Carnegie Mellon University
Software Engineering Institute

PRIORITIZING VULNERABILITY RESPONSE: A STAKEHOLDER-SPECIFIC VULNERABILITY CATEGORIZATION (VERSION 1.1)

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

Workshop on the Economics of Information Security; December 2020

1 Introduction

Many organizations use the Common Vulnerability Scoring System (CVSS) to prioritize actions during vulnerability management. This paper builds on prior work about prioritizing actions during vulnerability management by presenting a testable Stakeholder-Specific Vulnerability Categorization (SSVC) that avoids some problems with CVSS. SSVC takes the form of decision trees for different vulnerability management communities. We welcome others to test and improve it.

This paper 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; it is not a final proposal. The stakeholders in vulnerability management are diverse, and that diversity must be accommodated in the main functionality, rather than squeezed into hard-to-use optional features. Given this, as much as it is practical, we aim to avoid one-size-fits-all solutions.

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

The organizing concept of our decision-making procedure is decision trees. A decision tree represents important elements of a decision, possible decision values, and possible outcomes. We suggest decision trees as an adequate formalism for practical, widespread advice about vulnerability prioritization. We do not claim this approach is the only viable option. We also suggest that specific vulnerability management stakeholder communities use decision trees. These suggestions are hypotheses for viable replacements for CVSS in those communities, but the hypotheses require empirical testing before they can be justifiably considered fit for use. We propose a methodology for such testing.

[†] The authors thank the following people for helpful comments on prior drafts: Will Dormann, Madison Oliver, Vijay Sarvepalli, and Laurie Tyzenhaus (CERT/CC); Michel van Eeten and the anonymous WEIS reviewers; attendees at A Conference on Defense (ACoD), Austin TX 2020; Dale Peterson, Ralph Langer, and attendees at S4, Miami FL 2020; Muhammad Akbar and Manish Gaur (VMWare); David Oxley (McAfee).

The rest of the paper is organized as follows. Section 2 summarizes the current state of vulnerability management. Section 3 describes our design goals for an improved prioritization method. Section 4 proposes a definition of decision points and decision trees as a prioritization method. Section 5 describes an early test of this method against the design goals, as much to show an adequate usability test methodology as for the results. Section 6 provides examples of applying the methodology of Section 4 to sample vulnerabilities. Section 7 identifies future work. Section 8 identifies limitations in the design. Section 9 concludes with some final thoughts.

2 Current state of practice

Vulnerability management is a term of art for security practitioners, used to include "the discovery, analysis, and handling of new or reported security vulnerabilities in information systems [and] the detection of and response to known vulnerabilities in order to prevent them from being exploited."[1] Prioritization of organizational and analyst resources is an important precursor to vulnerability analysis, handling, and response. The general problem is: given limited resources, which vulnerabilities should be processed and which can be ignored for now. We approach this problem from a pragmatic, practitioner-centered angle.

The de facto standard prioritization language is CVSS.[2] CVSS avoids discussing decisions and, instead, takes *technical severity* as its fundamental concept. We understand severity's role as informing decision making about vulnerability management. The CVSS standard indicates vulnerability management decisions, and only those decisions, as what they expect CVSS scores to inform,[3] yet the standard does not provide clear advice about how CVSS scores might inform decisions.

How CVSS is used matters. Using just the base scores as a stand-alone prioritization method is not recommended.[4] However, as two examples, the U.S. government[5] and the global payment card industry[6] both have defined such misuse as expected practice in their vulnerability management requirements. CVSS has struggled to adapt to other stakeholder contexts; various stakeholder groups have

---

[1] Vilius Benetis, Olivier Caleff, Cristine Hoepers, Angela Horneman, Allen Householder, Klaus-Peter Kossakowski, Art Manion, Amanda Mullens, Samuel Perl, Daniel Roethlisberger, Sigitas Rokas, Mary Rossell, Robin M. Ruefle, D'esir'ee Sacher, Krassimir T. Tzvetanov, and Mark Zajicek. Computer security incident response team (CSIRT) services framework. Technical Report ver. 2.1, FIRST, Cary, NC, USA, July 2019.

[2] Spring, Jonathan. M., and Phyllis Illari. Review of human decision-making during computer security incident analysis. 2019. arXiv:1903.10080.

[3] "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 "Common Vulnerability Scoring System SIG" (https://www.first.org/cvss).

[4] The base score is defined as "the intrinsic characteristics of a vulnerability that are constant over time and across user environments." FIRST. Common Vulnerability Scoring System version 3.1: Specification Document.

[5] Suggested for use by federal civilian departments and agencies via NIST guidance (e.g., SP 800-115, p. 7-4 and SP 800-40r3 pg. 4) and the DHS directive on Critical Vulnerability Mitigation (https://cyber.dhs.gov/bod/15-01/).

[6] Via PCI DSS, see: https://www.pcisecuritystandards.org/documents/ASV_Program_Guide_v3.0.pdf

expressed dissatisfaction by making new versions of CVSS, such as medical devices,[7] robotics,[8] and industrial systems.[9] In these three examples, the modifications tend to add complexity to CVSS by adding metrics. Product vendors have varying degrees of adaptation of CVSS for development prioritization.[10] The vendors codify CVSS's recommended qualitative severity rankings in different ways, and Red Hat and Microsoft make the user interaction base metric more important. The various stakeholder re-adaptations of CVSS suggest a stakeholder-specific prioritization is important.

Unfortunately, all such re-adaptation of the basic CVSS mindset inherit its deeper issues. For example, the CVSS scoring algorithm has not been argued for transparently, and the standardization group has not justified the use of the formula either formally or empirically.[11] In addition, severity should only be a part of vulnerability response prioritization.[12] One complaint is that a high CVSS score is not predictive of which vulnerabilities will be commonly exploited or have exploits publicly released.[13] Studies of CVSS scoring consistency indicate that analysts do not consistently interpret the elements of a CVSSv3.0 score,[14] and as many adaptations of CVSS simply add additional metrics we expect they inherit such inconsistency. Analyst usability has so far been an afterthought, but we know from other areas of information security that usability is not well-served as an afterthought.[15]

Surveys of security metrics[16] and information sharing in cybersecurity[17] do not indicate any major efforts to conduct a wholesale rethinking of vulnerability prioritization. The surveys indicate some options for available measurements a prioritization method might consider, such as exploit availability or system attack surface. Section 3 describes our design goals for a pragmatic prioritization methodology that can improve and build on the state of current practice.

---

[7] Chase, Penny and Stevey Christey Coley. *Rubric for Applying CVSS to Medical Devices*. MITRE and the FDA. 2019.

[8] Vilches, Víctor Mayoral, Endika Gil-Uriarte, Irati Zamalloa Ugarte, Gorka Olalde Mendia, Rodrigo Izquierdo Pisón, Laura Alzola Kirschgens, Asier Bilbao Calvo, Alejandro Hernández Cordero, Lucas Apa, and César Cerrudo. *Towards an open standard for assessing the severity of robot security vulnerabilities, the Robot Vulnerability Scoring System (RVSS)*. arXiv:1807.10357 (2018).

[9] Santiago Figueroa-Lorenzo, Javier Añorga, and Saioa Arrizabalaga. A survey of IIoT protocols: A measure of vulnerability risk analysis based on cvss. ACM Comput. Surv., 53(2), April 2020.

[10] These include, but are not limited to :Red Hat (https://access.redhat.com/security/updates/classification), Microsoft (https://www.microsoft.com/en-us/msrc/security-update-severity-rating-system), and Cisco (https://tools.cisco.com/security/center/resources/security_vulnerability_policy.html#asr).

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

[12] For example: Farris KA, Shah A, Cybenko G, Ganesan R, Jajodia S. VULCON: A System for Vulnerability Prioritization, Mitigation, and Management. ACM Transactions on Privacy and Security (TOPS). 2018 Jun 12; 21(4):16.

[13] Allodi, Luca and Fabio Massacci. A Preliminary Analysis of Vulnerability Scores for Attacks in Wild: The EKITS and SYM Datasets. BADGERS'12, Oct 5, 2012, Raleigh, North Carolina, USA.

[14] 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

[15] Garfinkel, Simson, and Heather Richter Lipford. Usable security: History, themes, and challenges. Morgan & Claypool Publishers, 2014.

[16] Pendleton, Marcus, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. A survey on systems security metrics. ACM Comput. Surv., 49(4), December 2016.

[17] Stefan Laube and Rainer Böhme. Strategic aspects of cyber risk information sharing. ACM Comput. Surv., 50(5), November 2017.

3 Representing Information for Decisions About Vulnerabilities

We chose to build our model with *decisions* as the central concept. We propose that decisions—rather than severity—are a more useful output. Our design requirements for an adequate decision-making process is that it clearly define whose decisions are involved, properly use evidentiary categories, be based on reliably available evidence, be transparent, and be explainable. Our inspiration and justification for these design goals is that they are the features of a satisfactory scientific enterprise[18] adapted to this vulnerability management problem.

To consider decisions about managing the vulnerability rather than just technical severity, one 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 have different priorities than organizations that make word processors. These differences 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.

The goal of adequacy is more appropriate than optimality. Our search process need not be exhaustive; we are satisficing rather than optimizing.[19] Satisficing is more appropriate to qualitative criteria; we do not need to order different methods as to which are more transparent than others, for example. Finding any system that meets all of desired criteria is enough.

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 the design of the metrics. For example, CVSS scores were designed to be accurate with +/- 0.5 points of the given score.[20] 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 and 40% critical. We want decisions to be distinct and crisp; statistical overlaps of scores within 1.0 unit, for example, would muddy decision recommendations.

We 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

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

[19] Simon, Herbert A. The sciences of the artificial. 3rd ed. MIT press, 1996.

[20] Common Vulnerability Scoring System v3.1: Specification Document. 2019. See Section 7.5. https://www.first.org/cvss/v3.1/specification-document

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, are inextricably linked to managing it. Temporal and environmental considerations should be primary, not optional as they are in CVSS.

We make the 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 defined and used with its common meaning. This meaning is not the same as "explainable," as used in the research area of explainable artificial intelligence.) An explanation should make the process intelligible to an interested, competent, non-expert person. There are at least two reasons common explainability is important: (1) for troubleshooting and error correction and (2) for justifying proposed decisions.

To summarize, the following are our design goals for a vulnerability management process:

- Outputs are *decisions.*
- Pluralistic recommendations are made among a manageable number of stakeholder groups.
- Inputs are qualitative.
- Outputs are qualitative, and there are no (unjustified) shifts to quantitative calculations.
- Process justification is transparent.
- The results are explainable.

3.1 Formalization Options

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

We rule out many quantitative options, 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 since each individual decision tree can be traced and the decisions explained.[21] However, it's not transparent enough to simply know how the available decision trees are created or mutated and why a certain set of them works better. In any case, random forests are necessary only when decision trees get too complicated for humans to manage. We demonstrate below that in vulnerability management, useful decision trees are small enough for humans to manage.

[21] 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, in program verification, satisfiability problem (SAT) and satisfiability modulo theories (SMT) solvers are used to automate decisions about when some condition holds or whether software contains a certain kind of flaw. However, while the explanations provided by logical tools are accessible to experts, non-experts may struggle. However, 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.

Decision trees are used differently in operations research than in ML. In ML, decision trees are used as a predictive model to classify a target variable based on dependent variables. In operations research and decision analysis, a decision tree is a tool used to document a human process. In decision analysis "decision analysts frequently use specialized tools, such as decision tree techniques, to evaluate uncertain situations. Unfortunately, many people, some of them educators, have confused decision analysis with decision trees. This is like confusing surgery with the scalpel."[22] We use decision trees in the tradition of decision analysis, not ML.

Table 1: Comparison of Formalization Options for Vulnerability Prioritization Decisions

	Outputs Designed to be Decisions	Pluralistic Recommendations	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 First Order Logics	Maybe	Maybe	✓	✓	✓	Maybe
Decision Trees (as in decision analysis)	✓	✓	✓	✓	✓	✓

[22] Howard, Ronald A and James E Matheson, eds. Readings on the Principles and Applications of Decision Analysis. Strategic Decisions Group. 1983. Pg viii.

3.2 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. Each decision point can have more than two options and may have different options from other decision points.

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 need to be explained and justified, but this cost is paid infrequently.

There has been limited but positive use of decision trees in vulnerability management. For example, Vulnerability Response Decision Assistance (VRDA) studies how to make decisions about how to respond to vulnerability reports.[23] This paper continues roughly in the vein of such work to construct multiple decision trees for prioritization within the vulnerability management process.

4 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 address each of these parts, in turn, and should be taken as instructive examples. While we strive to make the examples realistic, we invite the community to engage and conduct empirical assessments to test examples. The following construction should be treated as an informed hypothesis rather than a conclusion.

4.1 Enumerating Stakeholders

Stakeholders in vulnerability management can be identified within multiple independent axes. For example, they can be identified by their responsibility: whether the organization *develops*, *applies*, 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 or not the stakeholder is in a safety-critical sector, the decision will be addressed.

The 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

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

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 "coordinating patches" as future work. Each of these trees will have different decision points that they take to arrive at a decision about a given unit of work.

The next two sections describe the decision space and the relevant decision points that we propose for these two responsibilities within the vulnerability management process.

The paper's 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 other roles, such as risk managers, technical managers, and incident responders. Although every layperson who owns a computing device makes decisions about managing it, this is not the target audience. The following decision system may help such laypeople, but we do not intend it to be used by that audience.

Relatedly, C-level executives and public policy professionals often make, shape, or incentivize decisions about managing information systems; however, this is not the target audience either. To the extent that decision trees for vulnerability management help higher level policy decisions, we believe the best way to help policy makers is by making the technical decisions more transparent and explainable to policy makers. While policy makers may see indirect benefit, they are not the primary target, and we are not designing an approach for them directly.

4.2 Enumerating Decisions

Stakeholders with different responsibilities in vulnerability management have largely different decisions to make. This section focuses 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 transport layer security (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 to the Linux kernel. The list goes on. Alternatively, one might view applying patches as, 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 genuine decisions do three things: (1) identify the decisions explicitly, (2) describe how they view their role(s), and (3) identify which software projects their decision relates to. If their decisions are explicit, then the decision makers can use the recommendations from this document that are 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 a 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 considers 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 only on the priority of building the patch, and we consider four categories of priority, as outlined 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 are 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, usually the action is to apply it. When a patch it not yet available, the action space is more diverse, but it should involve mitigating the vulnerability (e.g., shutting down services or applying additional security controls) or accepting the risk of not mitigating the vulnerability.

In this paper, we model the decision of "With what priority should the organization take action on a given vulnerability management work unit?" to be agnostic to whether or not a patch is available. A unit of work means 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.[23] VRDA is likely adequate for national-level CSIRTs that do general CVD, but other CSIRT types may have different needs. Future work may elicit those types and make a few different decision options. Specialized coordination organizations exist (e.g., ICS-CERT, which conducts CVD for safety-critical systems). We have not developed 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.[24]

Within each setting, the decisions are a kind of equivalence class for priority. That is, if an organization must deploy patches for three vulnerabilities, and if these vulnerabilities are all assigned the "scheduled" priority, then the organization can decide which to deploy first. The priority is equivalent. This approach may feel uncomfortable since CVSS gives the appearance of a finer grained priority. CVSS appears to say, "Not just 4.0 to 6.9 is 'medium' severity, but 4.6 is more severe than 4.5." However, as discussed previously (see page 4), CVSS is designed to be accurate only within +/- 0.5, and, in practice, is scored with errors of around +/- 1.5 to 2.5.[25] 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 approach. CVSS errors often cross decision boundaries; in other words, 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 affect the results.

Returning to the example of an organization with three vulnerabilities to patch that were assigned "scheduled" priority, in SSVC, 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.

4.3 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. 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, and are plausible, explicit, and accessible to all relevant decision makers.

---

[24] 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

[25] 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, Figure 1. The more accurate half of professionals estimated CVSS scores in ranges such as [+2,0] (i.e., between overestimating by 2 to being correct), [+2,-2], and [0,-2].

4.3.1.1 Boundaries of the Affected System

One distinction is 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 is what vulnerability management is focused on patching. Multiple pieces of software run on any given computer system. To consider software vulnerabilities in a useful scope, we follow prior work and propose that a vulnerability affects "the set of software instructions that executes in an environment with a coherent function and set of permissions."[26] 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 are parts of cyber-physical systems.

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, which is separating physical space. At the same time, an off-premises DNS server may be part of the site security assurance system if the on-premises security cameras rely on the DNS server to connect to the displays at the guard's desk. We define computer software as part of a cyber-physical system if the two systems are mutually manipulable; that is, changes in the part (the software) will (at least, often) make detectable and relevant changes to the whole (the cyber-physical system), and changes in the whole will (often) make relevant and detectable changes in the part.[27]

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

- *Nearest* is relative to the abstraction at which the vulnerability exists.
- *Relevant* implies that the impacted component must be in the chain of abstraction moving upward from the location of the flaw.
- *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*.
- *Discrete* means there is an identifiable thing that can be fixed (e.g., the unit of patching).

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

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

[27] Spring J.M. & Illari P. Building general knowledge of mechanisms in information security. *Philosophy & Technology*. **32**, 627–659. 2018.

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 (ERM) system would consider the technical impact to the database, not to the ERM system.

4.3.1.2 Reasoning Steps Forward

This aspect of scope is about 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 an exploitation of the software in the cyber-physical system contributes 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, 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*.

4.4 Likely Decision Points and Relevant Data

We propose the following decision points and associated values should be a factor when making decisions about vulnerability prioritization. Each decision point is tagged with the stakeholder it is relevant to: patch appliers, patch developers, or both. We emphasize that these descriptions are hypotheses to be further tested and validated. We made every effort to put forward informed and useful decision frameworks with wide applicability, but the goal of this paper is more to solicit feedback than make 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; of course, future events may require changes to these assumptions over time. Therefore, our assumptions 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; accommodating such tolerance or discounting explicitly is an area for future work. This flexibility fits into our overall goal of supplying a decision-making framework that is both transparent and fits the needs of different communities. 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 any "unknown" option.

We propose satisfactory decision points for vulnerability management in the next sections, in no particular order.

4.4.1 Exploitation (Developer, Applier)

Evidence of Active Exploitation of a Vulnerability

The intent of this measure is the present state of exploitation of the vulnerability. The intent is not to predict future exploitation but 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.[28]

Table 4: Exploitation Decision Values

None	There is no evidence of active exploitation and no public proof of concept (PoC) of how to exploit the vulnerability.
PoC (Proof of Concept)	One of the following cases is true: (1) exploit code sold or traded on underground or restricted fora; (2) typical public PoC in places such as Metasploit or ExploitDB; or (3) the vulnerability has a well-known method of exploitation. Some examples of condition (3) are open-source web proxies serve as 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 attacks on ethernet or WiFi networks.
Active	Shared, observable, reliable evidence that the exploit is being used in the wild by real attackers; there is credible public reporting.

4.4.2 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 where 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	The exploit gives the adversary *limited* control over, or information exposure about, the behavior of the software that 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	The exploit gives the adversary *total* control over the behavior of the software, or it gives total disclosure of all information on the system that contains the vulnerability

4.4.3 Utility (Developer, Applier[29])

The Usefulness of the Exploit to the Adversary

Heuristically, we base *utility* on a combination of value density of vulnerable components and virulence of potential exploitation. This framing makes it easier to analytically derive these categories

[28] Jay Jacobs; Sasha Romanosky; Idris Adjerid; & Wade Baker. *Improving Vulnerability Remediation Through Better Exploit Prediction*. WEIS. Boston, MA. June 2019.

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

from a description of the vulnerability and the affected component. Virulence (slow or rapid) and value density (diffuse or concentrated) are defined in Sections 4.4.3.1 and 4.4.3.2.

Roughly, *utility* is a combination of two things: (1) the value of each exploitation event and (2) the ease and speed with which the adversary can cause exploitation events. We define *utility* as laborious, efficient, or super effective, as described in Table 6.

Table 6: Utility Decision Values

Laborious	Slow virulence and diffuse value
Efficient	{Rapid virulence and diffuse value} OR {Slow virulence and concentrated value}
Super Effective	Rapid virulence and concentrated value

4.4.3.1 Virulence

Virulence is described as slow or rapid:

- **Slow**. Steps 1-4 of the kill chain[30] cannot be reliably automated for this vulnerability for some reason. These steps are reconnaissance, weaponization, delivery, and exploitation. Example reasons for why a step may not be reliably automatable include (1) the vulnerable component is not searchable or enumerable on the network, (2) weaponization may require human direction for each target, (3) delivery may require channels that widely deployed network security configurations block, and (3) exploitation may be frustrated by adequate exploit-prevention techniques enabled by default; ASLR is an example of an 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.

4.4.3.2 Value Density

Value density is described as diffuse or concentrated:

- **Diffuse**. The system that contains the vulnerable component has limited resources. That is, the resources that 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 that contains the vulnerable component is rich in resources. Heuristically, such systems are often the direct responsibility of "system operators" rather than users. Examples of concentrated value are database systems, Kerberos servers, web servers hosting login pages, and cloud service providers. However, usefulness and uniqueness of the resources on the vulnerable system also inform value density. For example, encrypted mobile messaging platforms

[30] Hutchins, E.M.; Cloppert, M.J.; & Amin, R.M. "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.

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 is visualized in Table 7.

Table 7: Utility to the Adversary, as a Combination of Virulence and Value Density

Virulence	Rapid	Efficient	Super Effective
	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,[31] make such pricing structures public. The valuable exploits track the virulence and value density heuristics for the most part. Within a single system—whether it is Apache, Windows, iOS or WhatsApp—more automated kill chain steps successfully 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. Price 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 look for and prevent large mismatches between the outputs of the *utility* decision point and the exploit markets.

4.4.4 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*.[32] 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[33] 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

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

[32] Centers for Disease Control and Prevention. "How is well-being defined?" https://www.cdc.gov/hrqol/wellbeing.htm#three. Health-Related Quality of Life (HRQOL). August 2019.

[33] By "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.

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 willfully ignore 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 of which it is part, that also supersedes our rough supply-chain depth considerations. For software used in a wide variety of sectors and deployments, the stakeholder may need to estimate an 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 for future work.

4.4.4.1 Advice for Gathering Information to Answer the Safety Impact Question

The factors that influence the safety impact level are diverse. This paper does not exhaustively discuss how a stakeholder should answer a question; that is a topic for future work. At a minimum, understanding safety impact should include gathering information about survivability of the vulnerable component, determining available operator actions to compensate for the vulnerable component, understanding relevant insurance, and determining the viability of existing backup measures. Each of these information items depends heavily on domain-specific knowledge, and so it is out of the scope of this paper 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.

Table 8: Safety Impact Decision Values

Safety Impact[34]	Type of Harm	Description
None	All	Does not mean *no impact* literally; it just means 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.) imposed 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

[34] These categories are based on hazard categories for aircraft software. See DO-187C (Software Considerations in Airborne Systems and Equipment Certification) and 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).

Safety Impact[34]	Type of Harm	Description
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 the actions require 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
	Environment	Major externalities (property damage, environmental damage, etc.) imposed on other parties
	Financial	Financial losses that likely 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; 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.) imposed 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.) imposed on other parties
	Financial	Social systems (elections, financial grid, etc.) supported by the software collapse
	Psychological	N/A

4.4.5 System Exposure (Applier)

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. Usually, the effect of these mitigations is 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 varies according to each case.

Table 9: Exposure Decision Values

Small	Local service or program; highly controlled network
Controlled	Networked service with some access restrictions or mitigations 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 (e.g., DNS servers, web servers, VOIP servers, email servers)

4.4.6 Mission Impact (Applier)

Impact on Mission Essential Functions[35] 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" (footnote 35, page A-1). Identifying MEFs is part of business continuity planning or crisis planning. The rough difference between MEFs and non-essential functions is that an organization "must perform a[n MEF] during a disruption to normal operations" (footnote 35, page B-2). The mission is the reason an organization exists, and MEFs are how that mission is affected. Non-essential functions do not directly support the mission per se; however, they support the smooth delivery or success of MEFs. Financial losses—especially to publicly traded for-profit corporations—are covered here as a (legally mandated) mission of such corporations is financial performance.

Table 10: Mission Impact Decision Values

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

4.4.6.1 Advice for Gathering Information to Answer the Mission Impact Question

The factors that influence the mission impact level are diverse. This paper does not exhaustively discuss how a stakeholder should answer a question; that is a topic for future work. At a minimum,

[35] For information about identification of mission essential functions, see *Federal Continuity Directive 2: Federal Executive Branch Mission Essential Functions and Candidate Primary Mission Essential Functions Identification and Submission Process* from June 2017 (https://www.fema.gov/media-library-data/1499702987348-c8eb5e5746bfc5a7a3cb954039df7fc2/FCD-2June132017.pdf).

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[35] or OCTAVE FORTE.[36] 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 question described in Section 4.4.3, Utility (Developer, Applier), 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."

4.5 Relationship to asset management

Our method is for prioritizing vulnerabilities based on the risk stemming from exploitation. There are other reasonable asset management considerations that may influence remediation timelines. There are at least three aspects of asset management that may be important but are out of scope for SSVC. First and most obvious is the transaction cost of conducting the mitigation or fix. System administrators are paid to develop or apply any fixes or mitigations, and there may be other transactional costs such as downtime for updates. Second is the risk of the fix or mitigation introducing a new error or vulnerability. Regression testing is part of managing this type of risk. Finally, there may be an operational cost of applying a fix or mitigation, representing an ongoing change of functionality or increased overhead. A decision maker could order work within one SSVC priority class (scheduled, out-of-band, etc.) based on these asset management considerations, for example. Once the organization fixes all the high-priority vulnerabilities, they can then fix the medium-level vulnerabilities with the same effort spent on the high-priority ones.

Asset management and risk management also drive some of the up-front work an organization would need to do to gather some of the necessary information. This situation is not new; an asset owner cannot prioritize which fixes to deploy to its assets if it does not know what assets it owns and their locations. The organization can pick its choice of tools for these things; there are about 200 asset management tools on the market.[37] Standards like the Software Bill of Materials (SBOM)[38] would likely reduce the burden on asset management, but these are still maturing. If an organization does not have an asset management or risk management (see Section 4.4.6.1) plan and process in place, then it will have a non-trivial amount of work to do to establish these processes before it can take full advantage of SSVC.

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

[37] Captera. IT Asset Management Software. May 24, 2020. https://www.capterra.com/it-asset-management-software/

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

4.6 Patch Developer Tree

Figure 1 shows the 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-of-Band = orange
- Immediate = red with black outline

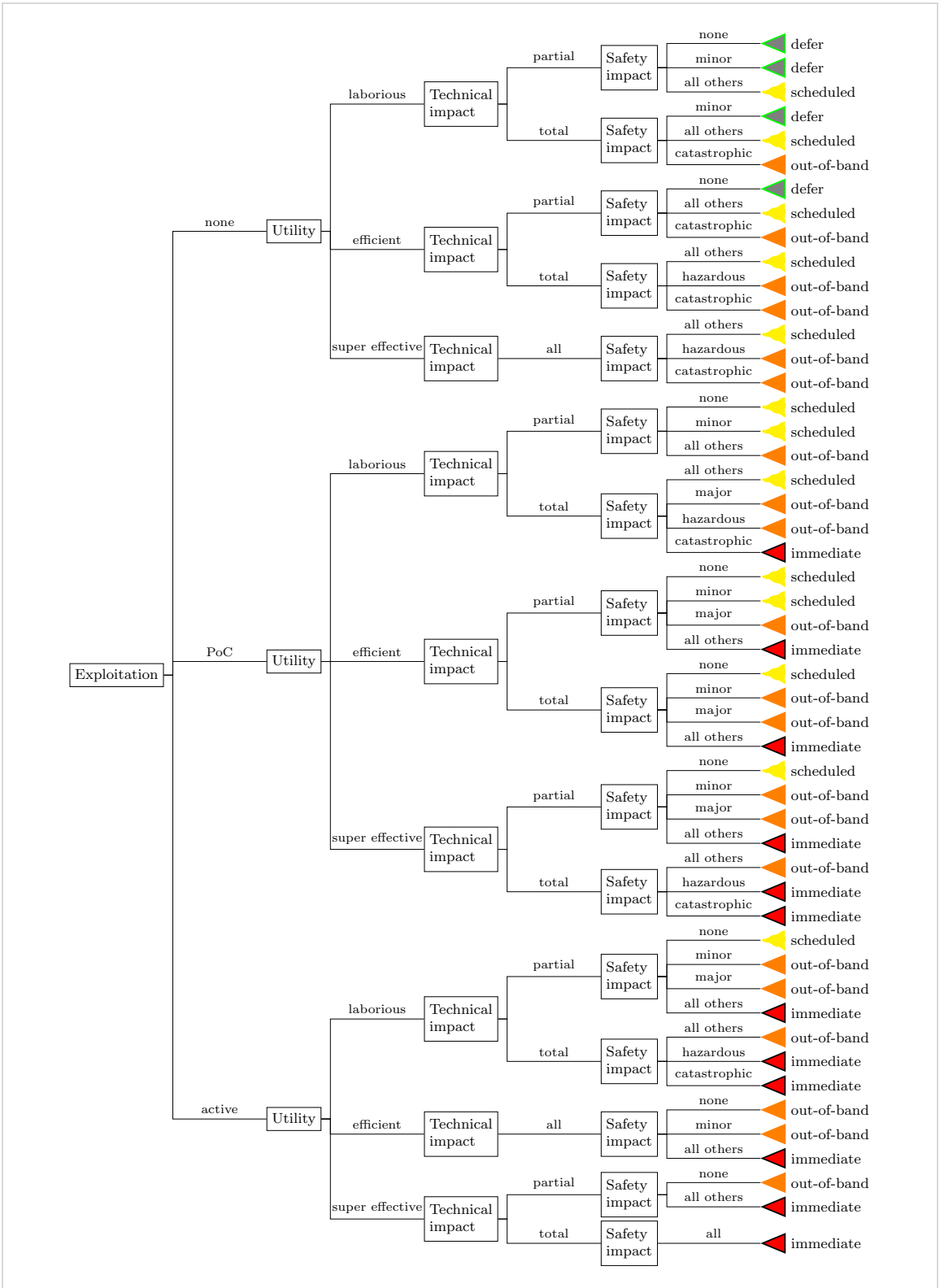

Figure 1: Proposed Vulnerability Prioritization Decision Tree for Patch Developer

4.7 Patch Applier Tree

*The proposed patch applier tree is depicted in Figure 2: Proposed Vulnerability Prioritization Decision Tree for Patch Appliers (Continued in Figure 3 and Figure 4**Error! Reference source not found.**)*

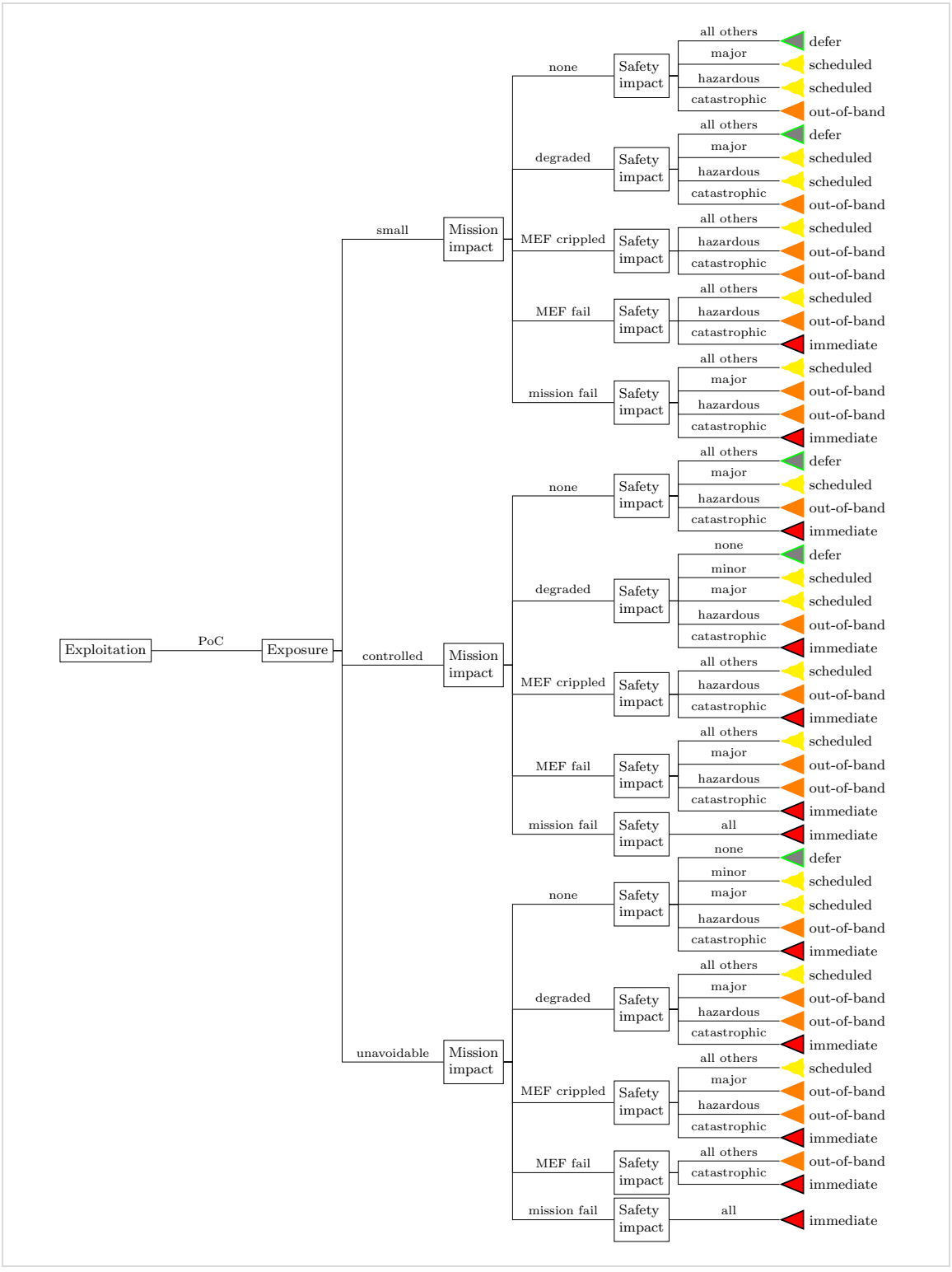

Figure 3, Figure 4, and **Error! Reference source not found.**. The state of *exploitation* is the first decision point, but in an effort to make the tree legible, we split the tree into three sub-trees over three pages. We suggest making the decision about *exploitation* as usual, and then going to the correct subtree.

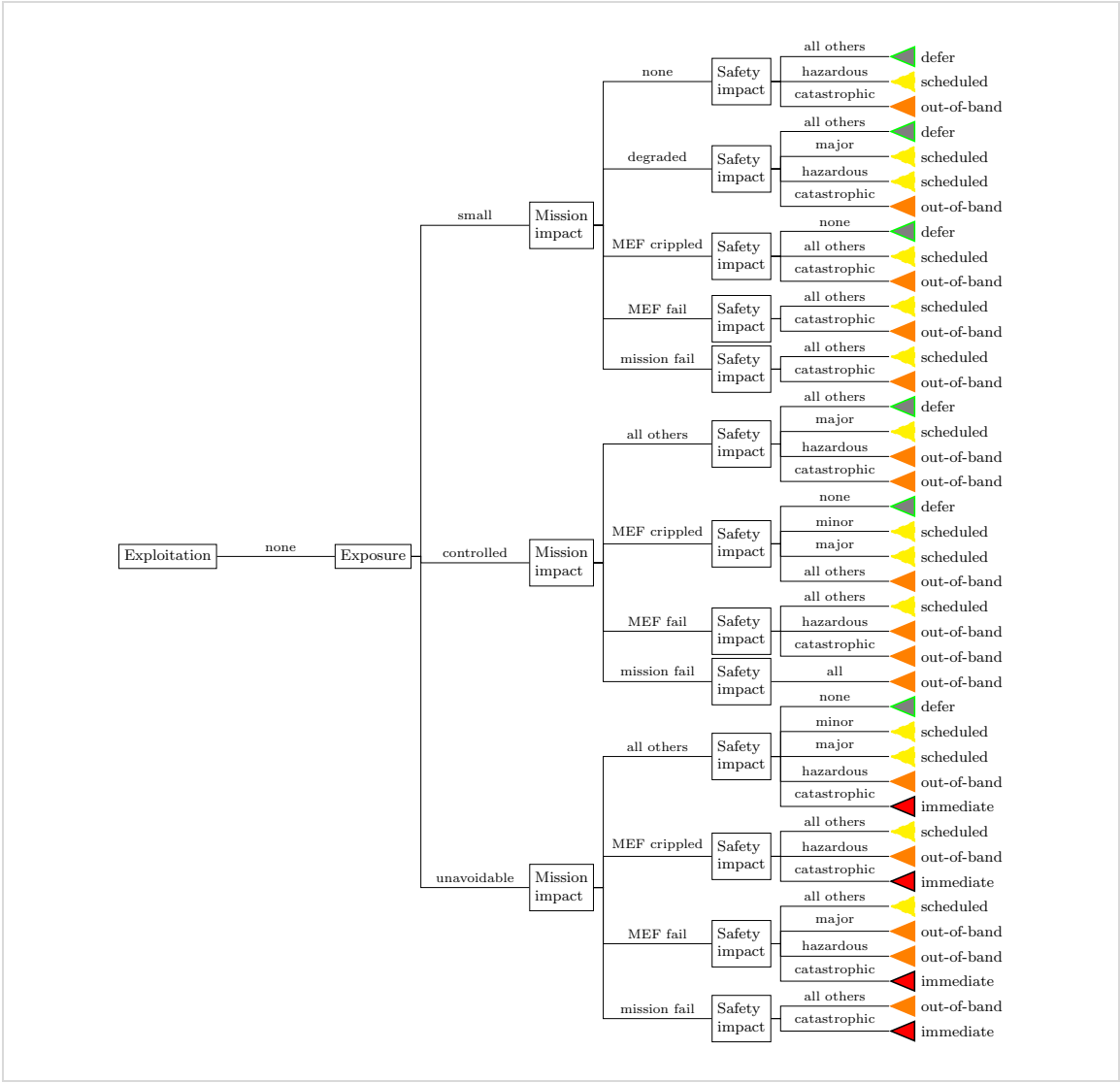

*Figure 2: Proposed Vulnerability Prioritization Decision Tree for Patch Appliers (Continued in Figure 3 and Figure 4**Error! Reference source not found.**)*

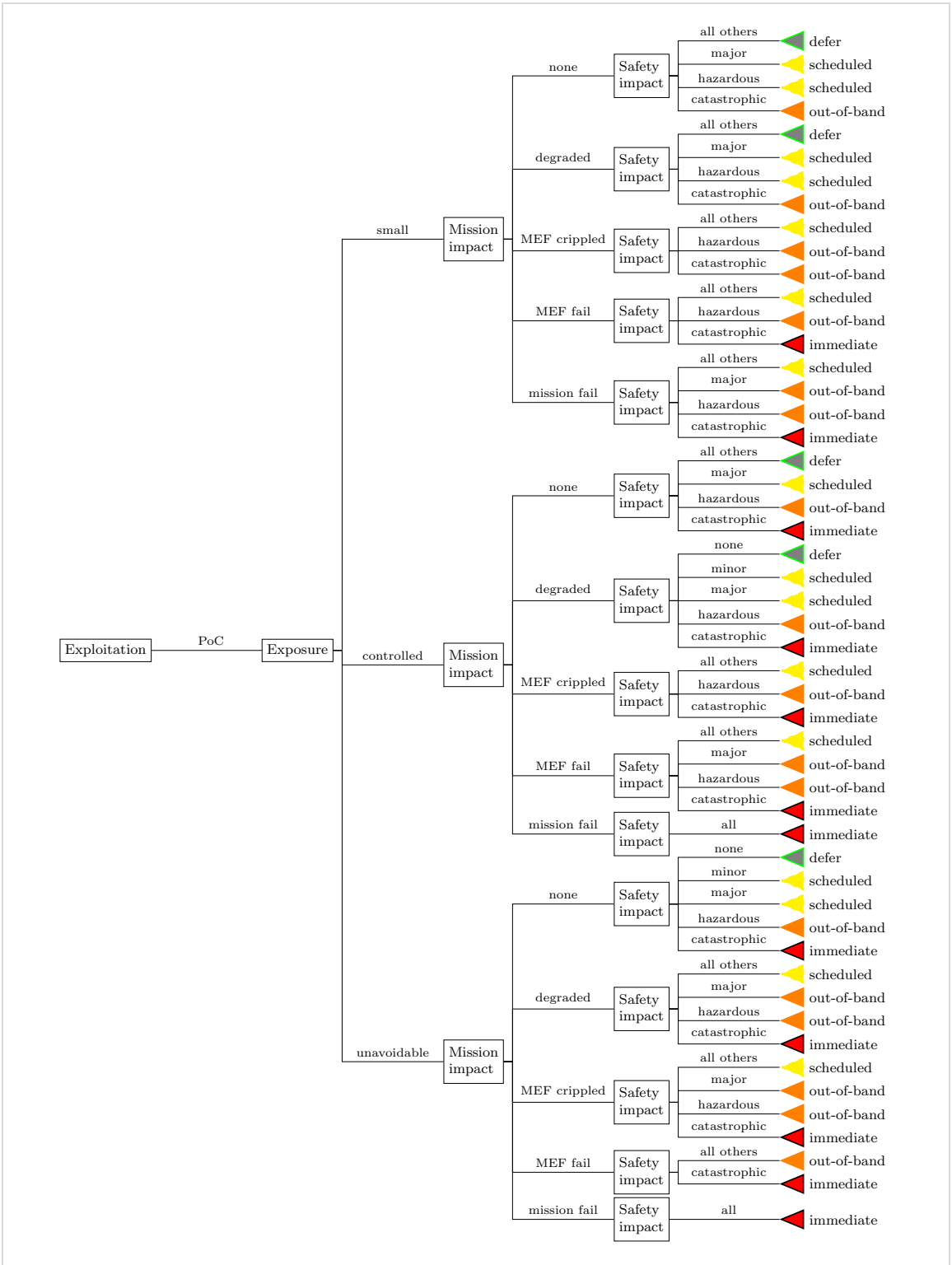

Figure 3: Proposed Vulnerability Prioritization Decision Tree for Patch Appliers (Continued from Figure 2 and in Figure 4).

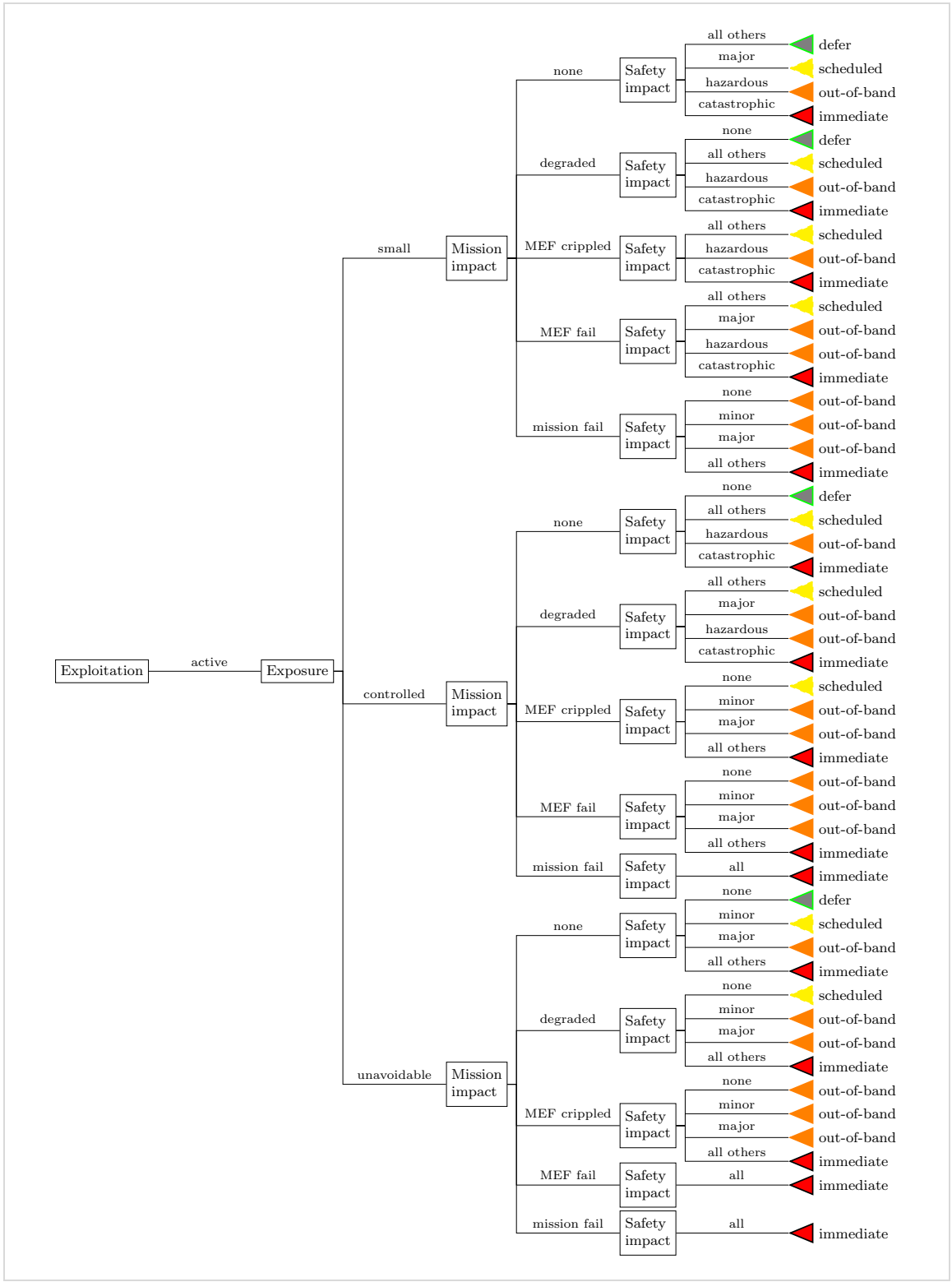

Figure 4: Proposed Vulnerability Prioritization Decision Tree for Patch Appliers (Continued from Figure 2 and Figure 3)

4.8 Evidence Gathering Guidance

To answer each of these decision points, a patch developer or patch applier should, as much as possible, have a repeatable evidence collection and evaluation process. However, we are proposing decisions for humans to make, so 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 must analyze 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 play a role in evaluating exposure, but the entire exposure question cannot be reduced to a binary question of whether an organization's assets appear in such databases. Once the applier has the situational awareness to understand MEFs 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 vulnerabilities were produced by the developer as "scheduled" patches, and one was "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, so they should assume exposure is unavoidable. If the decision maker 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 provide evidence that no one's well-being will suffer. The reach of software exploits is no longer limited to a research network. Similarly, with mission impact, 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.

4.9 Development Methodology

Our initial starting point for the decision trees was different than what we present here. For example, we initially hypothesized different trees for different sectors (e.g., safety-critical, highly regulated, and everyone else). The initial trees also included additional decision points, such as developer's patch distribution channels and 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 author would evaluate the priority of that vulnerability. This method quickly revealed some problems; 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 focused on decision trees for the patch developer and patch applier; we left the coordination decision for future work.

For this tabletop refinement, we could not select a mathematically representative set of CVEs. The goal was to select a handful of CVEs 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 one 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 satisfied that the decision trees were clearly defined and captured our intentions, we began the formal evaluation of the draft trees, which we describe in the next section.

5 Evaluation of the Draft Trees

We conducted a pilot test on the adequacy of the hypothesized decision trees. The method of the pilot test is described in Section 5.1. The study resulted in several changes to the hypothesized trees; we capture those changes and the reason for each of them in Section 5.2. The trees in Sections 4.6 and 4.7 include the improvements reported in Section 5.3.

5.1 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 did not form 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 agreed on the descriptions of the decision points and the descriptions of their values in (a prior version of) Section 4.4. 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 Section 5.3.

In the design of the pilot, we held constant the information available about the vulnerability. This strategy restricted the analyst to decisions based on the 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. After the framework is more stable, a separate study should be devised that shows how analysts should answer the questions in the framework. The basis for the 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 paper focuses only on the structure of decision making. Advice about how to search for information about a vulnerability is a separate project that will be part of 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 11 provides an example of the information provided to each analyst. The developer portfolio details use ~~strikeout font~~ because this decision item was removed after the pilot. The decision procedure for each case is as follows: for each analyst, for each vulnerability, for each stakeholder group, do the following.

1. Start at the root note of the relevant decision tree (patch applier or patch developer).
2. Document the decision branch that matches the vulnerability for this stakeholder context.
3. Document the evidence that supports that decision.
4. Repeat this decision-and-evidence process until the analyst reaches a leaf node in the tree.

Table 11: 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 content management system (CMS) seems to be fairly popular among 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-shopping 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."[39]
State of Exploitation	Appears to be active exploitation of this vulnerability according to NVD. They have linked to the exploit: 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 (i.e., 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 the 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 customers with a valuable product at a reasonable price, while still turning a profit to run the 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 the web designer Cheryl and the IT admin Sally are allowed to access the CMS as users. They login 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 and site are publicly available.

This test structure produced a series of lists similar in form to the contents of Table 12. Analysts also noted how much time they spent on each vulnerability in each stakeholder group.

Table 12: 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 stage produced 18 sets of decisions (nine vulnerabilities across each of two 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 the 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

[39] https://en.wikipedia.org/wiki/Content_management_system

amended. After the second stage, we calculated some statistical measures of inter-rater agreement to help guide the analysis of problem areas.

To assess agreement, we calculate Fleiss' kappa, both for the value in the leaf node reached for each case and every decision in a case.[40] 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. "Leaf node reached" is described 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 that agree on 0 of 4 decisions. So the two measures of agreement (per decision and per path) are complementary, and we report both.

5.1.1 Pilot participant details

The pilot participants are the five authors plus one analyst who had not seen the draft system before participating. Five of the six participants had spent at least one year as professional vulnerability analysts prior to the pilot (Spring was the exception). Three of the participants had at least ten years of experience each. The participants experience is primarily as coordinators at the CERT® Coordination Center. On the one hand, this is a different perspective than either developers or appliers; on the other, the coordinator role is an information broker that often interacts with these perspectives.[41]

These participant demographics limit the generalizability of the results of the pilot. Even though the results cannot be systematically generalized to other analysts, there are at least three benefits to conducting the pilot among this limited demographic. First, it should surface any material tacit disagreements about term usage among the authors. Tacit agreements that are not explained in the text likely survive the pilot study without being acknowledged, but places where the authors tacitly disagreed should be surfaced. We found this to be the case; Section 5.3 documents these results. Second, the pilot provides a case study that demonstrate SSVC is at least possible for some small group of analysts to use. This achievement is not large, but it is a first step. Thirdly, the pilot provides a proof of concept method and metric that any vulnerability prioritization method could use to examine usability for analysts more generally. While the effect of education on vulnerability assessment with CVSS has been tested,[42] we are not aware of any current vulnerability prioritization method that tests usability or agreement among analysts as part of the development process. Future work on SSVC as well as further development of other prioritization methods can benefit from using the method described in the pilot. Future instances should use more representative participant demographics.

[40] Fleiss, Joseph L., and Jacob Cohen. "The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability." *Educational and psychological measurement* 33, no. 3 (1973): 613-619.

[41] Allen D. Householder; Garret Wassermann; Art Manion; & Chris King. The CERT® Guide to Coordinated Vulnerability Disclosure. Section 3. https://vuls.cert.org/confluence/display/CVD/3.+Roles+in+CVD

[42] *Ibid.* 14

5.1.2 Vulnerabilities used as examples

The vulnerabilities used as case studies are as follows. All quotes are from the National Vulnerability Database (NVD) and are illustrative of the vulnerability; however, during the study each vulnerability was evaluated according to information analogous to that in Table 11.

Safety-Critical Cases

- CVE-2015-5374: "Vulnerability … in [Siemens] 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…"

5.2 Pilot Results

For each of the nine CVEs, six analysts rated the priority of the vulnerability as both a developer and applier. Table 13 summarizes the results by reporting the inter-rater agreement for each decision point. For all measures, agreement (κ) is above zero, which is generally interpreted as some agreement among analysts. Below zero is interpreted as noise or discord. Closer to 1 indicates 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. 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 as a descriptive statistic rather than a test statistic.

Table 13 is encouraging, though not conclusive. $\kappa<0$ is a strong sign of discordance. Although it is unclear how close to 1 is success, $\kappa<0$ would be clear sign of failure. In some ways, these results may be undercounting the agreement for SSVC as presented. These results are for SSVC prior to the improvements documented in Section 5.3, which are implemented in SSVC as presented in Section 4. On the other hand, the participant demographics may inflate the inter-rater agreement based on shared tacit understanding through the process of authorship. The one participant who was not an author surfaced two places where this was the case, but we expect the organizational homogeneity of the participants has inflated the agreement somewhat. The anecdotal feedback from vulnerability managers at several organizations (including VMware[43] and McAfee) is about refinement and tweaks, not gross disagreement. Therefore, while further refinement is necessary, this evidence suggests the results have some transferability to other organizations and are not a total artifact of the participant organization demographics.

Table 13: Inter-Rater Agreement for Decision Points

	Safety Impact	Exploitation	Technical Impact	Portfolio Value	Mission Impact	Exposure	Dev Result	Applier Result
Fleiss' κ	0.122	0.807	0.679	0.257	0.146	0.480	0.226	0.295
Disagreement range	2 max 4	1 max 2	1 max 1	1 max 1	2 max 4	1 max 2	1 max 3	2 max 3

[43] Muhammad Akbar. A Critical First Look at Stakeholder Specific Vulnerability Categorization (SSVC). Mar 6, 2020. https://blog.secursive.com/posts/critical-look-stakeholder-specific-vulnerability-categorization-ssvc/

For all decision points, the presumed goal is for κ to be close or equal to 1. The statistics literature has identified some limited cases in which Fleiss' k behaves strangely – for example it is lower than expected when raters are split between 2 of q ratings when q>2.[44] This paradox may apply to the safety and mission impact values, in particular. The paradox would bite hardest if the rating for each vulnerability was clustered on the same two values, for example, minor and major. Falotico and Quatto's proposed solution is to permute the columns, which is safe with unordered categorical data. Since the nine vulnerabilities do not have the same answers as each other (that is, the answers are not clustered on the same two values), we happen to avoid the worst of this paradox, but the results for safety impact and mission impact should be interpreted with some care.

This solution identifies another difficulty of Fleiss' kappa, namely that it does not preserve any order; none and catastrophic are considered the same level of disagreement as none and minor. Table 13 displays a sense of the range of disagreement to complement this weakness. This value is the largest distance between rater selections on a single vulnerability out of the maximum possible distance. So, for safety impact, the most two raters disagreed was by two steps (none to major, minor to hazardous, or major to catastrophic) out of the four possible steps (none to catastrophic). The only values of κ that are reliably comparable are those with the same number of options (that is, the same maximum distance). In other cases, closer to 1 is better, but how close is close enough to be considered "good" changes. In all but one case, if raters differed by two steps then there were raters who selected the central option between them. The exception was mission impact for CVE-201814781; it is unclear whether this discrepancy should be localized to a poor test scenario description, or to SSVC's mission impact definition. Given it is an isolated occurrence, we expect the scenario description at least partly.

Nonetheless, κ provides some way to measure improvement on this a conceptual engineering task. The 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. For a standard to be reliably applied across different analyst backgrounds, skill sets, and cultures, a 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.[45]

Table 14: SSVC pilot scores compared with the CVSS base scores for the vulnerabilities provided by NVD.

CVE-ID	Representative SSVC decision values[46]	SSVC recommendation (dev, applier)	NVD's CVSS base score
CVE-2014-0751	`E:N/T:T/U:L/S:H/X:C/M:C`	(Sched, OOC)	7.5 (High) (v2)

[44] Falotico, Rosa, and Piero Quatto. "Fleiss' kappa statistic without paradoxes." *Quality & Quantity* 49, no. 2 (2015): 463-470.

[45] This is not the case with CVSSv3; expert assessment of scores varies widely. See Figure 1 in *The Effect of Security Education and Expertise on Security Assessments: The Case of Software Vulnerabilities* by Luca Allodi, Marco Cremonini, Fabio Massacci, and Woohyun Shim, published in WEIS in 2018.

[46] We have not discussed a convenient compressed expression of a set of SSVC decision points. Initialisms similar to a CVSS vector are used here; Exploitation (E), Technical impact (T), Utility (U), Safety Impact (S), Exposure (X), and Mission impact (M). S:M is minor, S:J is major; M:F is MEF failure, M:M is mission failure. However, since we created Utility in response to the System Value metric's shortcomings, the pilot results do not include systematic consensus on Utility values.

CVE-2015-1014	E:N/T:T/U:L/S:J/X:S/M:F	(Sched, Sched)	7.3 (High) (v3.0)
CVE-2015-5374	E:A/T:P/U:L/S:H/X:C/M:F	(Immed, Immed)	7.8 (High) (v2)
CVE-2017-3183	E:N/T:T/U:E/S:M/X:C/M:C	(Sched, Sched)	8.8 (High) (v3.0)
CVE-2017-5638	E:A/T:T/U:S/S:M/X:U/M:C	(Immed, OOC)	10.0 (Critical) (v3.0)
CVE-2017-9590	E:P/T:T/U:E/S:M/X:U/M:D	(OOC, Sched)	5.9 (Medium) (v3.0)
CVE-2018-14781	E:P/T:P/U:L/S:H/X:C/M:F	(OOC, OOC)	5.3 (Medium) (v3.0)
CVE-2019-2691	E:N/T:P/U:E/S:M/X:C/M:C	(Sched, Sched)	4.9 (Medium) (v3.0)
CVE-2019-9042	E:A/T:T/U:L/S:N/X:C/M:C	(OOC, Sched)	7.2 (High) (v3.0)

14 presents the mode decision point value for each vulnerability tested, as well as the recommendation that would result from that set based on the trees in Sections 4.6 and 4.7. The comparison with the NVD's CVSS base scores mostly confirms that SSVC is prioritizing based on different criteria, as designed. In particular, differences in the state of exploitation and safety impact are suggestive.

Based on these results, we made about ten changes, some bigger than others. We did not execute 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.

5.3 Improvements Instigated by the Pilot

The following changes are reflected in Section 4, Decision Trees for Vulnerability Management.

- Technical impact: We clarified that partial/total is decided regarding the system scope definition, which considers a database or a web server program as the "whole" system. Furthermore, "total" also includes any technical impact that exposes authentication credentials to the adversary, if those credentials are to the whole system.
- We added advice for information gathering to answer safety impact and mission impact questions. This change is needed because of the particularly wide variety of background assumptions analysts made that influenced results and agreement.
- We clarified that "MEF failure" refers to any *one* essential function failing, not failure of all of them. We changed most severe mission impact to "mission failure" to better reflect the relationship between MEFs and the organization's mission.
- We removed the "developer portfolio value" question since it had poor agreement, and there is no clear way to fix it. We replaced this question with *utility*, which better captures the relevant kinds of value (namely, to the adversary) of the affected component while remaining amenable to pragmatic analysis.
- We clarified that "proof of concept" (see *exploitation*) includes cases in which existing tooling counts as a PoC. The examples listed are suggestive, not exhaustive.
- We reorganized the decision trees based on which items are easier to gather information for or which ones have a widely verifiable state. This change moved *exploitation* to the first question.

- We changed 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.

5.3.1 Questions Removed as Ineffective

In this section, we present 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 the reasons for excluding them are instructive, will help prevent others from re-inventing the proverbial wheel, and can guide thinking on future work.

Initially, we brainstormed approximately 12 potential decision points, most of which we removed early in our development process through informal testing. These decision points included adversary's strategic benefit of exploiting the vulnerability, state of legal or regulatory obligations, 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 points left marks on other decision points. The decision point "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 now a major safety impact. The "business losses to the applier" decision point is covered as a mission impact insofar as profit is a mission of publicly traded corporations.

Three of the above decision points left no trace on the current system. "State of legal or regulatory obligations," "cost of developing a patch," and "patch distribution readiness" 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 decision point, "adversary's strategic benefit of exploiting the vulnerability," 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 of exploiting the vulnerability" 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 13 demonstrates, there was relatively little agreement among the six analysts about how to evaluate this decision point. We replaced this sense of portfolio value with *utility*, which combines value density and virulence.

6 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 Section 4.7. The pilot used slightly different trees, as noted in Section 5.3.

The analyst's first question is related to exploitation. Technically, one could answer the questions in any order; however, exploitation is a good starting point because given an adequately defined search procedure, one can always answer whether it finds an available exploit or proof of concept. The scenario description for the pilot study reads as follows:

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

This information rules out "active" given the (perhaps limited) search procedure. While the search did not produce a precise PoC, based on the description of the vulnerability, 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 then ask about exposure. This considers the (fictional) applier scenario blurb and 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 the 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 on-site 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 the optional Bluetooth connection between the medical device and a phone app represents "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." Regardless, 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, since most of the hospital's clients have not installed the app, 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 the fictional pilot scenario, "Our mission dictates that the 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 MEFs. However, even from this abstract, it seems clear that "do no harm" is at risk due to this vulnerability. A mission essential function to that mission is each of the various medical devices works as expected, or at least if 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), the 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 its 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 13. For the purposes of this example, imagine that after gathering that information, the monitoring situation is adequate, and select "hazardous." Therefore, mitigate this vulnerability *out-of-band*, meaning that it should be addressed quickly, ahead of the usual update and patch cycle.

7 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.

7.1 Requirements Gathering via Sociological Research

The community should know what users of a vulnerability prioritization system want. To explore their needs, it's important to understand how people actually use CVSS and what they think it tells them. In general, such empirical, grounded evidence about what practitioners and decision makers want from vulnerability scoring is lacking. 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.

7.2 Further Decision Tree Testing

More testing with diverse analysts is necessary before the decision trees are 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.

7.3 Decision Tree for Vulnerability Coordination

Currently, only two stakeholders are addressed: patch appliers and patch developers. Expanding the work to include more types of stakeholders would be beneficial. We propose that the next stakeholder group could be vulnerability coordinators, as described in Section 4.1. 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.

8 Limitations

Even as a working proposal, SSVC has some limitations. These are inherent limits of the approach, which should be understood as tradeoffs. There are other limiting aspects of our implementation, but those have been covered as topics that need improvement and are described in Section 7.

We made two important tradeoffs compared to the current state of the practice with CVSS:

1. We eliminated numerical scores; this may make some practitioners uncomfortable. We explained the reasons for this in depth, but even though 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 we choose to convert to is immaterial, as 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.

2. We incorporated a wider variety of inputs from contexts beyond the affected component. Some organizations are not prepared or configured to reliably produce such data (e.g., around mission impact or safety impact). There is adequate guidance for how to elicit and curate this type information from various risk management frameworks, including OCTAVE.[47] Not every organization is going to have sufficiently mature risk management functions to apply SSVC.

   This limitation should be approached with two strategies: (1) organizations should be encouraged and enabled to mature their risk management capabilities and, in the meantime, (2) organizations such as NIST could consider developing 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 well-being" that is explicitly a sort of default advice for otherwise uninformed organizations that can then explicitly account for national priorities, such as critical infrastructure.

[47] 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

9 Conclusion

We presented a working hypothesis for how patch developers and patch appliers should prioritize their effort to mitigate different vulnerabilities. We have performed an initial pilot evaluation of the proposal and improved it, but the process we developed for evaluation is more important than the results. We invite further refinement of the prioritization mechanism. Further testing will be required before SSVC is ready for operational use. 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.

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