decision. Repeat this decision-and-evidence process until the analyst reaches a leaf node in the tree.

Table 10: Example of scenario information provided to analysts (using CVE-2019-9042 as the example)

Information item	Description provided to analysts
Use of cyber-physical system	Sitemagic's CMS seems to be fairly popular amongst smaller businesses because it starts out with a free plan to use it. Then it gradually has small increments in payment for additional features. It's ease-of-use, good designs, and one-stop-shop for businesses attracts a fair number of clients. Like any CMS, it "manages the creation and modification of digital content. These systems typically support multiple users in a collaborative environment, allowing document management with different styles of governance and workflows. Usually the content is a website." (https://en.wikipedia.org/wiki/Content_management_system)

State of exploitation	Appears to be active exploitation of this vulnerability according to NVD. They have linked to the exploit here, http://www.iwantacve.cn/index.php/archives/116/.
~~**Developer portfolio details**~~	~~Sitemagic is an open-source project. The only thing the brand name applies to is the CMS, and it does not appear to be part of another open-source umbrella. The project is under active maintenance (it's not a dead project).~~
Technical impact of exploit	An authenticated user can upload a .php file to execute arbitrary code with system privileges.
Scenario blurb	We are a small business that uses Sitemagic to help run business. Sitemagic handles everything from digital marketing and site design to facilitating the e-commerce transactions of our website. We rely on this website heavily, even though we do have a brick-and-mortar store. Many times, products are not available in-store, but are available online, so we point many customers to our online store.
Applier mission	We are a private company that must turn a profit to remain competitive. We have a desire to provide our customers with a valuable product at a reasonable price, while still turning a profit to run our business. As we are privately held (and not public), we are free to choose the best growth strategy (we do not legally bound to demonstrate quarterly earnings for shareholders, we can take a longer-term view if it makes us competitive).
Applier deployment of affected system	We have deployed this system in such that only our web designer Cheryl and our IT admin Sally are allowed to access the CMS as users. They log in through a password-protected portal that can be accessed anywhere in the world, for remote administration. The CMS publishes content to the web, and that web server / site are publicly available.

This test structure produced a series of lists similar in form to Table 11. Analysts also noted how much time they spent on each vulnerability in each stakeholder group.

Table 11: Example documentation of a single decision point

Decision point	Branch selected	Supporting evidence
Applier tree; exploitation=active	Controlled	The CMS has a limited number of authorized users and the vulnerability is exploitable only by an authenticated user.

We evaluated inter-rater agreement in two stages. In the first stage, each analyst independently documented their decisions. This produced 27 sets of decisions (nine vulnerabilities across each of three stakeholder groups) per analyst. In the second stage, we met to discuss decision points where at least one analyst differed from the others. If any analyst changed their decision, they appended what information and evidence they gained during this meeting in the "supporting evidence" value in their documentation. No changes to decisions were forced, and prior decisions were not erased, just amended. Following the second stage, we calculated some statistical measures of inter-rater agreement to help guide our analysis of problem areas.

To assess agreement, we calculate Fleiss' kappa, both for the value in the leaf node reached for each case as well as every decision in a case. (https://en.wikipedia.org/wiki/Fleiss%27_kappa). Evaluating individual decisions is complicated slightly because the different paths through the tree mean that a different number of analysts may have evaluated certain items, and Fleiss' kappa requires a fixed number of raters. The way "leaf node reached" is described is to pick out the specific path through the tree the analyst selected, and to treat that as a label holistically. Measuring agreement based on the path has the drawback that ostensibly similar paths, which agree on 3 of 4 decisions for example, are treated as no more similar than paths which are agree on 0 of 4 decisions. So the two measures of agreement (per decision and per path) are complementary, and we report both.

The vulnerabilities used as case studies are as follows. All quotes are from NVD, and are illustrative of the vulnerability; however, during the study each vulnerability was evaluated according to information analogous to that in Table 10.

Safety-critical cases

- CVE-2015-5374: "Vulnerability … in [Seimens] Firmware variant PROFINET IO for EN100 Ethernet module… Specially crafted packets sent to port 50000/UDP could cause a denial-of-service of the affected device…"
- CVE-2014-0751: "Directory traversal vulnerability in … GE Intelligent Platforms Proficy HMI/SCADA - CIMPLICITY before 8.2 SIM 24, and Proficy Process Systems with CIMPLICITY, allows remote attackers to execute arbitrary code via a crafted message to TCP port 10212, aka ZDI-CAN-1623."
- CVE-2015-1014: "A successful exploit of these vulnerabilities requires the local user to load a crafted DLL file in the system directory on servers running Schneider Electric OFS v3.5 with version v7.40 of SCADA Expert Vijeo Citect/CitectSCADA, OFS v3.5 with version v7.30 of Vijeo Citect/CitectSCADA, and OFS v3.5 with version v7.20 of Vijeo Citect/CitectSCADA. If the application attempts to open that file, the application could crash or allow the attacker to execute arbitrary code."

Regulated systems cases

- CVE-2018-14781: "Medtronic insulin pump [specific versions] when paired with a remote controller and having the "easy bolus" and "remote bolus" options enabled (non-default), are vulnerable to a capture-replay attack. An attacker can … cause an insulin (bolus) delivery."
- CVE-2017-9590: "The State Bank of Waterloo Mobile … app 3.0.2 … for iOS does not verify X.509 certificates from SSL servers, which allows man-in-the-middle attackers to spoof servers and obtain sensitive information via a crafted certificate."
- CVE-2017-3183: "Sage XRT Treasury, version 3, fails to properly restrict database access to authorized users, which may enable any authenticated user to gain full access to privileged database functions. Sage XRT Treasury is a business finance management application. …"

General computing cases

- CVE-2019-2691: "Vulnerability in the MySQL Server component of Oracle MySQL (subcomponent: Server: Security: Roles). Supported versions that are affected are 8.0.15 and prior. Easily exploitable vulnerability allows high privileged attacker with network access via multiple protocols to … complete DoS of MySQL Server."
- CVE-2019-9042: "[I]n Sitemagic CMS v4.4… the user can upload a .php file to execute arbitrary code, as demonstrated by 404.php. This can only occur if the administrator neglects to set FileExtensionFilter and there are untrusted user accounts. …"
- CVE-2017-5638: "The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via crafted [specific headers], as exploited in the wild in March 2017…"

Pilot results

For each of the nine CVEs, we had six analysts rate the priority of the vulnerability as both a developer and applier. Table 12 summarizes the results by reporting the inter-rater agreement for each of the decision points. For all measures, agreement (κ) is above zero, which is generally interpreted as some agreement among analysts. Below zero would be interpreted as noise or discord. Closer to 1 is more or stronger agreement.

How close κ should be to 1 before agreement can be considered strong enough or reliable enough is a matter of some debate. The value certainly depends on the number of options among which analysts select. Indeed, we find that for those decision points with five options (mission and safety impact), agreement is lowest. Although portfolio value has a higher κ than mission or safety impact, it may not actually have higher agreement because portfolio value only has two options. The results for portfolio value are nearly indistinguishable as far as level of statistical agreement from mission impact and safety impact. The statistical community does not have hard and fast rules for cut lines on adequate agreement. We treat κ here as a descriptive statistic, not a test statistic.

Table 12: Inter-rater agreement for decision points

	Safety Impact	Exploitation	Technical impact	Portfolio value	Mission impact	Exposure	Dev result	Applier result
Fleiss' κ	0.116	0.807	0.679	0.257	0.146	0.480	0.226	0.295

For all the decision points, our goal is to provide descriptions and decision procedures such that κ is 1. This will be a conceptual engineering task. Our pilot evaluation can be repeated, with more diverse groups of stakeholders after the descriptions have been refined by stakeholder input, to measure fit to this goal. In order for a standard to be reliably applied across different analyst backgrounds, skill sets, and cultures, an set of decision point descriptions should ideally achieve κ of 1 for each item in multiple

studies with diverse participants. Such a high level of agreement would be difficult to achieve, but it would ensure that when two analysts assign a priority with the system that they get the same answer.[18]

Based on these results, we made about ten changes, some bigger than others. We have not executed a new rater agreement experiment with the updated descriptions. The pilot results are encouraging, and we believe it is time to open up a wider community discussion.

Improvements instigated by the pilot

All of the following changes are reflected in the Section Decision trees for vulnerability management.

- Technical impact: Clarified that partial/total is in regards to the system scope definition, which considers a database or a web server program as the "whole" system. Furthermore, loss of authentication credentials is total loss of control if credentials are to the total system.
- Added note about advice for information gathering to answer both the safety impact and mission impact questions. This change is due to the particularly wide variety of background assumptions analysts had that influenced results and agreement.
- Clarified that "MEF failure" means if any one essential function fails, not all of them. Changed most severe mission impact to "mission failure" to better reflect the relationship between MEFs and the organization's mission.
- Removed the "developer portfolio value" question, as it had poor agreement and no clear way to fix it. Replaced it with *utility*. This change better captures the relevant kind of value (namely, to the adversary) of the affected component while remaining amenable to pragmatic analysis.
- Clarified that proof of concept (see *exploitation*) includes cases in which existing tooling counts as a PoC. The examples listed are suggestive, not exhaustive.
- Reorganized the decision trees based on what items are easier to gather information for or have a widely verifiable state. This change moved Exploitation to the first question.
- Change the decision tree results such that if Exposure is "small" then the resulting priority is lower than before the pilot study. That is, "small" exposure has a stronger effect on reducing urgency.

Questions removed as ineffective

This section discusses ideas we tried but rejected for various reasons. We are not presenting this section as the final word on excluding these ideas, but we hope our reasons for excluding them are instructive, will help prevent others from re-inventing the proverbial wheel, and can guide thinking on future work.

---

[18] This is not the case with CVSSv3; expert assessment of scores vary widely. See Figure 1 in Allodi L, Cremonini M, Massacci F, Shim W. The Effect of Security Education and Expertise on Security Assessments: the Case of Software Vulnerabilities. In: WEIS 2018.

We brainstormed around 12 potential decision points initially. Most of these were removed early on in our development process through informal testing. These included the adversary's strategic benefit of exploiting the vulnerability, the state of legal or regulatory obligations, the cost of developing a patch, patch distribution readiness, financial losses to customers due to potential exploitation, and business losses to the applier. Some of these have left marks on other decision points. Financial losses of customers led to an amendment of the definition of safety to include "well-being" such that, for example, bankruptcies of third parties are a major safety impact. Business losses are covered as a mission impact insofar as profit is a mission of publicly-traded corporations. Three of these items left no trace on the current system. Legal, patch cost, and distribution readiness items were dropped as either being too vaguely defined, too high level, or otherwise not within the scope of decisions by the defined stakeholders. The remaining item, "adversary's strategic gain," transmuted to a different sense of system value. In an attempt to be more concrete and not speculate about adversary motives, we considered a different sense of value: developer portfolio value.

The only decision point that we have removed following the pilot is developer portfolio value. This notion of value was essentially an over-correction to the flaws identified in the "adversary's strategic benefit" decision point. "Developer portfolio value" was defined as "the value of the affected component as a part of the developer's product portfolio. Value is some combination of importance of a given piece of software, number of deployed instances of the software, and how many people rely on each. The developer may also include lifecycle stage (early development, stable release, decommissioning, etc.) as an aspect of value." It had two possible values, low and high. As Table 12 demonstrates, there was relatively little agreement among our six analysts about how to evaluate this decision point. We have replaced this sense of portfolio value with *utility*, which combines value density and virulence.

Worked example

As an example, we will evaluate CVE-2018-14781 step by step from the patch applier point of view. The scenario here is that used for the pilot study. This example uses the decision tree in the Patch applier tree section. Note that the pilot used slightly different trees, as noted in the section Improvements instigated by the pilot.

The analyst's first question is exploitation. Technically, we could answer the questions in any order. However, this is this is good question to start with because given an adequately defined search procedure, we can always answer whether it finds an available exploit or proof of concept. The scenario description for the pilot study reads:

- State of exploitation: Metasploit and ExploitDB do not return results for this vul. NVD does not report any active exploitation of this vulnerability.

This information rules out "active" given our (perhaps limited) search procedure. While the search did not produce any precise PoC, based on the description of the vulnerability we see that it is a fairly

standard traffic capture and replay attack that, given access to the transmission medium, should be straightforward to conduct with Wireshark. Therefore, we select the "PoC" branch and move to asking about exposure. This considers the (fictional) applier scenario blurb as well as the notional deployment of the affected system, as follows.

- Scenario blurb: We are a hospital that uses Medtronic devices frequently because of their quality and popularity in our market. We give these devices out to clients who need to monitor and track their insulin intake. If clients need to access data on their device, they can physically connect it to their computer or connect via Bluetooth to an app on their phone for monitoring capabilities. Occasionally, clients who use this device will have a doctor's appointment in which the doctors have machines that can access the device as well to monitor or change settings. It is unknown how secure the doctor's computer that interfaces directly with this insulin pump is. If the doctor's computer is compromised, it potentially means that every device that connects to it is compromised as well. If an update to the insulin pump is required, a client can do this on their own through their computer or app or through a doctor while they are onsite at the hospital.

- Deployment of affected system These pumps are attached directly to the client. If an update is required, the client is permitted to do that through their own computer or app. However, we have not provided them with documentation on properly using their computer or app to securely access their device. This is done for convenience so that if the user needs to change something quickly, they can. They also can also come to us (hospital) for a change in their device's settings for dosage etc. The doctor's computer that directly handles interfacing with these devices is only connected to the intranet for the purpose of updating the client's settings on the device. Doctors authenticate with ID badge and password.

Exposure is less straightforward than exploitation. The option "unavoidable" is clearly ruled out. However, it is not clear whether optional Bluetooth connection between the medical device and a phone app is "controlled" or "small" exposure. The description does not explicitly handle the capture/replay aspect of the vulnerability. If the only way to exploit the vulnerability is to be within physical transmission range of the device, then that physical constraint argues for exposure being "small." However, if the client's phone app could be used to capture and replay attack packets, then unless that app is particularly well-secured the answer should be "controlled." The answer is not clear from the supplied information. Furthermore, if this fictional app is specific to the insulin pump, then even if it is not compromised, the attack might use its installation to remotely identify targets. However, we will make the assertion that the hospital's clients do not widely have the app even installed, and for nearly all cases physical proximity to the device is necessary. Therefore, we select "small" and move on to ask about mission impact.

According to our fictional pilot scenario, "Our mission dictates that our first and foremost priority is to contribute to human welfare and to uphold the Hippocratic oath (do no harm)." The continuity of operations planning for a hospital is complex, with many mission essential functions. However, even from this abstract remove it seems clear that "do no harm" is at risk due to this vulnerability. A MEF to that mission is each of the various medical devices works as expected, or at least when a device fails it cannot actively be used to inflict harm. Unsolicited insulin delivery would mean that MEF "fails for a period of time longer than acceptable," matching the description of "MEF failure." The question is then

whether the whole mission fails, which does not seem to be the case. The recovery of MEF functioning is not affected, and most MEFs (the emergency services, surgery, oncology, administration, etc.) would be unaffected. Therefore, we select "MEF failure" and move on to ask about safety impact.

Given the prior three answers (PoC, small, MEF failure), our safety analysis is somewhat constrained. If the result is none, minor, or major, the priority is *scheduled*. Hazardous will lead to *out-of-band*, and catastrophic to *immediate* action. In the pilot study, this information is conveyed as follows:

- Use of the cyber-physical system Insulin pumps are used to regulate blood glucose levels in diabetics. Diabetes is extremely common in the US. Misregulation of glucose can cause a variety of problems. Minor misregulation causes confusion or difficulty concentrating. Long-term minor mismanagement causes weigh management issues and blindness. Severe acute mismanagement can lead unconsciousness in a matter of minutes and death in a matter of hours. The impacted insulin pumps have a local (on-patient) wireless control, so wires to the pump do not have to be connected to the patient's control of the system, making the system lighter and less prone to be ripped out.

The closest match to "death in a matter of hours" would be hazardous, because that description reads "serious or fatal injuries, where fatalities are plausibly preventable via emergency services or other measures." Depending on the details of the hospital's contingency plans and monitoring of their patients, the safety impact could be catastrophic. If there is no way to tell whether the insulin pumps are misbehaving, for example, then exploitation could go on for some time, leading to a catastrophic safety impact. The pilot information is inadequate in this regard, which is the likely source of disagreement about safety impact in Table 12. For the purposes of this example, we imagine we gather that information, find the monitoring situation to be adequate, and select "hazardous." We should mitigate this vulnerability *out-of-band*, meaning that it should be addressed quickly, ahead of our usual update and patch cycle.

Future Work

We intend SSVC to offer a workable baseline from which to improve and refine a vulnerability prioritization methodology. While the method herein should be functional, we do not claim it is ready for use as is. Therefore, we lay out some aspects of future work that would help make it ready to use. We focus on further requirements gathering, further testing of the reliability of the decision process, and expanding to additional types of stakeholders beyond patch appliers and patch developers.

Requirements gathering via sociological research

We should know what users of a vulnerability prioritization system want. A start to this question would be to understand how people actually use CVSS and what they think it tells them. In general, we do not have empirical, grounded evidence about what practitioners and decision makers want from vulnerability scoring. We have based this paper's methodology on multiple decades of professional experience and myriad informal conversations with practitioners. Such evidence is not a bad place to start, but it

does not lend itself to examination and validation by others. The purpose of understanding practitioner expectations is to inform what a vulnerability prioritization methodology should actually provide by matching it to what people want or expect. The method this future work should take is long-form, structured interviews. We do not expect anyone to have access to enough consumers of CVSS to get statistically-valid results out of a short survey, nor to pilot a long survey.

Further decision tree testing

More testing with diverse analysts is necessary before we can consider the decision trees to be reliable. In this context, reliable means that two analysts, given the same vulnerability description and decision process description, will reach the same decision. Such reliability is important if scores and priorities are going to be useful. If they are not reliable, they will vary widely over time and among analysts. Such variability makes it impossible to tell whether a difference in scores is really due to one vulnerability being higher priority than other.

The pilot study provides a methodology for measuring and evaluating reliability of the decision process description based on the agreement measure κ. This study methodology should be repeated with different analyst groups, from different sectors and with different experience, feeding the results into changes in the decision process description until the agreement measure is adequately close to 1.

Decision tree for vulnerability coordination

Currently only two stakeholders are addressed, patch appliers and patch developers. Expanding our work to include more types of stakeholders would be beneficial. We propose that the next stakeholder group could be vulnerability coordinators, as described in the Enumerating stakeholders section. The development and testing methodology for any new stakeholder group should be roughly the same as that used to draft the applier and developer decision trees.

Limitations

Even as a working proposal, our Stakeholder-Specific Vulnerability Categorization (SSVC) has some limitations. These are inherent limits of the approach, which should be understood as trade-offs. There are other aspects of our implementation that are limiting, but those have been covered as topics needing improvement in Future Work.

We have made two important trade-offs compared to the current state of practice with CVSS. Firstly, we have done away with numerical scores, and this may make some practitioners uncomfortable. We explained our reasons for this in depth, but despite the fact that CVSS contains false precision we still must contend with the fact that, psychologically, users find that comforting. As this comfort gap may negatively impact adoption, this fact is a limitation. Although it is ungainly, it would be sound to convert the priority outcomes to numbers at the end of the process, if existing processes require it. Which numbers does not so much matter, so long as the ordering is preserved. CVSS has set a precedent that higher numbers are worse, so a scale [1, 2, 3, 4] would work, with defer = 1 and immediate = 4. However, if it

were important to maintain backwards compatibility to the CVSS range zero to ten, we could just as well relabel outcomes as [2, 5.5, 8, 9.5] for the midpoints of the current CVSS severity ranges.

Secondly, we incorporate a wider variety of inputs from contexts beyond the affected component. Some organizations are not prepared or configured to reliably produce such data, for example around mission impact or safety impact. There is adequate guidance for how to elicit and curate these kinds of information from various risk management frameworks, including OCTAVE.[19] Not every organization is going to have sufficiently mature risk management functions to apply our proposal. This limitation should be approached with two strategies. Organizations should be encouraged and enabled to mature their risk management capabilities. However, in the meantime, an organization such as NIST could consider developing a kind of default advice. The most practical framing of this approach might be for the NIST NVD to produce scores from the perspective of a new stakeholder, something like "national security" or "public wellbeing" that is explicitly a sort of default advice for otherwise uninformed organizations that can then explicitly account for national priorities, such as critical infrastructure.

Conclusion

We have presented a working hypothesis for how patch developers and patch appliers should prioritize their effort to mitigate different vulnerabilities. We have performed an initial evaluation of the proposal and improved it, but the process we develop for evaluation is more important than the results. We invite further refinement of the prioritization mechanism. We endeavored to be transparent about our process and provide justification for design decisions. We invite questions, comments, and further community refinement in moving forward with a transparent and justified vulnerability prioritization methodology that is inclusive for the various stakeholders and industries that develop and use information and computer technology.

[19] Caralli, Richard; Stevens, James; Young, Lisa; & Wilson, William. Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process. CMU/SEI-2007-TR-012. Software Engineering Institute, Carnegie Mellon University. 2007. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8419

Contact Us

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

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

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.

DM19-1222