Carnegie Mellon University
Software Engineering Institute

Designing Vultron:
A Protocol for Multi-Party Coordinated Vulnerability Disclosure (MPCVD)

(Vultron v0.4.0)

Allen D. Householder

September 2022

SPECIAL REPORT
CMU/SEI-2022-SR-012
DOI: 10.1184/R1/19852798

CERT Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM22-0559

Table of Contents

List of Figures

List of Tables

Acknowledgments

The author thanks his colleagues at the CERT Coordination Center (CERT/CC) for their contributions to the development of the protocol proposed in this report: Brad Runyon, Laurie Tyzenhaus, Chuck Yarbrough, Jonathan Spring, and Art Manion. Together, they helped keep the ball rolling by reviewing work in progress, paying attention to the details, asking insightful questions, and suggesting numerous improvements along the way.

I also owe my gratitude to Eric Hatleback, Timur Snoke, Vijay Sarvepalli, and Sam Perl, whose curiosity and conversations were helpful in sifting through a number of concepts integral to the underlying principles of the protocol proposed in this report.

Thanks as well to Barbara White and Sandy Shrum for their helpful editing advice.

Colledanchise and Ögren's book *Behavior Trees in Robotics and AI: An Introduction* [6], coupled with Dr. Ögren's Behavior Tree tutorials on YouTube provided essential background for Chapter 7.

This work was made possible through funding from the Cybersecurity and Infrastructure Security Agency (CISA) in support of the CERT/CC's Multi-Party Coordinated Vulnerability Disclosure (MPCVD) efforts.

Abstract

The Coordinated Vulnerability Disclosure (CVD) process addresses a human coordination problem that spans individuals and organizations. In this report, we propose a formal protocol specification for Multi-Party Coordinated Vulnerability Disclosure (MPCVD) with the goal of improving the interoperability of both CVD and MPCVD processes. The *Vultron* protocol is composed of three interacting Deterministic Finite Automata (DFAs) for each CVD case Participant representing the Report Management (RM), Embargo Management (EM), and CVD Case State (CS) processes. Additionally, we provide guidance and commentary on the associated MPCVD Participant capabilities and behaviors necessary for this interoperability to be realized.

1 Introduction

The Coordinated Vulnerability Disclosure (CVD) process addresses a human coordination problem that spans individuals and organizations. As we wrote in *The CERT® Guide to Coordinated Vulnerability Disclosure* [14, 13],

> Perhaps the simplest description of the CVD process is that it starts with at least one individual becoming aware of a vulnerability in a product. This discovery event immediately divides the world into two sets of people: those who know about the vulnerability, and those who don't. From that point on, those belonging to the set that knows about the vulnerability iterate on two questions:
>
> 1. What actions should I take in response to this knowledge?
> 2. Who else needs to know what, and when?
>
> The CVD process continues until the answers to these questions are "nothing," and "nobody."

CVD *Is* Multi-Party Coordinated Vulnerability Disclosure (MPCVD), and MPCVD *Is* CVD. Any given CVD case is made up of many individual disclosure events, for example,

- from a Finder to one or more Vendors and/or Coordinators
- from Vendors and Coordinators to other Vendors and Coordinators
- from Finders, Vendors, and Coordinators to Deployers and the Public

In recent years, software supply chains have evolved such that software library and component vulnerabilities have become just as much a part of the everyday CVD process as vulnerabilities in Vendors' proprietary code. This means that many CVD cases we encounter require coordination across multiple vendors. As a result, we find it decreasingly useful to differentiate between "traditional" (i.e., two-party) CVD and MPCVD. In this report, we use both terms interchangeably.

$$\text{CVD} \iff \text{MPCVD}$$

Practically speaking, this means that readers should not infer from our use of CVD in one place that we meant to exclude the multi-party scenario, nor that our use of MPCVD implies the exclusion of the single-vendor CVD scenario. Instead, our intent is to construct a protocol that adequately addresses the MPCVD scenario where $N_{vendors} \geq 1$ and for which the "traditional" CVD case is merely a special (and often simpler) case where $N_{vendors} = 1$.

Context of Our Recent Work. This report, in the context of recent CVD work at the CERT Coordination Center (CERT/CC), is one of four foundational documents aimed at increasing the professionalization of the CVD process. The following is the full set of foundational documents (thus far):

- *The CERT Guide to Coordinated Vulnerability Disclosure* (the *CVD Guide*) in both its original [14] and updated forms [13], provides a "field guide" perspective to the CVD process and its natural extension into MPCVD.
- *A Stakeholder-Specific Vulnerability Categorization* [27, 28, 29] provides decision support for prioritizing vulnerability response activities closely associated with the CVD process.

- *A State-Based Model for Multi-Party Coordinated Vulnerability Disclosure* [10] describes a model that encompasses all possible CVD case histories into a set of measures and metrics for the efficacy of CVD processes. That report is an expanded version of "Are We Skillful or Just Lucky? Interpreting the Possible Histories of Vulnerability Disclosures," an article published in the Association for Computing Machinery (ACM) Journal of Digital Threats: Research and Practice (DTRAP) [12].
- This report, which proposes an abstract formal protocol for MPCVD, ties together various concepts from all three of the above.

Whereas the *CVD Guide* offers a narrative description of both the CVD process and the many scenarios one can expect to encounter as a Participant therein, in this report, we provide an additional layer of formality in the form of a *protocol* for MPCVD.

What We Mean by *Protocol*. We first define what we mean by our use of the term *protocol* by providing a few common usages from the Oxford English Dictionary [26].

- (Computing and Telecommunications) A (usually standardized) set of rules governing the exchange of data between given devices, or the transmission of data via a given communications channel.
- (In extended use) the accepted or established code of behavior in any group, organization, or situation; an instance of this.

Both usages are relevant to this report. First, insofar as we seek to scale the MPCVD process through the use of automation and software-augmented human processes, we wish to propose a formal technical protocol that can serve as the basis of such technical tools. Chapter 6 addresses this first definition in specific detail after explicating its component parts and their interactions in Chapters 2, 3, 4, and 5.

Second, recognizing that MPCVD is primarily a coordination process among human Participants with the goal of remediating extant vulnerabilities in deployed information systems, a protocol must not only address the technical formalities of communicating code but also extend to the expected behaviors of the human Participants that rely on it. In this second sense, we address the term *protocol* in these ways:

- The *CVD Guide* offers a *narrative* protocol for practitioners to follow based on decades of accumulated experience and observation of the CVD process at the CERT/CC [14, 13].
- The CVD Case State (CS) model from the Householder and Spring 2021 report [10] offers a *prescriptive* protocol outlining the high-level goals of the CVD process, as derived from a first-principles approach to possible CVD case histories.
- This report describes a *normative* protocol designed to structure and guide practitioners toward those goals.

To that end, we offer this report as a proposal for such an MPCVD protocol.

Why *Vultron*? The working name for our protocol is *Vultron*, an homage to the fictional robot Voltron. In the Voltron animated series, a team of protectors joins forces to defend the universe from their adversaries. Their defensive mission requires a combination of independent defenders coordinating their efforts to achieve their goals. Like Voltron, our MPCVD protocol comprises a combination of humans and the technical processes and mechanisms that empower them. Together, those humans, processes, and mechanisms must function both individually and in coordination and cooperation with others to protect information systems and the people who depend on them from exploitation by adversaries.

Version Numbering Scheme. While we have not yet mapped out a future release schedule, in anticipation of future revisions, we have chosen a semantic versioning scheme for the Vultron protocol. Specifically, Vultron versions will be assigned according to the format *MAJOR.MINOR.MICRO*, where

- *MAJOR* represents the zero-indexed major version for the release.
- *MINOR* represents a zero-indexed counter for minor releases that maintain compatibility with their MAJOR version.
- *MICRO* represents an optional zero-indexed micro-version (patch) counter for versions that update a MINOR version.

Trailing zero values may be omitted (e.g., `3.1` and `3.1.0` denote the same version, similarly `5` and `5.0`). It may be useful at some point to use pre-release tags such as *-alpha*, *-beta*, *-rc* (with optional zero-indexed counters as needed), but we reserve that decision until their necessity becomes clear. The same goes for build-specific tags; while we do not currently have a use for them, we do not rule out their future use.

Because of the early nature of the current version of the protocol (`0.4.0`), as of this writing, no backward compatibility commitments are made or implied within the `0.x` versions. We anticipate this commitment will change as we get closer to a major release.

1.1 Goals

The overall goal of our MPCVD protocol effort is to achieve *interoperability* among CVD process implementations according to the broad definition of that term found in the SEI report, *Current Perspectives on Interoperability* [21]:

Interoperability. The ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics

This definition encompasses both (a) *syntactic* and (b) *semantic* interoperability. The goal of this report is to lay the foundation for the *semantic* interoperability of CVD processes across Participants. *Syntactic* interoperability, in the form of message formats and the like, will be left as future work (§9).

Addressing *semantic interoperability* first is a deliberate choice. If we were to go in the reverse order, we might wind up with systems that exchange data quickly and accurately yet still fail to accomplish the mutually beneficial outcomes that MPCVD sets out to achieve. Carney et al. illustrate the importance of semantic interoperability in their report *Some Current Approaches to Interoperability* [5]:

> There is a limited number of ways that agreements on meaning can be achieved. In the context of design-time interoperability, semantic agreements are reached in the same manner as interface agreements between the constituent systems... However, in the context of run-time interoperability, the situation is more complex, since there is need for some manner of universal agreement, so that a new system can join, ad-hoc, some other group of systems. The new system must be able to usefully share data and meaning with those other systems, and those other systems must be able to share data and meaning from an unfamiliar newcomer.

In this excerpt, replace the word "system" with the concept of a "CVD Case Participant," and the need for semantic interoperability as a means of achieving coordination in MPCVD becomes clear:

> [. . .] However, in the context of run-time interoperability, the situation is more complex, since there is need for some manner of universal agreement, so that a new CVD Participant can join, ad-hoc, some other group of CVD Participants [in a CVD Case]. The new CVD Case Participant must be able to usefully share data and meaning with those other CVD Case Participants, and those other Participants must be able to share data and meaning from an unfamiliar newcomer.

Elsewhere in the same report, Carney et al. write [5],

> In the hoped-for context of unbounded systems of systems, trust in the actions and capabilities provided by interoperating parties is essential. Each party to an interaction must have, develop, or perceive a sense of whether the actions of interoperating parties can be trusted. This sense of trust is not Boolean (e.g., parties can be trusted to varying degrees), is context dependent (Party A can be trusted in a particular context but not in another), and is time sensitive (Party A can be trusted for a certain period). Further, the absence of trust——distrust——is less dangerous than misplaced trust: it is better to know that you cannot trust a particular party than to misplace trust in a party

The protocol we propose is intended to promote trust between MPCVD Participants both within an individual case as well as over time and across cases.

1.2 Objectives

The objectives of this report are as follows:

1. Provide a set of common primitives to serve as an ontological foundation for CVD process definitions across organizations.
2. Construct abstract workflows that support the inter-organizational coordination and synchronization required for the CVD process to be successful.
3. From those primitives and workflows, identify a set of message types needed for the CVD process to function.
4. Provide high-level requirements for the semantic content of those message types.

1.3 What Does "Success" Mean in CVD

We take as a base set of criteria the ordering preferences given in the Householder and Spring 2021 report [10]. While we incorporate this model fully in Chapter 4, some notation is necessary to proceed here. The CS model is built on the idea that there are six events of significance in the lifespan of every vulnerability: Vendor Awareness, Fix Ready, Fix Deployed, Public Awareness, Exploit Public, and Attacks Observed. Brief descriptions of those events are listed in Table 1.1.

The Householder and Spring 2021 report defines a set of 12 ordering preferences over these 6 events [10]. We present them in roughly descending order of desirability according to the partial order developed in that report [10]. Items closer to the top of the list are indicators of CVD skill. The symbol $\prec$ is read as *precedes*.

Fix Deployed Before Public Awareness ($\mathbf{D} \prec \mathbf{P}$). For a fix to be deployed prior to public awareness, a lot has to go right in the CVD process: The vendor has to know about the vulnerability, create a fix, and deploy it——all without public knowledge——and has to achieve all that prior to any exploits being published or attacks becoming known to the public. Furthermore, it requires that the Vendor has the capability to deploy fixes without intervention by the system owner or user, which is a rare engineering feat unattain-

Table 1.1: CS Transition Events [10]

Case Event	Symbol	Description
Vendor Awareness	**V**	The Vendor knows the vulnerability exists.
Fix Ready	**F**	A fix has been created by the Vendor and is ready to be deployed.
Fix Deployed	**D**	The fix has been deployed to vulnerable instances.
Public Awareness	**P**	The public becomes aware of the existence of the vulnerability, whether through the successful completion of a CVD process or otherwise.
Exploit Public	**X**	An exploit for the vulnerability is made public.
Attacks Observed	**A**	Attacks against vulnerable instances of the software have been observed.

able by many software supply chains. More often, fix deployment ($\mathbf{D}$) requires users and/or system owners (Deployers) to take action. The need to inform Deployers implies a need for public awareness of the vulnerability, making this criteria impossible to achieve in those scenarios.

Fix Ready Before Public Awareness ($\mathbf{F} \prec \mathbf{P}$). Deployers (i.e., the public) can take no action until a fix is ready. Because public awareness also implies adversary awareness, the vendor-adversary race becomes even more critical when this condition is not met. Only Vendors who can receive *and act* on vulnerability reports—whether those reports originate from inside or outside of the organization—are able to achieve this goal.

Fix Deployed Before Exploit Public ($\mathbf{D} \prec \mathbf{X}$). Deploying a fix before an exploit is made public helps reduce the net risk to end users.

Fix Deployed Before Attacks Observed ($\mathbf{D} \prec \mathbf{A}$). Attacks occurring before a fix has been deployed are when there's maximum risk to users; therefore, we wish to avoid that situation.

Fix Ready Before Exploit Public ($\mathbf{F} \prec \mathbf{X}$). Exploit publication prior to fix readiness represents a period of increased threat to users since it means that attackers can exploit the vulnerability even if they lack exploit development skills. When fixes are ready before exploits are made public, defenders are better positioned to protect their users.

Vendor Awareness Before Public Awareness ($\mathbf{V} \prec \mathbf{P}$). Public awareness prior to vendor awareness can cause increased support costs for vendors at the same time they are experiencing increased pressure to prepare a fix.

Fix Ready Before Attacks Observed ($\mathbf{F} \prec \mathbf{A}$). As in the case with published exploits, when fixes exist before attacks are observed, defenders are in a substantially better position to protect their users.

Public Awareness Before Exploit Public ($\mathbf{P} \prec \mathbf{X}$). There is broad agreement that it is better for the public to find out about a vulnerability via a CVD process rather than because someone published an exploit for any adversary to use.

Exploit Public Before Attacks Observed ($\mathbf{X} \prec \mathbf{A}$). This criterion is not about whether exploits should be published or not. It is about whether we should prefer histories in which exploits are published *before* attacks happen over histories in which exploits are published *after* attacks happen. Because attackers have more advantages in the latter case than the former, histories in which $\mathbf{X} \prec \mathbf{A}$ are preferable to those in which $\mathbf{A} \prec \mathbf{X}$.

Public Awareness Before Attacks Observed ($\mathbf{P} \prec \mathbf{A}$). Similar to the exploit case above, public awareness via CVD is generally preferred over public awareness resulting from

incident analysis that results from attack observations.

Vendor Awareness Before Exploit Public ($\mathbf{V \prec X}$). If public awareness of the vulnerability prior to vendor awareness is bad, then a public exploit is at least as bad because it encompasses the former and makes it readily evident that adversaries have exploit code available for use.

Vendor Awareness Before Attacks Observed ($\mathbf{V \prec A}$). Attacks prior to vendor awareness represent a complete failure of the vulnerability remediation process because they indicate that adversaries are far ahead of defenders.

Taken together, these twelve ordering preferences constitute the minimum set of outcomes we hope to emerge from the protocol proposed in this report.

1.4 Report Preview

MPCVD is comprised of independent Participants performing their own CVD-related processes. Those processes can be represented by Finite State Machines (FSMs), specifically as Deterministic Finite Automata (DFAs). CVD processes (and the DFAs representing them) can be decomposed hierarchically. We propose three main DFAs as the core of our MPCVD protocol:

1. A Report Management (RM) DFA represents each CVD Participant's engagement with a particular report (Chapter 2).
2. An Embargo Management (EM) DFA negotiates and establishes the timing of future disclosures and publications (Chapter 3).
3. A CVD Case State DFA tracks the events in Table 1.1, as originally described in the Householder and Spring 2021 report [10] and summarized in Chapter 4 of this report.

Chapter 5 contains a discussion of the interactions among these three state machine models.

However, a set of agents independently executing processes is not coordinated, and if they are not coordinated, then whatever they are doing does not deserve the name CVD. Hence, there is a need for a protocol to describe the interactions necessary to coordinate these processes. Communicating FSMs provide a formal way to define a communications protocol, which coordinates the activities of independent DFAs through the interchange (e.g., sending and receiving) of messages [4]. We map our multiple DFA model onto a formal protocol definition in Chapter 6.

However, an MPCVD protocol needs to do more than just provide formally defined communication mechanisms. It also needs to normalize the expected behaviors and activities that the communication protocol enables. In this sense, our protocol expands upon ISO/IEC 29147:2018 *Vulnerability Disclosure*, ISO/IEC 30111:2019 *Vulnerability Handling Processes*, and ISO/IEC TR 5895:2022, which, taken together, provide a high-level normative standard for CVD activities [16, 17, 18].

Developed in response to the growing complexity of video game Non-Player Character (NPC) Artificial Intelligence (AI) FSMs, Behavior Trees offer a way to organize and describe agent behaviors in a straightforward, understandable way. Using Behavior Trees, agent processes can be modeled as sets of behaviors (e.g., pre-conditions, actions, and post-conditions) and the logic that joins them together. Today, Behavior Trees are used in video game software to develop realistic NPCs and in robotics to create reactive and adaptive behaviors from autonomous agents. Behavior Trees offer a high potential for automating complex tasks through a hierarchical decomposition of the logic and actions required to complete those tasks.

The behaviors we are interested in modeling are the various CVD activities described in the *CVD Guide* (e.g., find contacts, send reports, validate reports, prioritize reports, create fixes, publish reports, publish fixes, deploy fixes) [14]. Chapter 7 uses Behavior Trees to describe MPCVD Participant activities and their interactions with the MPCVD protocol proposed in Chapter 6.

Additional implementation notes, including a simplified case data model, can be found in Chapter 8. Chapter 9 covers future work not addressed here. Our conclusion is in Chapter 10.

Appendices are included to provide connections to closely related work: In Appendix A, we provide a mapping between the MPCVD protocol and relevant portions of the Stakeholder-Specific Vulnerability Categorization (SSVC), a vulnerability response prioritization model also developed by the CERT/CC. Appendix B contains a detailed crosswalk of our protocol with ISO/IEC 29147:2018 *Vulnerability Disclosure*, ISO/IEC 30111:2019 *Vulnerability Handling Processes*, and ISO/IEC TR 5895:2022 *Multi-party coordinated vulnerability disclosure and handling*. Appendix C maps concepts from the EM process onto the `iCalendar` protocol.

A list of acronyms is provided at the end of the report.

1.5 Terms and Definitions

Throughout this report, we refer to CVD Roles from the *CERT Guide to Coordinated Vulnerability Disclosure* [14, 13]:

Finder. The individual or organization that identifies the vulnerability

Reporter. The individual or organization that notifies the vendor of the vulnerability

Vendor (Supplier). The individual or organization that created or maintains the vulnerable product

Deployer (User). The individual or organization that must deploy a patch or take other remediation action

Coordinator. An individual or organization that facilitates the coordinated response process

The *Vendor* role is synonymous with the *Supplier* role as it appears in SSVC Version 2 [29]. The *Deployer* role is synonymous with the *User* role in ISO/IEC 29147:2018 and ISO/IEC 30111:2019 [16, 17], while the other roles are named consistent with those standards.

We also add a new role in this report, which we expect to incorporate into a future version of the *CVD Guide*:

Exploit Publisher. An individual or organization that publishes exploits (Exploit Publishers might be the same as Finders, Reporters, Coordinators, or Vendors, but this is not guaranteed. For example, Vendors that produce tools for Cybersecurity Red Teams might play a combination of roles: Finder, Reporter, Vendor, Coordinator, and/or Exploit Publisher.)

Finally, we have a few additional terms to define:

CVD Case. The unit of work for the overall CVD process for a specific vulnerability spanning the individual CVD Case Participants and their respective RM and EM processes

CVD Case Participant. Finder, Vendor, Coordinator, Deployer, etc., as defined above

Vulnerability Report. The unit of work for an individual Case Participant's RM process

A diagram showing the relationship between CVD Cases, Participants, and Reports can be found in Figure 8.1.

1.6 Notation

Before we proceed, we need to formally define our terms. In all of these definitions, we take the standard Zermelo-Fraenkel set theory. We adopt the following notation:

Set Theory Symbols

- $\{\dots\}$ an unordered set that makes no assertions about sequence
- $(\dots)$ depending on the context: (1) an ordered set in which the items occur in that sequence, or (2) a tuple of values
- $|x|$ the count of the number of elements in a list, set, tuple, or vector x
- the normal proper subset ($\subset$), equality ($=$), and subset ($\subseteq$) relations between sets
- $\in$ the membership (is-in) relation between an element and the set it belongs to
- $\prec$ the precedes relation on members of an ordered set: $\sigma_i \prec \sigma_j$ if and only if $\sigma_i, \sigma_j \in s$ and $i < j$ where s is an ordered set
- $|X|$ the size of (the number of elements in) a set X
- $\langle X_i \rangle_{i=1}^{N}$ a set of N sets X_i, indexed by i; used in Chapter 6 in the context of Communicating FSM, taken from the article on "On Communicating Finite State Machines"[4]

Logic Symbols

- $\implies$ implies
- $\iff$ if-and-only-if (bi-directional implication)
- $\wedge$ the logical AND operator
- $\neg$ the logical NOT operator

Directional Messaging Symbols

- $\rightharpoonup$ a message emitted (sent) by a process
- $\leftharpoondown$ a message received by a process

DFA Symbols

- $\rightarrow$ a transition between states, usually labeled with the transition type (e.g., $\xrightarrow{a}$)
- $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)$ specific symbols for individual DFA components that are introduced when needed in Chapters 2, 3, and 4
- $\left\langle \langle S_i \rangle_{i=1}^{N}, \langle o_i \rangle_{i=1}^{N}, \langle M_{i,j} \rangle_{i,j=1}^{N}, succ \right\rangle$ formal protocol symbols that are introduced at the beginning of Chapter 6

Our depictions of DFA as figures use common state diagram symbols (circles and arrows).

We follow Unified Modeling Language (UML) conventions for sequence and class diagrams in Chapters 6 and 8. We introduce a few additional notation details specific to Behavior Trees when needed at the beginning of Chapter 7.

2 Report Management (RM) Model

In this chapter, we describe a high-level workflow for CVD Report Management (RM). The RM process should be reasonably familiar to anyone familiar with Information Technology Service Management (ITSM) workflows such as problem, change, incident or service request management. In particular, any workflow in which work items (e.g., incident reports, problem tickets, change requests) are received, validated, prioritized, and work is subsequently completed, should map onto the RM process outlined in this chapter.

In the interest of maintaining the potential for interoperability among different organizations' internal processes, our protocol does not specify intra-organizational subprocesses within each state, although we give examples of such subprocesses in §7.5. For further reference, ISO/IEC 30111:2019(E) [17] provides recommendations for Vendors' *internal* processes that can be mapped into the RM process. We provide such a mapping in Appendix B.

2.1 RM State Machine

In this section, we first cover the states themselves before proceeding to a discussion of the transitions between them. Next, we provide a discussion of the Participant-specific semantics of the state transitions. We use DFA notation to describe our RM model.

A DFA is defined as a 5-tuple $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)$ [20]:

- $\mathcal{Q}$ is a finite set of states.
- $q_0 \in \mathcal{Q}$ is an initial state.
- $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final (or accepting) states.
- Σ is a finite set of input symbols.
- δ is a transition function $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$.

2.1.1 RM States

Our proposed RM DFA models a report lifecycle containing seven states, shown in (2.1).

$$\begin{aligned} \mathcal{Q}^{rm} = \{&\underline{S}tart,\ \underline{R}eceived,\ \underline{I}nvalid,\ \underline{V}alid,\\ &\underline{A}ccepted,\ \underline{D}eferred,\ \underline{C}losed\} \end{aligned} \tag{2.1}$$

In this example, we use underlined capital letters as a shorthand for the state names. We use this convention throughout the remainder of this report. Each Participant in a CVD case will have their own RM state.

RM states are not the same as CVD case states. Case states follow the Householder-Spring model summarized in §4, as originally described in the 2021 report [10]. Further discussion of the interactions of the RM and CS models is found in §5.2.

2.1.1.1 The *Start* State (S)

The *Start* state is a simple placeholder state for reports that have yet to be received. It is, in effect, a null state that no CVD Participant would be expected to reflect in their report

tracking system. We include it here because it will become useful when we are modeling coordination that spans multiple Participants in a formal protocol in Chapter 6. Otherwise, the discussion until then will mostly ignore it.

2.1.1.2 The *Received* State (*R*)

Reports initially arrive in the *Received* state.

Vendors lacking the ability to receive reports will find it exceedingly difficult if not impossible to participate in the CVD process. Therefore,

- Vendors SHOULD have a clearly defined and publicly available mechanism for receiving reports.

Similarly, those who coordinate others' responses to vulnerability reports also need to have a report receiving capability; otherwise, they are not capable of coordinating vulnerability disclosures. Hence,

- Coordinators MUST have a clearly defined and publicly available mechanism for receiving reports.

Exiting the *Received* state requires a Participant to assess the validity of a report. Note that validation is distinct from prioritization, as covered in §2.1.1.4. As an example, a Vendor might later choose to $defer$ further response on a $Valid$ report due to other priorities.

Validity criteria need not be limited to technical analysis. For instance, a Coordinator might only accept reports within their specific scope of concern and consider reports outside their scope to be $Invalid$ even if they believe the report accurately describes a real vulnerability. Alternatively, a Vendor might institute a policy designating reports unaccompanied by a working proof-of-concept exploit as $Invalid$ by default.

- All Participants SHOULD have a clearly defined process for validating reports in the *Received* state.
- Participants SHOULD perform at least a minimal credibility check on reports as a minimum validation process before exiting the *Received* state.
- Participants MAY perform a more technical report validation process before exiting the *Received* state.
- Regardless of the technical rigor applied in the validation process, Participants SHOULD proceed only after validating the reports they receive.
- Participants SHOULD transition all valid reports to the $Valid$ state and all invalid reports to the $Invalid$ state.
- Regardless of the content or quality of the initial report, once a Vendor confirms that a reported vulnerability affects one or more of their product(s) or service(s), the Vendor SHOULD designate the report as $Valid$.

2.1.1.3 The *Invalid* State (*I*)

Reports in the $Invalid$ state have been evaluated and found lacking by the recipient. This state allows time for the Reporter to provide additional information and for the receiver to revisit the validation before moving the report to *Closed*.

The reasons for a report to be put in this state will vary based on each recipient's validation criteria, and their technical capability and available resources. The $Invalid$ state is intended to be used as a temporary holding place to allow for additional evidence to be sought to contradict that conclusion.

- Participants SHOULD temporarily hold reports that they cannot validate pending additional information.
- Participants SHOULD provide Reporters an opportunity to update their report with additional information in support of its validity before closing the report entirely.
- Participants MAY set a timer to move reports from *Invalid* to *Closed* after a set period of inactivity.

2.1.1.4 The *Valid* State (*V*)

Reports in the *Valid* state are ready to be prioritized for possible future work. The result of this prioritization process will be to either accept the report for follow-up or defer further effort.

- Once a report is in the *Valid* state, Participants MAY choose to perform a shallow technical analysis on it to prioritize any further effort relative to other work.
- Participants SHOULD have a bias toward accepting rather than deferring cases up to their work capacity limits.

In other words, prioritization is only necessary if the workload represented by active valid reports exceeds the organization's capacity to process those reports.

Prioritization schemes, such as SSVC [29] or the Common Vulnerability Scoring System (CVSS) [24], are commonly used to prioritize work within the CVD process; however, specific details are left to Participant-specific implementation.[1]

2.1.1.5 The *Accepted* State (*A*)

The *Accepted* state is where the bulk of the work for a given CVD Participant occurs. Reports reach this state for a Participant only once the Participant has deemed the report to be both valid and of sufficient priority to warrant further action. The *Accepted* state has a different meaning for each different Participant.

- For our purposes, Finders/Reporters enter the *Accepted* state only for reports that they intend to put through the CVD process. If they have no intention of pursuing CVD, there is no need for them to track their actions using this protocol. See §2.2.1.
- Vendors usually do root cause analysis, understand the problem, and produce a fix or mitigation.
- Coordinators typically identify potentially affected Vendors, notify them, and possibly negotiate embargoes.

We provide additional elaboration on the sorts of activities that might happen in the *Accept* state in §7.5.

- A report MAY enter and exit the *Accepted* state a number of times in its lifespan as a Participant resumes or pauses work (i.e., transitions to/from the *Deferred* state).

2.1.1.6 The *Deferred* State (*D*)

The *Deferred* state is reserved for valid, unclosed reports that are otherwise not being actively worked on (i.e., those in *Accepted*). It parallels the *Invalid* state for reports that fail to meet the necessary validation criteria in that both states are awaiting closure once it is determined that no further action is necessary.

[1] See also Appendix A, where we connect a few of the dots between SSVC and this protocol model.

For example, a Participant might use the $Deferred$ state when a valid report fails to meet their prioritization criteria (2.7), or when a higher priority task takes precedence over an active report, as in (2.9).

- A report MAY enter and exit the $Deferred$ state a number of times in its lifespan as a Participant pauses or resumes work (i.e., transitions from/to the *Accepted* state).
- Reports SHOULD exit the $Deferred$ state when work is resumed (2.10), or when the Participant has determined that no further action will be taken (2.11).
- CVD Participants MAY set a policy timer on reports in the $Deferred$ state to ensure they are moved to *Closed* after a set period of inactivity.

2.1.1.7 The *Closed* State (C)

The *Closed* state implies no further work is to be done; therefore, any pre-closure review (e.g., for quality assurance purposes) should be performed before the case moves to the *Closed* state (i.e., while the report is in $Invalid$, $Deferred$, or *Accepted*).

- Reports SHOULD be moved to the *Closed* state once a Participant has completed all outstanding work tasks and is fairly sure that they will not be pursuing any further action on it.

2.1.1.8 RM Start and End States

The RM process starts in the *Start* state.

$$q_0^{rm} = Start \tag{2.2}$$

The RM process ends in the *Closed* state.

$$\mathcal{F}^{rm} = \{Closed\} \tag{2.3}$$

2.1.2 RM State Transitions

The actions performed in the RM process represent the allowed state transitions in the corresponding DFA.

- A Participant's RM process begins when the Participant *receive*s a report.
- Each Participant SHOULD subject each *Received* report to some sort of validation process, resulting in the report being designated as *valid* or *invalid* based on the Participant's particular criteria.

In other words, the *Received* state corresponds to the Validation phase of *The CERT Guide to Coordinated Vulnerability Disclosure* [14].

- For $Valid$ reports, the Participant SHOULD perform a prioritization evaluation to decide whether to *accept* or $defer$ the report for further work.

Similarly, the $Valid$ state is equivalent to the Prioritization (Triage) phase of the *CVD Guide* [14]. The SSVC model is illustrative here, although any prioritization scheme could be substituted as long as it emits a result that can be mapped onto the semantics of "continue work" or "defer further action" [29]. Appendix A takes a closer look at how SSVC fits into the protocol we are defining.

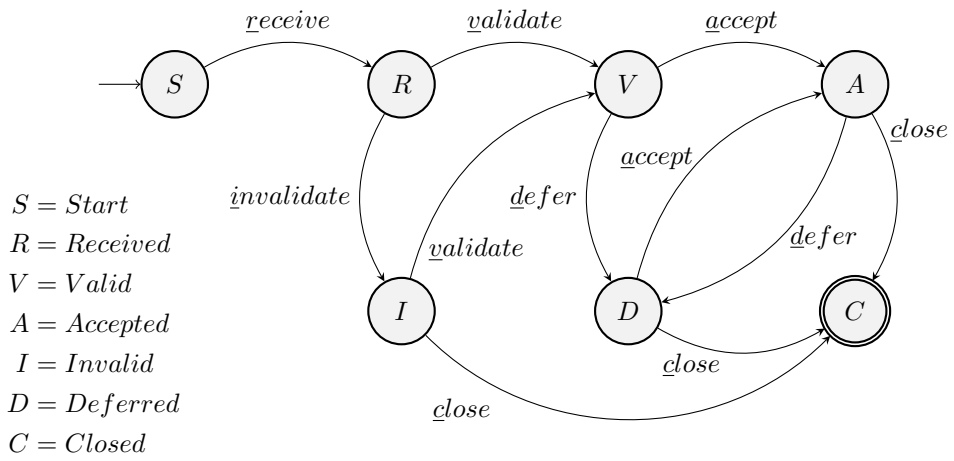


Figure 2.1: The RM Process

Note that the $Start$ state is a placeholder to assist with modeling the coordination process as a formal protocol; it is not a "real" state. Underlined letters in transition names will be used as shorthand in both the text and subsequent figures.

- Participants SHOULD *close* reports that require no further work (e.g., those that have been in $Invalid$ or $Deferred$ for some length of time, or those in *Accepted*, where all necessary tasks are complete.)

These actions constitute the set of symbols for the RM DFA, as shown in (2.4).

$$\Sigma^{rm} = \{\underline{r}eceive,\ \underline{v}alidate,\ \underline{i}nvalidate,\ \underline{a}ccept,\ \underline{d}efer,\ \underline{c}lose\} \tag{2.4}$$

2.1.2.1 RM Transitions Defined

In this section, we define the allowable transitions between states in the RM process model. The RM process, including its states and transitions, is depicted in Figure 2.1.

To begin, a Participant must receive a report. Recall that the *Start* state is a placeholder, so this action simply puts the receiving Participant into the *Received* state at the beginning of their involvement in the case.

$$Start \xrightarrow{receive} Received \tag{2.5}$$

The Participant must validate the report to exit the *Received* state. Depending on the validation outcome, the report will be in either the $Valid$ or $Invalid$ state.

$$Received \rightarrow \begin{cases} \xrightarrow{validate} Valid \\ \xrightarrow{invalidate} Invalid \end{cases} \tag{2.6}$$

Once a report has been validated (i.e., it is in the RM $Valid$ state, $q^{rm} \in V$), the Participant must prioritize it to determine what further effort, if any, is necessary. Appendix A contains an example of how the SSVC model can be applied here, although any prioritization scheme could be substituted. Prioritization ends with the report in either the *Accepted* or $Deferred$ state.

$$Valid \rightarrow \begin{cases} \xrightarrow{accept} Accepted \\ \xrightarrow{defer} Deferred \end{cases} \tag{2.7}$$

Some Participants (e.g., Finders and Coordinators) need to engage someone else (e.g., a Vendor) to resolve a case. To do this, the *sender* Participants must also be in the *Accepted* state; otherwise, why are they working on the case? In the following equation, we use brackets and subscripts to indicate the interaction between two instances of the RM model: one bracket represents the *sender* and *receiver* states before the message is transmitted, while the other is for the end state of both Participants. Although the *sender*'s state does not change, the *recipient*'s state moves from *Start* to *Received*.

$$\begin{bmatrix} Accepted_{sender} \\ Start_{recipient} \end{bmatrix} \xrightarrow{receive_{recipient}} \begin{bmatrix} Accepted_{sender} \\ Received_{recipient} \end{bmatrix} \tag{2.8}$$

A Participant might choose to pause work on a previously *Accepted* report after revisiting their prioritization decision. When this happens, the Participant moves the report to the $Deferred$ state.

$$Accepted \xrightarrow{defer} Deferred \tag{2.9}$$

Similarly, a Participant might resume work on a $Deferred$ report, moving it to the *Accepted* state.

$$Deferred \xrightarrow{accept} Accepted \tag{2.10}$$

Finally, a Participant can complete work on an *Accepted* report or abandon further work on an $Invalid$ or $Deferred$ report.

$$\begin{bmatrix} Accepted \text{ or} \\ Invalid \text{ or} \\ Deferred \end{bmatrix} \xrightarrow{close} Closed \tag{2.11}$$

Our model assumes that $Valid$ reports cannot be closed directly without first passing through either *Accepted* or $Deferred$. It is reasonable to wonder why *close* is not a valid transition from the $Valid$ state. The answer is that we wanted to allow prioritization and closure to be distinct activities; deferral is reversible, whereas closure is not. Often a Participant might initially $defer$ a case only to resume work later, once more information has arrived. However, there is nothing stopping a Participant from instituting a process that goes from $Valid$ to $Deferred$ to $Closed$ in rapid (even immediate) succession.

2.1.2.2 RM as a Regular Grammar

Following the state machine diagram in Figure 2.1, equation (2.12) represents the RM process model as a right-linear grammar:

$$\delta^{rm} = \begin{cases} S & \to rR \\ R & \to vV \mid iI \\ I & \to vV \mid cC \\ V & \to aA \mid dD \\ A & \to dD \mid cC \\ D & \to aA \mid cC \\ C & \to \epsilon \end{cases} \tag{2.12}$$

The strings generated in the language defined by this grammar can be useful for exploring the possible sequences of states each report might encounter for each Participant. The 15 shortest paths are ric, $rvac$, $rvdc$, $rivac$, $rivdc$, $rvadc$, $rvdac$, $rivadc$, $rvadac$, $rvdadc$, $rivdac$, $rivdadc$,

rvdadac, *rivadac*, and *rvadadc*. Due to the structure of the RM DFA, longer strings just add more $defer\text{-}accept$ (da) or $accept\text{-}defer$ (ad) cycles prior to closure (c). The usually limited duration of the RM process coupled with the tendency for CVD Participants to prefer to avoid frequent starts and stops means that we expect the vast majority of reports to follow one of the above paths, with the remainder falling into marginal extensions. Further discussion of a reward function to evaluate RM DFA strings is discussed as future work in §9.2.1.

2.1.3 RM DFA Fully Defined

Taken in combination, the full definition of the RM DFA $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)^{rm}$ is given by equations (2.1), (2.2), (2.3), (2.4), and (2.12). For convenience, we assembled them into (2.13).

$$RM = \begin{pmatrix} \mathcal{Q}^{rm} = \{S, R, I, V, A, D, C\}, & (2.1) \\ q_0^{rm} = S, & (2.2) \\ \mathcal{F}^{rm} = \{C\}, & (2.3) \\ \Sigma^{rm} = \{r, i, v, a, d, c\}, & (2.4) \\ \delta^{rm} = \begin{cases} S \rightarrow & rR \\ R \rightarrow & vV \mid iI \\ I \rightarrow & vV \mid cC \\ V \rightarrow & aA \mid dD \\ A \rightarrow & dD \mid cC \\ D \rightarrow & aA \mid cC \\ C \rightarrow & \epsilon \end{cases} & (2.12) \end{pmatrix} \tag{2.13}$$

2.2 RM Discussion

State transitions represent messaging opportunities to communicate CVD case status among Participants.

- CVD Participants SHOULD announce their RM state transitions to the other Participants in a case.

This is the lynchpin that makes the RM model point toward a technical protocol. Every state transition implies a different message type.

2.2.1 The Secret Lives of Finders

While the Finder's *Received*, $Valid$, and $Invalid$ states are useful for modeling and simulation purposes, they are less useful to us as part of a potential CVD protocol. Why? Because for anyone else to know about the vulnerability (and as a prerequisite to CVD happening at all), the Finder must have already validated the report and prioritized it as worthy of further effort to have any reason to attempt to coordinate its disclosure. In other words, CVD only starts *after* the Finder has already reached the *Accepted* state for any given vulnerability to be reported. Correspondingly, this also represents their transition from *Finder* to *Reporter*. Nevertheless, for now, we retain these states for completeness. We revisit this topic in our derivation of a protocol state model for Reporters in §6.3.4.

2.2.2 RM Interactions Between CVD Participants

Each Participant in a case has their own instance of the RM state model. Participants can change their local state independent of the state of other Participants. Events within a CVD case may trigger a state transition in one Participant while no transition occurs in another.

Table 2.1: Common CVD Actions by Role and Their Effects on the RM Model

CVD Role(s)				RM
Finder	Vendor	Coordinator	Action	transition
✓			*Discover vulnerability* (hidden)	(2.5)
✓			*Analyze discovery* (hidden)	(2.6)
✓			*Decide whether to initiate CVD* (hidden)	(2.7)
✓	✓	✓	*Notify Vendor*	(2.8)
✓	✓	✓	*Notify Coordinator*	(2.8)
	✓	✓	*Receive report*	(2.5)
	✓	✓	*Validate report*	(2.6)
✓	✓	✓	*Prioritize report*	(2.7)
✓	✓	✓	*Pause work*	(2.9)
✓	✓	✓	*Resume work*	(2.10)
✓	✓	✓	*Close report*	(2.11)

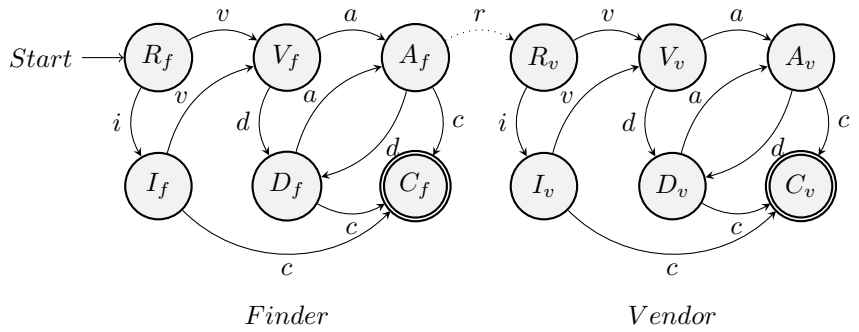


Figure 2.2: Notional Diagram of a Finder RM Deterministic Finite Automaton (DFA) Interacting with a Vendor RM DFA

The Finder proceeds from discovery ($start \rightarrow R_f$) and must reach the A_f state prior to notifying the Vendor (corresponding to the r transition shown in the center), who, in turn, starts in their own R_v state. The goal of a Finder notifying the Vendor is to get the Vendor into state A_v, implying that the report provided should be both valid and of sufficiently high priority that the Vendor will make both the v and a transitions upon receipt. The report submission transition (r) triggers the $S \xrightarrow{r} R$ transition in the Vendor but does not alter the Finder's state.

For example, the *notify another Participant* action in (2.8) shows that even though the *sender* is the one taking the action, it is the *recipient*'s state that changes. Table 2.1 lists role-based actions. A few examples of this model applied to common CVD and MPCVD case scenarios follow.

Finder-Vendor CVD. A simple Finder-Vendor CVD scenario is shown in Figure 2.2. As explained in §2.2.1, many of the Finder's states would be hidden from view until they reach the *Accepted* (A_f) state. The *receive* action bridging $A_f \xrightarrow{r} R_v$ corresponds to the *notify other participants* action defined by (2.8).

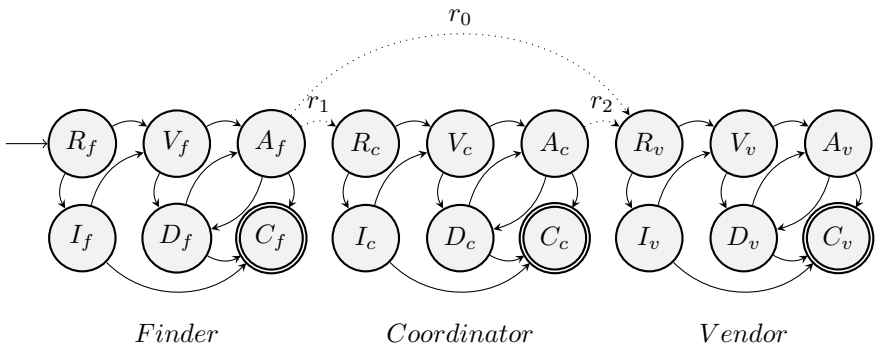


Figure 2.3: Notional Diagram of a Finder RM DFA Interacting with RM DFAs from Both a Coordinator and a Vendor

As in Figure 2.2, the Finder must reach the A_f state prior to notifying anyone. In this scenario, the Finder first attempts to notify the Vendor (r_0) but fails (for whatever reason) to achieve the goal of inducing the Vendor into state A_v. The Finder then notifies the Coordinator (r_1), who, in turn, starts in their own R_c state. The Finder's goal in notifying the Coordinator is to get the Coordinator into state A_c, implying that the report provided should be both valid and of sufficiently high priority that the Coordinator will make both the v and a transitions upon receipt. In turn, the Coordinator must reach the A_c state prior to notifying the Vendor (r_2), who subsequently starts in their own R_v state. Once again, the goal of notifying the Vendor is to get the Vendor into state A_v, implying the need to prompt the Vendor to make both the v and a transitions upon receipt. State transition labels are omitted for clarity but are consistent with Figures 2.1 and 2.2. The report submission transitions (r_0, r_1, r_2) correspond to the $S \xrightarrow{r} R$ transition for each recipient, although they do not necessarily trigger a state transition in the sender.

Finder-Coordinator-Vendor CVD. A slightly more complicated scenario in which a Finder engages a Coordinator after failing to reach a Vendor is shown in Figure 2.3. This scenario is very common in our experience at the CERT/CC, which should come as no surprise considering our role as a Coordinator means that we do not participate in cases following the previous example. Here we see three notification actions corresponding to (2.8):

- First, $A_f \xrightarrow{r_0} R_v$ represents the Finder's initial attempt to reach the Vendor.
- Next, $A_f \xrightarrow{r_1} R_c$ is the Finder's subsequent attempt to engage with the Coordinator.
- Finally, the Coordinator contacts the Vendor in $A_c \xrightarrow{r_2} R_v$.

MPCVD with a Coordinator and Multiple Vendors. A small MPCVD scenario is shown in Figure 2.4. As with the other examples, each notification shown is an instance of the *notify other participants* action from (2.8). Contrary to the previous example, this scenario starts with the Finder contacting a Coordinator, perhaps because they recognize the increased complexity of coordinating multiple Vendors' responses.

- First, $A_f \xrightarrow{r_0} R_c$ represents the Finder's initial report to the Coordinator.
- Next, $A_c \xrightarrow{r_1} R_{v_1}$ shows the Coordinator contacting the first Vendor.
- Finally, the Coordinator contacts a second Vendor in $A_c \xrightarrow{r_2} R_{v_2}$.

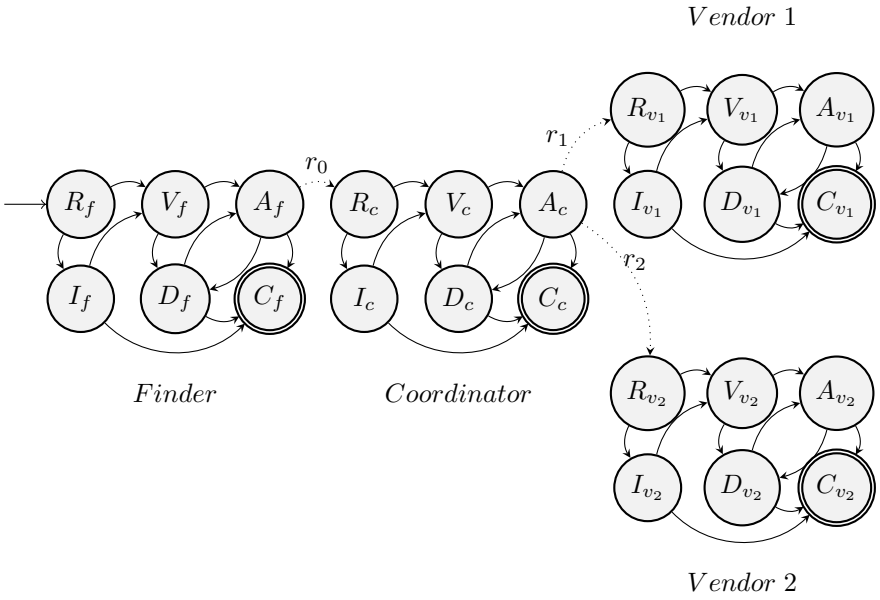
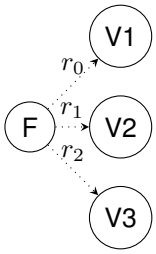
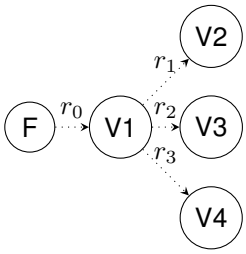


Figure 2.4: Notional Diagram of a Finder RM DFA Interacting with a Coordinator and Two Vendor RM DFAs in an MPCVD Case

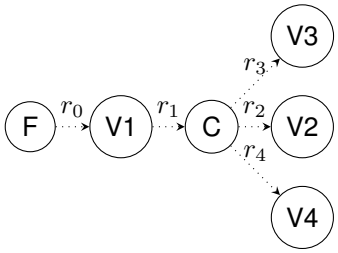
In this scenario, the Finder must reach the A_f state prior to notifying the Coordinator (r_0), who, in turn, starts in their own R_c state. The goal of r_0 is to get the Coordinator into state A_c, implying that the report provided should be both valid and of sufficiently high priority that the Coordinator will make both the v and a transitions upon receipt. In turn, the Coordinator must reach the A_c state prior to notifying the Vendors (r_1, r_2), who, in turn, start in their own R_{v_1} and R_{v_2} states, respectively. The goal of r_1 and r_2 is to get each Vendor into their respective A_v states, implying that the report provided should be both valid and of sufficiently high priority that each Vendor will make both the v and a transitions upon receipt. State transition labels are omitted for clarity but are consistent with Figures 2.1 and 2.2. The report submission transitions (r_0, r_1, r_2) correspond to the $S \xrightarrow{r} R$ transition for the recipient, although they do not necessarily trigger a state transition in the sender.



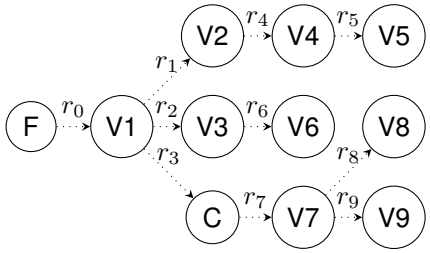
(a) Finder Coordinates MPCVD



(b) Vendor Coordinates MPCVD



(c) Vendor Engages Coordinator for MPCVD



(d) Supply-Chain MPCVD Incl. a Coordinator

Figure 2.5: High-Level Examples of Other Common MPCVD Scenarios

Each node in the diagrams represents a single Participant's entire RM state machine as detailed in Figure 2.1. We have collapsed each Participant into a single node to save space. Note that, as with previous examples, any given Participant must reach the $Accept\ (A)$ state in their own RM process to notify others.

A Menagerie of MPCVD Scenarios. Other MPCVD RM interaction configurations are possible, of course. We demonstrate a few such scenarios in Figure 2.5, where this time each circle represents a Participant's entire RM model. We have observed all of the following interactions at the CERT/CC:

- A Finder notifies multiple Vendors without engaging a Coordinator (Figure 2.5a).
- A Finder notifies a Vendor, who, in turn, notifies other Vendors (Figure 2.5b) or engages a Coordinator to do so (Figure 2.5c).
- Supply-chain oriented MPCVD often has two or more tiers of Vendors being notified by their upstream component suppliers, with or without one or more Coordinators' involvement (Figure 2.5d).

We intend the RM model to be sufficiently composable to accommodate all such permutations.

2.2.3 RM State Subsets

Before proceeding, we pause to define a few useful subsets of RM states ($\cdots \subset \mathcal{Q}^{rm}$) for future use:

$$Open = \{R, I, V, D, A\} \tag{2.14a}$$

$$Valid\ Yet\ Unclosed = \{V, D, A\} \tag{2.14b}$$

$$Potentially\ Valid\ Yet\ Unclosed = \{R, V, D, A\} \tag{2.14c}$$

$$Active = \{R, V, A\} \tag{2.14d}$$

$$Inactive = \{I, D, C\} \tag{2.14e}$$

3 Embargo Management (EM) Model

In this chapter, we describe the basic primitives necessary for the CVD Embargo Management (EM) process. For our purposes, an embargo is an *informal* agreement among peer CVD case Participants to refrain from publishing information about a vulnerability until some future point in time relative to the report at hand. Once an embargo has expired, there is no further restriction on publishing information about the vulnerability.[2]

CVD case Participants must be able to propose, accept, and reject embargo timing proposals according to their individual needs. Additionally, Participants may want to attempt to gain agreement that enables specific details about a vulnerability to be shared with other Participants or made public. Such content considerations are outside the scope of this proposal. We focus our discussion on the *when* of an embargo, not the *what*.

Unlike the RM model, in which each Participant has their own instance of the RM DFA, EM states are a global property of a CVD case.

- A CVD case SHALL NOT have more than one active embargo at a time.

Even in an MPCVD case having a vertical supply chain—in which Vendors must wait on their upstream suppliers to produce fixes before they can take action on their own, as in Figure 2.5d—our intent is that the embargo period terminates when as many Vendors as possible have been given an adequate opportunity to produce a fix.

Embargoes Are Not NDAs. Importantly, CVD embargoes are *not* Non-Disclosure Agreements (NDAs). An NDA (also known as a Confidentiality agreement) is a legally binding contract between parties, often accompanied by a reward for compliance and/or some penalty in the event of unauthorized disclosure. NDAs do, on occasion, have a place in CVD processes, but the relatively rapid pace and scale of most MPCVD embargoes makes per-case NDAs prohibitively difficult. As a result, we are intentionally setting aside NDA negotiation as beyond the scope of this proposal.

On the surface, many bug bounty programs may appear to fall outside our scope because they are often structured as NDAs in which compliance is rewarded. For some bounty programs, the penalty for non-compliance or early disclosure is limited to the loss of the reward. For others, non-compliance can lead to the forfeiture of a promise of amnesty from the pursuit of civil or criminal charges that might otherwise apply because of security or acceptable-use policy violations. Nevertheless, we are optimistic that the bulk of this protocol (i.e., the parts that do not interfere with the contractual provisions of bounty-associated NDAs) will be found to be compatible with the full variety of bounty-oriented CVD programs existing now and in the future.

3.1 EM State Machine

As with our definition of the RM model in Chapter 2, we describe our EM model using DFA notation.

To recap, A DFA is defined as a 5-tuple $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)$ [20].

- $\mathcal{Q}$ is a finite set of states
- $q_0 \in \mathcal{Q}$ is an initial state

[2] Reminder: Exploits are information about vulnerabilities too.

- $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final (or accepting) states
- Σ is a finite set of input symbols
- δ is a transition function $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$

3.1.1 EM States

CVD cases are either subject to an active embargo or they are not. We begin with a simple two-state model for the embargo state:

$$\mathcal{Q}^{em}_{simple} = \{None,\ Active\} \tag{3.1}$$

However, because embargo management is a process of coordinating across Participants, it will be useful to distinguish between the $None$ state and an intermediate state in which an embargo has been proposed but not yet accepted or rejected. We might call this the $None + Proposed$ state, but we shortened it to $Proposed$.

Similarly, we want to be able to discriminate between an $Active$ embargo state and one in which a revision has been proposed but is not yet accepted or rejected, which we will denote as the $Active + Revise$ state, shortened to $Revise$. Finally, we wish to distinguish between the state in which no embargo has ever been established ($None$), and the final state after an active embargo has ended ($eXited$). Combining these, we get the following set of EM states, which we denote as $\mathcal{Q}^{em}$ in (3.2).

$$\begin{aligned} \mathcal{Q}^{em} = \{&\underline{N}one, \\ &\underline{P}roposed, \\ &\underline{A}ctive, \\ &\underline{R}evise, \\ &e\underline{X}ited\} \end{aligned} \tag{3.2}$$

As a reminder, we use the underlined capital letters as shorthand for EM state names later in the document. Also note that $q^{em} \in A$ is distinct from $q^{rm} \in A$. An embargo can be $Active$, while a Report can be $Accepted$, and these are independent states. Be sure to check which model a state's shorthand notation is referring to.

Start and Final States. The EM process starts in the $None$ state. The process ends in one of two states: If an embargo agreement is eventually reached, the EM process ends in the $eXited$ state. Otherwise, if no agreement is ever reached, the EM process ends in the $None$ state. Formal definitions of each are shown in (3.3) and (3.4), respectively.

$$q^{em}_0 = None \tag{3.3}$$

$$\mathcal{F}^{em} = \{None,\ eXited\} \tag{3.4}$$

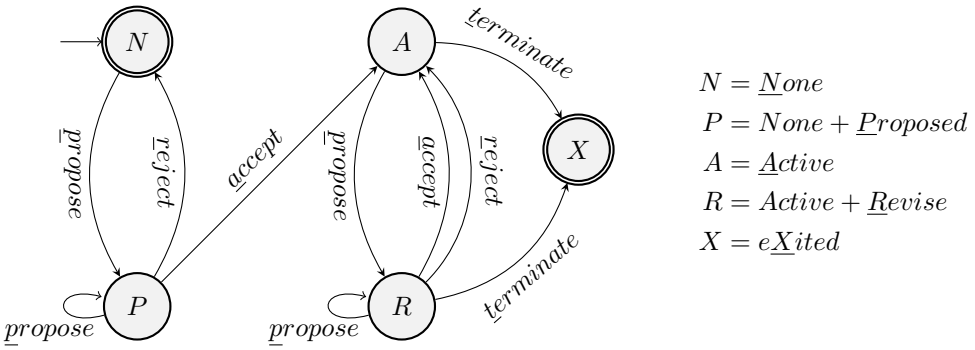


Figure 3.1: The EM Process

Initial embargo negotiations occur on the left side of the diagram, while renegotiation happens on the right.

3.1.2 EM State Transitions

The symbols of our EM DFA are the actions that cause transitions between the states:

- An embargo MAY be *propose*d.
- Once proposed, it MAY be *accept*ed or *reject*ed.
- Once accepted, revisions MAY be *propose*d, which MAY, in turn, be *accept*ed or *reject*ed.
- Finally, accepted embargoes MUST eventually *terminate*.

A summary of the available actions is shown as Σ^{em} in (3.5).

$$\Sigma^{em} = \{\ \underline{p}ropose,\ \underline{r}eject,\ \underline{a}ccept,\ \underline{t}erminate\} \tag{3.5}$$

Once again, the underlined lowercase letters will be used as shorthand for the EM transition names in the remainder of the document.

3.1.2.1 EM Transitions Defined

Now we define the possible state transitions. Figure 3.1 summarizes the EM process DFA states and transitions.

Propose a new embargo when none exists:

$$None \xrightarrow{propose} Proposed \tag{3.6}$$

Accept or reject a proposed embargo:

$$Proposed \begin{cases} \xrightarrow{accept} Active \\ \xrightarrow{reject} None \end{cases} \tag{3.7}$$

An existing embargo can also be renegotiated by proposing a new embargo. The existing embargo remains active until it is replaced by accepting the revision proposal.

$$Active_{old} \xrightarrow{propose(new)} Revise \tag{3.8}$$

If the newly proposed embargo is accepted, then the old one is abandoned. On the other hand, if the newly proposed embargo is rejected, the old one remains accepted.

$$Revise \begin{cases} \xrightarrow{accept(new)} Active_{new} \\ \xrightarrow{reject(new)} Active_{old} \end{cases} \tag{3.9}$$

Existing embargoes can terminate due to timer expiration or other reasons to be discussed in §3.2.7. Termination can occur even if there is an open revision proposal.

$$\begin{bmatrix} Active \\ Revise \end{bmatrix} \xrightarrow{terminate} eXited \tag{3.10}$$

3.1.2.2 A Regular Grammar for EM

Based on the actions and corresponding state transitions just described, we define the transition function δ^{em} for the EM process as a set of production rules for the right-linear grammar using our single-character shorthand in (3.11).

$$\delta^{em} = \begin{cases} N \rightarrow \ pP \mid \epsilon \\ P \rightarrow \ pP \mid rN \mid aA \\ A \rightarrow \ pR \mid tX \\ R \rightarrow \ pR \mid aA \mid rA \mid tX \\ X \rightarrow \ \epsilon \end{cases} \tag{3.11}$$

Due to the numerous loops in the DFA shown in Figure 3.1, the EM grammar is capable of generating arbitrarily long strings of *propose-propose* and *propose-reject* histories matching the regular expression `(p*r)*(pa(p*r)*(pa)?t)?`. As an example, here is an exhaustive list of all the possible traces of length seven or fewer:

> *pr*, *pat*, *ppr*, *ppat*, *papt*, *prpr*, *pppr*, *ppppr*, *pprpr*, *prppr*, *pappt*, *ppapt*, *pppat*, *papat*, *paprt*, *prpat*, *pppppr*, *papppt*, *prpppr*, *ppprpr*, *ppappt*, *pppapt*, *prprpr*, *papapt*, *pprppr*, *pappat*, *paprpt*, *prppat*, *prpapt*, *ppaprt*, *pprpat*, *ppapat*, *papprt*, *ppapprt*, *ppappat*, *prpppat*, *prpapat*, *ppprpat*, *ppppppr*, *pprppat*, *papapat*, *paprpat*, *ppapapt*, *prprpat*, *paprprt*, *prppapt*, *pppapat*, *pprpapt*, *pppaprt*, *pppppat*, *prpappt*, *papaprt*, *pappppt*

However, because EM is a human-oriented scheduling process, our experience suggests that we should expect there to be a natural limit on CVD Participants' tolerance for churn during embargo negotiations. Hence, we expect most paths through the EM DFA to be on the short end of this list in practice. We offer some thoughts on a potential reward function over EM DFA strings as future work in §9.2.2.

For example, it is often preferable for a Vendor to accept whatever embargo the Reporter initially proposes followed closely by proposing a revision to their preferred timeline than it is for the Vendor and Reporter to ping-pong proposals and rejections without ever establishing an embargo in the first place. In the worst case (i.e., where the Reporter declines to extend their embargo), a short embargo is preferable to none at all. This implies a preference for strings

starting with *par* over strings starting with *ppa* or *prpa*, among others. We will come back to this idea in §3.2.6 and in the worked protocol example at the end of Chapter 6, specifically in §6.9.2.

3.1.3 EM DFA Fully Defined

Taken together, the complete DFA specification for the EM process $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)^{em}$ is shown in equations (3.2), (3.3), (3.4), (3.5), and (3.11), respectively. For convenience, we have assembled them in shorthand form in (3.12).

$$EM = \left(\begin{array}{ll} \mathcal{Q}^{em} = \{N, P, A, R, X\}, & (3.2) \\ q_0^{em} = N, & (3.3) \\ \mathcal{F}^{em} = \{N, X\}, & (3.4) \\ \Sigma^{em} = \{p, r, a, t\}, & (3.5) \\ \delta^{em} = \begin{cases} N \rightarrow\ pP \mid \epsilon \\ P \rightarrow\ pP \mid rN \mid aA \\ A \rightarrow\ pR \mid tX \\ R \rightarrow\ pR \mid aA \mid rA \mid tX \\ X \rightarrow\ \epsilon \end{cases} & (3.11) \end{array} \right) \tag{3.12}$$

3.2 EM Discussion

Embargoes are a means of inhibiting public disclosure of a vulnerability while defenses are prepared (e.g., until fix development has completed for a reasonable quorum of Vendors). The goal of the EM process is not to establish an exact publication schedule for every Participant to adhere to. Rather, it is to establish a window spanning from the present to some future time, during which Participants are expected not to publish or otherwise disclose information about the vulnerability to non-Participants outside of the CVD case.

3.2.1 Embargo Principles

An embargo is a social agreement between independent parties acting in the interest of providing vulnerability fixes to users in a timely manner while minimizing attacker advantage in the interim. However, embargoes are not always appropriate or useful within the context of any given CVD case.

With that in mind, we offer the following principles as guidance. We begin with some behavioral norms that define what it means to cooperate with an embargo.

- Embargo Participants SHOULD NOT knowingly release information about an embargoed case until either
  1. all proposed embargoes have been explicitly rejected
  2. no proposed embargo has been explicitly accepted in a timely manner
  3. the expiration date/time of an accepted embargo has passed
  4. an accepted embargo has been terminated prior to the embargo expiration date and time due to other reasons (e.g., those outlined in §3.2.7)
- Additional Participants MAY be added to an existing embargo upon accepting the terms of that embargo.

- Adding Participants to an existing embargo SHALL NOT constitute "release" or "publication" so long as those Participants accept the terms of the embargo. See §3.2.10.

Furthermore, we need to establish a few norms related to embargo timing.

- An embargo SHALL specify an unambiguous date and time as its endpoint.
- An embargo SHALL NOT be used to indefinitely delay publication of vulnerability information, whether through repeated extension or by setting a long-range endpoint.
- An embargo SHALL begin at the moment it is accepted.
- Embargoes SHOULD be of short duration, from a few days to a few months.

3.2.2 Embargo Scale and Duration

Given all other facts about a vulnerability report being equal, there are two factors that contribute significantly to the success or failure of an embargo: scale and duration. The more people involved in an embargo, the more likely the embargo is to fail.

- Embargo participation SHOULD be limited to the smallest possible set of individuals and organizations needed to adequately address the vulnerability report.

Similarly, the longer an embargo lasts, the more likely it is to fail.

- Embargo duration SHOULD be limited to the shortest duration possible to adequately address the vulnerability report.

3.2.3 Embargo Participants Are Free to Engage or Disengage

As we described at the beginning of the chapter, an embargo is not the same thing as an NDA, even if they have similar effects. Because it is a contract, an NDA can carry civil or even criminal penalties for breaking it. CVD embargoes have no such legal framework. Hence, CVD Participants are free to enter or exit an embargo at any time, for any reason. In fact, CVD Participants are not obliged to agree to any embargo at all. However, regardless of their choices, Participants should be clear about their status and intentions with other Participants. There are a few good reasons to exit an embargo early. (See §3.2.7.)

- Participants MAY propose a new embargo or revision to an existing embargo at any time within the constraints outlined in §3.2.4.
- Participants MAY reject proposed embargo terms for any reason.
- Participants in an embargo MAY exit the embargo at any time.

Note that a Participant leaving an embargo is not necessarily the same as the embargo itself terminating. Embargo termination corresponds to the $q^{em} \in \{A, R\} \xrightarrow{t} X$ transition in the EM model and reflects a consensus among case Participants that the embargo no longer applies. A Participant leaving an *Active* embargo means that the embargo agreement between other Participants remains intact, but that the leaving Participant is no longer involved in the case.

- Participants stopping work on a case SHOULD notify remaining Participants of their intent to adhere to or disregard any existing embargo associated with the case.
- Participants SHOULD continue to comply with any active embargoes to which they have been a part, even if they stop work on the case.
- Participants who leave an *Active* embargo SHOULD be removed by the remaining Participants from further communication about the case.

These points imply a need for Participants to track the status of other Participants with respect to their adherence to the embargo and engagement with the case. We will return to these concepts with the *case_engagement* and *embargo_adherence* attributes described in §8.1.5.

CVD is an iterated game, and actions have consequences. Leaving an embargo early in one case may have repercussions to Participants' willingness to cooperate in later cases.

- A Participant's refusal to accept embargo terms MAY result in that Participant being left out of the CVD case entirely.
- Participants SHOULD consider other Participants' history of cooperation when evaluating the terms of a proposed embargo.

Finally, embargo termination removes a constraint rather than adding an obligation.

- Participants SHOULD not publish information about the vulnerability when there is an active embargo.
- Participants MAY publish information about the vulnerability when there is no active embargo.
- Embargo termination SHALL NOT be construed as an obligation to publish.

A discussion of how to decide who to invite to participate in a CVD case is addressed in §3.2.10.

3.2.4 Entering an Embargo

Negotiating and entering into a new embargo for a case is only possible within an embargo "habitable zone" defined in terms of the CS model as laid out below. The notation for CS model states is explained in Chapter 4, but the contextual explanation below should suffice for now.

- CVD Participants MUST NOT *propose* or *accept* a new embargo negotiation when any of the following conditions are true:
  1. Information about the vulnerability is already known to the public ($q^{cs} \in \cdots P \cdot\cdot$).
  2. An exploit for the vulnerability is publicly available ($q^{cs} \in \cdots\cdot X \cdot$).
  3. There is evidence that the vulnerability is being actively exploited by adversaries ($q^{cs} \in \cdots\cdot\cdot A$).
- CVD Participants MAY *propose* or *accept* an embargo in all other case states ($q^{cs} \in \cdots pxa$).
- CVD Participants SHOULD NOT *propose* or *accept* a new embargo negotiation when the fix for a vulnerability has already been deployed ($q^{cs} \in VFDpxa$). Counterexamples include (a) when an embargo is desired to allow for a downstream Vendor to synchronize their fix delivery or deployment, and (b) when a Vendor has deployed a fix but wants to complete their root cause analysis prior to releasing information about the vulnerability.
- CVD Participants MAY *propose* or *accept* a new embargo when the fix for a vulnerability is ready but has neither been made public nor deployed ($q^{cs} \in VFdpxa$). Such an embargo SHOULD be brief and used only to allow Participants to prepare for timely publication or deployment.

3.2.5 Negotiating Embargoes

Asymmetry is inherent in the CVD process because those who currently have the vulnerability information get to decide who they will share it with. This asymmetry puts Reporters at somewhat of an advantage when it comes to the initial report submission to another Participant. We will discuss some ways to improve (but not fully remove) this asymmetry in §3.2.6, but for now we just need to acknowledge that it exists.

- Participants MAY *accept* or *reject* any proposed embargo as they see fit.
- Receivers SHOULD *accept* any embargo proposed by Reporters.
- Receivers MAY *propose* embargo terms they find more favorable as they see fit.
- Participants MAY withdraw (*reject*) their own unaccepted *Proposed* embargo.

Respond Promptly. Timely response to embargo proposals is important. Explicit acceptance is expected.

- Participants SHOULD explicitly *accept* or *reject* embargo proposals in a timely manner. (For example, embargo agreement or rejection SHOULD NOT be tacit.)
- Participants MAY interpret another Participant's failure to respond to an embargo proposal in a timely manner as a *reject*ion of that proposal.
- In the absence of an explicit *accept* or *reject* response from a Receiver in a timely manner, the Sender MAY proceed in a manner consistent with an EM state of $None$ ($q^{em} \in N$).

Don't Give Up. Once an embargo negotiation has failed the first time, Participants have no further obligations. They are, however, encouraged to try again.

- In a case where the embargo state is $None$ and for which an embargo has been *propose*d and either explicitly or tacitly *reject*ed, Participants MAY take any action they choose with the report in question, including immediate publication.
- Participants SHOULD make reasonable attempts to retry embargo negotiations when prior proposals have been *reject*ed or otherwise failed to achieve *accept*ance.

Participants need not wait for embargo negotiations to conclude before submitting a report. However, by doing so, they might give up some of their leverage over the Receiver in the embargo negotiations.

- Participants MAY withhold a report from a Recipient until an initial embargo has been accepted.
- Submission of a report when an embargo proposal is pending ($q^{em} \in P$) SHALL be construed as the Sender's acceptance ($q^{em} \in P \xrightarrow{a} A$) of the terms proposed regardless of whether the Sender or Receiver was the proposer.

Addressing Validation Uncertainty. Participants might prefer to delay accepting or rejecting a proposed embargo until after they have had an opportunity to review the report through the validation and (possibly) prioritization processes. However, because other Participants are under no obligation to withhold publication of cases not covered by an active embargo, we recommend that a short embargo be used until the validation process concludes, at which point, it can be extended with a revision.

- Participants MAY use short embargo periods to cover their report validation process, and subsequently revise the embargo terms pending the outcome of their report validation and/or prioritization processes.

- Participants SHOULD remain flexible in adjusting embargo terms as the case evolves.

3.2.6 Default Embargoes

As described in §3.1.2.2, the EM process has the potential for unbounded *propose-reject* churn. To reduce the potential for this churn and increase the likelihood that *some* embargo is established rather than a stalemate of unaccepted proposals, we offer the following guidance.

Declaring Defaults. First, we note that all CVD Participants (including Reporters) are free to establish their own default embargo period in a published vulnerability disclosure policy. In particular, we recommend that CVD report recipients (typically Vendors and Coordinators) do so.

- Participants MAY include a default embargo period as part of a published Vulnerability Disclosure Policy.
- Recipients SHOULD post a default embargo period as part of their Vulnerability Disclosure Policy to set expectations with potential Reporters.

Using Defaults. Next, we work through the possible interactions of published policies with proposed embargoes. Each of the following scenarios assumes a starting state of $q^{em} \in N$, and a negotiation between two parties. We cover the extended situation (adding parties to an existing embargo) in §3.2.10. For now, we begin with the simplest case and proceed in an approximate order of ascending complexity.

In each of the following, subscripts on transitions indicate the Participant whose proposal is being acted upon, not the Participant who is performing the action. For example, a_{sender} indicates acceptance of the Sender's proposal, even if it is the Receiver doing the accepting.

- If neither Sender nor Receiver proposes an embargo, and no policy defaults apply, no embargo SHALL exist.

$$q^{em} \in N \tag{3.13}$$

- If the Sender proposes an embargo and the Receiver has no default embargo specified by policy, the Receiver SHOULD accept the Sender's proposal.

$$q^{em} \in N \xrightarrow{p_{sender}} P \xrightarrow{a_{sender}} A \tag{3.14}$$

- The Receiver MAY then propose a revision.

$$q^{em} \in A \xrightarrow{p_{receiver}} R \tag{3.15}$$

- A Receiver's default embargo specified in its vulnerability disclosure policy SHALL be treated as an initial embargo proposal.

$$q^{em} \in N \xrightarrow{p_{receiver}} P \tag{3.16}$$

- If the Receiver has declared a default embargo in its vulnerability disclosure policy and the Sender proposes nothing to the contrary, the Receiver's default embargo SHALL be considered as an accepted proposal.

$$q^{em} \in N \xrightarrow{p_{receiver}} P \xrightarrow{a_{receiver}} A \tag{3.17}$$

- If the Sender proposes an embargo *longer* than the Receiver's default embargo, the Receiver's default SHALL be taken as accepted and the Sender's proposal taken as a proposed revision.

$$q^{em} \in N \xrightarrow{p_{receiver}} P \xrightarrow{p_{sender}} P \xrightarrow{a_{receiver}} A \xrightarrow{r_{sender}} R \tag{3.18}$$

- The Receiver MAY then *accept* or *reject* the proposed extension.

$$q^{em} \in \begin{cases} R \xrightarrow{a_{sender}} A \\ R \xrightarrow{r_{sender}} A \end{cases} \tag{3.19}$$

- If the Sender proposes an embargo *shorter* than the Receiver's default embargo, the Sender's proposal SHALL be taken as accepted and the Receiver's default taken as a proposed revision.

$$q^{em} \in N \xrightarrow{p_{receiver}} P \xrightarrow{p_{sender}} P \xrightarrow{a_{sender}} A \xrightarrow{r_{receiver}} R \tag{3.20}$$

- The Sender MAY then *accept* or *reject* the proposed extension.

$$q^{em} \in \begin{cases} R \xrightarrow{a_{receiver}} A \\ R \xrightarrow{r_{receiver}} A \end{cases} \tag{3.21}$$

A Game Theory Argument for Accepting the Shortest Proposed Embargo. Readers may notice that we have taken a "shortest proposal wins" approach to the above guidance. This is intentional, and it results directly from the asymmetry mentioned in §3.2.5: The Receiver is faced with a choice to either *accept* the Reporter's proposal and attempt to extend it or to *reject* the proposal and end up with no embargo at all. Therefore, if we take the stance that for a vulnerability with no fix available, *any* embargo is better than *no* embargo, it should be obvious that it is in the Receiver's interest to *accept* even a short proposed embargo before immediately working to revise it.

The alternative is impractical because the Reporter is not obligated to provide the report to the Receiver at all. In the scenario where a Reporter *proposes* a short embargo and the Receiver *rejects* it because it is not long enough, the Reporter might choose to exit the negotiation entirely and publish whenever they choose without ever providing the report to the Receiver. That is not to say that we recommend this sort of behavior from Reporters. In fact, we specifically recommend the opposite in §3.2.5. Rather, it once more acknowledges the time-dependent informational asymmetry inherent to the CVD process.

A Logical Argument for Accepting the Shortest Proposed Embargo. Perhaps the above reasoning comes across as too Machiavellian for some readers. Here is a different perspective: Say a Reporter proposes an embargo of n days, while the Vendor would prefer m days. If n and m are given in units of days, we can look at them as a series of individual agreements, each of 1 day in length. We will represent each Participant as a vector representing that Participant's willingness to perpetuate the embargo on each day. Embargo willingness will be represented as a 1 if the Participant is willing to commit to keeping the embargo on that day, and a 0 if they are not. For simplicity's sake, we assume that each Participant is willing to maintain the embargo up to a certain point, and then their willingness goes away. In other words, each vector will be a series of zero or more 1s followed by zero or more 0s. For example, $[1, 1, 1, 1, 0, 0, 0]$ represents a Participant's willingness to engage in a 4-day embargo.

For our two Participants, let $\mathbf{x}$ and $\mathbf{y}$ be zero-indexed vectors of length $max(n, m)$.

$$|\mathbf{x}| = max(n, m) \tag{3.22}$$
$$|\mathbf{y}| = max(n, m) \tag{3.23}$$

The elements of each vector represent each respective Participant's willingness for the embargo to persist on each consecutive day.

$$\mathbf{x} = \left[x_i : x_i = \begin{cases} 1 & \text{if } i < n \\ 0 & \text{otherwise} \end{cases} \quad \text{for } 0 \leq i < max(n, m) \right] \tag{3.24}$$
$$\mathbf{y} = \left[y_i : y_i = \begin{cases} 1 & \text{if } i < m \\ 0 & \text{otherwise} \end{cases} \quad \text{for } 0 \leq i < max(n, m) \right] \tag{3.25}$$

Note that we have constructed these vectors so that each vector's scalar sum is just the length of embargo they prefer.

$$\Sigma(\mathbf{x}) = n \tag{3.26}$$
$$\Sigma(\mathbf{y}) = m \tag{3.27}$$

Now we can define an agreement vector $\mathbf{z}$ as the pairwise logical *AND* ($\wedge$) of elements from $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{z} = \left[z_i : z_i = x_i \wedge y_i \quad \text{for } 0 \leq i < max(n, m) \right] \tag{3.28}$$

For example, if one party prefers an embargo of length $n = 4$ days while another prefers one of length $m = 7$ days, we can apply (3.24), (3.25), and (3.28) as follows:

$$\begin{aligned} \mathbf{x} &= [1, 1, 1, 1, 0, 0, 0] \\ \wedge\ \mathbf{y} &= [1, 1, 1, 1, 1, 1, 1] \\ \hline \mathbf{z} &= [1, 1, 1, 1, 0, 0, 0] \end{aligned}$$

From this, we can see that the scalar sum of the agreement vector—and therefore the longest embargo acceptable to both parties—is simply the lesser of n and m:

$$\Sigma(\mathbf{z}) = min(n, m) \tag{3.29}$$

The Shortest Proposed Embargo Wins. In other words, if a Reporter proposes a 90-day embargo, but the Vendor prefers a 30-day embargo, we can think of this as a series of 1-day agreements in which both parties agree to the first 30 days of the embargo and disagree beyond that. By accepting the shorter 30-day embargo, the Reporter now has 30 days to continue negotiations with the Vendor to extend the embargo. Even if those continued negotiations fail, both parties get at least the 30-day embargo period they agreed on in the first place. This should be preferable to both parties versus the alternative of no embargo at all

were they to simply reject the shorter proposal. Typically it is the Reporter who desires a shorter embargo than the Vendor. We chose our example to demonstrate that this analysis works between any two parties, regardless of which party wants the shorter embargo.

On our way to making this principle explicit, we immediately came across a second scenario worth a brief diversion: What to do when multiple revisions are up for negotiation simultaneously? Based on the idea of extending the above to an efficient pairwise evaluation of multiple proposals, we suggest the following heuristic:

1. Sort the proposals in order from earliest to latest according to their expiration date.
2. Set the current candidate to the earliest proposal.
3. Loop over each remaining (later) proposal, evaluating it against the current candidate.
4. If the newly evaluated proposal is accepted, it becomes the current candidate and the loop repeats.
5. Otherwise, the loop exits at the first *reject*ed proposal.
6. The current candidate (i.e., the latest *accept*ed proposal) becomes the new *Active* embargo.
7. If the earliest proposed revision is rejected—implying that none of the later ones would be acceptable either—then the existing *Active* embargo remains intact.

Summarizing the principles just laid out as rules

- When two or more embargo proposals are open (i.e., none have yet been accepted) and $q^{em} \in P$, Participants SHOULD accept the shortest one and propose the remainder as revisions.
- When two or more embargo revisions are open (i.e., an embargo is active yet none of the proposals have been decided) and $q^{em} \in R$, Participants SHOULD *accept* or *reject* them individually, in earliest to latest expiration order.

3.2.7 Early Termination

Embargoes sometimes terminate prior to the agreed date and time. This is an unavoidable, if inconvenient, fact arising from three main causes:

1. Vulnerability discovery capability is widely distributed across the world, and not all Finders become cooperative Reporters.
2. Even among otherwise cooperative CVD Participants, leaks sometimes happen.
3. Adversaries are unconstrained by CVD in their vulnerability discovery, exploit code development, and use of exploit code in attacks.

While many leaks are unintentional and due to miscommunication or errors in a Participant's CVD process, the effect is the same regardless of the cause. As a result,

- Participants SHOULD be prepared with contingency plans in the event of early embargo termination.

Some reasons to terminate an embargo before the agreed date include the following:

- Embargoes SHALL terminate immediately when information about the vulnerability becomes public. Public information may include reports of the vulnerability or exploit code. ($q^{cs} \in \{\cdots P \cdot\cdot, \cdot\cdot\cdot\cdot X \cdot\}$)
- Embargoes SHOULD terminate early when there is evidence that the vulnerability is being actively exploited by adversaries. ($q^{cs} \in \{\cdot\cdot\cdot\cdot\cdot A\}$)

- Embargoes SHOULD terminate early when there is evidence that adversaries possess exploit code for the vulnerability.
- Embargoes MAY terminate early when there is evidence that adversaries are aware of the technical details of the vulnerability.

The above is not a complete list of acceptable reasons to terminate an embargo early. Note that the distinction between the *SHALL* in the first item and the *SHOULD* in the second is derived from the reasoning given in §4.3.1, where we describe the CS model's transition function. Embargo termination is the set of transitions described by (3.10).

Waiting for All Vendors to Reach Fix Ready May Be Impractical. It is not necessary for all Vendor Participants to reach $q^{cs} \in VF\cdots$ before publication or embargo termination. Especially in larger MPCVD cases, there comes a point where the net benefit of waiting for every Vendor to be ready is outweighed by the benefit of delivering a fix to the population that can deploy it. No solid formula for this exists, but factors to consider include the market share of the Vendors in $q^{cs} \in VF\cdots$ compared to those with $q^{cs} \in \cdot f\cdots$; the software supply chain for fix delivery to Deployers; the potential impact to critical infrastructure, public safety/health, or national security; etc.

- Embargoes MAY terminate early when a quorum of Vendor Participants is prepared to release fixes for the vulnerability ($q^{cs} \in VF\cdots$), even if some Vendors remain unprepared ($q^{cs} \in \cdot f\cdots$).
- Participants SHOULD consider the software supply chain for the vulnerability in question when determining an appropriate quorum for release.

3.2.8 Impact of Case Mergers on Embargoes

While relatively rare, it is sometimes necessary for two independent CVD cases to be merged into a single case. This can happen, for example, when two Finders independently discover vulnerabilities in separate products and report them to their respective (distinct) Vendors. On further investigation, it might be determined that both reported problems stem from a vulnerability in a library shared by both products. In this scenario, each Reporter-Vendor pair might have already negotiated an embargo for the case. Once the cases merge, the best option is usually to renegotiate a new embargo for the new case.

- A new embargo SHOULD be proposed when any two or more CVD cases are to be merged into a single case and multiple parties have agreed to different embargo terms prior to the case merger.
- If no new embargo has been proposed, or if agreement has not been reached, the earliest of the previously accepted embargo dates SHALL be adopted for the merged case.
- Participants MAY propose revisions to the embargo on a merged case as usual.

3.2.9 Impact of Case Splits on Embargoes

It is also possible that a single case needs to be split into multiple cases after an embargo has been agreed to. For example, consider a vulnerability that affects two widely disparate fix supply chains, such as a library used in both Software-as-a-Service (SAAS) and Operational Technology (OT) environments. The SAAS Vendors might be well positioned for a quick fix deployment while the OT Vendors might need considerably longer to work through the logistics of delivering deployable fixes to their customers. In such a case, the case Participants might choose to split the case into its respective supply chain cohorts to better coordinate within each group.

- When a case is split into two or more parts, any existing embargo SHOULD transfer to the new cases.
- If any of the new cases need to renegotiate the embargo inherited from the parent case, any new embargo SHOULD be later than the inherited embargo.
- In the event that an earlier embargo date is needed for a child case, consideration SHALL be given to the impact that ending the embargo on that case will have on the other child cases retaining a later embargo date. In particular, Participants in each child case should assess whether earlier publication of one child case might reveal the existence of or details about other child cases.
- Participants in a child case SHALL communicate any subsequently agreed changes from the inherited embargo to the Participants of the other child cases.

Note that it may not always be possible for the split cases to have different embargo dates without the earlier case revealing the existence of a vulnerability in the products allocated to the later case. For this reason, it is often preferable to avoid case splits entirely.

3.2.10 Inviting Others to an Embargoed Case

As anyone who has tried to schedule a meeting with multiple attendees can attest, multi-party scheduling can be difficult. When that schedule must also accommodate work completion schedules for an MPCVD case, it becomes even harder. In §3.2.6, we laid out a heuristic for resolving multiple embargo proposals, "The Shortest Embargo Proposed Wins." More specifically, we recommended that Participants *accept* the earliest proposed end date and immediately propose and evaluate the rest as potential revisions. This principle applies to any MPCVD case, even at its outset.

Embargo negotiations can become contentious in larger cases. Many MPCVD cases grow over time, and it is usually easier to establish an embargo with a smaller group than a larger one. Conflict resolution via consensus agreement is fine if it works. In fact, in scenarios where Participants who have already agreed to an embargo get to choose who else to add to the embargo, the existing consensus can be a strong influence for the new Participant to consent to the existing agreement.

In other words, it is usually preferable to present an already-accepted embargo to new Participants and get their agreement before potentially revising the embargo than it is to wait for a large multi-party negotiation to succeed before establishing an embargo in the first place. When consensus fails, however, it may be helpful for the group to appoint a case lead to resolve any conflicts. Such scenarios are often an opportunity for a third-party Coordinator to be engaged [16].

Therefore,

- Participants SHOULD attempt to establish an embargo as early in the process of handling the case as possible.
- Participants SHOULD follow consensus agreement to decide embargo terms.
- When consensus fails to reach agreement on embargo terms, Participants MAY appoint a case lead to resolve conflicts.
- Participants MAY engage a third-party Coordinator to act as a neutral third-party case lead to resolve conflicts between Participants during the course of handling a case.

3.2.10.1 Who to Invite

The Finder/Reporter is, by definition, a Participant in any CVD case by virtue of their knowledge of the vulnerability in the first place. Additional Participants usually fall into one of three categories:

- All known Vendors of affected software SHOULD be included as Participants.
- Third-party Coordinators MAY be included as Participants.
- Other parties MAY be included as Participants when necessary and appropriate. Examples we have observed in past cases include Deployers, subject matter experts, and government agencies with relevant regulatory oversight or critical infrastructure protection responsibilities.

3.2.10.2 Adding Participants to an Existing Embargo

Adding new Participants to a case with an existing embargo might require the new Participant to accept the embargo prior to receiving the report.

- When inviting a new Participant to a case with an existing embargo, the inviting Participant SHALL propose the existing embargo to the invited Participant.
- A newly invited Participant to a case with an existing embargo SHOULD accept the existing embargo.
- The inviting Participant SHOULD NOT share the vulnerability report with the newly invited Participant unless the new Participant has accepted the existing embargo.
- The inviting Participant MAY interpret the potential Participant's default embargo contained in their published vulnerability disclosure policy in accordance with the default acceptance strategies listed in §3.2.6.
- A newly invited Participant to a case with an existing embargo MAY propose a revision after accepting the existing embargo.

3.2.10.3 When to Invite Participants

In MPCVD there are practical considerations to be made regarding the timing of *when* to notify individual Participants. The primary factor in these decisions stems from the interaction of the *Active* embargo with the potential Participant's existing (explicit or implicit) disclosure policy.

Participants with Disclosure Policies Shorter Than an Existing Embargo. Adding a potential Participant with a known default disclosure policy *shorter* than an extant embargo leaves Participants with these options to choose from:

1. Shorten the existing embargo to match the potential Participant's policy.
2. Propose the existing embargo to the potential Participant, and, upon acceptance, share the report with them.
3. Delay notifying the potential Participant until their default policy aligns with the existing embargo.
4. Avoid including the potential Participant in the embargo entirely.

For example, say a Vendor has a seven-day maximum public disclosure policy. Participants in a case with an existing embargo ending in three weeks might choose to notify that Vendor two weeks from now to ensure that even the default disclosure timeline remains compatible with the extant embargo.

- Participants with short default embargo policies SHOULD consider accepting longer embargoes in MPCVD cases.
- Participants in an MPCVD case MAY delay notifying potential Participants with short default embargo policies until their policy aligns with the agreed embargo.

Participants with Disclosure Policies Longer Than an Existing Embargo. Similarly, adding a Participant with a known default disclosure policy *longer* than an extant embargo leaves Participants with the following options to choose from:

1. Lengthen the existing embargo to match the potential Participant's policy.
2. Propose the existing embargo to the potential Participant, and, upon acceptance, share the report with them.
3. Avoid including the potential Participant in the embargo entirely.

In the case of a Vendor with a *longer* default policy than the existing embargo, it is still preferable to give them as much lead time as possible *even* if it is not possible to extend the embargo to their preferred timing.

- In the interest of receiving the report in the first place, potential Participants with a longer default policy than an existing case SHOULD accept the embargo terms offered.
- After accepting an existing embargo, newly invited Participants with a longer default policy than an existing case MAY propose a revision to the existing embargo, if desired, to accommodate their preferences.
- Existing Participants MAY *accept* or *reject* such a proposed revision as they see fit.
- Participants in a case with an existing embargo SHOULD notify Vendors with a longer default embargo policy.
- Participants in a case with an existing embargo MAY choose to extend the embargo to accommodate a newly added Participant.

Untrustworthy Participants. Unfortunately, not all potential CVD Participants are equally trustworthy with vulnerability information. For example, a Vendor might have sub-par operational security or even business practices that result in adversaries often finding out about vulnerabilities in their products before the end of an embargo period. Participants might also be subject to regulatory regimes in which they are required by law to share known vulnerabilities with government agencies having oversight responsibilities.

- Participants that are known to leak or provide vulnerability information to adversaries either as a matter of policy or historical fact SHOULD be treated similar to Participants with brief disclosure policies.

Acknowledging that *adversary* is not a universally agreed-upon category, the definition of *adversary* in the above is left to individual Participants.

The maximal interpretation of the above is that untrustworthy Participants are left to be notified by the publication of the vulnerability report. This is the equivalent of treating them like a Participant with a default zero-day maximum embargo policy.

Coordinators. Third-party Coordinators, as Participants who are neither Finders nor Vendors, often play an important role in MPCVD cases, especially those with broad impact across the software supply chain or with acute critical infrastructure or public safety impacts. Most Coordinators strive to be consistent in their MPCVD practices and have well-documented disclosure policies along with significant histories of handling previous cases. All of these factors

make the argument for including third-party Coordinators in CVD cases of sufficient complexity, impact, or importance.

Other Parties. Some Participants in CVD have their own policies that prohibit notification of any parties unable to directly contribute to the development of a fix for a particular vulnerability. Typically, these policies take the form of "only Vendors of affected products" or similar such restrictions.

The CERT/CC's position as a third-party Coordinator in numerous cases is that this approach can be appropriate for straightforward scenarios, such as those in which a Vendor is in direct contact with their downstream Vendors and can coordinate the response within that community. However, it falls short in some cases, such the following:

- Vulnerabilities are found to affect a broad spectrum of Vendors and products, especially when cases cross industry sectors or otherwise include Participants having widely divergent operational tempos or software delivery models.
- Vulnerabilities affect systems deployed in high-impact niches, such as critical infrastructure, public safety, and national security.
- Outside expertise is needed to understand the implications or impact of a vulnerability beyond the participating Vendors; sometimes the most knowledgeable parties work for someone else.

3.2.11 Consequences of Non-Compliance

Considering multiple cases over time, MPCVD can be thought of as an iterated game analogous to the Prisoner's Dilemma. One notable strategy for the Prisoner's Dilemma is *tit for tat* in which non-cooperation from one party in one round can be met with non-cooperation from the opposite party in the next. While MPCVD is usually much bigger than a toy two-player game, we feel it is necessary to encode the possibility that non-cooperation will have downstream consequences.

- Participants MAY decline to participate in future CVD cases involving parties with a history of violating previous embargoes.

4 CVD Case State Model

In this chapter, we revisit the CVD Case State (CS) model from the Householder and Spring 2021 report [10]. A complete derivation of the CS model can be found in that report. Here, we are primarily interested in the final model, which comprises 32 states and their transitions.

As in the previous two chapters, we wish to define a DFA 5-tuple $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ [20], this time for the CS model. However, due to the size of the final CS model, we begin with some necessary background on the substates of the model in §4.1 prior to defining the Case States in §4.2.

4.1 CVD Case Substates

In our model, the state of the world is a specification of the current status of all the events in the vulnerability lifecycle model described in the Householder and Spring 2021 report [10]. We describe the relevant factors as substates below. For notational purposes, each substate status is represented by a letter for that part of the state of the world. For example, v means no Vendor awareness and V means the Vendor is aware. The complete set of status labels is shown in Table 4.1.

4.1.1 The $Vendor\ Awareness$ Substate (v, V)

The *Vendor Awareness* substate corresponds to *Disclosure* in the Arbaugh, Fithen, and McHugh article, "Windows of Vulnerability: A Case Study analysis" [1] and *vulnerability discovered by Vendor* in Bilge and Dumitraş's article, "Before we knew it: an empirical study of zero-day attacks in the real world" [3]. In the interest of model simplicity, we are not concerned with *how* the Vendor finds out about the vulnerability's existence—whether it was found via internal testing, reported within a CVD process, or noticed as the result of incident or malware analysis.

4.1.2 The $Fix\ Readiness$ Substate (f, F)

The *Fix Readiness* substate refers to the Vendor's creation and possession of a fix that *could* be deployed to a vulnerable system *if* the system owner knew of its existence. Here we differ somewhat from previous models [1, 9, 3]; their models address the *release* of the fix rather than its *readiness* for release. This distinction is necessary because we are interested in modeling the activities and states leading up to disclosure. Fix *release* is a goal of the CVD process, whereas fix *readiness* is a significant process milestone along the way.

Table 4.1: CVD Case Status Labels

Status	Meaning	Status	Meaning
v	Vendor is not aware of vulnerability.	V	Vendor is aware of vulnerability.
f	Fix is not ready.	F	Fix is ready.
d	Fix is not deployed.	D	Fix is deployed.
p	Public is not aware of vulnerability.	P	Public is aware of vulnerability.
x	No exploit has been made public.	X	Exploit has been made public.
a	No attacks have been observed.	A	Attacks have been observed.

4.1.3 The *Fix Deployed* Substate (d, D)

The *Fix Deployed* substate reflects the deployment status of an existing fix. The model in the Householder and Spring 2021 report [10] was initially designed to treat this substate as a singular binary state for a case, but we intend to relax that here to reflect a more realistic perspective in which each Deployer maintains their own instance of this state value. It remains a binary state for each Deployer, which, however, is still a simplification.

4.1.4 The *Public Awareness* Substate (p, P)

The *Public Awareness* substate corresponds to *Publication* in the Arbaugh, Fithen, and McHugh article [1], *time of public disclosure* in Frei et al.'s article Modeling the Security Ecosystem—The Dynamics of (In)Security [9]; and *vulnerability disclosed publicly* in Bilge and Dumitraş's article [3]. The public might find out about a vulnerability through the Vendor's announcement of a fix, a news report about a security breach, a conference presentation by a researcher, or a variety of other means. As above, we are primarily concerned with the occurrence of the event itself rather than the details of *how* the public awareness event arises.

4.1.5 The *Exploit Public* Substate (x, X)

The *Exploit Public* substate reflects whether the method of exploiting a vulnerability has been made public in sufficient detail to be reproduced by others. Posting Proof of Concept (PoC) code to a widely available site or including the exploit code in a commonly available exploit tool meets this criteria; privately held exploits do not.

4.1.6 The *Attacks Observed* Substate (a, A)

The *Attacks Observed* substate reflects whether attacks have been observed in which the vulnerability was exploited. This substate requires evidence that the vulnerability was exploited; we can then presume the existence of exploit code regardless of its availability to the public. Analysis of malware from an incident might meet *Attacks Observed* but not *Exploit Public*, depending on how closely the attacker holds the malware. Use of a public exploit in an attack meets both *Exploit Public* and *Attacks Observed*.

4.1.7 CS Model Design Choices

We chose to include the *Fix Ready*, *Fix Deployed*, and *Public Awareness* events so that our model could better accommodate two common modes of modern software deployment:

- *shrinkwrap* is a traditional distribution mode where the Vendor and Deployer are distinct entities, and Deployers must be made aware of the fix before it can be deployed. In this case, both *Fix Ready* and *Public Awareness* are necessary for *Fix Deployment* to occur.
- *SAAS* is a more recent delivery mode where the Vendor also plays the role of Deployer. In this distribution mode, *Fix Ready* can lead directly to *Fix Deployed* with no dependency on *Public Awareness*.

We note that so-called *silent fixes* by Vendors can sometimes result in a fix being deployed without public awareness even if the Vendor is not the Deployer. Thus, it is possible (but unlikely) for *Fix Deployed* to occur before *Public Awareness* even in the shrinkwrap mode above. It is also possible, and somewhat more likely, for *Public Awareness* to occur before *Fix Deployed* in the SAAS mode as well.

4.2 CVD Case States

In the CS model, a state q^{cs} represents the status of each of the six substates. State labels inherit the substate notation from above: lowercase letters designate events that have not occurred, and uppercase letters designate events that have occurred in a particular state. For example, the state $VFdpXa$ represents Vendor is aware, fix is ready, fix not deployed, no public awareness, exploit is public, and no attacks. The order in which the events occurred does not matter when defining the state. However, we will observe a notation convention keeping the letter names in the same case-insensitive order (v, f, d, p, x, a).

CS states can be any combination of statuses, provided that a number of caveats elaborated in §4.3 are met. One such caveat worth noting here is that valid states must follow what we call the *Vendor fix path*.[3] The reason is causal: For a fix to be deployed (D), it must have been ready (F) for deployment. And for it to be ready, the Vendor must have already known (V) about the vulnerability—symbolically, $D \implies F \implies V$. As a result, valid states must begin with one of the following strings: vfd, Vfd, VFd, or VFD.

The CS model is thus composed of 32 possible states, which we define as $\mathcal{Q}^{cs}$ in (4.1).

$$\mathcal{Q}^{cs} = \begin{Bmatrix} vfdpxa, & vfdPxa, & vfdpXa, & vfdPXa, \\ vfdpxA, & vfdPxA, & vfdpXA, & vfdPXA, \\ Vfdpxa, & VfdPxa, & VfdpXa, & VfdPXa, \\ VfdpxA, & VfdPxA, & VfdpXA, & VfdPXA, \\ VFdpxa, & VFdPxa, & VFdpXa, & VFdPXa, \\ VFdpxA, & VFdPxA, & VFdpXA, & VFdPXA, \\ VFDpxa, & VFDPxa, & VFDpXa, & VFDPXa, \\ VFDpxA, & VFDPxA, & VFDpXA, & VFDPXA \end{Bmatrix} \tag{4.1}$$

4.2.1 CS Start and End States

All vulnerabilities start in the base state q_0^{cs} in which no events have occurred.

$$q_0^{cs} = vfdpxa \tag{4.2}$$

The lone final state in which all events have occurred is $VFDPXA$.

$$\mathcal{F}^{cs} = \{VFDPXA\} \tag{4.3}$$

Note that this is a place where our model of the vulnerability lifecycle diverges from what we expect to observe in CVD cases in the real world. There is ample evidence that most vulnerabilities never have exploits published or attacks observed [11, 19]. Therefore, practically speaking, we might expect vulnerabilities to wind up in one of

$$\mathcal{F}' = \{VFDPxa, VFDPxA, VFDPXa, VFDPXA\}$$

at the time a report is closed (i.e., when $q^{rm} \xrightarrow{c} C$). In fact, most count a CVD as successful when reports are closed in $q^{cs} \in VFDPxa$ because it means that the defenders won the race against adversaries. The distinction between the RM and CS processes is important; Participants can close cases whenever their RM process dictates, independent of the CS state. In other words, it remains possible for exploits to be published or attacks to be observed long after the RM process has closed a case.

[3] See §2.4 of the Householder and Spring 2021 report [10] for an expanded explanation of the *Vendor fix path*.

4.2.2 CS Model Wildcard Notation

We frequently need to refer to subsets of $\mathcal{Q}^{cs}$. To do so, we will use a dot ($\cdot$) to represent a single character wildcard. For example, $VFdP\cdot\cdot$ refers to the subset of $\mathcal{Q}^{cs}$ in which the Vendor is aware, a fix is ready but not yet deployed, and the public is aware of the vulnerability, yet we are indifferent to whether exploit code has been made public or attacks have been observed. Specifically,

$$VFdP\cdot\cdot = \{VFdPxa, VFdPxA, VFdPXa, VFdPXA\} \subset \mathcal{Q}^{cs}$$

4.3 CS Transitions

In this section, we elaborate on the input symbols and transition functions for our CS DFA. A row-wise reading of Table 4.1 implies a set of events corresponding to each specific substate change, which we correspond to the symbols in the CS DFA.

- **V** – A Vendor becomes aware of a vulnerability $vfd\cdots \rightarrow Vfd\cdots$
- **F** – A Vendor readies a fix for a vulnerability $Vfd\cdots \rightarrow VFd\cdots$
- **D** – A Deployer deploys a fix for a vulnerability $VFd\cdots \rightarrow VFD\cdots$
- **P** – Information about a vulnerability becomes known to the public $\cdots p\cdot\cdot \rightarrow \cdots P\cdot\cdot$
- **X** – An exploit for a vulnerability is made public $\cdots\cdot x\cdot \rightarrow \cdots\cdot X\cdot$
- **A** – Attacks exploiting a vulnerability are observed $\cdots\cdot\cdot a \rightarrow \cdots\cdot\cdot A$

We define the set of symbols for our CS DFA as Σ^{cs}:

$$\Sigma^{cs} = \{\mathbf{V}, \mathbf{F}, \mathbf{D}, \mathbf{P}, \mathbf{X}, \mathbf{A}\} \tag{4.4}$$

Here we diverge somewhat from the notation used for the RM and EM models described in previous chapters, which use lowercase letters for transitions and uppercase letters for states. Because CS state names already use both lowercase and uppercase letters, here we use a bold font for the symbols of the CS DFA to differentiate the transition from the corresponding substate it leads to: e.g., $vfd\cdots \xrightarrow{\mathbf{V}} Vfd\cdots$.

For the CS model, an input symbol $\sigma^{cs} \in \Sigma^{cs}$ is "read" when a Participant observes a change in status (a Vendor is notified and exploit code has been published, etc.). For the sake of simplicity, we begin with the assumption that observations are globally known—that is, a status change observed by any CVD Participant is known to all. In the real world, we believe the MPCVD protocol proposed in §6 is poised to ensure eventual consistency with this assumption through the communication of perceived case state across coordinating parties.

4.3.1 CS Transitions Defined

Here we define the allowable transitions between states in the CS model. A diagram of the CS process, including its states and transitions, is shown in Figure 4.1.

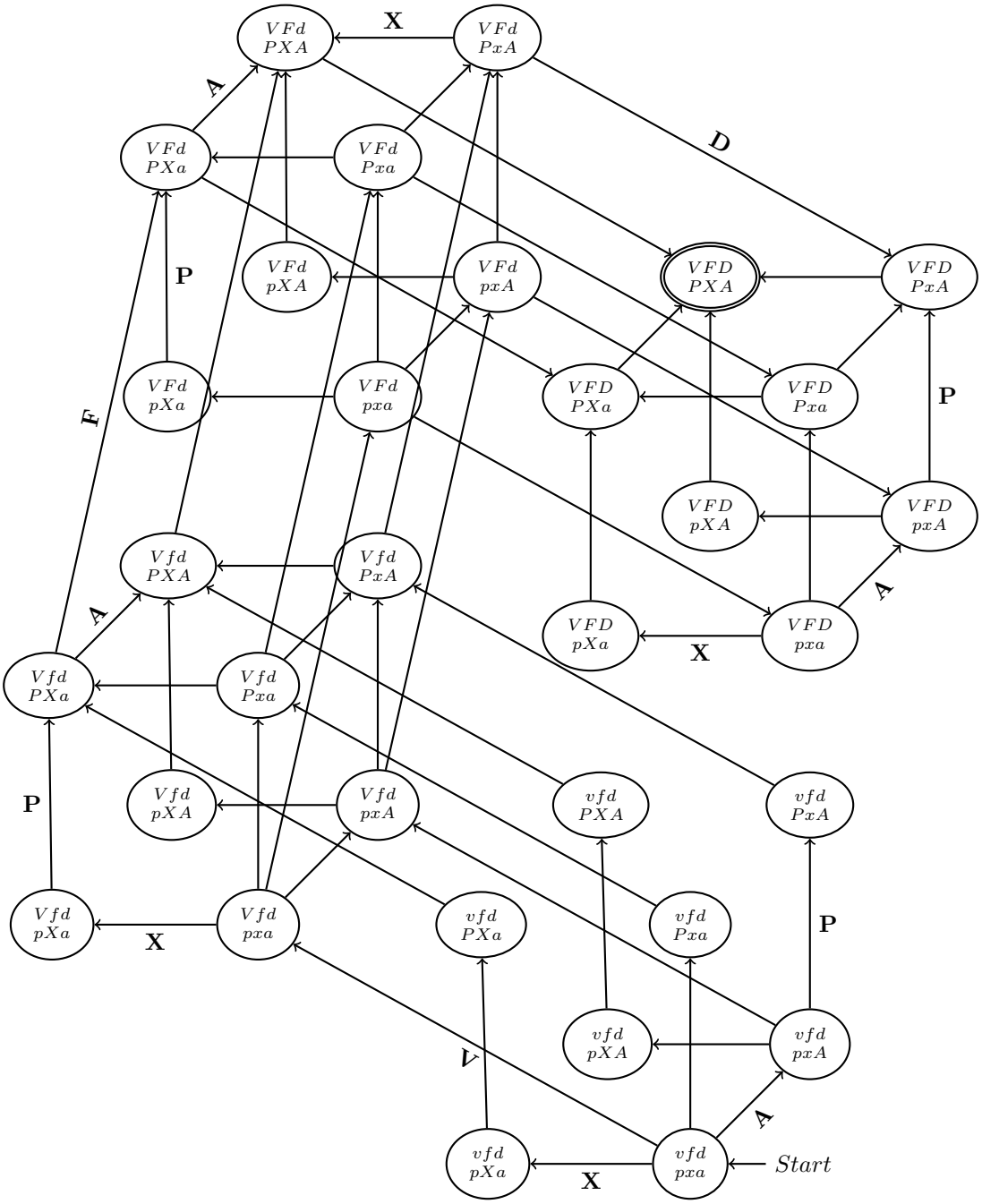


*Figure 4.1: Complete Map of the 32 Possible States in the CS Model of Vulnerability Disclosure and Their Allowed Transitions (***V***, **F**, **D**, **P**, **X**, **A**)*

Transitions in the CS model follow a few rules described in detail in §2.4 of the Householder and Spring 2021 report [10], which we summarize here:

- Because states correspond to the status of events that have or have not occurred, and all state transitions are irreversible (i.e., we assume history is immutable), the result will be an acyclic directed graph of states beginning at $q_0^{cs} = vfdpxa$ and ending at $\mathcal{F}^{cs} = \{VFDPXA\}$ with allowed transitions as the edges. In practical terms for the CS model, this means there is an arrow of time from $vfdpxa$ through $VFDPXA$ in which each individual state transition changes exactly one letter from lowercase to uppercase.
- The *Vendor fix path* ($vfd\cdots \xrightarrow{\mathbf{V}} Vfd\cdots \xrightarrow{\mathbf{F}} VFd\cdots \xrightarrow{\mathbf{D}} VFD\cdots$) is a causal requirement as outlined in §4.2.
- Vendors are presumed to know at least as much as the public does; therefore, $v\cdot\cdot P\cdot\cdot$ can only lead to $V\cdot\cdot P\cdot\cdot$.
- Exploit publication is tantamount to public awareness; therefore, $\cdots pX\cdot$ can only lead to $\cdots\cdot PX\cdot$.

In this model, attacks observed when a vulnerability is unknown to the public ($\cdots p\cdot A$) need not immediately cause public awareness ($\cdots P\cdot A$), although, obviously, that can and does happen. Our reasoning for allowing states in $\cdots p\cdot A$ to persist is twofold:

- First, the connection between attacks and exploited vulnerabilities is often made later during incident analysis. While the attack itself may have been observed much earlier, the knowledge of *which* vulnerability it targeted may be delayed until after other events have occurred.
- Second, attackers are not a monolithic group. An attack from a niche set of threat actors does not automatically mean that the knowledge and capability of exploiting a particular vulnerability is widely available to all possible adversaries. Publication, in that case, might assist other adversaries more than it helps defenders.

In other words, although $\cdots p\cdot A$ does not require an immediate transition to $\cdots P\cdot A$ the way $\cdots pX\cdot \xrightarrow{\mathbf{P}} \cdots PX\cdot$ does, it does seem plausible that the likelihood of **P** occurring increases when attacks are occurring. Logically, this is a result of there being more ways for the public to discover the vulnerability when attacks are happening than when they are not. For states in $\cdots p\cdot a$, the public depends on the normal vulnerability discovery and reporting process. States in $\cdots p\cdot A$ include that possibility and add the potential for discovery as a result of security incident analysis. Hence,

- Once attacks have been observed, fix development SHOULD accelerate, the embargo teardown process SHOULD begin, and publication and deployment SHOULD follow as soon as is practical.

4.3.2 A Regular Grammar for the CS model

Following the complete state machine diagram in Figure 4.1, we can summarize the transition functions of the CS model as a right-linear grammar δ^{cs}:

$$\delta^{cs} = \begin{cases} vfdpxa & \to \mathbf{V}\ Vfdpxa \mid \mathbf{P}\ vfdPxa \mid \mathbf{X}\ vfdpXa \mid \mathbf{A}\ vfdpxA \\ vfdpxA & \to \mathbf{V}\ VfdpxA \mid \mathbf{P}\ vfdPxA \mid \mathbf{X}\ vfdpXA \\ vfdpXa & \to \mathbf{P}\ vfdPXa \\ vfdpXA & \to \mathbf{P}\ vfdPXA \\ vfdPxa & \to \mathbf{V}\ VfdPxa \\ vfdPxA & \to \mathbf{V}\ VfdPxA \\ vfdPXa & \to \mathbf{V}\ VfdPXa \\ vfdPXA & \to \mathbf{V}\ VfdPXA \\ Vfdpxa & \to \mathbf{F}\ VFdpxa \mid \mathbf{P}\ VfdPxa \mid \mathbf{X}\ VfdpXa \mid \mathbf{A}\ VfdpxA \\ VfdpxA & \to \mathbf{F}\ VFdpxA \mid \mathbf{P}\ VfdPxA \mid \mathbf{X}\ VfdpXA \\ VfdpXa & \to \mathbf{P}\ VfdPXa \\ VfdpXA & \to \mathbf{P}\ VfdPXA \\ VfdPxa & \to \mathbf{F}\ VFdPxa \mid \mathbf{X}\ VfdPXa \mid \mathbf{A}\ VfdPxA \\ VfdPxA & \to \mathbf{F}\ VFdPxA \mid \mathbf{X}\ VfdPXA \\ VfdPXa & \to \mathbf{F}\ VFdPXa \mid \mathbf{A}\ VfdPXA \\ VfdPXA & \to \mathbf{F}\ VFdPXA \\ VFdpxa & \to \mathbf{D}\ VFDpxa \mid \mathbf{P}\ VFdPxa \mid \mathbf{X}\ VFdpXa \mid \mathbf{A}\ VFdpxA \\ VFdpxA & \to \mathbf{D}\ VFDpxA \mid \mathbf{P}\ VFdPxA \mid \mathbf{X}\ VFdpXA \\ VFdpXa & \to \mathbf{P}\ VFdPXa \\ VFdpXA & \to \mathbf{P}\ VFdPXA \\ VFdPxa & \to \mathbf{D}\ VFDPxa \mid \mathbf{X}\ VFdPXa \mid \mathbf{A}\ VFdPxA \\ VFdPxA & \to \mathbf{D}\ VFDPxA \mid \mathbf{X}\ VFDPXA \\ VFdPXa & \to \mathbf{D}\ VFDPXa \mid \mathbf{A}\ VFdPXA \\ VFdPXA & \to \mathbf{D}\ VFDPXA \\ VFDpxa & \to \mathbf{P}\ VFDPxa \mid \mathbf{X}\ VFDpXa \mid \mathbf{A}\ VFDpxA \\ VFDpxA & \to \mathbf{P}\ VFDPxA \mid \mathbf{X}\ VFDpXA \\ VFDpXa & \to \mathbf{P}\ VFDPXa \\ VFDpXA & \to \mathbf{P}\ VFDPXA \\ VFDPxa & \to \mathbf{X}\ VFDPXa \mid \mathbf{A}\ VFDPxA \\ VFDPxA & \to \mathbf{X}\ VFDPXA \\ VFDPXa & \to \mathbf{A}\ VFDPXA \\ VFDPXA & \to \varepsilon \end{cases} \tag{4.5}$$

A more thorough examination of the strings generated by this grammar, their interpretation as the possible histories of all CVD cases, and implications for measuring the efficacy of the overall CVD process writ large can be found in the Householder and Spring 2021 report [10].

4.4 CS Model Fully Defined

In combination, the full definition of the CS DFA $(\mathcal{Q}, q_0, \mathcal{F}, \Sigma, \delta)^{cs}$ is given by equations (4.1), (4.2), (4.3), (4.4), and (4.5). For convenience, we have assembled them into (4.6).

$$
\begin{aligned}
CS = \Bigg(\quad & \\
\mathcal{Q}^{cs} =& \left\{ \begin{matrix} vfdpxa, & vfdPxa, & vfdpXa, & vfdPXa, \\ vfdpxA, & vfdPxA, & vfdpXA, & vfdPXA, \\ Vfdpxa, & VfdPxa, & VfdpXa, & VfdPXa, \\ VfdpxA, & VfdPxA, & VfdpXA, & VfdPXA, \\ VFdpxa, & VFdPxa, & VFdpXa, & VFdPXa, \\ VFdpxA, & VFdPxA, & VFdpXA, & VFdPXA, \\ VFDpxa, & VFDPxa, & VFDpXa, & VFDPXa, \\ VFDpxA, & VFDPxA, & VFDpXA, & VFDPXA \end{matrix} \right\}, && (4.1) \\
q_0^{cs} =& vfdpxa, && (4.2) \\
\mathcal{F}^{cs} =& \{VFDPXA\}, && (4.3) \\
\Sigma^{cs} =& \{\mathbf{V}, \mathbf{F}, \mathbf{D}, \mathbf{P}, \mathbf{X}, \mathbf{A}\}, && (4.4) \\
\delta^{cs} =& \begin{cases} vfdpxa & \to \mathbf{V}\ Vfdpxa \mid \mathbf{P}\ vfdPxa \mid \mathbf{X}\ vfdpXa \mid \mathbf{A}\ vfdpxA \\ vfdpxA & \to \mathbf{V}\ Vfdpxa \mid \mathbf{P}\ vfdPxA \mid \mathbf{X}\ vfdpXA \\ vfdpXa & \to \mathbf{P}\ vfdPXa \\ vfdpXA & \to \mathbf{P}\ vfdPXA \\ vfdPxa & \to \mathbf{V}\ VfdPxa \\ vfdPxA & \to \mathbf{V}\ VfdPxA \\ vfdPXa & \to \mathbf{V}\ VfdPXa \\ vfdPXA & \to \mathbf{V}\ VfdPXA \\ Vfdpxa & \to \mathbf{F}\ VFdpxa \mid \mathbf{P}\ VfdPxa \mid \mathbf{X}\ VfdpXa \mid \mathbf{A}\ VfdpxA \\ Vfdpxa & \to \mathbf{F}\ VFdpxA \mid \mathbf{P}\ VfdPxA \mid \mathbf{X}\ VfdpXA \\ VfdpXa & \to \mathbf{P}\ VfdPXa \\ VfdpXA & \to \mathbf{P}\ VfdPXA \\ VfdPxa & \to \mathbf{F}\ VFdPxa \mid \mathbf{X}\ VfdPXa \mid \mathbf{A}\ VfdPxA \\ VfdPxA & \to \mathbf{F}\ VFdPxA \mid \mathbf{X}\ VfdPXA \\ VfdPXa & \to \mathbf{F}\ VFdPXa \mid \mathbf{A}\ VfdPXA \\ VfdPXA & \to \mathbf{F}\ VFdPXA \\ VFdpxa & \to \mathbf{D}\ VFDpxa \mid \mathbf{P}\ VFdPxa \mid \mathbf{X}\ VFdpXa \mid \mathbf{A}\ VFdpxA \\ VFdpxA & \to \mathbf{D}\ VFDpxA \mid \mathbf{P}\ VFdPxA \mid \mathbf{X}\ VFdpXA \\ VFdpXa & \to \mathbf{P}\ VFdPXa \\ VFdpXA & \to \mathbf{P}\ VFdPXA \\ VFdPxa & \to \mathbf{D}\ VFDPxa \mid \mathbf{X}\ VFdPXa \mid \mathbf{A}\ VFdPxA \\ VFdPxA & \to \mathbf{D}\ VFDPxA \mid \mathbf{X}\ VFDPXA \\ VFdPXa & \to \mathbf{D}\ VFDPXa \mid \mathbf{A}\ VFdPXA \\ VFdPXA & \to \mathbf{D}\ VFDPXA \\ VFDpxa & \to \mathbf{P}\ VFDPxa \mid \mathbf{X}\ VFDpXa \mid \mathbf{A}\ VFDpxA \\ VFDpxA & \to \mathbf{P}\ VFDPxA \mid \mathbf{X}\ VFDpXA \\ VFDpXa & \to \mathbf{P}\ VFDPXa \\ VFDpXA & \to \mathbf{P}\ VFDPXA \\ VFDPxa & \to \mathbf{X}\ VFDPXa \mid \mathbf{A}\ VFDPxA \\ VFDPxA & \to \mathbf{X}\ VFDPXA \\ VFDPXa & \to \mathbf{A}\ VFDPXA \\ VFDPXA & \to \varepsilon \end{cases} && (4.5) \\
\Bigg) & && (4.6)
\end{aligned}
$$

5 Model Interactions

In this chapter, we reflect on the interactions between the Report Management (RM), Embargo Management (EM), and CVD Case State (CS) models within the overall MPCVD process.

5.1 Interactions Between the RM and EM Models

There are additional constraints on how the RM and EM processes interact.

Start Embargo Negotiations As Early as Possible

- The EM process MAY begin (i.e., the initial *propose* transition $q^{em} \in N \xrightarrow{p} P$) prior to the report being sent to a potential Participant ($q^{rm} \in S$), for example, when a Participant wishes to ensure acceptable embargo terms prior to sharing a report with a potential recipient.
- If it has not already begun, the EM process SHOULD begin when a recipient is in RM *Received* ($q^{rm} \in R$) whenever possible.

Negotiate Embargoes for Active Reports

- Embargo Management MAY begin in any of the active RM states ($q^{rm} \in \{R, V, A\}$).
- Embargo Management SHOULD NOT begin in an inactive RM state ($q^{rm} \in \{I, D, C\}$).

Negotiate Embargoes Through Validation and Prioritization

- Embargo Management MAY run in parallel to validation ($q^{rm} \in \{R, I\} \xrightarrow{\{v,i\}} \{V, I\}$) and prioritization ($q^{rm} \in V \xrightarrow{\{a,d\}} \{A, D\}$) activities.

Renegotiate Embargoes While Reports Are Valid Yet Unclosed

- EM revision proposals ($q^{em} \in A \xrightarrow{p} R$) and acceptance or rejection of those proposals ($q^{em} \in R \xrightarrow{\{a,r\}} A$) MAY occur during any of the valid yet unclosed RM states ($q_{rm} \in \{V, A, D\}$).

Avoid Embargoes for Invalid Reports...

- Embargo Management SHOULD NOT begin with a proposal from a Participant in RM *Invalid* ($q^{rm} \in I$).

...but Don't Lose Momentum if Validation Is Pending

- Outstanding embargo negotiations ($q^{em} \in P \xrightarrow{\{r,p\}} \{N, P\}$) MAY continue in RM *Invalid* ($q^{rm} \in I$) (e.g., if it is anticipated that additional information may be forthcoming to promote the report from *Invalid* to *Valid*) ($q^{rm} \in I \xrightarrow{v} V$).

Only Accept Embargoes for Possibly Valid Yet Unclosed Reports

- Embargo Management MAY proceed from EM *Proposed* to EM *Accepted* ($q^{em} \in P \xrightarrow{a} A$) when RM is neither *Invalid* nor *Closed* ($q^{rm} \in \{R, V, A, D\}$).
- Embargo Management SHOULD NOT proceed from EM *Proposed* to EM *Accepted* when RM is *Invalid* or *Closed* ($q^{rm} \in \{I, C\}$).
- Embargo Management MAY proceed from EM *Proposed* to EM *None* ($q^{em} \in P \xrightarrow{r} N$) when RM is *Invalid* or *Closed*.

Report Closure, Deferral, and Active Embargoes

- Participants SHOULD NOT close reports ($q^{rm} \in \{I, D, A\} \xrightarrow{c} C$) while an embargo is active ($q^{em} \in \{A, R\}$).
- Instead, reports with no further tasks SHOULD be held in either $Deferred$ or *Invalid* ($q^{rm} \in \{D, I\}$) (depending on the report validity status) until the embargo has terminated ($q^{em} \in X$). This allows Participants to stop work on a report but still maintain their participation in an extant embargo.
- Notwithstanding, Participants who choose to close a report ($q^{rm} \in \{I, D, A\} \xrightarrow{c} C$) while an embargo remains in force ($q^{em} \in \{A, R\}$) SHOULD communicate their intent to either continue to adhere to the embargo or terminate their compliance with it.
- Report closure or deferral does not terminate an embargo. A Participant's closure or deferral ($q^{rm} \in \{C, D\}$) of a report while an embargo remains active ($q^{em} \in \{A, R\}$) and while other Participants remain engaged ($q^{rm} \in \{R, V, A\}$) SHALL NOT automatically terminate the embargo.
- Any changes to a Participant's intention to adhere to an active embargo SHOULD be communicated clearly in addition to any necessary notifications regarding RM or EM state changes.

5.2 RM - CVD and EM - CVD Model Interactions

The RM and EM models interact with the CS model described in Chapter 4. Here we will review the constraints arising from the interaction of the RM and EM models with each of the CS transition events represented by its symbols. As a reminder, the CS transition symbols (Σ^{cs}) from the Householder and Spring 2021 report [10] are represented as bold capital letters.

$$\Sigma^{cs} = \{\mathbf{V},\ \mathbf{F},\ \mathbf{D},\ \mathbf{P},\ \mathbf{X},\ \mathbf{A}\} \qquad \text{(4.4 revisited)}$$

Global vs. Participant-Specific Aspects of the CS Model. The CS model encompasses both Participant-specific and global aspects of a CVD case. In particular, the Vendor fix path substates—Vendor unaware (vfd), Vendor aware (Vfd), fix ready (VFd), and fix deployed (VFD)—are specific to each Vendor Participant in a case. On the other hand, the remaining substates represent global facts about the case status—public awareness (p, P), exploit public (x, X), and attacks observed (a, A). This local versus global distinction will become important in Chapter 6.

5.2.1 Vendor Notification

Vendor Awareness ($\mathbf{V}$) occurs when a Participant—typically a Finder, Coordinator, or another Vendor—is in RM *Accepted* and notifies the Vendor ($q^{cs} \in vfd\cdots \xrightarrow{\mathbf{V}} Vfd\cdots$). In turn, the Vendor starts in $q^{rm} = Received$ and proceeds to follow their validation and prioritization routines. We previously outlined this in Table 2.1.

Depending on which parties are involved in a CVD case, the EM process might already be underway prior to Vendor notification (e.g., $q^{em} \in \{P, A, R\}$). For example, a Reporter and Coordinator might have already agreed to a disclosure timeline. Or, in an MPCVD case, other Vendors may have already been coordinating the case under an embargo and only recently realized the need to engage with a new Vendor on the case. The latter example is consistent with public narratives about the Meltdown/Spectre vulnerabilities [31].

Once a case has reached $q^{cs} \in Vfdpxa$ for at least one Vendor,

- If the EM process has not started, it SHOULD begin as soon as possible.
- Any proposed embargo SHOULD be decided (*accept*, *reject*) soon after the first Vendor is notified.

$$q^{cs} \in Vfdpxa \implies q^{em} \in \begin{cases} None \xrightarrow{propose} Proposed \\ Proposed \begin{cases} \xrightarrow{reject} None \\ \xrightarrow{accept} Accepted \end{cases} \\ Accepted \\ Revise \end{cases} \tag{5.1}$$

5.2.2 Fix Ready

Fix Readiness (**F**) can occur only when a Vendor is in the *Accepted* state. As a reminder, in MPCVD cases, each affected Vendor has their own RM state, so this constraint applies to each Vendor individually.

With respect to EM, when the case state is $q^{cs} \in VF{\cdot}pxa$, it's usually too late to start a new embargo. Once a case has reached $q^{cs} \in VF{\cdot}pxa$,

- New embargo negotiations SHOULD NOT start.
- Proposed but not-yet-agreed-to embargoes SHOULD be rejected.
- Existing embargoes ($q^{em} \in \{Active,\ Revise\}$) MAY continue but SHOULD prepare to *terminate* soon.

$$q^{cs} \in VF{\cdot}pxa \implies q^{em} \in \begin{cases} None \\ Proposed \xrightarrow{reject} None \\ Accepted \\ Revise \end{cases} \tag{5.2}$$

In MPCVD cases, where some Vendors are likely to reach $q^{cs} \in VF{\cdot}{\cdot}{\cdot}{\cdot}$ before others,

- Participants MAY propose an embargo extension to allow trailing Vendors to catch up before publication.
- Participants SHOULD accept reasonable extension proposals for such purposes when possible (e.g., when other constraints could still be met by the extended deadline).

5.2.3 Fix Deployed

For vulnerabilities in systems where the Vendor controls deployment, the Fix Deployment (**D**) event can only occur if the Vendor is in $q^{rm} = Accepted$.

For vulnerabilities in systems where Public Awareness must precede Deployment ($\mathbf{P} \prec \mathbf{D}$), the Vendor status at the time of deployment might be irrelevant—assuming, of course, that

they at least passed through $q^{rm} = Accepted$ at some point as is required for Fix Ready (**F**), which, in turn, is a prerequisite for deployment (**D**).

As regards EM, by the time a fix has been deployed ($q^{cs} \in VFD\cdots$),

- New embargoes SHOULD NOT be sought.
- Any existing embargo SHOULD terminate.

$$q^{cs} \in VFD\cdots \implies q^{em} \in \begin{cases} None \\ Proposed \xrightarrow{reject} None \\ Accepted \xrightarrow{terminate} eXited \\ Revise \xrightarrow{terminate} eXited \end{cases} \tag{5.3}$$

As with the *Fix Ready* scenario in §5.2.2, MPCVD cases may have Vendors in varying states of *Fix Deployment*. Therefore the embargo extension caveats from that section apply to the *Fix Deployed* state as well.

5.2.4 Public Awareness

Within the context of a coordinated publication process, (**P**) requires at least one Participant to be in the $q^{rm} = Accepted$ state because Participants are presumed to publish only on cases they have accepted. Ideally, the Vendor is among those Participants, but as outlined in the *CERT Guide to Coordinated Vulnerability Disclosure* [14], that is not strictly necessary.

That said, the publishing party might be outside of *any* existing coordination process. For example, this is the situation when a report is already in the midst of a CVD process and a party outside the CVD case reveals the vulnerability publicly (e.g., parallel discovery, embargo leaks).

As for EM, the whole point of an embargo is to prevent **P** from occurring until other objectives (e.g., $q^{cs} \in VF{\cdot}px{\cdot}$) have been met. Therefore, once **P** has happened and the case state reaches $q^{cs} \in \cdots P{\cdot}{\cdot}$,

- New embargoes SHALL NOT be sought.
- Any existing embargo SHALL terminate.

$$q^{cs} \in \cdots P{\cdot}{\cdot} \implies q^{em} \in \begin{cases} None \\ Proposed \xrightarrow{reject} None \\ Accepted \xrightarrow{terminate} eXited \\ Revise \xrightarrow{terminate} eXited \end{cases} \tag{5.4}$$

5.2.5 Exploit Public

Exploit publishers may also be presumed to have a similar RM state model for their own work. Therefore, we can expect them to be in an RM *Accepted* state at the time of exploit code publication (**X**). However, we cannot presume that those who publish exploit code will be Participants in a pre-public CVD process. That said,

- Exploit Publishers who *are* Participants in pre-public CVD cases ($q^{cs} \in \cdots p{\cdot}{\cdot}$) SHOULD comply with the protocol described here, especially when they also fulfill other roles (e.g., Finder, Reporter, Coordinator, Vendor) in the process.

For example, as described in the Householder and Spring 2021 report [10], the preference for $\mathbf{P} \prec \mathbf{X}$ dictates that

- Exploit publishers SHOULD NOT release exploit code while an embargo is active ($q^{em} \in \{A, R\}$).

In the Householder and Spring 2021 report [10], the authors argue that public exploit code is either preceded by Public Awareness (**P**) or immediately leads to it. Therefore, once **X** has occurred ($q^{cs} \in \cdots\cdot X \cdot$),

- New embargoes SHALL NOT be sought.
- Any existing embargo SHALL terminate.

$$q^{cs} \in \cdots\cdot X \cdot \implies q^{em} \in \begin{cases} None \\ Proposed \xrightarrow{reject} None \\ Accepted \xrightarrow{terminate} eXited \\ Revise \xrightarrow{terminate} eXited \end{cases} \tag{5.5}$$

5.2.6 Attacks Observed

Nothing in this or any other CVD process model should be interpreted as constraining adversary activity.

- Participants MUST treat attacks as an event that could occur at any time and adapt their process as needed in light of the available information.

As we outlined in §3.2.7, when attacks are occurring, embargoes can often be of more benefit to adversaries than defenders. However, we also acknowledged in §4.3.1 that narrowly scoped attacks need not imply widespread adversary knowledge of the vulnerability. In such scenarios, it is possible that early embargo termination—leading to publication—might be of more assistance to other adversaries than it is to defenders. Thus, we need to allow room for Participant judgment based on their case-specific situation awareness.

Formally, once attacks have been observed ($q^{cs} \in \cdots\cdot\mathbf{A}$),

- New embargoes SHALL NOT be sought.
- Any existing embargo SHOULD terminate.

$$q^{cs} \in \cdots\cdot A \implies q^{em} \in \begin{cases} None \\ Proposed \xrightarrow{reject} None \\ Accepted \xrightarrow{terminate} eXited \\ Revise \xrightarrow{terminate} eXited \end{cases} \tag{5.6}$$

6 A Formal Protocol Definition for MPCVD

The MPCVD process can be described as a Communicating Hierarchical State Machine. In this chapter, we begin by laying out the requirements for a formal protocol definition followed by a step-by-step walkthrough of each of those requirements as they relate to the RM, EM, and CS models we described in preceding chapters.

6.1 Communication Protocol Definitions

A communication protocol allows independent processes, represented as finite state machines, to coordinate their state transitions through the passing of messages. Brand and Zafiropulo [4] defined a protocol as follows. A **protocol** with N processes is a quadruple:

$$protocol = \left\langle \langle S_i \rangle_{i=1}^{N}, \langle o_i \rangle_{i=1}^{N}, \langle M_{i,j} \rangle_{i,j=1}^{N}, succ \right\rangle \tag{6.1}$$

Where

- N is a positive integer representing the number of processes.
- $\langle S_i \rangle_{i=1}^{N}$ are N disjoint finite sets (S_i represents the set of states of process i).
- Each o_i is an element of S_i representing the initial state of process i.
- $\langle M_{ij} \rangle_{i,j=1}^{N}$ are N^2 disjoint finite sets with M_{ii} empty for all i. M_{ij} represents the messages that can be sent from process i to process j,
- $succ$ is a partial function mapping for each i and j,

  $$S_i \times M_{ij} \rightarrow S_i \text{ and } S_i \times M_{ji} \rightarrow S_i$$

  $succ(s, x)$ is the state entered after a process transmits or receives message x in state s. It is a transmission if x is from M_{ij} and a reception if x is from M_{ji}.

The **global state** of a protocol given by (6.1) is a pair $\langle S, C \rangle$, where

- S is an N-tuple of states $\langle s_1, \ldots, s_N \rangle$ with each s_i representing the current state of process i.
- C is an N^2-tuple $\langle c_{1,1}, \ldots, c_{1,N}, c_{2,1}, \ldots\ldots, c_{N,N} \rangle$, where each $c_{i,j}$ is a sequence of messages from $M_{i,j}$. The message sequence $c_{i,j}$ represents the contents of the channel from process i to j. (Note that $c_{i,j}$ is empty when $i = j$ since processes are presumed to not communicate with themselves.)

We detail each of these in the subsequent sections of this chapter: N in §6.2, $\langle S_i \rangle_{i=1}^{N}$ in §6.3, $\langle o_i \rangle_{i=1}^{N}$ in §6.5, $\langle M_{i,j} \rangle_{i,j=1}^{N}$ in §6.6, and $\langle succ \rangle_{i=1}^{N}$ in §6.7.

6.2 Number of Processes

The processes we are concerned with represent the different Participants in their roles (Finder, Vendor, Coordinator, Deployer, and Other). Each Participant has their own process, but Participants might take on multiple roles in a given case. The total number of processes N is simply the count of unique Participants, as shown in (6.2).

$$N = |Participants| = |Reporters \cup Vendors \cup Coordinators \cup Deployers \cup Others| \tag{6.2}$$

6.3 States

Each Participant in an MPCVD case has a corresponding RM state, an EM state, and an overall CS state. Therefore, we can represent a Participant's state as a triple comprising the state of each of these models as in (6.3).

$$q_{Participant} = (q^{rm}, q^{em}, q^{cs}) \tag{6.3}$$

Good Participant situation awareness makes for good CVD decision making.

- Participants SHOULD track the state of other Participants in a case to inform their own decision making as it pertains to the case.

An example object model to facilitate such tracking is given in §8.1. However, the protocol we are developing is expected to function even when incomplete information is available to any given Participant.

- Adequate operation of the protocol MUST NOT depend on perfect information across all Participants.

A generic state model for a CVD Participant can be composed from the Cartesian product of $\mathcal{Q}^{rm}$, $\mathcal{Q}^{em}$, and $\mathcal{Q}^{cs}$ as shown in (6.4).

$$S_i = \underbrace{\begin{bmatrix} S \\ R \\ I \\ V \\ D \\ A \\ C \end{bmatrix}}_{\mathcal{Q}^{rm}} \times \underbrace{\begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix}}_{\mathcal{Q}^{em}} \times \underbrace{\left[\begin{bmatrix} \varnothing \\ vfd \\ Vfd \\ VFd \\ VFD \end{bmatrix} \times \begin{bmatrix} p \\ P \end{bmatrix} \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right]}_{\mathcal{Q}^{cs}} \tag{6.4}$$

Note that (6.4) splits the case state ($\mathcal{Q}^{cs}$) into chunks corresponding to the Vendor fix path ($vfd \xrightarrow{\mathbf{V}} Vfd \xrightarrow{\mathbf{F}} VFd \xrightarrow{\mathbf{D}} VFD$) and the public-exploit-attack ($pxa \xrightarrow{\cdots} PXA$) sub-models detailed in the Householder and Spring 2021 report [10]. This is done for two reasons. First, it gives us a more compact notation to represent the 32 states of the CS model. Second, it highlights the fact that the Vendor fix path represents the state of an individual Participant, whereas the public-exploit-attack sub-model represents facts about the world at large. Because not all Participants are Vendors or Deployers, Participants might not have a corresponding state on the $vfd \rightarrow VFD$ axis. Therefore, we add a null element $\varnothing$ to the set of states representing the Vendor fix path.

Thus, one might conclude that a total of 1,400 states is possible for each Participant, as shown in (6.5).

$$|S_i| = |\mathcal{Q}^{rm}| \cdot |\mathcal{Q}^{em}| \cdot |\mathcal{Q}^{cs}| = 7 \cdot 5 \cdot (5 \cdot 2 \cdot 2 \cdot 2) = 1400 \tag{6.5}$$

However, this dramatically overstates the possibilities for individual CVD Participant Roles because many of these states will be unreachable to individual Participants. In the remainder of this section, we detail these differences.

6.3.1 Unreachable States

For any Participant, the RM *Closed* state implies that the EM and CVD Case states do not matter. Similarly, for any Participant, the RM *Start* state represents a case that the Participant doesn't even know about yet. Therefore, the *Start* state also implies that the EM and CVD Case states do not matter. We use $*$ to represent the "don't care" value.

$$q^{rm} \in \{S, C\} \implies (q^{em} \in *) \cup (q^{cs} \in *) \tag{6.6}$$

A public exploit implies the vulnerability is public as well. In other words, $q^{cs} \in \cdots pX\cdot$ is an ephemeral state that resolves quickly to $q^{cs} \in \cdots PX\cdot$. (As a reminder, dots $(\cdot)$ in CVD case state notation indicate single-character wildcards.)

$$q^{cs} \in \cdots\cdot X\cdot \implies q^{cs} \in \cdots PX\cdot \tag{6.7}$$

Furthermore, when a vulnerability becomes public, the EM state no longer matters.

$$q^{cs} \in \cdots P\cdot\cdot \implies q^{em} \in * \tag{6.8}$$

Taken together, we can modify (6.4) in light of (6.6), (6.7), and (6.8). The result is shown in (6.9).

$$S_i = \begin{cases} (S, *, *) \\ \begin{bmatrix} R \\ I \\ V \\ D \\ A \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\begin{bmatrix} \varnothing \\ vfd \\ Vfd \\ VFd \\ VFD \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] \\ \begin{bmatrix} R \\ I \\ V \\ D \\ A \end{bmatrix} \times \begin{bmatrix} * \end{bmatrix} \times \left[\begin{bmatrix} \varnothing \\ vfd \\ Vfd \\ VFd \\ VFD \end{bmatrix} \times \begin{bmatrix} P \end{bmatrix} \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] \\ (C, *, *) \end{cases} \tag{6.9}$$

This means that each Participant must be in one of 352 possible states.

$$\begin{aligned} |S_i| &= 1 + \big(5 \cdot 5 \cdot (5 \cdot 1 \cdot 1 \cdot 2)\big) + \big(5 \cdot 1 \cdot (5 \cdot 1 \cdot 2 \cdot 2\big) + 1 \\ &= 352 \end{aligned} \tag{6.10}$$

6.3.2 Vendors (Fix Suppliers)

Vendors are the sole providers of fixes. Therefore, they are the only Participants in a CVD case for which the $Vfd \xrightarrow{\mathbf{F}} VFd \xrightarrow{\mathbf{D}} VFD$ path is possible. Furthermore, since they are Vendors by definition, they do not have access to the vfd state or the $\varnothing$ state that was just

added. As a Vendor has a report in *Received*, it is, by definition, at least in the Vfd case state.

Vendors create fixes only when they are in the *Accepted* RM state. Because the *Received*, *Invalid*, and *Valid* states come strictly *before* the *Accepted* state in the RM DFA, there is no way for the Vendor to be in either VFd or VFD while in any of those states.

$$q^{rm}_{Vendor} \in \{R, I, V\} \implies q^{cs}_{Vendor} \in Vfd\cdots \tag{6.11}$$

Vendors with the ability to deploy fixes themselves have access to three states in the fix path: $\{Vfd,\ VFd,\ VFD\}$. However, this is not always the case. Vendor Participants without a deployment capability can only create fixes, limiting them to the middle two states in the fix path: $\{Vfd,\ VFd\}$. Additional discussion of the distinction between Vendors with and without a deployment capability can be found in the Householder and Spring 2021 report [10].

We apply these caveats to the generic model from (6.9) to arrive at a Vendor state model in in (6.12)

$$S_{i_{Vendor}} = \begin{cases} (S, *, *) \\ \begin{bmatrix} R \\ I \\ V \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\left[Vfd \right] \times \left[p \right] \times \left[x \right] \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{unprioritized, maybe embargoed} \\ \begin{bmatrix} D \\ A \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\begin{bmatrix} Vfd \\ VFd \\ VFD^{\dagger} \end{bmatrix} \times \left[p \right] \times \left[x \right] \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{prioritized, maybe embargoed} \\ \begin{bmatrix} R \\ I \\ V \end{bmatrix} \times \left[* \right] \times \left[\left[Vfd \right] \times \left[P \right] \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{unprioritized, embargo irrelevant} \\ \begin{bmatrix} D \\ A \end{bmatrix} \times \left[* \right] \times \left[\begin{bmatrix} Vfd \\ VFd \\ VFD^{\dagger} \end{bmatrix} \times \left[P \right] \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{prioritized, embargo irrelevant} \\ (C, *, *) \end{cases} \tag{6.12}$$

The † on VFD in (6.12) indicates that the VFD state is accessible only to Vendors with a deployment capability. As tallied in (6.13) and (6.14) respectively, there are 128 possible states for a Vendor with deployment capability and 100 for those without.

$$
\begin{aligned}
|S_{i_{\frac{Vendor}{Deployer}}}| =& 1 + \big(3 \cdot 5 \cdot (1 \cdot 1 \cdot 1 \cdot 2)\big) + \big(2 \cdot 5 \cdot (3 \cdot 1 \cdot 1 \cdot 2)\big) \\
&+ \big(3 \cdot 1 \cdot (1 \cdot 1 \cdot 2 \cdot 2)\big) + \big(2 \cdot 1 \cdot (3 \cdot 1 \cdot 2 \cdot 2)\big) + 1 \\
=& 128
\end{aligned}
\tag{6.13}
$$

$$
\begin{aligned}
|S_{i_{Vendor}}| =& 1 + \big(3 \cdot 5 \cdot (1 \cdot 1 \cdot 1 \cdot 2)\big) + \big(2 \cdot 5 \cdot (2 \cdot 1 \cdot 1 \cdot 2)\big) \\
&+ \big(3 \cdot 1 \cdot (1 \cdot 1 \cdot 2 \cdot 2)\big) + \big(2 \cdot 1 \cdot (2 \cdot 1 \cdot 2 \cdot 2)\big) + 1 \\
=& 100
\end{aligned}
\tag{6.14}
$$

6.3.3 Non-Vendor Deployers

We just explained that not all Vendors are Deployers. Likewise, not all Deployers are Vendors. Most CVD cases leave Non-Vendor Deployers entirely out of the CVD process, so their appearance is expected to be rare in actual cases. However, there are scenarios when an MPCVD case may include Non-Vendor Deployers, such as when a vulnerability in some critical infrastructure component is being handled or when the MPCVD protocol is used in the context of a Vulnerability Disclosure Program (VDP). These Non-Vendor Deployers participate only in the $d \xrightarrow{\mathbf{D}} D$ transition on the fix path. Similar to the Vendor scenario in §6.3.2, it is expected that Deployers actually deploy fixes only when they are in the RM *Accepted* state (implying their intent to deploy). Therefore, their set of possible states is even more restricted than Vendors, as shown in (6.15).

$$
S_{i_{Deployer}} = \begin{cases}
(S, *, *) & \\
\begin{bmatrix} R \\ I \\ V \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\begin{bmatrix} d \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{unprioritized, maybe embargoed} \\
\begin{bmatrix} D \\ A \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\begin{bmatrix} d \\ D \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{prioritized, maybe embargoed} \\
\begin{bmatrix} R \\ I \\ V \end{bmatrix} \times \begin{bmatrix} * \end{bmatrix} \times \left[\begin{bmatrix} d \end{bmatrix} \times \begin{bmatrix} P \end{bmatrix} \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{unprioritized, embargo irrelevant} \\
\begin{bmatrix} D \\ A \end{bmatrix} \times \begin{bmatrix} * \end{bmatrix} \times \left[\begin{bmatrix} d \\ D \end{bmatrix} \times \begin{bmatrix} P \end{bmatrix} \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{prioritized, embargo irrelevant} \\
(C, *, *) &
\end{cases}
\tag{6.15}
$$

Thus, Non-Vendor Deployers can be expected to be in 1 of 100 possible states, as shown in (6.16).

$$\begin{aligned} |S_{i_{Deployer}}| =& 1 + \big(3 \cdot 5 \cdot (1 \cdot 1 \cdot 1 \cdot 2)\big) + \big(2 \cdot 5 \cdot (2 \cdot 1 \cdot 1 \cdot 2)\big) \\ &+ \big(3 \cdot 1 \cdot (1 \cdot 1 \cdot 2 \cdot 2)\big) + \big(2 \cdot 1 \cdot (2 \cdot 1 \cdot 2 \cdot 2)\big) + 1 \\ =& 100 \end{aligned} \tag{6.16}$$

6.3.4 Non-Vendor, Non-Deployer Participants

Finally, CVD cases often involve Participants who are neither Vendors nor Deployers. Specifically, Finder/Reporters fall into this category, as do Coordinators. Other roles, as outlined in the *CERT Guide to Coordinated Vulnerability Disclosure* [14], could be included here as well. Because they do not participate directly in the Vendor fix path, these Non-Vendor, Non-Deployer CVD Participants fall into the ∅ case substate we added to (6.4). Their state model is shown in (6.17).

$$S_{i_{Other}} = \begin{cases} (S, *, *) \\ \begin{bmatrix} R \\ I \\ V \\ D \\ A \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\left[\varnothing \right] \times \left[p \right] \times \left[x \right] \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{(maybe embargoed)} \\ \begin{bmatrix} R \\ I \\ V \\ D \\ A \end{bmatrix} \times \left[* \right] \times \left[\left[\varnothing \right] \times \left[P \right] \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{(embargo irrelevant)} \\ (C, *, *) \end{cases} \tag{6.17}$$

Non-Vendor Non-Deployer CVD Participants (Finder/Reporters, Coordinators, etc.) will be in 1 of 72 states, as calculated in (6.18).

$$\begin{aligned} |S_{i_{Other}}| =& 1 + \big(5 \cdot 5 \cdot (1 \cdot 1 \cdot 1 \cdot 2)\big) + \big(5 \cdot 1 \cdot (1 \cdot 1 \cdot 2 \cdot 2)\big) + 1 \\ =& 72 \end{aligned} \tag{6.18}$$

Finder-Reporters. As we discussed in §2.2.1, the early Finder states are largely hidden from view from other CVD Participants unless they choose to engage in the CVD process in the first place. Therefore, for a CVD protocol, we only need to care about Finder states once they have reached RM *Accepted*. Coincidentally, this is also a convenient way to mark the transition from Finder to Reporter.

$$S_{i_{Reporter}} = \begin{cases} (S,*,*) \quad \text{(hidden)} \\ (R,*,*) \quad \text{(hidden)} \\ (I,*,*) \quad \text{(hidden)} \\ (V,*,*) \quad \text{(hidden)} \\ \begin{bmatrix} D \\ A \end{bmatrix} \times \begin{bmatrix} N \\ P \\ A \\ R \\ X \end{bmatrix} \times \left[\begin{bmatrix} \varnothing \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{maybe embargoed} \\ \begin{bmatrix} D \\ A \end{bmatrix} \times \begin{bmatrix} * \end{bmatrix} \times \left[\begin{bmatrix} \varnothing \end{bmatrix} \times \begin{bmatrix} P \end{bmatrix} \times \begin{bmatrix} x \\ X \end{bmatrix} \times \begin{bmatrix} a \\ A \end{bmatrix} \right] & \text{embargo irrelevant} \\ (C,*,*) \end{cases} \tag{6.19}$$

Thus, for all practical purposes, we can ignore the hidden states in (6.19) and conclude that Finders who go on to become Reporters have only 29 possible states during a CVD case.

$$\begin{aligned} |S_{i_{Reporter}}| =& \big(2 \cdot 5 \cdot (1 \cdot 1 \cdot 1 \cdot 2)\big) + \big(2 \cdot 1 \cdot (1 \cdot 1 \cdot 2 \cdot 2)\big) + 1 \\ =& 29 \end{aligned} \tag{6.20}$$

6.4 A Lower Bounds on MPCVD State Space

Now we can touch on the lower bounds of the state space of an MPCVD case. Generically, we would expect the state space for N Participants to take the form of equation (6.21).

$$|S_{total}| = \prod_{i=1}^{N} |S_i| \tag{6.21}$$

The upper bound on the MPCVD state space is simply $352^N \approx 10^{2.55N}$. However, because of the Role-specific limits just described in §6.3, we already know that this overcounts the possible states significantly. We can do better still. If we ignore transient states while Participants converge on a consistent view of the global state of a case, we can drastically reduce the state space for an MPCVD case. Why? There are two reasons:

1. Because they represent facts about the outside world, the eight $\cdots pxa \rightarrow \cdots PXA$ CS substates are global to the case, not to individual Participants. This means all Participants should rapidly converge to the same substate.
2. Similarly, the five EM states are also global to the case and should converge rapidly.

Given these two observations, we can pull those global terms out of the state calculations for individual Participants,

$$|S_{global}| = 8 \times 5 = 40 \tag{6.22}$$

which leaves

$$|S_{Participant}| = \begin{cases} Reporter = 1 + 2 = 3 \\ Vendor = 2 + 3 + (2 \cdot 2) + 3 + (2 \cdot 2) = 16 \\ Vendor/Deployer = 2 + 3 + (2 \cdot 3) + 3 + (2 \cdot 3) = 20 \\ Coordinator = 2 + 5 = 7 \\ Deployer = 2 + 3 = 5 \\ Others = 2 + 5 = 7 \end{cases} \tag{6.23}$$

So our state space looks like

$$\begin{aligned} |S_{total}| = 40 \times 3^{N_{Reporter}} \times 16^{N_{Vendor}} \times 20^{N_{\frac{Vendor}{Deployer}}} \\ \times 7^{N_{Coordinator}} \times 5^{N_{Deployer}} \times 7^{N_{Other}} \end{aligned} \tag{6.24}$$

With these values in mind, we see that

- A two-party (Finder-Vendor) case might have a lower bound state space of $40 \times 3 \times 16 = 1,920$ states.
- A case like Meltdown/Spectre (with six Vendors and no Coordinators) might have $40 \times 3 \times 16^6 \approx 10^9$ states.
- A large, but not atypical, 200-Vendor case handled by the CERT/CC might have $40 \times 3 \times 16^{200} \times 7 \approx 10^{244}$ possible configurations.
- In the case of the log4j vulnerability CVE-2021-44228 in December 2021, the CERT/CC notified around 1,600 Vendors after the vulnerability had been made public [22]. Had this been an embargoed disclosure, the case would have a total state space around 10^{2000}.

That said, while these are dramatic numbers, the reader is reminded that the whole point of the MPCVD protocol is to *coordinate* the process so that it is not just hundreds or thousands of Participants behaving randomly.

6.5 Starting States

Each Participant begins a case in the state where the report management process is in the start state, there is no embargo in place, and the case has not made any progress.

A formal definition of the Participant start state is shown in (6.25).

$$o_i = (o_i^{rm},\ o_i^{em},\ o_i^{cs}) = (S, N, vfdpxa) \tag{6.25}$$

Following the discussion in §6.3, the starting states for Vendors, Deployers, and other Participants are shown in (6.26) (6.27), and (6.28), respectively.

$$o_{i_{Vendor}} = (S, N, vfdpxa) \tag{6.26}$$

$$o_{i_{Deployer}} = (S, N, dpxa) \tag{6.27}$$

$$o_{i_{Other}} = (S, N, pxa) \tag{6.28}$$

For a case to really begin, the Finder must at least reach the A state. Therefore, at the point when a second party finds out about the vulnerability from a Finder, the Finder is presumed to be already at $q_{Finder} = (A, N, pxa)$.

$$o_{i_{Finder}} = (A, N, pxa) \tag{6.29}$$

We will show in §6.7 how this plays out. But first, we need to define the message types that can be exchanged between Participants.

6.6 Message Types

In §6.3, we identified four main roles in the MPCVD process: Finder/Reporter, Vendor, Coordinator, and Deployer. Here we will examine the messages passed between them. Revisiting the definitions from §6.1,

> $\langle M_{ij} \rangle_{i,j=1}^{N}$ are N^2 disjoint finite sets with M_{ii} empty for all i: M_{ij} represents the messages that can be sent from process i to process j.

The message types in our proposed MPCVD protocol arise primarily from the following principle taken directly from the CVD Guide [14]:

> **Avoid Surprise** – As with most situations in which multiple parties are engaged in a potentially stressful and contentious negotiation, surprise tends to increase the risk of a negative outcome. The importance of clearly communicating expectations across all parties involved in a CVD process cannot be overemphasized. If we expect cooperation between all parties and stakeholders, we should do our best to match their expectations of being "in the loop" and minimize their surprise. Publicly disclosing a vulnerability without coordinating first can result in panic and an aversion to future cooperation from Vendors and Finders alike. CVD promotes continued cooperation and increases the likelihood that future vulnerabilities will also be addressed and remedied.

Now we condense that principle into the following protocol recommendation:

- Participants whose state changes in the RM, EM, or CVD State Models SHOULD send a message to other Participants for each transition.

If you are looking for a one-sentence summary of the entire MPCVD protocol, that was it.

As a reminder, those transitions are

- RM state transitions $\Sigma^{rm} = \{r, v, a, i, d, c\}$
- EM state transitions $\Sigma^{em} = \{p, a, r, t\}$
- CVD state transitions $\Sigma^{cs} = \{\mathbf{V}, \mathbf{F}, \mathbf{D}, \mathbf{P}, \mathbf{X}, \mathbf{A}\}$

We will address the specific circumstances when each message should be emitted in §6.7, but first we need to introduce the message types this recommendation implies. We cover messages associated with each state model, in turn, below, concluding the section with a few message types not directly connected to any particular state model.

6.6.1 RM Message Types

With the exception of the Finder/Reporter, each Participant's involvement in a CVD case starts with the receipt of a report from another Participant who is already in the *Accepted* ($q^{rm} \in A$) state.[4]

Report Submission (RS) is a message from one Participant to a new Participant containing a vulnerability report.

We continue with a list of state-change messages *originating* from a Participant in the RM process:

Report Invalid (RI) is a message indicating the Participant has designated the report as invalid.

Report Valid (RV) is a message indicating the Participant has designated the report as valid.

Report Deferred (RD) is a message indicating the Participant is deferring further action on a report.

Report Accepted (RA) is a message indicating the Participant has accepted the report for further action.

Report Closed (RC) is a message indicating the Participant has closed the report.

Report Acknowledgement (RK) is a message acknowledging the receipt of a report.

Report Error (RE) is a message indicating a Participant received an unexpected RM message.

A summary of the RM message types is shown in (6.30).

$$M^{rm} = \{RS, RI, RV, RD, RA, RC, RK, RE\} \tag{6.30}$$

All state changes are from the Participant's (sender's) perspective, not the recipient's perspective. We will see in §6.7 that the receipt of a *Report Submission* is the only message whose *receipt* directly triggers an RM state change in the receiver. All other RM messages are used to convey the sender's status.

- Participants SHOULD act in accordance with their own policy and process in deciding when to transition states in the RM model.
- Participants SHOULD NOT mark duplicate reports as invalid.
- Instead, duplicate reports SHOULD pass through $Valid$ ($q^{rm} \in V$), although they MAY be subsequently (immediately or otherwise) deferred ($q^{rm} \in V \xrightarrow{d} D$) in favor of the original.
- Participants SHOULD track the RM states of the other Participants in the case.

An example object model for such tracking is described in §8.1. Furthermore, while these messages are expected to inform the receiving Participant's choices in their own RM process, this protocol intentionally does not specify any other recipient RM state changes upon receipt of an RM message.

[4] As we discuss in §2.2.1, the Finder's states $q^{rm} \in \{R, I, V\}$ are not observable to the CVD process because Finders start coordination only when they have already reached $q^{rm} = A$.

6.6.2 EM Message Types

Whereas the RM process is unique to each Participant, the EM process is global to the case. Therefore, we begin with the list of message types a Participant SHOULD emit when their EM state changes.

Embargo Proposal (EP) is a message containing proposed embargo terms (e.g., date/time of expiration).

Embargo Proposal Rejection (ER) is a message indicating the Participant has rejected an embargo proposal.

Embargo Proposal Acceptance (EA) is a message indicating the Participant has accepted an embargo proposal.

Embargo Revision Proposal (EV) is a message containing a proposed revision to embargo terms (e.g., date/time of expiration).

Embargo Revision Rejection (EJ) is a message indicating the Participant has rejected a proposed embargo revision.

Embargo Revision Acceptance (EC) is a message indicating the Participant has accepted a proposed embargo revision.

Embargo Termination (ET) is a message indicating the Participant has terminated an embargo (including the reason for termination). Note that an *Embargo Termination* message is intended to have immediate effect.

- If an early termination is desired but the termination date/time is in the future, this SHOULD be achieved through an *Embargo Revision Proposal* and additional communication as necessary to convey the constraints on the proposal.

Embargo Acknowledgement (EK) is a message acknowledging receipt of an EM message.

Embargo Error (EE) is a message indicating a Participant received an unexpected EM message.

A summary of the EM message types is shown in (6.31).

$$M^{em} = \{EP, ER, EA, EV, EJ, EC, ET, EK, EE\} \tag{6.31}$$

6.6.3 CS Message Types

From the CS process in §4, the following is the list of messages associated with CS state changes:

Vendor Awareness (CV) is a message to other Participants indicating that a report has been delivered to a specific Vendor. Note that this is an announcement of a state change for a Vendor, not the actual report to the Vendor, which is covered in the **Report Submission** (RS) above.

- **Vendor Awareness** messages SHOULD be sent only by Participants with direct knowledge of the notification (i.e., either by the Participant who sent the report to the Vendor or by the Vendor upon receipt of the report).

Fix Readiness (CF) is a message from a Participant (usually a Vendor) indicating that a specific Vendor has a fix ready.

Fix Deployed (CD) is a message from a Participant (usually a Deployer) indicating that they have completed their fix deployment process. This message is expected to be rare in most MPCVD cases because Deployers are rarely included in the coordination effort.

Public Awareness (CP) is a message from a Participant indicating that they have evidence that the vulnerability is known to the public. This message might be sent after a Participant has published their own advisory or if they have observed public discussion of the vulnerability.

Exploit Public (CX) is a message from a Participant indicating that they have evidence that an exploit for the vulnerability is publicly available. This message might be sent after a Participant has published their own exploit code, or if they have observed exploit code available to the public.

Attacks Observed (CA) is a message from a Participant indicating that they have evidence that attackers are exploiting the vulnerability in attacks.

CVD Case State Acknowledgement (CK) is a message acknowledging receipt of a CS message.

CVD Case State Error (CE) is a message indicating a Participant received an unexpected CS message.

A summary of the CS message types is shown in (6.32).

$$M^{cs} = \{CV, CF, CD, CP, CX, CA, CK, CE\} \tag{6.32}$$

6.6.4 Other Message Types

Finally, there are a few additional message types required to tie the coordination process together. Most of these message types are *not* associated with a specific state change, although they might trigger activities or events that could cause a state change in a Participant (and therefore trigger one or more of the above message types to be sent).

General Inquiry (GI) is a message from a Participant to one or more other Participants to communicate non-state-change information. Examples of general inquiry messages include but are not limited to

- asking or responding to a question
- requesting an update on a Participant's status
- requesting review of a draft publication
- suggesting a potential Participant to be added to a case
- coordinating other events
- resolving a loss of Participant state synchronization

General Acknowledgement (GK) is a message from a Participant indicating their receipt of any of the other messages listed here.

General Error (GE) is a message indicating a general error has occurred.

A summary of the General message types is shown in (6.33).

$$M^{*} = \{GI, GK, GE\} \tag{6.33}$$

6.6.5 Message Type Redux

Thus, the complete set of possible messages between processes is $M_{i,j} = M^{rm} \cup M^{em} \cup M^{cs} \cup M^{*}$. For convenience, we collected these into (6.34) and provide a summary in Table 6.1.

Table 6.1: MPCVD Protocol Message Types (M_{ij}) and the Corresponding Sender State Changes

Process Model	M_{ij}	Message Type	Emit when
RM	RS	Report Submission	A
	RI	Report Invalid	$R \xrightarrow{i} I$
	RV	Report Valid	$\{R, I\} \xrightarrow{v} V$
	RD	Report Deferred	$\{V, A\} \xrightarrow{d} D$
	RA	Report Accepted	$\{V, D\} \xrightarrow{a} A$
	RC	Report Closed	$\{I, D, A\} \xrightarrow{c} C$
	RK	Report Acknowledgement	$< receipt >$
	RE	Report Error	$< error >$
EM	EP	Embargo Proposal	$\{N, P\} \xrightarrow{p} P$
	ER	Embargo Proposal Rejection	$P \xrightarrow{r} N$
	EA	Embargo Proposal Acceptance	$P \xrightarrow{a} A$
	EV	Embargo Revision Proposal	$A \xrightarrow{p} R$
	EJ	Embargo Revision Rejection	$R \xrightarrow{r} A$
	EC	Embargo Revision Acceptance	$R \xrightarrow{a} A$
	ET	Embargo Termination	$\{A, R\} \xrightarrow{t} X$
	EK	Embargo Acknowledgement	$< receipt >$
	EE	Embargo Error	$< error >$
CS	CV	Vendor Aware	$vfd\cdots \xrightarrow{\mathbf{V}} Vfd\cdots$
	CF	Fix Ready	$Vfd\cdots \xrightarrow{\mathbf{F}} VFd\cdots$
	CD	Fix Deployed	$VFd\cdots \xrightarrow{\mathbf{D}} VFD\cdots$
	CP	Public Aware	$\cdots p\cdot\cdot \xrightarrow{\mathbf{P}} \cdots P\cdot\cdot$
	CX	Exploit Public	$\cdots\cdot x\cdot \xrightarrow{\mathbf{X}} \cdots\cdot X\cdot$
	CA	Attacks Observed	$\cdots\cdot\cdot a \xrightarrow{\mathbf{A}} \cdots\cdot\cdot A$
	CK	CVD Case State Acknowledgement	$< receipt >$
	CE	CVD Case State Error	$< error >$
General	GI	General Inquiry	$< anytime >$
	GK	General Acknowledgement	$< receipt >$
	GE	General Error	$< error >$

$$M_{i,j} = \left\{ \begin{array}{l} RS, RI, RV, RD, RA, RC, RK, \\ RE, EP, ER, EA, EV, EJ, EC, \\ ET, EK, EE, CV, CF, CD, CP, \\ CX, CA, CK, CE, GI, GK, GE \end{array} \right\} \begin{array}{l} \text{where } i \neq j; \\ \varnothing \text{ otherwise;} \\ \text{for } i, j \leq N \end{array} \tag{6.34}$$

Message formats are left as future work in §8.3.1.

6.7 Transition Functions

Revisiting the formal protocol definition from the beginning of the chapter,

succ is a partial function mapping for each i and j,

$$S_i \times M_{ij} \rightarrow S_i \text{ and } S_i \times M_{ji} \rightarrow S_i$$

$succ(s, x)$ is the state entered after a process transmits or receives message x in state s. It is a transmission if x is from M_{ij} and a reception if x is from M_{ji}.

Table 6.2: RM Messages Sent and State Transitions ($S_i \times M_{ij}^{rm} \rightarrow S_i$)

Sender Precondition ($s_n \in S_i$)			Sender Transition ($s_n \rightarrow s_{n+1}$)			Message Type(s)
q^{cs}	q^{rm}	q^{em}	q^{cs}	q^{rm}	q^{em}	M_{ij}
$*$	A	$*$	$-$	$-$	$-$	RS
$*$	$\{R, V\}$	$*$	$-$	$\xrightarrow{i} I$	$-$	RI
$*$	$\{R, I\}$	$*$	$-$	$\xrightarrow{v} V$	$-$	RV
$*$	$\{V, A\}$	$*$	$-$	$\xrightarrow{d} D$	$-$	RD
$*$	$\{V, D\}$	$*$	$-$	$\xrightarrow{a} A$	$-$	RA
$*$	$\{I, D, A\}$	$*$	$-$	$\xrightarrow{c} C$	$-$	RC
$*$	$*$	$*$	$-$	$-$	$-$	RE
$*$	$*$	$*$	$-$	$-$	$-$	RK

In this section, we describe the transition functions for the RM, EM, and CVD Case processes, respectively. Note that while the RM process is largely independent of the other two process models, the EM and CVD process models have some noteworthy interactions, which we will cover in detail.

6.7.1 RM Transition Functions

Because it only reflects an individual Participant's report handling status, the RM process operates largely independent of both the EM and CS processes. Otherwise,

- Participants MUST be in RM *Accepted* to send a report (RS) to someone else.
- Participants SHOULD send RI when the report validation process ends in an *invalid* determination.
- Participants SHOULD send RV when the report validation process ends in a *valid* determination.
- Participants SHOULD send RD when the report prioritization process ends in a $deferred$ decision.
- Participants SHOULD send RA when the report prioritization process ends in an *accept* decision.
- Participants SHOULD send RC when the report is closed.
- Participants SHOULD send RE regardless of the state when any error is encountered.
- Recipients MAY ignore messages received on $Closed$ cases.
- Recipients SHOULD send RK in acknowledgment of any $R*$ message except RK itself.
- Vendor Recipients should send both CV and RK in response to a report submission (RS). If the report is new to the Vendor, it MUST transition $q^{cs} \xrightarrow{\mathbf{V}} Vfd\cdots$.
- Any $R*$ message, aside from RS, received by recipient in $q^{rm} \in S$ is an error because it indicates the sender thought the receiver was aware of a report they had no knowledge of. The Recipient SHOULD respond with both an RE to signal the error and GI to find out what the sender expected.
- Recipients SHOULD acknowledge RE messages (RK) and inquire (GI) as to the nature of the error.

Table 6.3: RM Messages Received and State Transitions ($S_i \times M_{ji}^{rm} \to S_i$)

Because the RM process is independent of the EM process, q^{em} is omitted as a subcolumn of both Receiver Precondition ($q^{em} \in *$) and Receiver Transition ($q_n^{em} = q_{n+1}^{em}$).

Recv. Msg.	Receiver Precondition ($s_n \in S_i$)		Receiver Transition ($s_n \to s_{n+1}$)		Response Msg(s).
M_{ji}	q^{cs}	q^{rm}	q^{cs}	q^{rm}	M_{ij}
$*$	$*$	C	–	–	–
RS	$vfd\cdots$	S	$\xrightarrow{\mathbf{V}} Vfd\cdots$	$\xrightarrow{r} R$	$RK + CV$ (Vendor)
		$\{R, I, V, D, A\}$		–	
	$V\cdots\cdots$	S	–	$\xrightarrow{r} R$	
		$\{R, I, V, D, A\}$		–	
	$*$	S	–	$\xrightarrow{r} R$	RK (Non-Vendor)
		$\{R, I, V, D, A\}$		–	
$RI, RV, RD,$	$*$	$\{R, I, V, D, A\}$	–	–	RK
RA, RC	$*$	S	–	–	$RE + GI$
RE	$*$	$*$	–	–	$RK + GI$
RK	$*$	$*$	–	–	–

Table 6.2 lists each RM message type and the states in which that message is appropriate to send along with the corresponding sender state transition. Table 6.3 lists the effects of receiving RM messages on the receiving Participant's state coupled with the expected response message.

6.7.2 EM Transition Functions

The appropriate Participant behavior in the EM process depends on whether the case state q^{cs} is in $\cdots pxa$ or not.

- Participants SHALL NOT negotiate embargoes where the vulnerability or its exploit is public or attacks are known to have occurred.
- Participants MAY begin embargo negotiations before sending the report itself in an RS message. Therefore, it is *not* an error for an $E*$ message to arrive while the Recipient is unaware of the report ($q^{rm} \in S$).
- Participants MAY reject any embargo proposals or revisions for any reason.
- If information about the vulnerability or an exploit for it has been made public, Participants SHALL terminate the embargo ($q^{cs} \in \{\cdots P\cdot\cdot, \cdots\cdot X\cdot\}$).
- If attacks are known to have occurred, Participants SHOULD terminate the embargo ($q^{cs} \in \cdots\cdots A$).
- Participants SHOULD send EK in acknowledgment of any other $E*$ message except EK itself.
- Participants SHOULD acknowledge (EK) and inquire (GI) about the nature of any error reported by an incoming EE message.

Table 6.4 lists each EM message type and the states in which that message is appropriate to send along with the corresponding sender state transition. Table 6.5 lists the effects of receiving an EM message to the receiving Participant's state, grouped by the EM message type received.

Table 6.4: EM Messages Sent and State Transitions ($S_i \times M_{ij}^{em} \rightarrow S_i$)

Sender Precondition ($s_n \in S_i$)			Sender Transition ($s_n \rightarrow s_{n+1}$)			Message Type(s)
q^{cs}	q^{rm}	q^{em}	q^{cs}	q^{rm}	q^{em}	M_{ij}
$\cdots pxa$	$\neg C$	N	–	–	$\xrightarrow{p} P$	EP
		P	–	–	$\xrightarrow{a} A$	EA
		A	–	–	$\xrightarrow{p} R$	EV
		R	–	–	$\xrightarrow{r} A$	EJ
			–	–	$\xrightarrow{a} A$	EC
$*$	$\neg C$	P	–	–	$\xrightarrow{r} N$	ER
		A	–	–	$\xrightarrow{t} X$	ET
		R				
		$*$	–	–	–	EK
						EE

Table 6.5: EM Messages Received and State Transitions ($S_i \times M_{ji}^{em} \rightarrow S_i$).

Incoming EM Messages do not trigger any change in q^{cs} or q^{rm}; therefore, those subcolumns are omitted from the Receiver Transition column. When CS is $q^{cs} \notin \cdots pxa$, embargoes are not viable.

Recv. Msg.	Receiver Precondition ($s_n \in S_i$)			Receiver Transition ($s_n \rightarrow s_{n+1}$)	Response Msg(s).
M_{ji}	q^{cs}	q^{rm}	q^{em}	q^{em}	M_{ij}
EP	$\cdots pxa$	$\neg C$	N	$\xrightarrow{p} P$	EK
			P	–	
EA				$\xrightarrow{a} A$	
EV			A	$\xrightarrow{p} R$	
			R	–	
EJ				$\xrightarrow{r} A$	
EC				$\xrightarrow{a} A$	
ER	$*$	$\neg C$	P	$\xrightarrow{r} N$	
ET			A	$\xrightarrow{t} X$	
			R		
			X	–	
EP	$\neg\cdots pxa$	$\neg C$	N	–	ER
			P	$\xrightarrow{r} N$	
EA					
EV			A	$\xrightarrow{t} X$	ET
			R		
EJ					
EC					
EE	$*$	$\neg C$	$*$	–	$EK + GI$
EK	$*$	$*$	$*$	–	–
Any EM msg. not addressed above	$*$	$\neg C$	$*$	–	EE

Table 6.6: CS Messages Sent and State Transitions ($S_i \times M_{ij}^{cs} \rightarrow S_i$)

By convention, case states are labeled in the order $vfdpxa$. Participant state is a tuple of the individual CVD Case, RM, and EM states $S_i = (q^{cs}, q^{rm}, q^{em})$. Note that when a CS message induces a q^{rm} or q^{em} state change, the corresponding RM or EM message should be sent as indicated in Tables 6.2 and 6.4, respectively. Dots (·) in states indicate single wildcards. For example, $Vfd...$ includes $Vfdpxa, VfdPxA, VfdPXA$, etc. Asterisks (∗) indicate arbitrary wildcards. Dashes (−) indicate no state change.

Sender Precondition ($s_n \in S_i$)			Sender Transition ($s_n \rightarrow s_{n+1}$)			Message Type(s)
q^{cs}	q^{rm}	q^{em}	q^{cs}	q^{rm}	q^{em}	M_{ij}^{cs}
$vfd\cdots$	S	$*$	$\xrightarrow{\mathbf{V}} Vfd\cdots$	$\xrightarrow{r} R$	$-$	CV
$Vfd\cdots$	$\neg C$	$*$	$\xrightarrow{\mathbf{F}} VFd\cdots$	$-$	$-$	CF
$VFd\cdots$	$\neg C$	$*$	$\xrightarrow{\mathbf{D}} VFD\cdots$	$-$	$-$	CD
$\cdots p\cdot\cdot$	$\neg C$	$\{N, X\}$ P $\{A, R\}$	$\xrightarrow{\mathbf{P}} \cdots P\cdot\cdot$	$-$	$-$ $\xrightarrow{r} N$ $\xrightarrow{t} X$	CP
$\cdots px\cdot$	$\neg C$	$\{N, X\}$ P $\{A, R\}$	$\xrightarrow{\mathbf{X}}\xrightarrow{\mathbf{P}} \cdots PX\cdot$	$-$	$-$ $\xrightarrow{r} N$ $\xrightarrow{t} X$	$CX + CP$
$\cdots Px\cdot$	$\neg C$	$\{N, X\}$ P $\{A, R\}$	$\xrightarrow{\mathbf{X}} \cdots PX\cdot$	$-$	$-$ $\xrightarrow{r} N$ $\xrightarrow{t} X$	CX
$\cdots\cdot a$	$\neg C$	$\{N, X\}$ P $\{A, R\}$	$\xrightarrow{\mathbf{A}} \cdots\cdot A$	$-$	$-$ $\xrightarrow{r} N$ $\xrightarrow{t} X$	CA

6.7.3 CVD Transition Functions

The Vendor-specific portions of the CS (*Vendor Awareness*, *Fix Ready*, and *Fix Deployed*) are per-Participant states. Therefore, the receiver of a message indicating another Participant has changed their $\{v, V\}$, $\{f, F\}$ or $\{d, D\}$ status is not expected to change their own state as a result.[5]

However, this is not the case for the remainder of the CS substates. As above, the appropriate Participant response to receiving CS messages (namely, those surrounding *Public Awareness*, *Exploit Public*, or *Attacks Observed*) depends on the state of the EM process.

- Participants SHALL initiate embargo termination upon becoming aware of publicly available information about the vulnerability or its exploit code.
- Participants SHOULD initiate embargo termination upon becoming aware of attacks against an otherwise unpublished vulnerability.

Table 6.6 lists each CVD message type and the states in which that message is appropriate to send along with the corresponding sender state transition. Table 6.7 lists the effects of receiving a CS message to the receiving Participant's state coupled with the expected response message.

[5] Effective coordination is usually improved with Participants' mutual awareness of each other's state, of course.

Table 6.7: CS Messages Received and State Transitions

Note that when a CS message induces a q^{em} state change, the corresponding EM message should be sent as indicated in Table 6.4. Dots (·) in states indicate single wildcards. Asterisks ($*$) indicate arbitrary wildcards. Dashes ($-$) indicate no state change.

Recv. Msg.	Receiver Precondition ($s_n \in S_i$)			Receiver Transition ($s_n \to s_{n+1}$)			Response Msg.
M_{ji}	q^{cs}	q^{rm}	q^{em}	q^{cs}	q^{rm}	q^{em}	M_{ij}
CV	$*$	$\neg C$	$*$	$-$	$-$	$-$	CK
CF							
CD							
CP	$\cdots p\cdot\cdot$	$\neg C$	P $\{A,R\}$ $\{N,X\}$	$\xrightarrow{\mathbf{P}} \cdots P\cdot\cdot$	$-$	$\xrightarrow{r} N$ $\xrightarrow{t} X$ $-$	
	$\cdots P\cdot\cdot$	$\neg C$	$*$	$-$	$-$	$-$	
CX	$\cdots px\cdot$	$\neg C$	P $\{A,R\}$ $\{N,X\}$	$\xrightarrow{\mathbf{X}}\xrightarrow{\mathbf{P}} \cdots PX\cdot$	$-$	$\xrightarrow{r} N$ $\xrightarrow{t} X$ $-$	
	$\cdots Px\cdot$	$\neg C$	$*$	$\xrightarrow{\mathbf{X}} \cdots PX\cdot$	$-$	$-$	
	$\cdots PX\cdot$	$\neg C$	$*$	$-$	$-$	$-$	
CA	$\cdots p\cdot a$	$\neg C$	P $\{A,R\}$ $\{N,X\}$	$\xrightarrow{\mathbf{A}} \cdots p\cdot A$	$-$	$\xrightarrow{r} N$ $\xrightarrow{t} X$ $-$	
	$\cdots P\cdot a$	$\neg C$	$*$	$\xrightarrow{\mathbf{A}} \cdots P\cdot A$	$-$	$-$	
	$\cdots\cdot A$	$\neg C$	$*$	$-$	$-$	$-$	
CE	$*$	$\neg C$	$*$	$-$	$-$	$-$	$CK + GI$
CK	$*$	$\neg C$	$*$	$-$	$-$	$-$	$-$

Table 6.8: General Messages Sent and State Transitions ($S_i \times M_{ij} \to S_i$)

Sender Precondition ($s_n \in S_i$)			Sender Transition ($s_n \to s_{n+1}$)			Message Type(s)
q^{cs}	q^{rm}	q^{em}	q^{cs}	q^{rm}	q^{em}	M_{ij}
$*$	$*$	$*$	$-$	$-$	$-$	GI
$*$	$*$	$*$	$-$	$-$	$-$	GK
$*$	$*$	$*$	$-$	$-$	$-$	GE

Table 6.9: General Messages Received and State Transitions ($S_i \times M_{ji} \to S_i$)

Receiver Precondition ($s_n \in S_i$)			Recv. Msg	Receiver Transition ($s_n \to s_{n+1}$)			Response Msg(s).
q^{cs}	q^{rm}	q^{em}	M_{ji}	q^{cs}	q^{rm}	q^{em}	M_{ij}
$*$	$*$	$*$	GI	$-$	$-$	$-$	GK
$*$	$*$	$*$	GK	$-$	$-$	$-$	$-$
$*$	$*$	$*$	GE	$-$	$-$	$-$	GI

6.7.4 General Transition Functions

Finally, for the sake of completeness, in Tables 6.8 and 6.9, we show that general inquiries, acknowledgments, and errors are otherwise independent of the rest of the processes. No state changes are expected to occur based on the receipt of a General message. Note that we do not mean to imply that the *content* of such a message is expected to have no effect on the progression of a case, merely that the act of sending or receiving a general message itself does not imply any necessary state change to either the sender or receiver Participants.

Table 6.8 lists each general message and the states in which it is appropriate to send along with the corresponding sender state. Table 6.9 lists the effects of receiving a general message to the receiving Participant's state coupled with the expected response message.

6.8 Formal MPCVD Protocol Redux

In this chapter, we have formally defined an MPCVD protocol

$$protocol_{MPCVD} = \left\langle \langle S_i \rangle_{i=1}^{N}, \langle o_i \rangle_{i=1}^{N}, \langle M_{i,j} \rangle_{i,j=1}^{N}, succ \right\rangle$$

where

- N is a positive integer representing the number of MPCVD Participants in a case and
- $\langle S_i \rangle_{i=1}^{N}$ are N disjoint finite sets in which each S_i represents the set of states of a given Participant i (6.9), as refined by (6.12), (6.15), (6.19), and (6.17).
- $\langle o_i \rangle_{i=1}^{N}$ is the set of starting states across all Participants in which each o_i is an element of S_i representing the initial state of each Participant i, as detailed in (6.26) (6.27), (6.28), and (6.29).
- $\langle M_{ij} \rangle_{i,j=1}^{N}$ are N^2 disjoint finite sets with M_{ii} empty for all i. M_{ij} represents the messages that can be sent from process i to process j. A list of message types is defined in (6.34) and summarized in Table 6.1.
- $succ$ is a partial function mapping for each i and j,

$$S_i \times M_{ij} \to S_i \text{ and } S_i \times M_{ji} \to S_i$$

indicating the state changes arising from the sending and receiving of messages between Participants. The full set of transition function definitions for our protocol is shown in Tables 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, and 6.9.

A summary diagram of the MPCVD state model S_i for an individual Participant is shown in Figure 6.1.

6.9 Worked Example

We conclude the chapter with a brief worked example showing a few usage scenarios of the protocol. We use UML Sequence Diagrams to show the interaction between Participant roles.

6.9.1 A Finder Becomes a Reporter

As mentioned in §2.2.1, Finders have a few hidden state transitions before the CVD process really begins. An example of this is shown in Figure 6.2a. The Finder must discover, validate, and prioritize their finding before initiating the CVD process.

Finders become Reporters when they report a vulnerability to someone else. Figure 6.2b shows a Finder sending a report (RS) in conjunction with an embargo proposal (EP) to a Vendor. The Vendor receives the report and updates their state accordingly. Then the Vendor replies to acknowledge receipt of the report and the embargo proposal, and confirms that they (i.e., the Vendor) are aware of the report (RK, EK, and CV, respectively). Note that the EK response is intended to convey receipt of the embargo proposal (EP) but does not constitute acceptance of the proposal. We will discuss that in the next subsection.

6.9.2 Vendor Evaluates Embargo

In Figure 6.3, we show a variety of responses a Vendor might have to an embargo proposal. First is a basic accept sequence in which the Vendor accepts the proposed embargo and tells the Reporter this through an EA message, as shown in Figure 6.3a. The Reporter acknowledges this with an EK in response.

Figure 6.3b shows a rejected proposal. As above, this is a simple sequence where the Vendor indicates their rejection of the proposal with an ER message, and the Reporter acknowledges this with an EK message.

Figure 6.3c demonstrates a Vendor embargo counterproposal. The Vendor responds to the Reporter's prior EP message with an EP message of their own. The Reporter initially acknowledges the counterproposal with an RK message and then evaluates it and accepts with an EA message. Finally, the Vendor acknowledges the acceptance with an EK message. Note, however, that there is no active embargo until the Reporter accepts it. This method of counterproposal might delay the establishment of an embargo.

Finally, Figure 6.3d offers what we think is a better approach than a simple counterproposal. In this "Accept-then-Counter" sequence, we see that the Vendor initially accepts the Reporter's proposed embargo and immediately follows up with a revision proposal of their own. The difference is that by initially accepting the proposal, the Vendor ensures that they are in an active embargo state before attempting to renegotiate. The sequence shown in Figure 6.3d is intended to be consistent with the previous discussion surrounding default embargo strategies in §3.2.6. One might think of this as the "Yes-And" rule for embargo negotiations.[6]

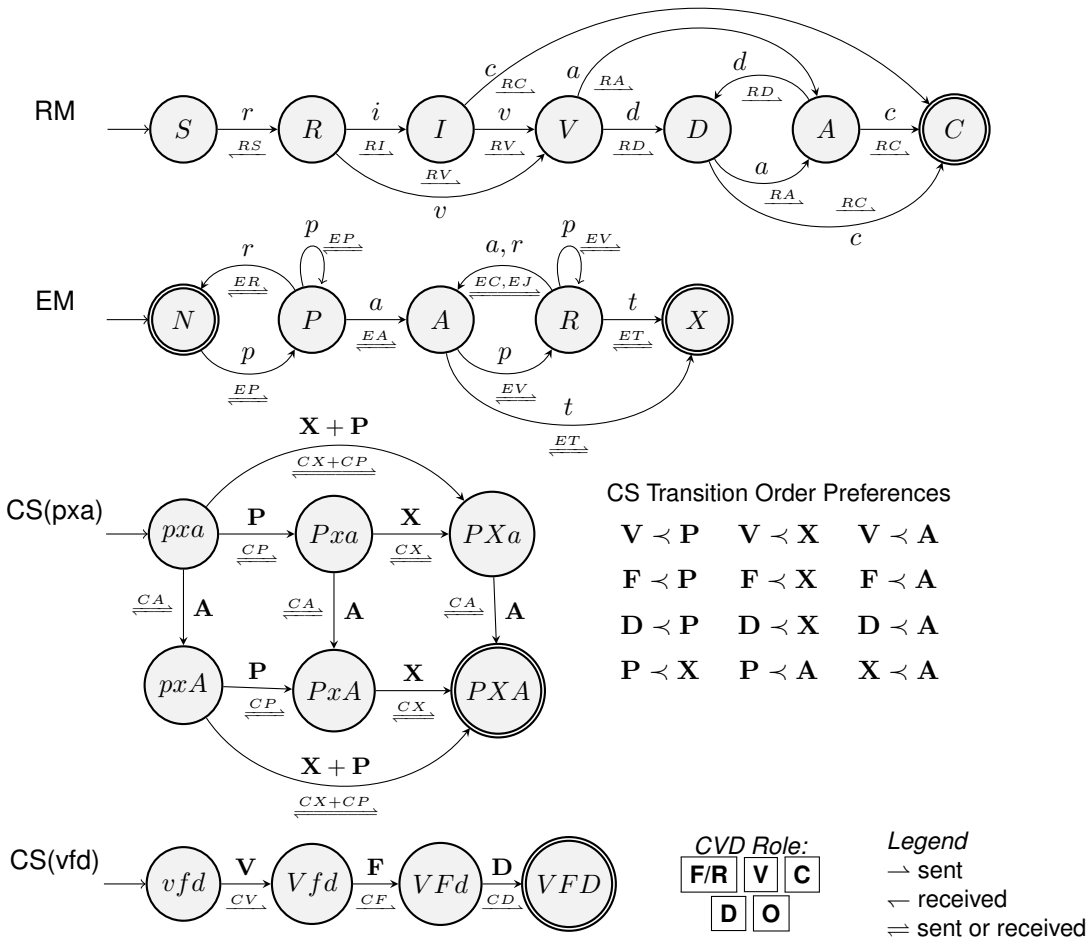
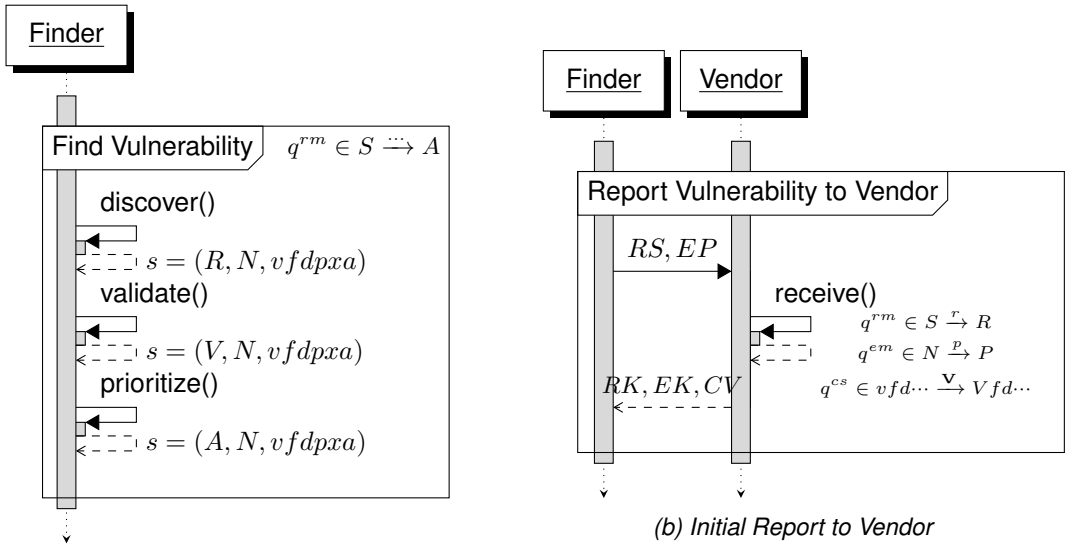


Figure 6.1: MPCVD Protocol State Model Summary for a Single Participant

*This diagram is offered as a quick-reference "score card" to facilitate conversations about the model. It is not meant to be a complete representation of the model presented in §6. Omitted here are the interactions between the various state models. Acknowledgments (RK, EK, CK, GK) and error messages (RE, EE, CE, GE) are also omitted. A summary of outgoing message types is shown in Table 6.1. Some states are not reachable by certain CVD Roles. CVD Roles include Finder/Reporter (**F**/**R**) Vendor (**V**), Coordinator (**C**), Deployer (**D**), and Other (**O**). Also note that the CS model is more complicated than what is shown here. Here it is broken up into two parts, CS(pxa) and CS(vfd), to simplify the diagram. The CS model split also reflects the idea that the CS(pxa) substates are a global property of the overall case, while the CS(vfd) substates are local to each Participant. Similarly, the RM model is per-Participant whereas the EM model belongs to the case and shares its state across all Participants.*



(a) A Finder Chooses to Do CVD

(b) Initial Report to Vendor

Figure 6.2: A Finder Becomes a Reporter, and a Vendor Acknowledges the Report Without Yet Accepting the Embargo

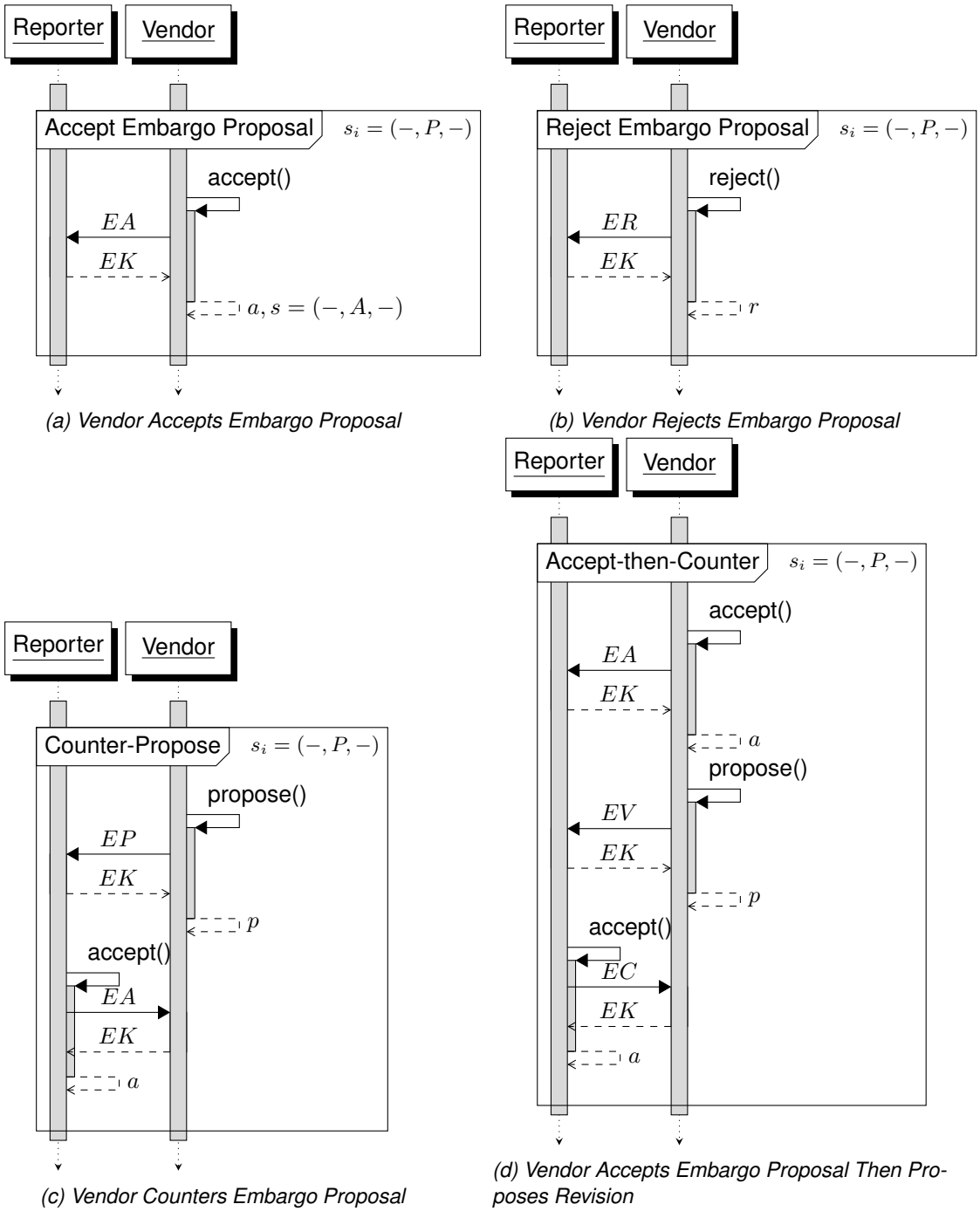
6.9.3 Vendor Sets Priority

Figure 6.4 offers two responses from a Vendor in the course of prioritizing a report. Figure 6.4a shows a Vendor accepting the report for further work (presumably to develop a patch) with an RA message. On the contrary, Figure 6.4b shows the Vendor deferring the report with an RD message. In both cases, the Reporter acknowledges the Vendor's messages with an RK message.

6.9.4 Coordination With a Coordinator

Figure 6.5 shows the process of a Reporter engaging a Coordinator, who, in turn, engages a Vendor. The process begins in Figure 6.5a with the Reporter sending a report along with an embargo proposal to the Coordinator (RS, EP). The Coordinator acknowledges receipt with an RK, EK response. After evaluating the proposed embargo, the Coordinator accepts it with an EA message. The Coordinator proceeds to validate and prioritize the report, emitting an RV and RA along the way.

Proceeding to Figure 6.5b, the Coordinator now acts as a proxy for the Reporter, notifying the Vendor and passing along the embargo information through an RS, EP message of its own. The Vendor accepts the existing embargo (EA) and proceeds to validate (RV) and prioritize (RA) the report. Relevant responses from the Vendor are passed through to the Reporter. Having accepted the report for further work, the Vendor continues with creating a fix for the reported vulnerability. When complete, the Vendor conveys their readiness to the Coordinator, who in turn passes this information along to the Reporter through the CF message.



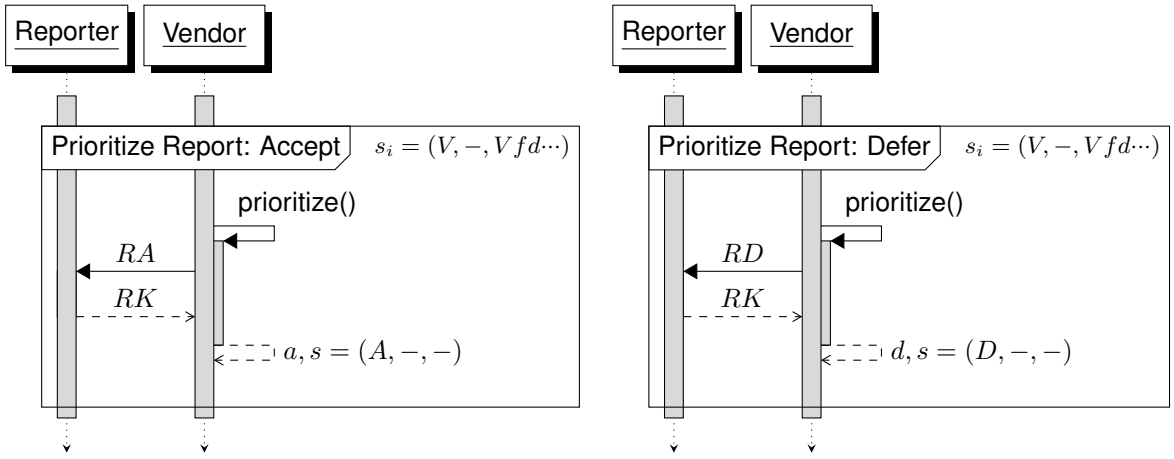
(a) Vendor Accepts Embargo Proposal

(b) Vendor Rejects Embargo Proposal

(c) Vendor Counters Embargo Proposal

(d) Vendor Accepts Embargo Proposal Then Proposes Revision

Figure 6.3: A Vendor Evaluates a Proposed Embargo and Responds in a Variety of Ways



(a) Vendor Accepts Report for Further Work *(b) Vendor Defers Further Work on Report*

Figure 6.4: A Vendor Prioritizes a Report

6.9.5 Embargo Teardown, Publish, and Close

Any Participant can initiate an embargo teardown. We happened to show the case where the Coordinator initiates it in Figure 6.6a, sending an embargo termination message (ET) to all parties in the case (Reporter and Vendor in this scenario). Recipients of the ET message acknowledge receipt and update their EM state accordingly.

Once the embargo has been exited, any Participant may now publish. In Figure 6.6b, we show the Vendor publishing first. They notify the Coordinator that they have published using a CP message to convey that information about the vulnerability is now public. The Coordinator relays this information to the Reporter. Both the Reporter and the Coordinator publish their own reports shortly thereafter.

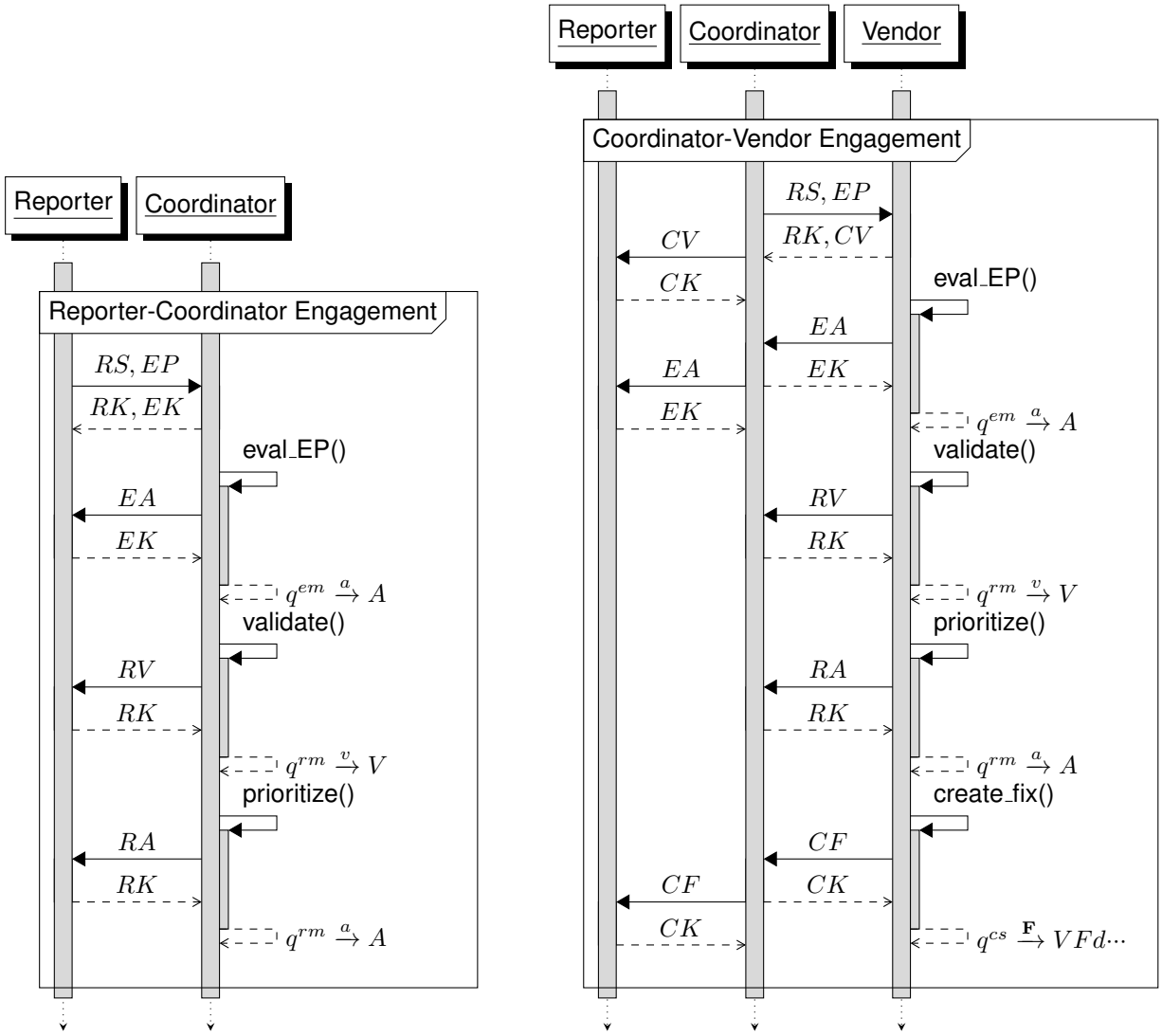
Having no further work to be done on the case, the Reporter closes their report and tells the Coordinator using an RC message in Figure 6.6c. This prompts the Coordinator to review their outstanding tasks and decide to initiate the closure of their own report. In turn, the Coordinator relays this to the Vendor, who also closes their report.

Note that for all three scenarios shown in Figure 6.6, there is no specific order in which Participants must act. We could just as easily have shown the Reporter initiating an embargo teardown because of a leaked media report or the Vendor exiting an embargo early because they had their fix ready sooner than expected.

Furthermore, our protocol only sets a discrete end to the embargo period, it intentionally does *not* address a publication schedule. Once the embargo has been exited, *any* Participant may publish at any time. Participants might choose to coordinate publication schedules more closely, but there is nothing in the protocol to require it. With the recognition that more concise publication scheduling might be needed in some situations, we revisit this concern as future work in §9.3.

Finally, report closure is a per-Participant choice. We chose to show a simple case where all Participants agreed at approximately the same time that there was nothing further to be done. This will not always be the case, nor is it necessary.

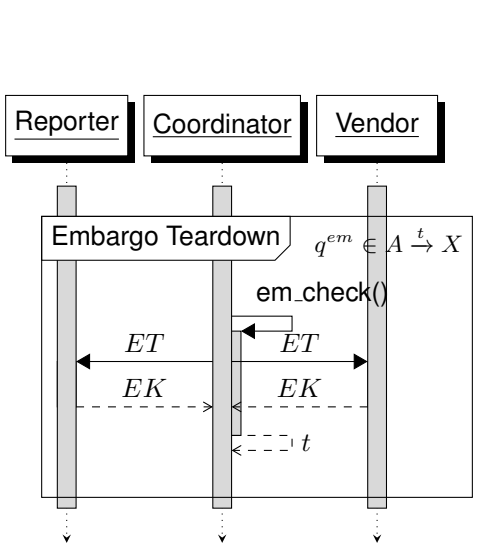
[6] "Yes-And" is a heuristic taken from improvisational theatre in which Participants are encouraged to agree with whatever their counterpart suggests and add to it rather than reject it outright. It serves as a good model for cooperation among parties who share an interest in a positive outcome.



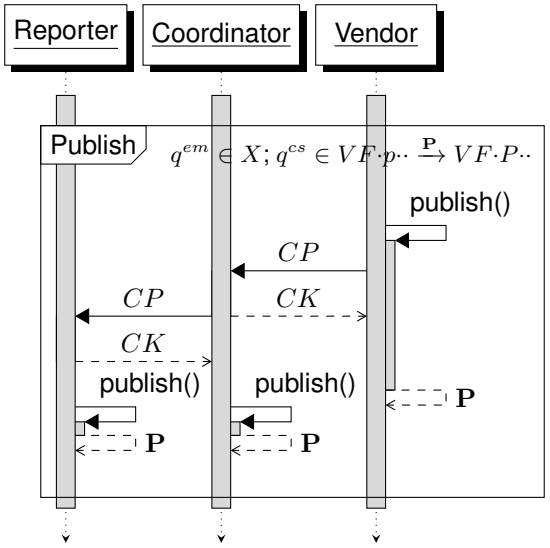
(a) A Reporter Engages a Coordinator *(b) A Coordinator Engages a Vendor*

Figure 6.5: A Reporter Engages a Coordinator, Who, in Turn, Engages a Vendor

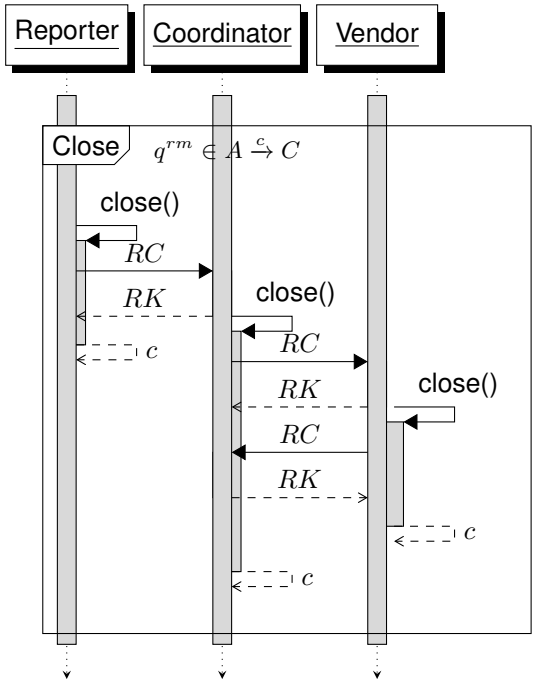
Both receiving parties (Coordinator and Vendor) perform their own validation and prioritization behaviors. The Vendor develops a fix and communicates readiness to the other Participants. (This process is continued in Figure 6.6.)



(a) A Coordinator Initiates an Embargo Teardown



(b) A Vendor Publishes First



(c) A Reporter Closes the Case

Figure 6.6: Embargo Teardown, Publication, and Report Closure

A Coordinator initiates embargo teardown. This is followed by Vendor publication, which, in turn, triggers both the Coordinator and Reporter to publish. Finally, the Reporter's report closure causes both the Coordinator and Vendor to choose to close their respective reports.

7 Modeling an MPCVD AI Using Behavior Trees

With the formal definition of our proposed MPCVD protocol behind us, we now turn our attention to reflect on one of many possible paths toward implementation. We find that Behavior Trees have a number of desirable properties when it comes to automating the kinds of complex behaviors our protocol demands.

Behavior Trees are a way of designing and programming hierarchical behaviors [6]. They originated in the computer gaming industry to develop realistic Artificial Intelligences (AIs) to control Non-Player Characters (NPCs) [23, 15] in games. More recently, Behavior Trees have been used in robotics to create adaptive behaviors using autonomous AI agents [25, 2]. Behavior Trees offer a high potential for automating complex tasks. Agent processes can be modeled as sets of behaviors (pre-conditions, actions, and post-conditions) and the logic that joins them. Behavior Trees offer a way to organize and describe agent behaviors in a straightforward, understandable way.

In this chapter, we use Behavior Trees as a method for describing MPCVD Participant activities and their interactions with the MPCVD protocol model from Chapter 6. These behaviors map approximately to the activities described in the *CVD Guide* (e.g., validate report, prioritize report, create fix, publish report, publish fix, deploy fix) [14, 13].

If Behavior Trees were merely a notational convention, they would already have been useful enough to include here to structure the high-level business logic of the MPCVD protocol. But they also offer a way to prototype software agents that reflect the activities of CVD Participants. Because Behavior Trees are inherently hierarchical, they are composable. Both conditions and actions can be composed into small task-oriented behaviors, which can, in turn, be composed to represent more complex agent behaviors. As a result, independent agents using Behavior Trees can be composed into multi-agent behaviors that achieve goals.

A Brief Introduction to Behavior Tree Notation. Behavior Trees consist of a hierarchy of nodes represented as a Directed Acyclic Graph (DAG). A Behavior Tree execution always begins at the root node, and execution is passed along the tree by *ticking* each child node according to the logic built into the tree. When *ticked*, each node does its job and returns one of three statuses: *Success*, *Failure*, or *Running*. A full introduction to Behavior Trees can be found in Colledanchise and Ögren's book *Behavior Trees in Robotics and AI: An Introduction* [6].

Node types include

Root has no parent nodes, has one or more child nodes, and can be of any of the control-flow types.

Condition does not change the state of the world, has no child nodes, and returns only *Success* or *Failure*.

Task has no child nodes; performs a task, which might change the state of the world; and returns *Success*, *Failure*, or *Running*.

Sequence → ticks each child node, returning the last *Success* or the first *Failure*, or *Running* if a child returns *Running*.

Fallback ? ticks each child node, returning the first *Success* or the last *Failure*, or *Running* if a child returns *Running*.

Loop ↻ repeatedly ticks child nodes until an exit condition is met.

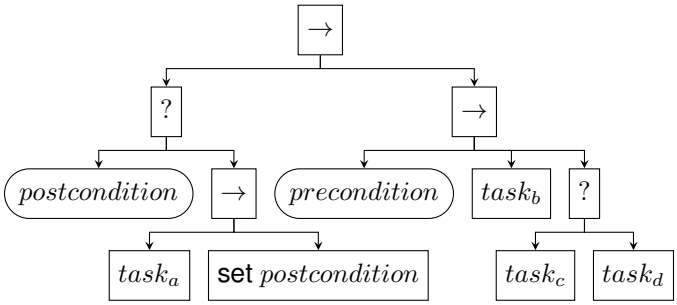


Figure 7.1: Basic Behavior Tree

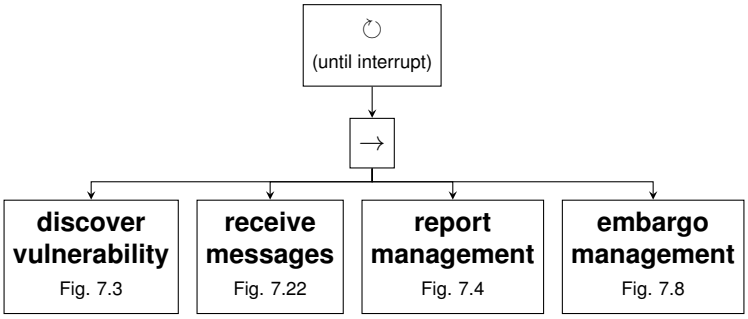


Figure 7.2: CVD Process Behavior Tree

Parallel ⇉ ticks all child nodes simultaneously, and returns *Success* when m of n children have returned *Success*.

A basic Behavior Tree is shown in Figure 7.1. In it, we see two motifs that come up through the remainder of the chapter. On the left side is a Fallback node (?), which short-circuits to *Success* when the *postcondition* is already met. Otherwise, some activity will occur in $task_a$ and, assuming that it succeeds, the *postcondition* is set. As a result, the fallback node ensures that *Success* means that the *postcondition* is met.

On the right side is a sequence (→) that hinges on a *precondition* being met prior to some set of actions being taken. Assuming the *precondition* is met, $task_b$ fires and, assuming it succeeds execution, proceeds to another fallback node. This fallback node represents a set of tasks in which one only needs to succeed for the fallback to return *Success*. If $task_c$ succeeds, then $task_d$ does not run.

Behavior Trees are composable—that is, a task node in one tree can be replaced with a more refined Behavior Tree in another. We leverage this feature throughout the remainder of this chapter to describe an agent model for an MPCVD Participant as a set of nested Behavior Trees that reflect the protocol described in the previous chapters.

7.1 CVD Behavior Tree

We begin at the root node of the CVD Behavior Tree shown in Figure 7.2. The root node is a simple loop that continues until an interrupt condition is met, representing the idea that the CVD practice is meant to be continuous. In other words, we are intentionally not specifying the interrupt condition.

The main sequence is comprised of four main tasks:

- *Discover vulnerability.* Although not all Participants have the ability or motive to dis-

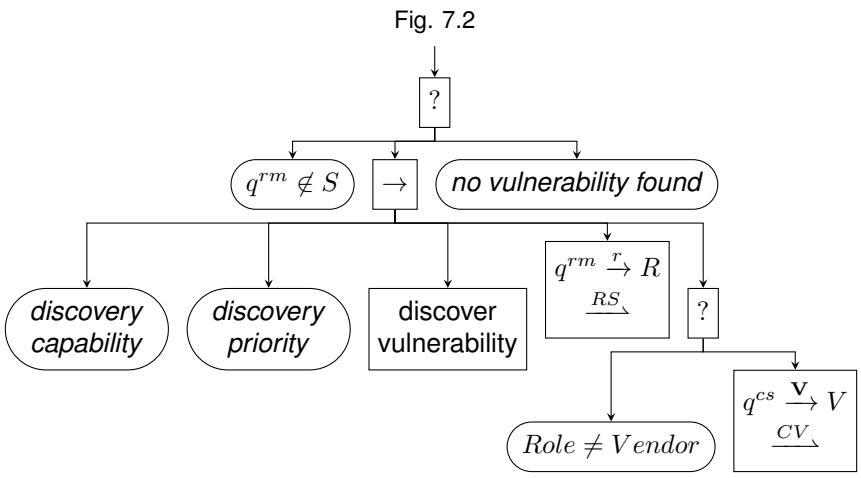


Figure 7.3: Discover Vulnerability Behavior Tree

cover vulnerabilities, we include it as a task here to call out its importance to the overall CVD process. We show in §7.2 that this task returns *Success* regardless of whether a vulnerability is found to allow execution to pass to the next task.

- *Receive messages.* All coordination in CVD between Participants is done through the exchange of messages, regardless of how those messages are conveyed, stored, or presented. The receive messages task represents the Participant's response to receiving the various messages defined in Chapter 6. Due to the degree of detail required to cover all the various message types, decomposition of this task node is deferred until §7.6 so we can cover the next two items first.
- *Report management.* This task embodies the RM process described in Chapter 2 as integrated into the MPCVD protocol of Chapter 6. The RM Behavior Tree is described in §7.3.
- *Embargo management.* Similarly, this task represents the EM process from Chapter 3 as integrated into the MPCVD protocol of Chapter 6. The EM Behavior Tree is decomposed in §7.4

A further breakdown of a number of CVD tasks that fall outside the scope of the formal MPCVD protocol of Chapter 6 can be found in §7.5. In that section, we examine a number of behaviors that Participants may include as part of the work they do for reports in the *Accepted* RM state ($q^{rm} \in A$).

Behaviors and state changes resulting from changes to the CS model are scattered throughout the other Behavior Trees where relevant.

7.2 Vulnerability Discovery Behavior

CVD is built on the idea that vulnerabilities exist to be found. There are two ways for a CVD Participant to find out about a vulnerability. Either they discover it themselves, or they hear about it from someone else. The discovery behavior is modeled by the Discover Vulnerability Behavior Tree shown in Figure 7.3. External reports are covered in §7.6.1.

The goal of the Discover Vulnerability Behavior is for the Participant to end up outside of the *Start* state of the Report Management process ($q^{rm} \notin S$). Assuming this has not already occurred, the discovery sequence is followed. If the Participant has both the means and the motive to find a vulnerability, they might discover it themselves. Should this succeed, the branch

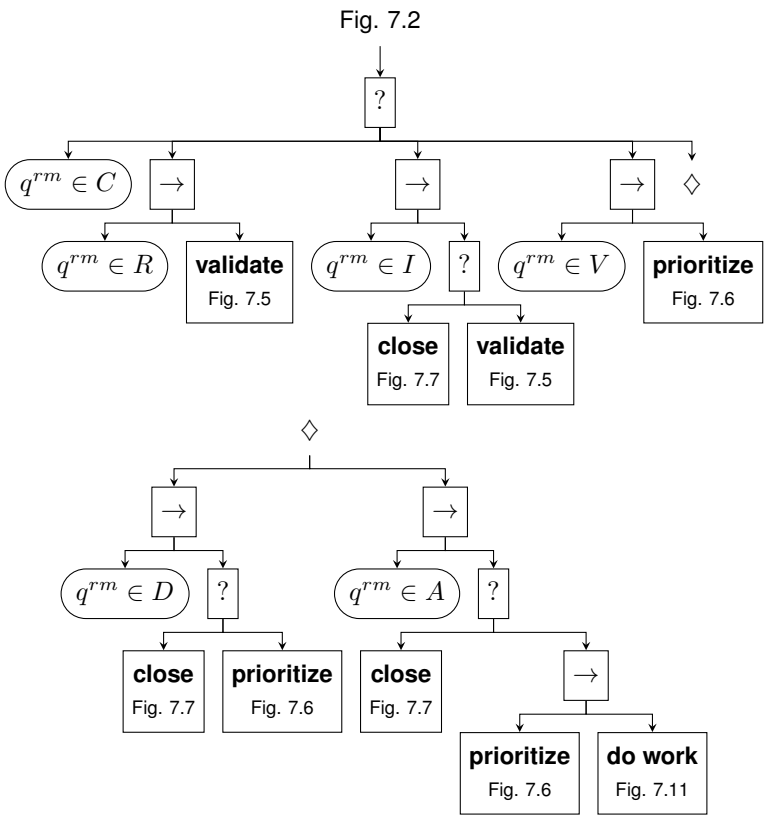


Figure 7.4: Report Management Behavior Tree

sets $q^{rm} \in S \xrightarrow{r} R$ and returns *Success*. We also show a report submission (RS) message being emitted as a reminder that even internally discovered vulnerabilities can trigger the CVD process—although, at the point of discovery, the Finder is the only Participant, so the RS message in this situation might be an internal message within the Finder organization (at most).

Should no discovery occur, the branch returns *Success* so that the parent process in Figure 7.1 can proceed to receive messages from others. Because of the amount of detail necessary to describe the *receive messages* behavior, we defer it to §7.6. Before we proceed, it is sufficient to know that a new report arriving in the *receive messages* behavior sets $q^{rm} \in S \xrightarrow{r} R$ and returns *Success*.

7.3 Report Management Behavior Tree

A Behavior Tree for the Report Management model is shown in Figure 7.4. The Report Management process is represented by a Fallback node. Note that we assume that completing the process will require multiple *ticks* of the Behavior Tree since each tick can complete, at most, only one branch.

The first check is to see whether the case is already *Closed* ($q^{rm} \in C$). If that check succeeds, the branch returns *Success*, and we're done. If it doesn't, we move on to the next branch, which addresses reports in the *Received* state ($q^{rm} \in R$).

The only action to be taken from $q^{rm} \in R$ is to validate the report. We address report validation in §7.3.1, but, for now, it is sufficient to say that the validate report behavior returns

Fig. 7.4

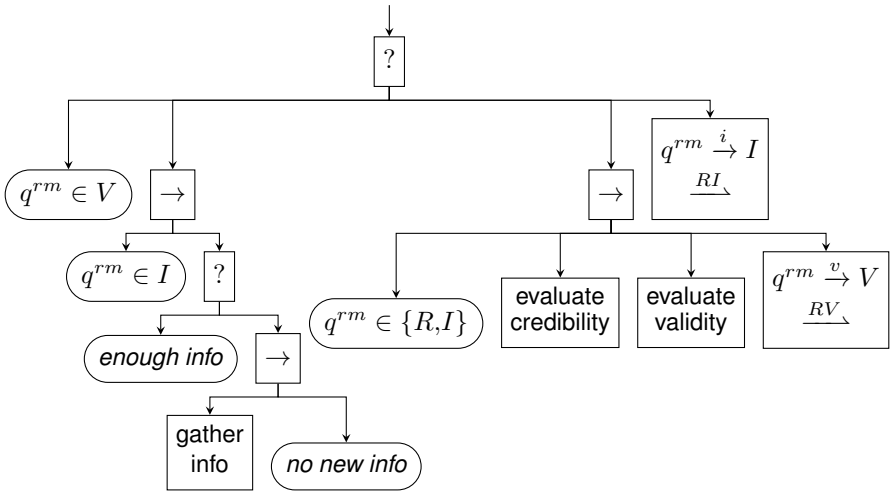


Figure 7.5: Validate Report Behavior Tree

Success after moving the report to either *Valid* ($q^{rm} \xrightarrow{v} V$) or *Invalid* ($q^{rm} \xrightarrow{i} I$).

The next branch covers reports in the *Invalid* state ($q^{rm} \in I$). Here we have two options: either close the report (move to $q^{rm} \xrightarrow{c} C$, §7.3.3), or retry the validation.

For reports that have reached the *Valid* state ($q^{rm} \in V$), our only action is to prioritize the report. Report prioritization is addressed in detail in §7.3.2 but returns *Success* after moving the report to either *Accepted* ($q^{rm} \xrightarrow{a} A$) or *Deferred* ($q^{rm} \xrightarrow{d} D$).

Directing our attention to the lower ($\diamondsuit$) tier of Figure 7.4, we reach behaviors associated with reports that have been both validated and prioritized. *Deferred* reports ($q^{rm} \in D$) can be *Closed* or have their priority reevaluated, but otherwise are not expected to receive additional work.

Similarly, *Accepted* reports ($q^{rm} \in A$) can also be *Closed* or have their priority reevaluated. However, they are also expected to receive more effort—the *do work* task node, which we explore further in §7.5.

We are taking advantage of the composability of Behavior Trees to simplify the presentation. Behaviors that appear in multiple places can be represented as their own trees. We explore the most relevant of these subtrees in the next few subsections.

7.3.1 Report Validation Behavior

A Report Validation Behavior Tree is shown in Figure 7.5. To begin with, if the report is already *Valid*, no further action is needed from this behavior.

When the report has already been designated as *Invalid*, the necessary actions depend on whether further information is necessary, or not. If the current information available in the report is sufficient, no further action is necessary and the entire behavior returns *Success*. However, a previous validation pass might have left some indicator that more information was needed. In that case, execution proceeds to the sequence in which the *gather info* task runs. If nothing new is found, the entire branch returns *Success*, and the report remains *Invalid*. If new information *is* found, though, the branch fails, driving execution over to the main validation sequence.

Fig. 7.4

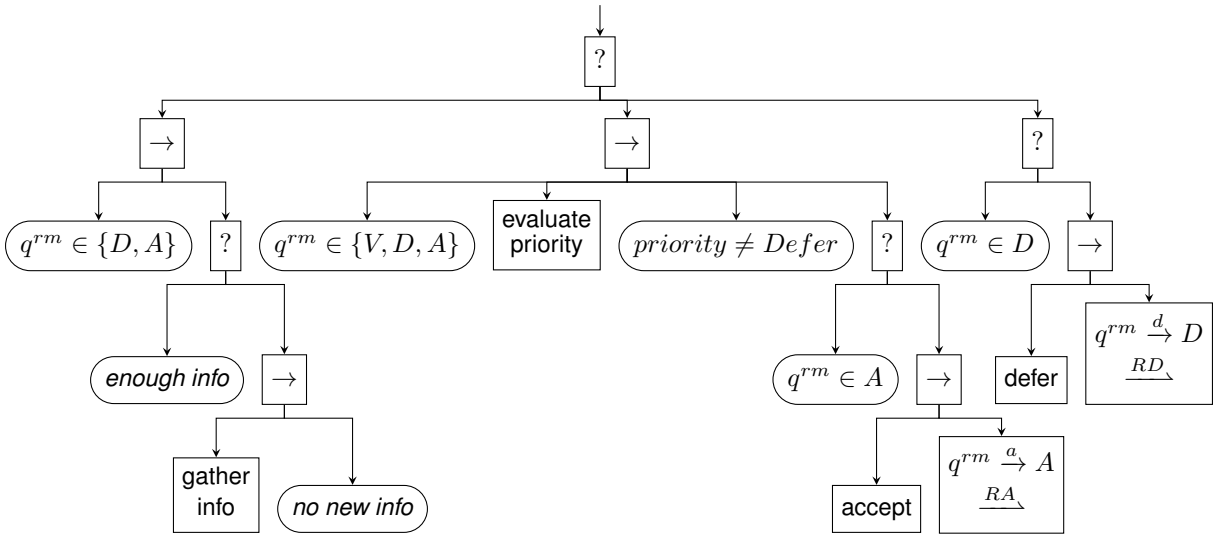


Figure 7.6: Prioritize Report Behavior Tree

The main validation sequence follows when none of the above conditions have been met. In other words, the validation sequence is triggered when the report is in *Received* and its validity has never been evaluated or when the report was originally determined to be *Invalid* but new information is available to prompt reconsideration. The validation process shown here is comprised of two main steps: a credibility check followed by a validity check as outlined in §2.1.1.2.

As a reminder, a report might be in one of three categories: (a) neither credible nor valid, (b) credible but invalid, or (c) both credible and valid. Assuming the report passes both the credibility and validity checks, it is deemed *Valid*, moved to $q^{rm} \xrightarrow{v} V$, and an RV message is emitted.

Should either check fail, the validation sequence fails, the report is deemed *Invalid* and moves (or remains in) $q^{rm} \in I$. In that case, an RI message is sent when appropriate to update other Participants on the corresponding state change.

7.3.2 Report Prioritization Behavior

The Report Prioritization Behavior Tree is shown in Figure 7.6. It bears some structural similarity to the Report Validation Behavior Tree just described: An initial post-condition check falls back to the main process leading toward *accept*, which, in turn, falls back to the deferral process. If the report is already in either the *Accepted* or *Deferred* states and no new information is available to prompt a change, the behavior ends.

Failing that, we enter the main prioritization sequence. The preconditions of the main sequence are that either the report has not yet been prioritized out of the *Valid* state ($q^{rm} \in V$) or new information has been made available to a report in either $q^{rm} \in \{D, A\}$ to trigger a reevaluation.

Assuming the preconditions are met, the report priority is evaluated. For example, a Participant using SSVC [29] could insert that process here. The evaluation task is expected to always set the report priority. The subsequent check returns *Failure* on a defer priority or *Success* on any non-deferral priority. On *Success*, an *accept* task is included as a placeholder for any intake process that a Participant might have for *Accepted* reports. Assuming that it suc-

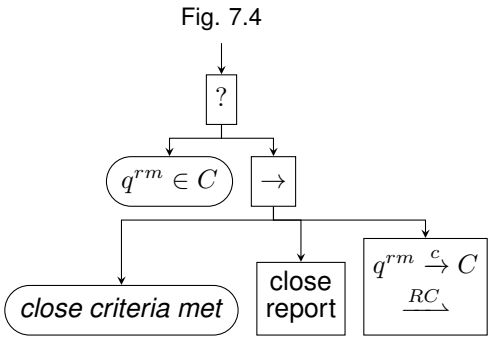


Figure 7.7: Close Report Behavior Tree

ceeds, the report is explicitly moved to the *Accepted* ($q^{rm} \xrightarrow{a} A$) state, and an RA message is emitted.

Should any item in the main sequence fail, the case is deferred, its state set to $q^{rm} \xrightarrow{d} D$, and an RD message is emitted accordingly. Similarly, a *defer* task is included as a callback placeholder.

7.3.3 Report Closure Behavior

The Report Closure Behavior Tree is shown in Figure 7.7. As usual, the post-condition is checked before proceeding. If the case is already *Closed* ($q^{rm} \in C$), we're done. Otherwise, the main close sequence begins with a check for whether the report closure criteria have been met. Report closure criteria are Participant specific and are, therefore, out of scope for this report. Nevertheless, once those closure criteria are met, the actual *close report* task is activated (e.g., an `OnClose` callback). The sequence ends with setting the state to *Closed* ($q^{rm} \xrightarrow{c} C$) and emitting an RC message.

7.4 Embargo Management Behavior Tree

The Embargo Management Behavior Tree is shown in Figure 7.8. It follows the state transition function in Table 6.4. Recall that the EM process begins in the $q^{em} \in N$ state and ends in one of two states:

- in the *eXited* ($q^{em} \in X$) state after having established an *Active* embargo, or
- in the *None* ($q^{em} \in N$) state after having exhausted all attempts to reach an agreement

The tree starts with a check to see whether no report has arrived or whether the report has already *Closed* ($q^{rm} \in \{S,C\}$). If either of these conditions is met, no further effort is needed, and the tree succeeds. Next, the tree checks whether the embargo has already *eXited* ($q^{em} \in X$). If it has, that leads the tree to succeed. Failing that, the treat checks to see if the case has moved outside the "habitable zone" for embargoes. The $q^{cs} \notin \cdots pxa$ condition is true when attacks have been observed, an exploit has been made public, or information about the vulnerability has been made public. If one of those conditions is met and the embargo state is *None* ($q^{em} \in N$), the check returns *Success*, and the tree terminates, consistent with §3.2.4.

Otherwise, we continue through each remaining EM state. When there is no embargo and there are no outstanding proposals ($q^{em} \in N$), the only options are to either stop trying or propose a new embargo. The decision to stop trying to achieve an embargo is left to individual Participants, although we did provide some relevant guidance in §3.2.5.

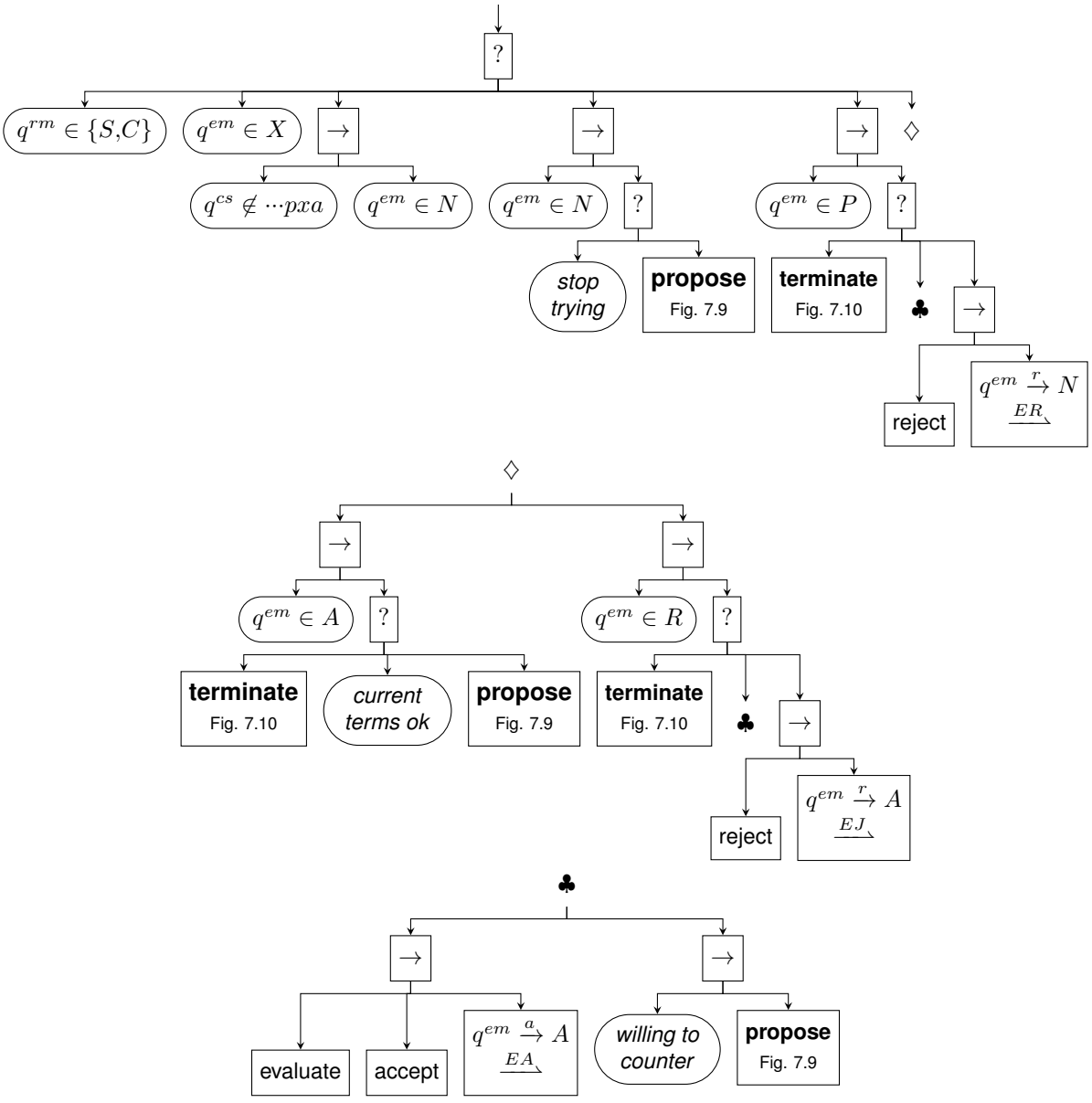


Figure 7.8: Embargo Management Behavior Tree

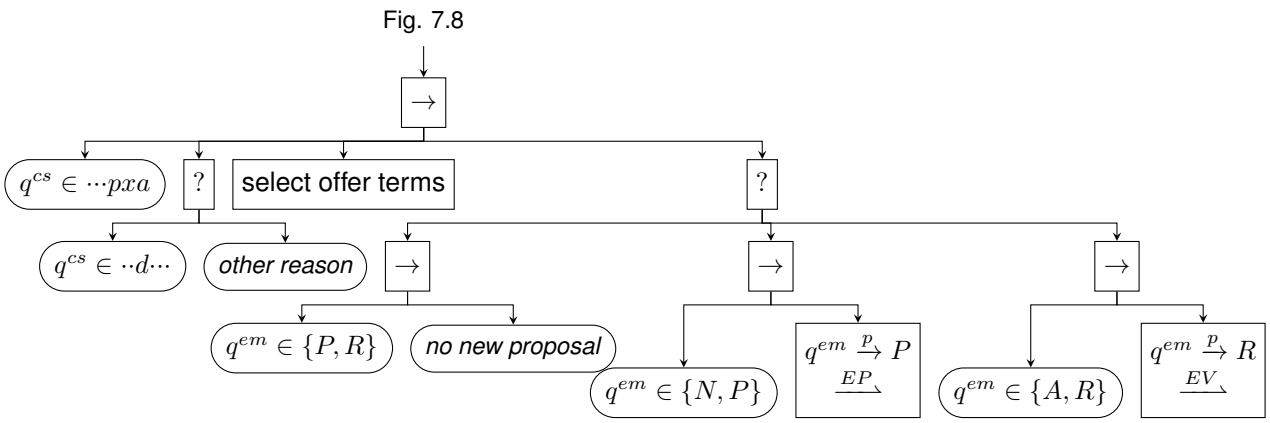


Figure 7.9: Propose Embargo Behavior Tree

When there is an outstanding embargo proposal ($q^{em} \in P$), we first attempt the terminate task. We shall see in §7.4.2 that this task returns *Success* if there is a reason for $q^{em} \in P \xrightarrow{r} N$.

Otherwise we proceed to the bottom (♣) tier of Figure 7.8 to evaluate and possibly accept the proposal. Acceptance leads to an EM state transition to $q^{em} \in A$ and emission of an EA message.

On the other hand, the proposed terms may not be acceptable. In this case, the Participant might be willing to offer a counterproposal. The counterproposal is covered by the propose behavior described in §7.4.1.

Assuming neither of these succeeds, we return to the top tier of Figure 7.8 and reject the proposal, returning to $q^{em} \in N$ and emitting a corresponding ER message.

This brings us to the middle (♢) tier of Figure 7.8. The process within the *Active* ($q^{em} \in A$) state is similarly straightforward. If there is reason to terminate the embargo, do so. Otherwise, either the current embargo terms are acceptable, or a new embargo should be proposed.

Finally, we handle the *Revise* EM state ($q^{em} \in R$). The structure of this branch mirrors that of the *Proposed* state discussed above. Again, we check to see if there is cause to terminate doing so, if needed. If termination is not indicated, we proceed once again to the bottom (♣) tier to evaluate the proposed revision, either accepting or countering the proposal. When neither of these succeed, the revision is rejected and the EM state returns to $q^{em} \in A$ with the original embargo terms intact. An EJ message conveys this information to the other Participants.

7.4.1 Propose Embargo Behavior

The Propose Embargo Behavior Tree is shown in Figure 7.9. It consists of a sequence that begins with a check for embargo viability as outlined in §3.2.4. Once the checks succeed, it proceeds to selecting embargo terms to propose. Implementations of this task might simply draw from a default policy, as in §3.2.6, or it might be a case-specific decision made by a Participant. Embargo terms can be proposed from any of the non-*eXited* states ($q^{em} \in \{N, P, A, R\}$). If a new or revised embargo has already been proposed, the tree then checks whether a counterproposal is desired. Assuming it is not, no proposal is made, and the behavior succeeds. Otherwise, proposals from state $q^{em} \in N$ emit EP and transition $q^{em} \xrightarrow{p} P$, whereas those from $q^{em} \in A$ emit EV and move to $q^{em} \xrightarrow{p} R$. Proposals from states $q^{em} \in P$ or $q^{em} \in R$ represent counterproposals and, therefore, do not change the EM state. They do,

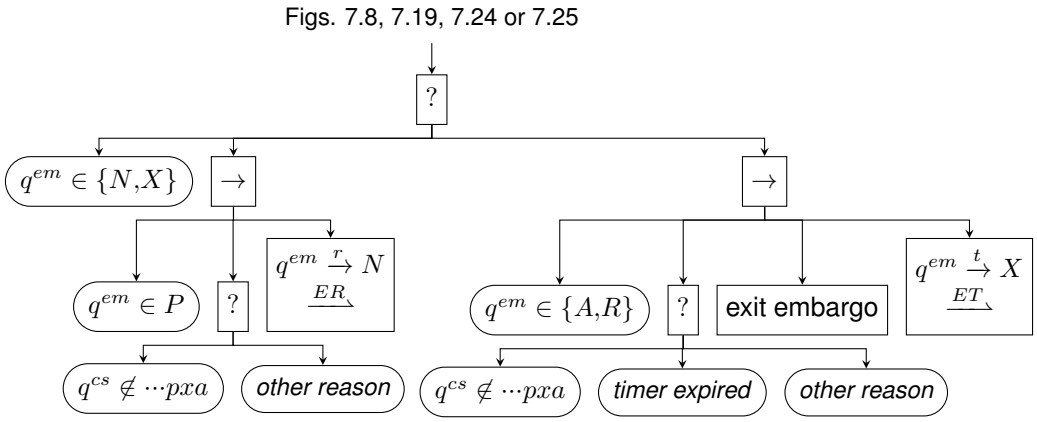


Figure 7.10: Terminate Embargo Behavior Tree

however, emit EP or EV messages as appropriate.

7.4.2 Terminate Embargo Behavior

The Terminate Embargo Behavior Tree is shown in Figure 7.10. It consists of two major behaviors depending on whether an embargo has been established or not.

If the EM state is *None* or *eXited*, ($q^{em} \in \{N,X\}$), the tree succeeds immediately. The next node handles the scenario where no embargo has been established. The behavior descends into a sequence that checks whether we are in *Propose* ($q^{em} \in P$). If we are, we check to see if there is a reason to exit the embargo negotiation process. One such reason is that the case state is outside the embargo "habitable zone," but there may be others that we leave unspecified. If any reason is found, then the proposal is rejected, the state returns to *None*, and an ER message is sent.

Should that branch fail, we still need to handle the situation where an embargo has already been established. Following a confirmation that we are in either *Active* or *Revise*, we again look for reasons to exit, this time adding the possibility of timer expiration to the conditions explicitly called out. Terminating an existing embargo might have some other teardown procedures to be completed, which we represent as the *exit embargo* task. Finally, the EM state is updated to *eXited* and an ET message is emitted.

The Terminate Embargo Behavior Tree appears in multiple locations in the larger tree. We will encounter it again as a possible response to evidence collected via threat monitoring (§7.5.5) as well as in response to certain CS or EM messages in states when an embargo is no longer viable (§7.25 and §7.24, respectively).

7.5 Do Work Behavior

Although it is not directly addressed by the formal MPCVD protocol defined in Chapter 6, the *do work* node of the RM Behavior Tree in Figure 7.4 and §7.3 is where much of the CVD effort happens. As a result, it is worth spending some time reviewing what some of the underlying work actually entails.

In this section, we will expand on the set of behaviors shown in Figure 7.11. Specifically, we will cover the following tasks, each in its own subsection.

- Deployment
- Developing a fix

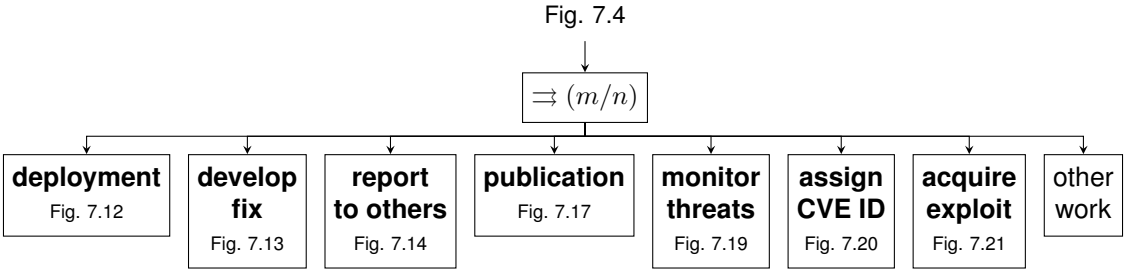


Figure 7.11: Do Work Behavior Tree

- Reporting to others
- Publication
- Monitoring threats
- Assigning CVE IDs
- Acquiring exploits

The *other work* task is included as a placeholder for any Participant-specific tasks not represented here.

Note that Figure 7.11 models this behavior as a parallel Behavior Tree node, although we intentionally do not specify any *Success* criteria regarding what fraction of its children must succeed. Decisions about which (and how many) of the following tasks are necessary for a Participant to complete work on their *Accepted* CVD cases are left to the discretion of individual Participants.

7.5.1 Deployment Behavior

The Deployment Behavior Tree is shown in Figure 7.12. The goal of this behavior is either for the case to reach the $q^{cs} \in D$ state or for the Participant to be comfortable with remaining in a *Deferred* deployment state.

Assuming neither of these conditions has been met, the main deployment sequence falls to the Developer role. It consists of two subprocesses: prioritize deployment and deploy. The prioritize deployment behavior is shown in the fallback node in the center of Figure 7.12. The subgoal is for the deployment priority to be established, as indicated by the Deployer's RM state $q^{rm} \in \{D, A\}$. For example, a Deployer might use the SSVC Deployer Tree [29] to decide whether (and when) to deploy a fix or mitigation. If the deployment priority evaluation indicates further action is needed, the report management state is set to $q^{rm} \in A$. An RA message is emitted, and the overall prioritization behavior succeeds. Otherwise, when the deployment is *Deferred*, it results in a transition to state $q^{rm} \in D$ and emission of an RD message.

The deploy behavior is shown in the diamond tier ($\diamondsuit$) of Figure 7.12. It short-circuits to *Success* if either the deployment is *Deferred* or has already occurred. The main sequence can fire in two cases:

1. when the Deployer is also the Vendor and a fix is ready ($q^{cs} \in F$)
2. when the Deployer is not the Vendor but the fix is both ready and public ($q^{cs} \in P$ and $q^{cs} \in F$)

Assuming either of these conditions is met, the deploy fix task can run, the case status is updated to $q^{cs} \in D$, and CD emits on *Success*. Should the deployment sequence fail for any

Fig. 7.11

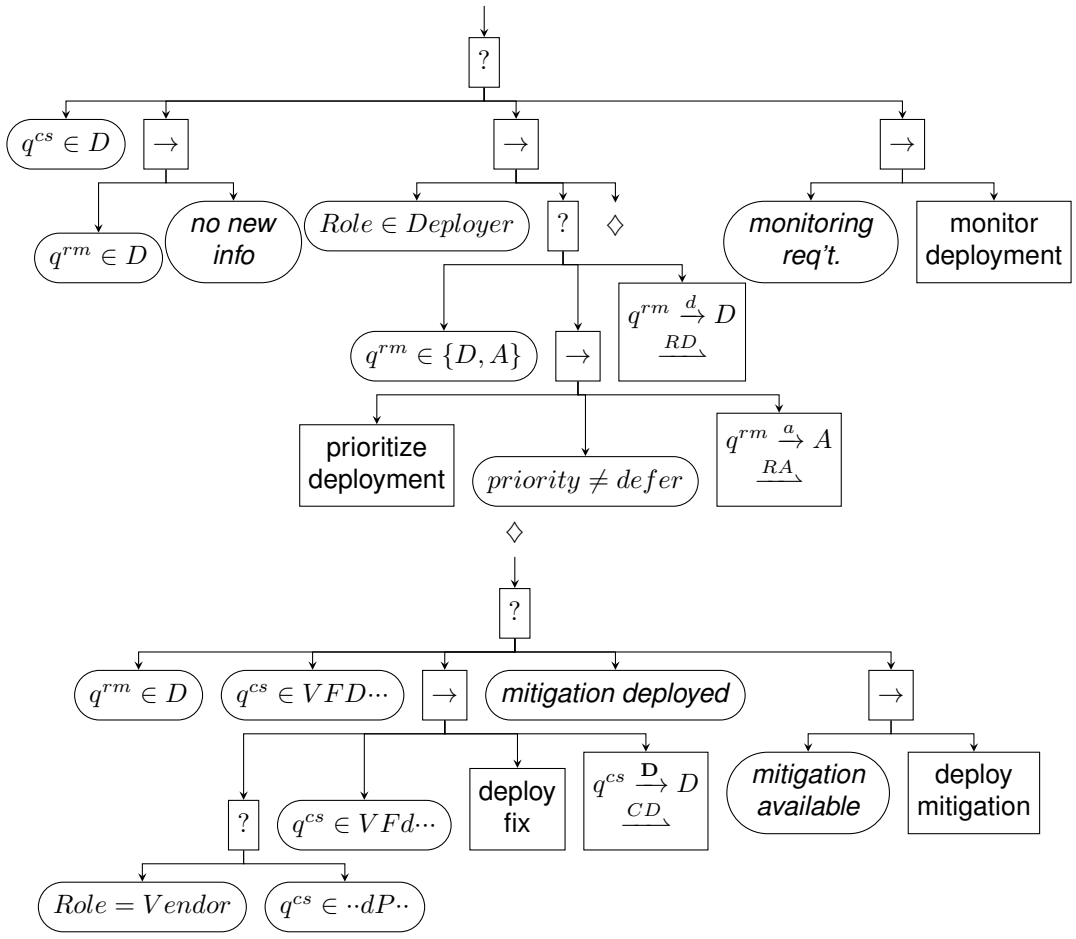


Figure 7.12: Deployment Behavior Tree

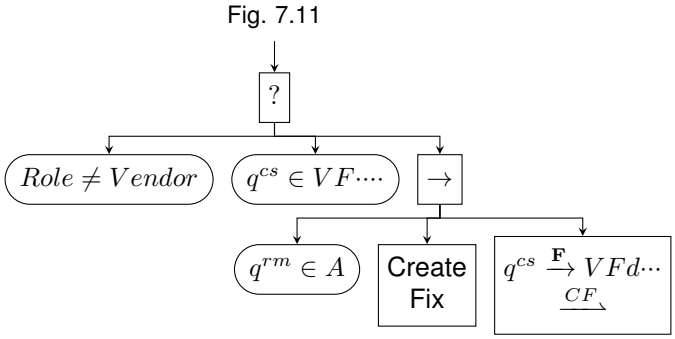


Figure 7.13: Fix Development Behavior Tree

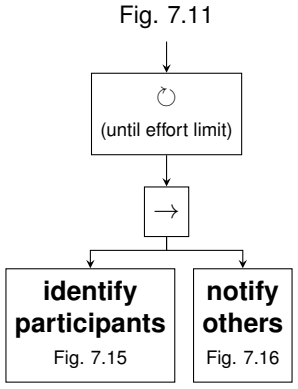


Figure 7.14: Reporting Behavior Tree

reason, a fallback is possible if undeployed mitigations are available.

Finally, returning to the top part of the tree, Participants might choose to monitor the deployment process should they have the need to.

7.5.2 Fix Development Behavior

The Fix Development Behavior Tree is shown in Figure 7.13. Fix development is relegated to the Vendor role, so Non-Vendors just return *Success* since they have nothing further to do. For Vendors, if a fix is ready (i.e., the case is in $q^{cs} \in VF\cdots$), the tree returns *Success*. Otherwise, engaged Vendors ($q^{rm} \in A$) can create fixes, set $q^{cs} \in Vfd\cdots \xrightarrow{\mathbf{F}} VFd\cdots$ and emit CF upon completion.

7.5.3 Reporting Behavior

The *CERT Guide to Coordinated Vulnerability Disclosure* describes the reporting phase as the process of identifying parties that need to be informed about the vulnerability and then notifying them [14]. Reporting only works if the intended recipient has the ability to receive reports, as outlined in §7.2.

The Reporting Behavior Tree is shown in Figure 7.14. The tree describes a Participant that performs reporting until either their effort limit is met, or they run out of Participants to notify.

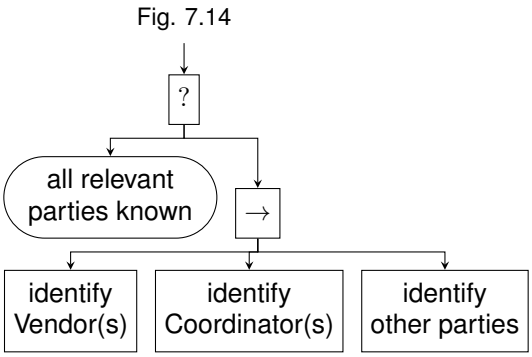


Figure 7.15: Identify Participants Behavior Tree

7.5.3.1 Identify Participants Behavior

The Identify Participants Behavior Tree, shown in Figure 7.15, ends when all relevant parties have been identified. Until that condition is met, the Participant can proceed with identifying Vendors, Coordinators, or other parties relevant to the coordination of the case. Note that we are intentionally avoiding setting any requirements about *who* is relevant to a case since we leave that to each Participant's judgment.

7.5.3.2 Notify Others Behavior

The Notify Others Behavior Tree is shown in Figure 7.16. Its goal is for all intended recipients to receive the report, thereby reaching the $q^{rm} \in R$ state. Each pass through this part of the tree chooses a Participant from a list of eligible recipients constructed in the Identify Participants Behavior. The method for choosing the recipient is left unspecified since Participants can prioritize recipients how they see fit.

The process proceeds to clean up the eligible recipients list when either the recipient is already believed to be in $q^{rm} \in R$ or if the effort expended in trying to reach the recipient has exceeded the Participant's limit. Such limits are entirely left to the discretion of each Participant. If the chosen recipient is pruned by this branch, the branch returns *Success*.

If the chosen recipient was not pruned, then the cleanup branch fails and execution transfers to the second branch to notify the recipient. The first step in the notification branch is a check for an existing embargo. If the embargo management state is one of $q^{em} \in \{N, P, X\}$, there is no active embargo, and the Participant can proceed with notification. Otherwise, in the case of an already active embargo (i.e., $q^{em} \in \{A, R\}$), there is an additional check to ensure that the potential recipient's policy is compatible with any existing embargo. This check allows for a reporting Participant to skip a recipient if they are likely to cause premature termination of an extant embargo.

Once at least one of these checks is passed, the notification sequence proceeds through finding the recipient's contact information, sending the report, and updating the Participant's knowledge of the recipient's report management state.

Fig. 7.14

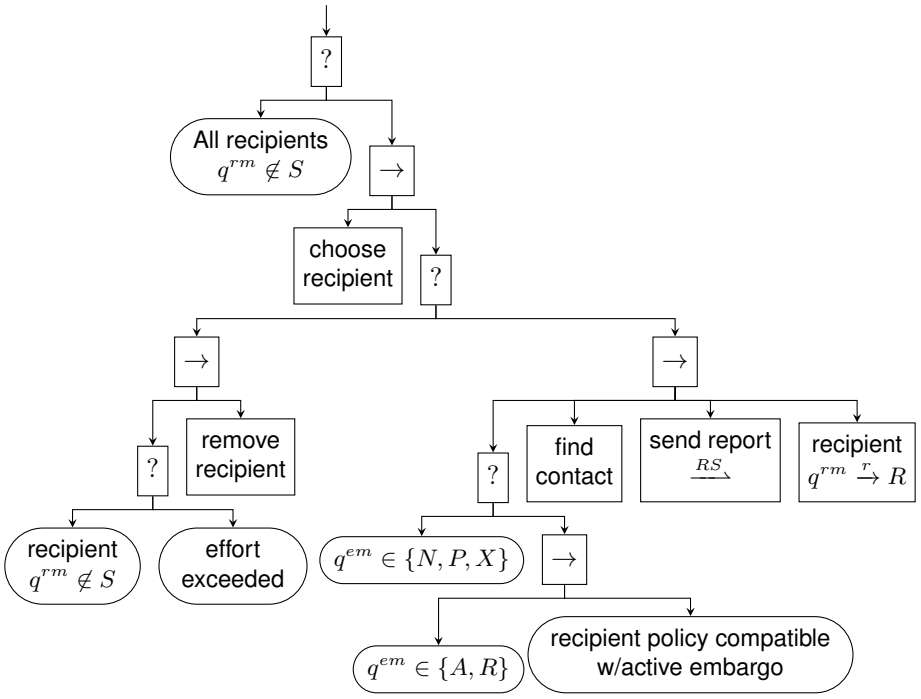


Figure 7.16: Notify Others Behavior Tree

7.5.4 Publication Behavior

The Publication Behavior Tree is shown in Figure 7.17. It begins by ensuring that the Participant knows what they intend to publish, followed by a check to see if that publication has been achieved. Assuming that work remains to be done, the main publish sequence commences on the right-hand branch.

The process begins with preparation for publication, described in §7.5.4.1, followed by a pre-publication embargo check. This behavior is a simple check to ensure that no embargo remains active prior to publication. Note that the embargo management process may result in early termination of an existing embargo if the Participant has sufficient cause to do so. (See the detailed description of the EM behavior in §7.4.)

Once these subprocesses complete, the publish task fires, the case state is updated to $q^{cs} \in P$, and a CP message emits.

7.5.4.1 Prepare Publication Behavior

The Prepare Publication Behavior Tree is shown in Figure 7.18. There are separate branches for publishing exploits, fixes, and reports. The publish exploit branch succeeds if either no exploit publication is intended, if it is intended and ready, or if it can be acquired and prepared for publication. The publish fix branch succeeds if the Participant does not intend to publish a fix (e.g., if they are not the Vendor), if a fix is ready, or if it can be developed and prepared for publication. The publish report branch is the simplest and succeeds if either no publication is intended or if the report is ready to go.

Once all three branches have completed, the behavior returns *Success*.

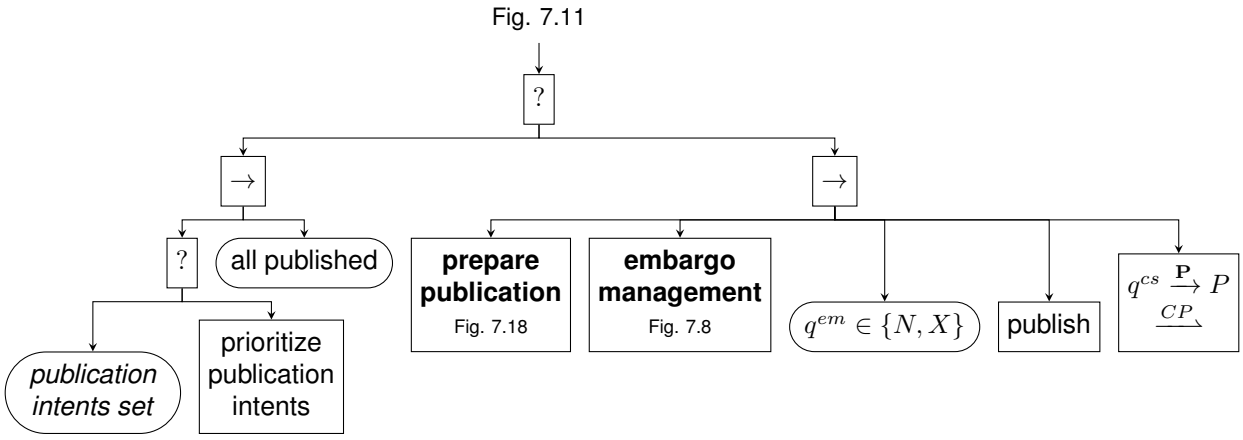


Figure 7.17: Publication Behavior Tree

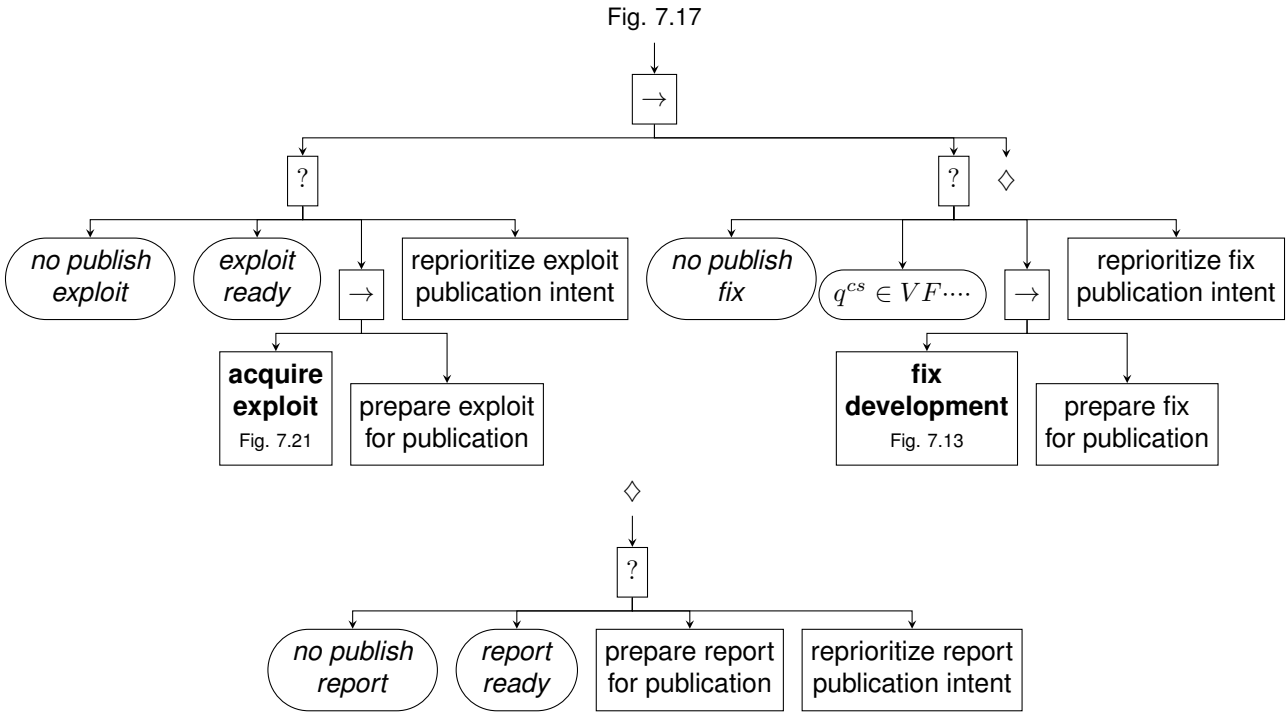


Figure 7.18: Prepare Publication Behavior Tree

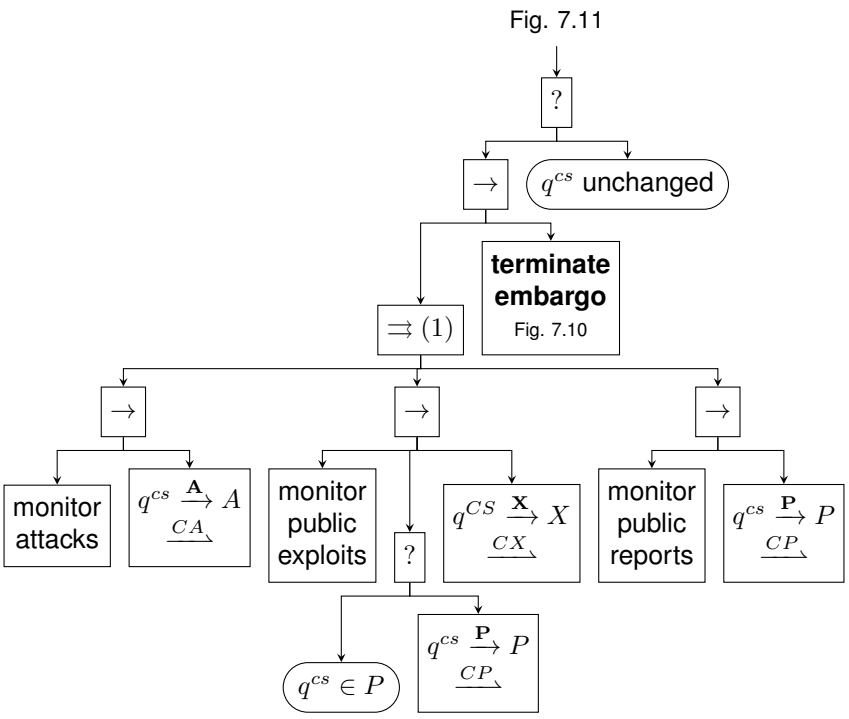


Figure 7.19: Monitor Threats Behavior Tree

7.5.5 Monitor Threats Behavior

The Monitor Threats Behavior Tree is shown in Figure 7.19. For our purposes, monitoring consists of a set of parallel tasks, any one of which can lead to embargo termination. The three conditions of interest are taken straight from the embargo exit criteria:

- If attacks are observed, the $q^{cs} \xrightarrow{\mathbf{A}} A$ transition occurs, and a CA message is emitted.
- If a public exploit is observed, the $q^{cs} \xrightarrow{\mathbf{X}} X$ transition occurs, and a CX message is emitted. In the special case where the exploit is made public prior to the vulnerability itself being made public,[7] there is an additional $q^{cs} \xrightarrow{\mathbf{P}} P$ transition and CP emission.
- Finally, if the vulnerability information has been made public, then the $q^{cs} \xrightarrow{\mathbf{P}} P$ and emits CP.

In the event that one or more of these events is detected, the terminate embargo behavior is triggered.

There are many other good reasons to monitor and maintain awareness of cybersecurity threats. The behavior shown here is intended as a minimal set of things that CVD Participants should watch out for in the course of performing their CVD practice.

[7] Corresponding to a Type 3 Zero Day Exploit as defined in §6.5.1 of [10]

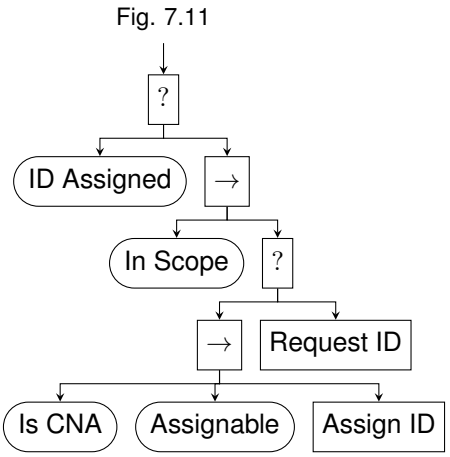


Figure 7.20: CVE Assignment Behavior Tree

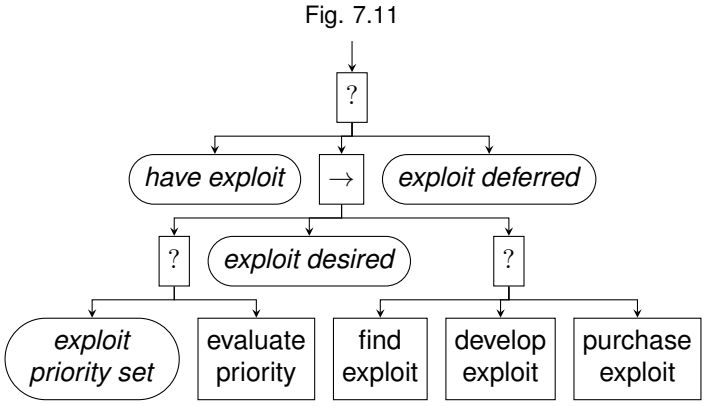


Figure 7.21: Acquire Exploit Behavior Tree

7.5.6 CVE ID Assignment Behavior

Many CVD practitioners want to assign identifiers to the vulnerabilities they coordinate. The most common of these is a Common Vulnerabilities and Exposures (CVE) ID, so we provide an example CVE ID Assignment Behavior Tree, shown in Figure 7.20. While this tree is constructed around the CVE ID assignment process, it could be easily adapted to any other identifier process as well.

The goal is to end with an ID assigned. If that has not yet happened, the first check is whether the vulnerability is in scope for an ID assignment. If it is, the Participant might be able to assign IDs directly, assuming they are a CVE Numbering Authority (CNA) and the vulnerability meets the criteria for assigning IDs.

Otherwise, if the Participant is not a CNA, they will have to request an ID from a CNA.

Should both assignment branches fail, the behavior fails. Otherwise, as long as one of them succeeds, the behavior succeeds.

7.5.7 Acquire Exploit Behavior

Some Vendors or other CVD Participants might require a proof-of-concept exploit to accompany an incoming report for it to pass their validation checks. To that end, an Acquire Ex-

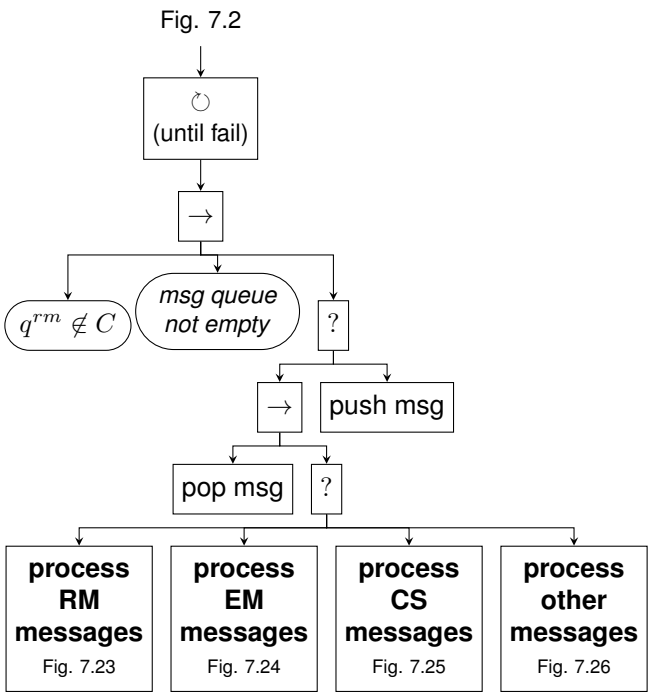


Figure 7.22: Receive Messages Behavior Tree

ploit Behavior Tree is shown in Figure 7.21. The goal of this behavior is for the Participant to be in possession of an exploit.

If the Participant does not already have one, the main acquisition sequence is triggered. The sequence begins by ensuring that the exploit acquisition activity has sufficient priority to continue. If it does, the Participant has one of three options to choose from: they can find one somewhere else, develop it themselves, or pay someone for the privilege.

The overall behavior returns *Success* when either an exploit is acquired or when one is not desired and is therefore deferred. It can fail in the scenario where an exploit is desired but not acquired.

7.6 Receiving and Processing Messages Behavior

Now we return to the CVD Behavior Tree in Figure 7.2 to pick up the last unexplored branch, Receive Messages. The Receive Messages Behavior Tree is shown in Figure 7.22. It is a loop that continues until all currently received messages have been processed. Each iteration checks for unprocessed messages and handles them.

First, we encounter a case closure check. We assume that messages to existing cases will have a case ID associated with all messages about that case and that new report submissions will not have a case ID assigned yet, implying they are in the RM *Start* state ($q^{rm} \in S$). Therefore, new reports will pass this check every time. However, messages received on an already *Closed* case will short-circuit here and take no further action.

Assuming the message is regarding a new or unclosed case and the message has not yet been processed, each message type is handled by a process-specific Behavior Tree, which we discuss in the following sections. Although each process-specific behavior is described in a subsection and shown in its own figure, they are all part of the same fallback node shown here.

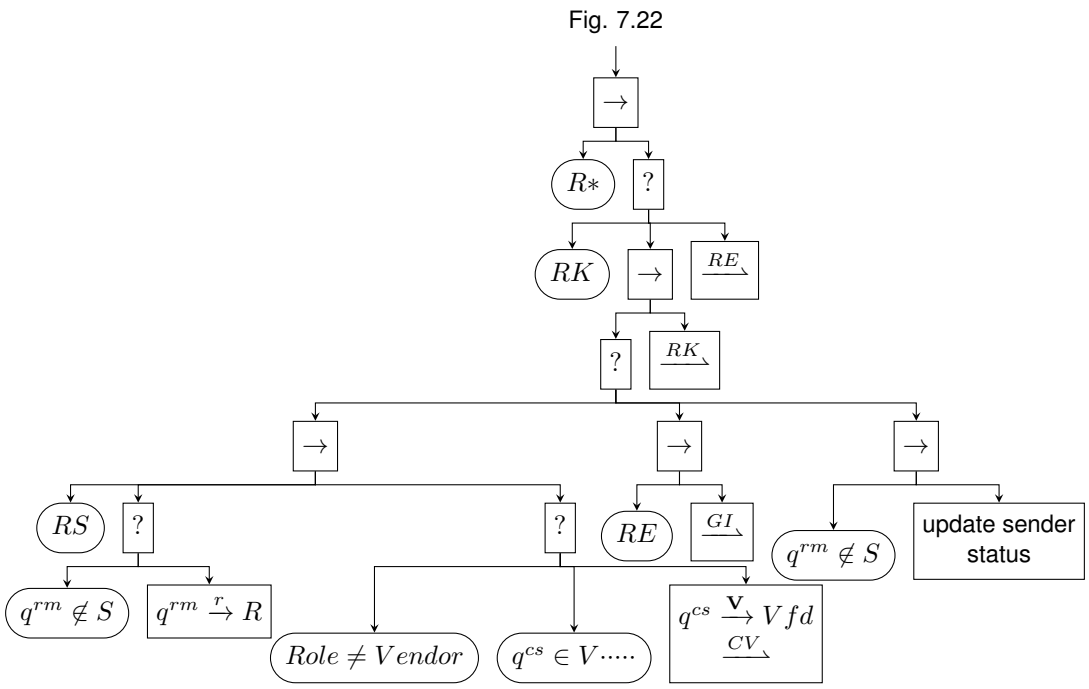


Figure 7.23: Process RM Messages Behavior Tree

7.6.1 Process RM Messages Behavior

The Process RM Messages Behavior Tree is shown in Figure 7.23. It is a child of the fallback node started in Figure 7.22. Beginning with a precondition check for any RM message type, the tree proceeds to a fallback node. RM acknowledgment messages (RK) receive no further attention and return *Success*.

Next comes the main RM message processing sequence. A fallback node covers three major cases:

- First comes a sequence that handles new reports (RS when $q^{rm} \in S$). This branch changes the recipient's RM state regardless of the Participant's role. If the Participant happens to be a Vendor and the Vendor was previously unaware of the vulnerability described by the report, the Vendor would also note the CS transition from $q^{cs} \in vfd \xrightarrow{\mathbf{V}} Vfd$ and emit a corresponding CV message.
- Next, we see that an RM Error (RE) results in the emission of a general inquiry (GI) for Participants to sort out what the problem is, along with an RK to acknowledge receipt of the error.
- Finally, recall that the RM process is unique to each CVD Participant, so most of the remaining RM messages are simply informational messages about other Participants' statuses that do not directly affect the receiver's status. Therefore, if there is already an associated case ($q^{rm} \notin S$), the recipient might update their record of the sender's state, but no further action is needed.

For all three cases, an RK message acknowledges receipt of the message. Any unhandled message results in an RE response, indicating an error.

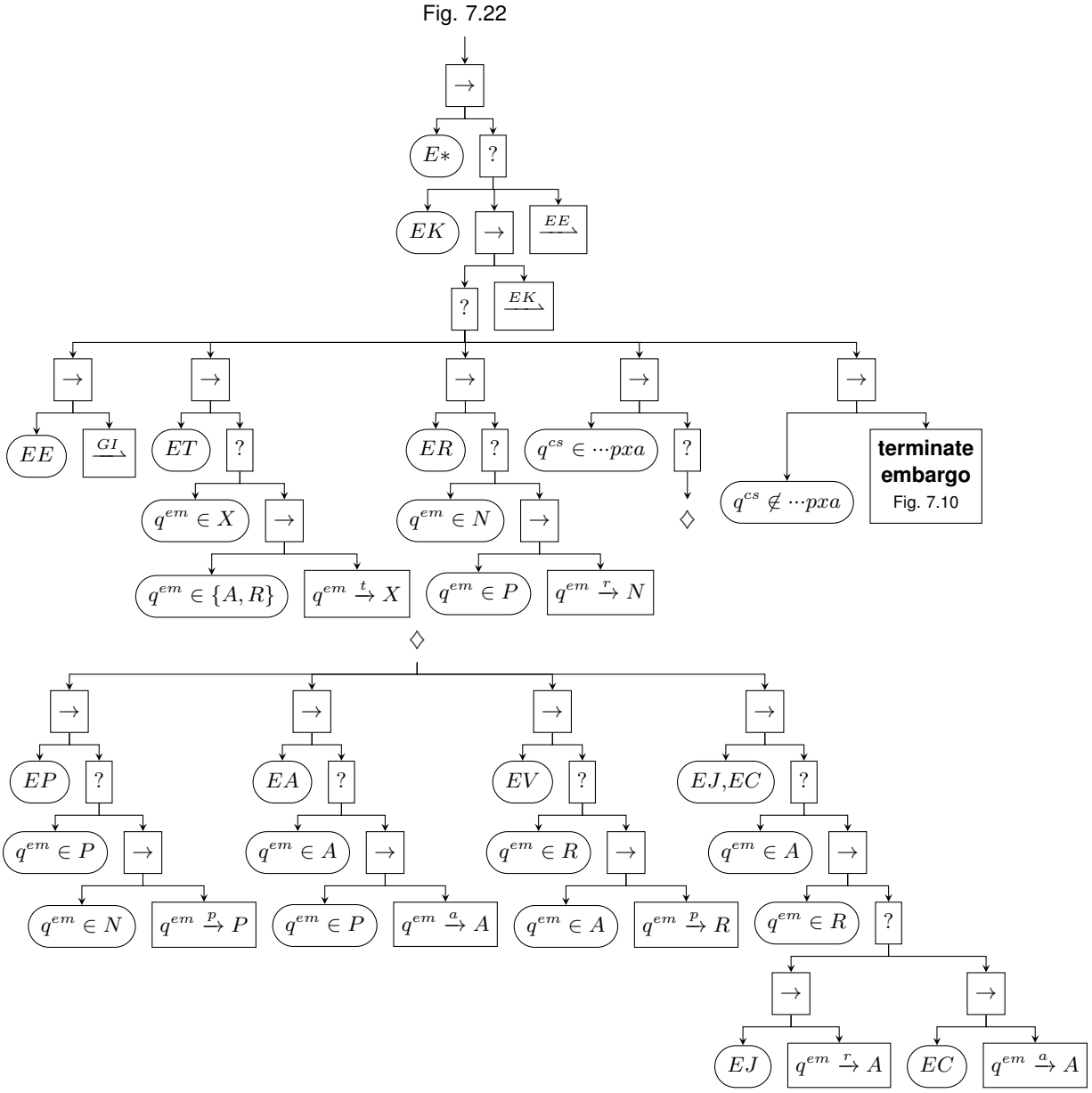


Figure 7.24: Process EM Messages Behavior Tree

7.6.2 Process EM Messages Behavior

The Process EM Messages Behavior Tree is shown in Figure 7.24. As above, it is a child of the fallback node started in Figure 7.22. A precondition check for EM message types is followed by a fallback node. EM acknowledgment messages (EK) receive no further attention and return *Success*.

Messages That Lead to a Simple Acknowledgment. Next is a branch handling all the messages that will result in a simple acknowledgment (EK). First, we handle embargo error messages (EE), which additionally trigger a general inquiry (GI) message to attempt to resolve the problem. Second are embargo termination messages (ET). If the Participant is already in the EM *eXited* state (X), no further action is taken (aside from the EK). Otherwise, if the Participant is in either *Active* or *Revise* EM states, the ET message triggers a state transition $q^{em} \xrightarrow{t} X$. Embargo rejections are handled next in a simple sequence that returns the state from *Proposed* to *None*.

The final chunk of the simple acknowledge branch handles EM messages received when the case state permits embargo viability ($q^{cs} \in \cdots pxa$). A variety of actions can be taken in this case state, as shown in the lower ($\diamondsuit$) tier of Figure 7.24. An embargo proposal (EP) results in either a move from *None* to *Proposed* or stays in *Proposed*, if that was already the case. An embargo acceptance (EA) transitions from *Proposed* to *Active*. Similar to the EP behavior, an embargo revision proposal (EV) either moves from *Active* to *Revise* or stays in *Revise*, as appropriate. Finally, we deal with revision rejection (EJ) or acceptance (EC) when in the *Revise* state. Climbing back up the tree, we see that *Success* in any of the branches in this or the previous paragraph results in an acknowledgment message EK.

Messages That Require More than a Simple Acknowledgment. Returning to the top portion of the tree in Figure 7.24, we come to a branch focused on handling EM messages when an embargo is no longer viable—in other words, when the case has reached a point where attacks are occurring, or either the exploit or the vulnerability has been made public ($q^{cs} \notin \cdots pxa$). This branch takes us to the Terminate Embargo tree in Figure 7.10 (§7.4.2).

Finally, back at the top of Figure 7.24, when no other branch has succeeded, we emit an embargo error (EE) message to relay the failure.

7.6.3 Process CS Messages Behavior

The Process CS Messages Behavior Tree is shown in Figure 7.25. We are still working through the children of the fallback node in Figure 7.22. And as we've come to expect, a precondition check leads to a fallback node in which CS acknowledgement messages (CK) receive no further attention and return *Success*. The main CS message-handling sequence comes next, with all matching incoming messages resulting in emission of an acknowledgment message (CK).

Messages That Change the Participant's Case State. The tree first handles messages indicating a global CS change. Information that the vulnerability has been made public (CP) is met with a transition to the *Public Aware* state in the CS model when necessary. Similarly, information that an exploit has been made public forces both the **X** and **P** transitions, as necessary. Because the **P** transition, should it occur in response to a CX message, represents possibly new information to the case, it triggers the emission of a CP message to convey this information to the other Participants. Likewise, a message indicating attacks underway triggers the **A** transition.

Again, we note that any of the **P**, **X**, or **A** transitions in the CS model imply that no new embargo should be entered, and any existing embargo should be terminated. Hence, the se-

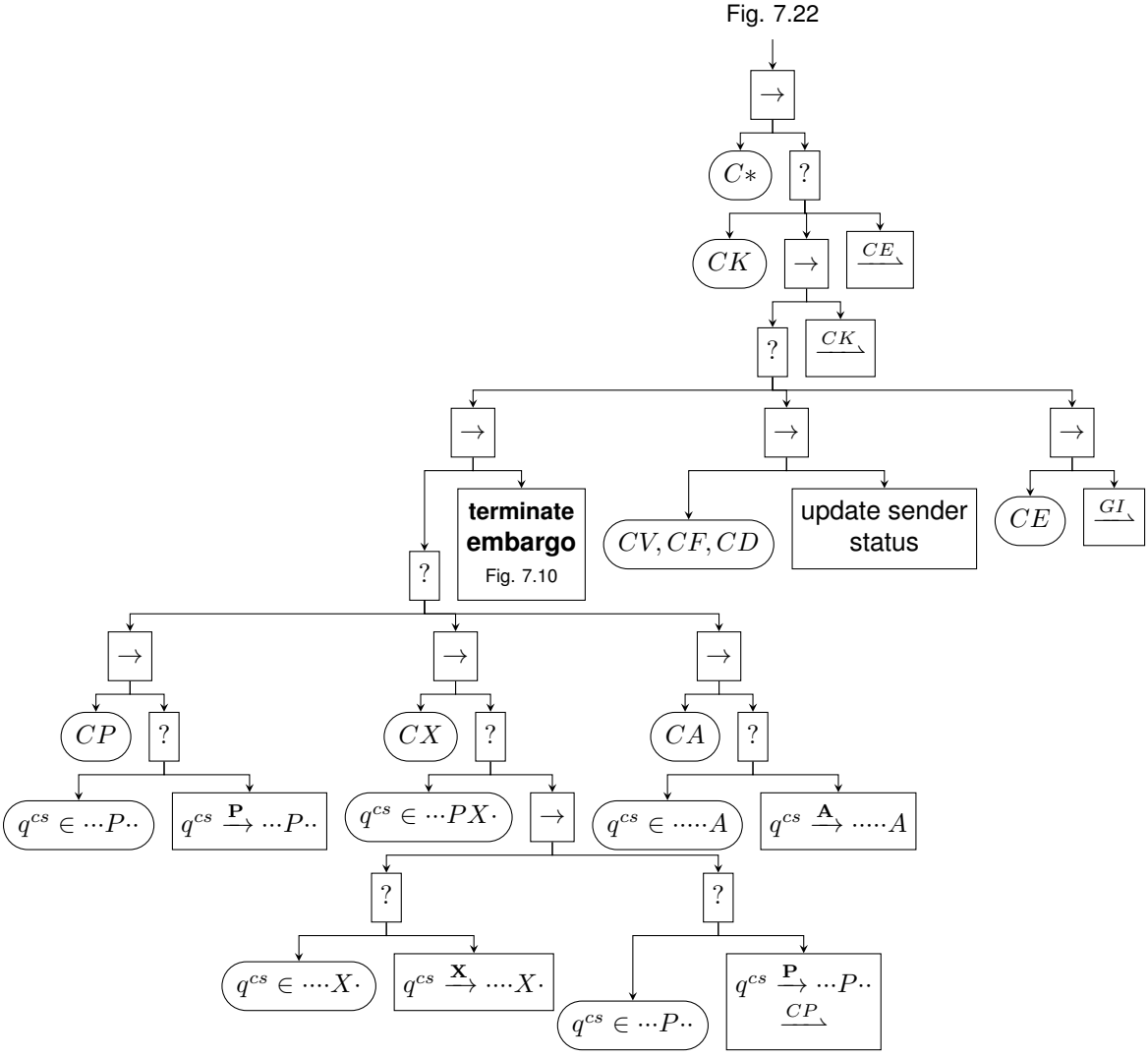


Figure 7.25: Process CS Messages Behavior Tree

Fig. 7.22

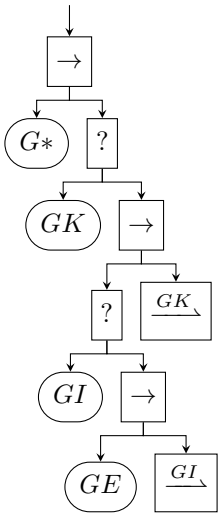


Figure 7.26: Process Other Messages Behavior Tree

quence described in the previous paragraph leads to the embargo termination described in §7.4.2.

Messages That Do Not Change the Participant's Case State. Next, we see that messages indicating *Vendor Awareness* (CV), *Fix Readiness* (CF), and *Fix Deployed* (CD) are treated as mere status updates for the Participant because they are recognized and acknowledged but trigger no further action directly. Recall from §§5.2 and 6.3.2 that the $vfd\cdots \rightarrow \cdots \rightarrow VFD\cdots$ portion of the CS model is unique to each Vendor Participant, and similarly, from §6.3.3, that the $\cdot\cdot d\cdots \rightarrow \cdot\cdot D\cdots$ portion is unique to each Participant in the Deployer role. Therefore, messages representing another Participant's status change for this portion of the CS do not directly affect the receiving Participant's status. This is not to say that the Participant might not choose to take some action based on their knowledge of a Vendor's (or Deployer's) status. Rather, such follow-up would be expected to occur as part of the Participant's *do work* process outlined in §7.5.

Following the tree to the right, we encounter the familiar motif of an error (CE) triggering a general inquiry (GI) to seek resolution.

In the top of Figure 7.25, we have handled all expected messages, so anything else would result in an error condition and emission of a CE message accordingly.

7.6.4 Process Other Messages Behavior

The Process Other Messages Behavior Tree is shown in Figure 7.26. This tree represents the final chunk of the fallback node in Figure 7.22. And here, for the final time, we see a message type check and that general acknowledgment messages (GK) receive no further attention and return *Success*. General inquiries (GI) get at least an acknowledgment, with any follow-up to be handled by *do work* as described in §7.5. As usual, errors (GE) also trigger follow-up inquiries (GI) in the interest of resolution.

Chapter Wrap-Up. In this chapter, we described a complete Behavior Tree for a CVD Participant following the formal MPCVD protocol described in Chapter 6. Next, we discuss a few notes regarding the eventual implementation of this protocol.

8 Implementation Notes

While a complete MPCVD protocol implementation specification is out of scope for this report, we do have a few additional suggestions for future implementations. In this chapter, we describe an abstract case object for use in tracking MPCVD cases. Next, we touch on the core MPCVD protocol subprocesses (RM, EM, and CS), including how the CS model might integrate with other processes. Finally, we provide a few general notes on future implementations.

8.1 An MPCVD Case Object

In this section, we describe a notional MPCVD *Case* object that incorporates the state machines and formal protocol of the previous chapters. The object model we describe is intended to provide the necessary core information for an implementation of the protocol described in §6 to function. Figure 8.1 depicts a UML Class Diagram of the *Case* model. It is not the minimal possible model required by the MPCVD protocol of this report; for example, strictly speaking, a Participant does not need to attempt to track the state of every other Participant, but it might help to do so. Rather, this model is intended to be compact yet sufficient enough for an implementation to effectively track the coordination effort of an MPCVD case.

The remainder of this section provides details about Figure 8.1.

8.1.1 The *Case* Class

The *Case* class has attributes to track the EM state as described in §3 and the global portion of the CS (i.e., the *pxa* substates), as outlined in §4. The *Case* class aggregates one or more *Report*s and *Participant*s, and 0 or more *Message*s and *LogEvent*s.

8.1.2 The *Report* Class

The *Report* class represents the vulnerability report that serves as the impetus for the case. Since it is possible for multiple reports to arrive that describe the same vulnerability, the cardinality of the composition relationship allows for a *Case* to have many *Report*s. In most *Case*s, however, there will be only a single associated *Report*.

8.1.3 The *Message* Class

The *Message* class represents a protocol message as outlined in §6.6. We expect that any implementation of this model will expand this data type to include numerous message-related attributes. Here, we highlight the minimum requirements that the protocol demands: Each *Message* has an identified sender (who is a *Participant* in the case) and one or more message types from §6.6. Message types are represented as flags since a single actual message might represent multiple message types. For example, a report submission that includes an embargo proposal might have both the *RS* and *EP* message type flags set.

Conceptually, one might think of the *Case* as a shared object among *engaged Participants* and that *Messages* are sent to the *Case* for all *Participants* to see. In other words, the *Case* acts as a broadcast domain, a topic queue, or a blackboard pattern (depending on your preferences for networking or software engineering terminology). Because of this shared-channel assumption, we omit a *receiver* attribute from the *Message* class, as the *Case* itself can serve as the recipient of each message emitted by any *Participant*. Implementations of this model could, of course, choose a more traditional messaging model with specified recipients.

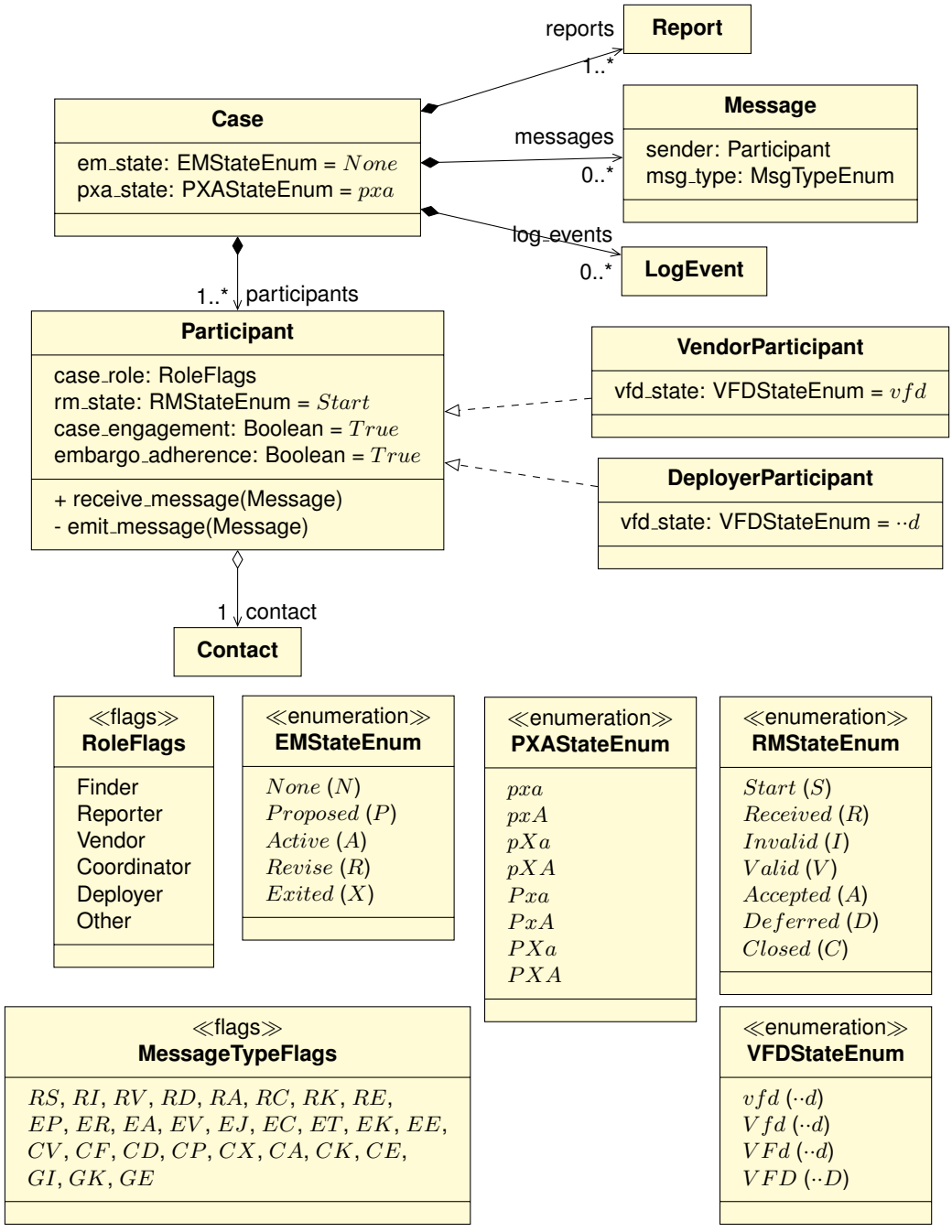


Figure 8.1: UML Class Diagram of a Notional MPCVD Case Object

8.1.4 The *LogEvent* Class

The *LogEvent* class is a placeholder to represent an event log or history for the *Case*. Although not required for the protocol to operate, it is a good idea for *Case* tracking to include a timestamped list of events (e.g., state changes or messages sent or received) so that new *Participants* can be brought up to speed and so that cases can be analyzed for process improvement in the future.

8.1.5 The *Participant* Class

The *Participant* class represents an individual or organization's involvement in the case. The attributes of the *Participant* class are as follows:

case_role. A set of flags indicating the Role(s) this *Participant* plays in the *Case* (Flags are used instead of an enumeration to convey that a *Participant* may have multiple roles in a single *Case*. Roles may differ for the same actor across different cases. For example, an organization might be the Vendor in one case and the Coordinator in another.)

rm_state. An enumeration attribute that captures the RM state for this *Participant* consistent with §2

case_engagement. A Boolean attribute that indicates whether the *Participant* should be included in future communications about the *Case* (This attribute is provided to allow other *Participant*s to recognize the status of other *Participants*. For example, a Reporter who bows out of a case shortly after reporting it to a Coordinator might be listed as a *Participant* with $case\_engagement = False$ and could, therefore, be left out of further communication about the case.)

embargo_adherence. A Boolean attribute that indicates the expectation that a *Participant* is adhering to any existing embargo (As discussed in §3.2.3, it is possible for a *Participant* to exit a case while still agreeing to abide by the terms of the extant embargo. Continuing our example of a Reporter leaving a case early, they might still be cooperative and indicate their $embargo\_adherence = True$. A more hostile *Participant* exit could warrant setting $embargo\_adherence = False$, likely triggering an embargo teardown procedure as a consequence.)

*Participant*s can also emit (send) and receive messages. The + on *receive_message* indicates that this capability is accessible to others (i.e., you can send a *Participant* a message). On the contrary the - on *emit_message* conveys that this capability is only accessible to the *Participant* class itself (i.e., each *Participant* gets to decide if, when, and what messages to send).

***Vendor* and *Deployer Participant* Classes.** The presence of the *VendorParticipant* and *DeployerParticipant* classes—depicted as implementations of the *Participant* class—is necessitated by the discussion in §6.3.2 and §6.3.3, where we described how Vendors and Deployers have a unique part to play in the creation, delivery, and deployment of fixes within the CVD process. These two classes add the *vfd_state* attribute with different possible values. Vendors can take on one of four possible values (vfd, Vfd, VFd, and VFD), whereas Deployers only have two possible values ($\cdot\cdot d$ and $\cdot\cdot D$). Other than that, Vendors and Deployers have the same attributes as other *Participant*s.

8.1.6 The *Contact* Class

Since a *Participant* is a specific relationship between an individual or organization and the *Case* itself, we can safely assume that those individuals or organizations exist and persist independently of the *Case*s they participate in. Hence, each *Participant* class in a *Case* is associated with a long-lived *Contact* record that represents an individual or organization. Defining

the *Contact* class is outside the scope of this report, so we will simply say that there is nothing particularly special about it. One might reasonably expect *Contact*s to have names, email addresses, phone numbers, etc.

A separate contact management process and accompanying directory service is a likely candidate for future integration work. We revisit this topic in §9.1. For now, we observe that similar directories already exist, although there is room for improvement:

- The Forum of Incident Response and Security Teams (FIRST) maintains a directory of member teams for incident response purposes (`https://www.first.org/members/teams/`).
- Disclose.io offers a searchable list of bug bounty and vulnerability disclosure programs (`https://disclose.io/programs/`). Contributions are solicited as pull requests on GitHub (`https://github.com/disclose/diodb`).
- Many vulnerability disclosure platform service providers host directories of the programs hosted on their platforms.

8.1.7 The Enumeration Classes

The remainder of Figure 8.1 consists of classes representing the Role and Message Type flags and various enumerations. The Roles are the same set we have used throughout this report, as taken from the *CVD Guide* [14]. Message Type flags are consistent with §6.6. The enumeration classes are consistent with the RM, EM, and CS state machines described in Chapters 2, 3, and 4, respectively.

8.2 Process Implementation Notes

Integrating the MPCVD protocol into everyday MPCVD operations requires each Participant to consider how their business processes interact with the individual RM, EM, and CS process models we described in Chapters 2, 3, and 4. In this section, we offer some thoughts on where such integration might begin.

8.2.1 RM Implementation Notes

Roughly speaking, the RM process is very close to a normal ITSM incident or service request workflow. As such, the RM process could be implemented as a JIRA ticket workflow, as part of a Kanban process, etc. The main modifications needed to adapt an existing workflow are to intercept the key milestones and emit the appropriate RM messages:

- when the reports are received (RK)
- when the report validation process completes (RI, RV)
- when the report prioritization process completes (RA, RD)
- when the report is closed (RC)

Vulnerability Draft Pre-Publication Review. MPCVD case Participants often share pre-publication drafts of their advisories during the embargo period [16]. Our protocol proposal is mute on this subject because it is not strictly necessary for the MPCVD process to complete successfully. However, as we observe in Appendix B, the GI and GK message types appear to provide sufficient mechanics for this process to be fleshed out as necessary. This draft-sharing process could be built into the *prepare publication* process outlined in §7.5.4.1, where appropriate.

8.2.2 EM Implementation Notes

In terms of the proposal, acceptance, rejection, etc., the EM process is strikingly parallel to the process of scheduling a meeting in a calendaring system. In Appendix C, we suggest a potential mapping of many of the concepts from the EM process onto `iCalendar` protocol semantics.

8.2.3 CS Implementation Notes

Because part of the CS model is Participant specific and the other is global to the case, we address each part below.

The vfd Process. Similar to the RM process, which is specific to each Participant, the vfd process is individualized to each Vendor (or Deployer, for the simpler $d \xrightarrow{\mathbf{D}} D$ state transition). Modifications to the Vendor's development process to implement the MPCVD protocol are expected to be minimal and are limited to the following:

- acknowledging the Vendor's role on report receipt with a CV message
- emitting a CF message when a fix becomes ready (and possibly terminating any active embargo to open the door to publication)
- (if relevant) issuing a CD message when the fix has been deployed

Non-Vendor Deployers are rarely involved in MPCVD cases, but when they are, their main integration point is to emit a CD message when deployment is complete.

The pxa Process. On the other hand, the pxa process hinges on monitoring public and private sources for evidence of information leaks, research publications, and adversarial activity. In other words, the pxa process is well positioned to be wired into Participants' threat intelligence and threat analysis capabilities. Some portions of this process can be automated:

- Human analysts and/or automated search agents can look for evidence of early publication of vulnerability information.
- Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) signatures might be deployed prior to fix availability to act as an early warning of adversary activity.
- Well-known code publication and malware analysis platforms can be monitored for evidence of exploit publication or use.

8.3 General Notes

The protocol and data structures outlined in this report are intended to facilitate interoperability among individual organizations' workflow management systems. As such, they are focused on the exchange of information and data necessary for the MPCVD process to function and will not likely be sufficient to fully address any individual organization's vulnerability response process.

We conclude the chapter with a few general implementation notes.

8.3.1 Message Formats

We defined a number of message types in §6.6 and showed how they fit into a case in §8.1, but we did not specify any format for these messages. Message formats can be thought of as two related problems:

Structure and Semantic Content of Each Message Type. In addition to the commentary throughout this chapter, messages will likely need to have some sort of consistent header information and some content specifically designed to address the semantic needs of each message type. Such a format must include fields, datatypes, and an underlying formatting structure.

Container Syntax for Messaging and Data Exchange. While we have a predilection for JavaScript Object Notation (JSON) Schema-defined formats, other format specifications (e.g., Extensible Markup Language (XML) Schema Definition standard (XSD) or Protocol Buffers (protobuf)) could serve implementers' needs just as well. In fact, to the degree possible, it seems preferable for the container syntax to remain a late-binding decision in implementation. As long as the structure and semantics are well defined, most standard data formats should be adaptable to the task.

- MPCVD Protocol Messages SHOULD use well-defined format specifications (e.g., JSON Schema, protobuf, XSD).

We anticipate that emerging formats like the OASIS Common Security Advisory Framework (CSAF) [8, 7] and ontologies like the National Institute of Standards and Technology (NIST) Vulnerability Data Ontology (Vulntology) [30] will play a part.

8.3.2 Transport Protocol

We have not specified how MPCVD protocol implementations connect to each other. Presumably, technologies such as Representational State Transfer (REST) Application Programming Interfaces (APIs) or WebSockets would be leading candidates to resolve this gap. However, other system architectures could be adapted as well. For example, an Extensible Messaging and Presence Protocol (XMPP) message-routing system might be desired, or even blockchain-related technologies might be adaptable to portions of this protocol as well.

- MPCVD Protocol Implementations SHOULD use common API patterns (e.g., REST, WebSockets).

8.3.3 Identity Management

We have not addressed Participant authentication as part of the protocol, but obviously implementers will need to determine how Participants know who they are talking to. Individual user accounts with multi-factor authentication are the de facto standard for modern CVD tools, but in an interoperable MPCVD world, the assumption of centralized identity management may not be practical. Federated identity protocols such as OAuth, Security Assertion Markup Language (SAML), and/or OpenID Connect may be useful.

9 Future Work

In this chapter, we review a number of items remaining as future work. We start with a discussion of the need for a CVD Directory and some of the difficulties it might pose. Next, we revisit the concept of churn in the RM and EM processes and elaborate a few ideas to reward efficient behavior. We wrap up the chapter with some brief thoughts on publication scheduling, the potential use of ontologies for process interoperability, and ideas for future modeling and simulation to further improve the MPCVD process.

9.1 CVD Directory

The idea of CVD embargoes implies a means of dividing the world into those who belong in the embargo and those who do not. Because authentication is not the same as authorization, we cannot simply rely on knowing *who* a Participant *is*; we also have to be able to identify *why* they are *relevant* to a particular case.

Thus, we must ask this question: How do Participants find other relevant potential Participants to invite to a case? In small CVD cases, the answer might be straightforward: The affected product comes from a known Vendor, so the only question to answer is how best to contact them. As a first approximation, Internet search engines offer a de facto baseline CVD directory service simply because they allow any potential Reporter to search for `<vendor name> vulnerability report` or similar terms to find an individual Vendor contact.[8]

But in larger MPCVD cases, there are a few entangled problems:

1. It can be difficult and inefficient to collect contact information for all possibly relevant parties.
2. Even if contact information is widely available using searchable resources, many Vendors' preferred contact methods might preclude automation of mass notification (or require customized integration to ensure interoperability between report senders and receivers). Some Vendors only want email. Others require Reporters to create an account on their bespoke bug-tracking system before reporting. Others ask for submissions via a customized web form. All of these examples hinder the interoperability of MPCVD processes.
3. It is not always clear which *other* Vendors' products contain the affected product, which limits MPCVD cases' ability to follow the software supply chain.
4. Sometimes vulnerabilities arise in protocols or specifications where multiple implementations are affected. It can be difficult to identify Vendors whose products implement specific technologies. Software reverse engineering methods can be used to identify affected products in some cases.
5. At the same time, some Vendors treat their product's subcomponents as proprietary close-hold information for competitive advantage; this might happen, for example, with Original Equipment Manufacturer (OEM) or white label licensing agreements. While it is certainly their prerogative to do so, this desire to avoid disclosing internal components of a product can inhibit discovery—and therefore disclosure to the Vendor—that a vulnerability affects a product.

[8] Vendors can improve their discoverability by using a `security.txt` file on their websites. See the security.txt website for more information (`https://securitytxt.org/`).

When it comes to larger scale MPCVD, the inefficiency of ad hoc contact collection via search engines is evident. Creating a directory of software Vendors and Coordinators and their vulnerability disclosure programs would be a step in the right direction. Community-operated directories such as the FIRST member list or Disclose.io serve as proof-of-concept of the value such systems can provide.[9] We especially like the open source model that Disclose.io uses, which solicits contributions from the community.[10]

But further improvements to MPCVD contact management could be made by standardizing the following:

- contact information records and the APIs to access them
- contact methods, including common protocols such as the one we just proposed, in conjunction with common data object models and vocabularies or ontologies
- Software Bill of Materials (SBOM) publication and aggregation services
- mechanisms for Vendors to register their interest in specific technologies

The last of these suggested improvements is not without its challenges. It is difficult to prevent adversarial parties (including Participants who might be competitors or have motives incompatible with CVD principles) from registering interest in receiving vulnerability reports about technologies in others' products.

9.2 Reward Functions

Further optimization of the MPCVD protocol can be studied with the development of reward functions to evaluate preferences for certain CVD case histories over others. Householder and Spring [10] provide a method to measure skill (α_d) in CVD based on a partial order over the CVD success criteria that make up the CS process, as outlined in §1.3. While not yet a fully-realized reward function, we feel that the α_d skill measure has potential as the basis of a reward function for the CS model.

The following subsections describe two additional reward functions.

9.2.1 A Reward Function for Minimizing RM Strings

In §2.1.2.2, we described a grammar that generates RM histories. The state machine can generate arbitrarily long histories because of the cycles in the state machine graph; however, we found that human Participants in any real CVD case would likely check the amount of churn. That sort of reliance on human intervention will not scale as well as a more automatable solution might.

As a result, we suggest that future work might produce a reward function that can be used to optimize RM histories. Such a function would need to include the following:

- a preference for shorter paths over longer ones
- a preference for paths that traverse through $q^{rm} \in A$ over ones that do not
- a preference for Vendor attentiveness (The default path for an organization with no CVD capability is effectively $q^{em} \in S \xrightarrow{r} R \xrightarrow{i} I \xrightarrow{c} C$, which is short (good!). However, assuming the vulnerability is legitimate, half of the desired CS criteria can never be achieved (bad!). In other words, $\mathbf{F} \prec \mathbf{P}$, $\mathbf{F} \prec \mathbf{X}$, $\mathbf{F} \prec \mathbf{A}$, $\mathbf{D} \prec \mathbf{P}$, $\mathbf{D} \prec \mathbf{X}$, $\mathbf{D} \prec \mathbf{A}$ are impossible when the Vendor ignores the report. No reward function should provide incentive for willful Vendor ignorance.)

[9] For more information, see the FIRST (`https://www.first.org/members/teams/`) and Disclose.io (`https://disclose.io/programs/`) websites.

[10] `https://github.com/disclose/diodb`

- a preference for validation accuracy (Real vulnerabilities should pass through $q^{rm} \in V$, while bogus reports should pass through $q^{rm} \in I$. The only RM paths path not involving at least one step through $q^{rm} \in A$ are the following.)

$$q^{em} \in \begin{cases} S \xrightarrow{r} R \xrightarrow{i} I \xrightarrow{c} C & \text{Ignore an invalid case.} \\ S \xrightarrow{r} R \xrightarrow{v} V \xrightarrow{d} D \xrightarrow{c} C & \text{Defer a valid case.} \\ S \xrightarrow{r} R \xrightarrow{i} I \xrightarrow{v} V \xrightarrow{d} D \xrightarrow{c} C & \text{Initially ignore an invalid case, then validate, but defer it anyway.} \end{cases}$$

  To an outside observer, any of these could be interpreted as inattentiveness from an uncommunicative Participant. Yet any of these paths might be fine, assuming that (1) the Participant communicates about their RM state transitions, and (2) the a transition was possible but intentionally just not taken.

These last two imply some capacity for independent validation of reports, which, on the surface, seems poised to add cost or complexity to the process. However, in any MPCVD case with three or more Participants, a consensus or voting heuristic could be applied. For example, should a quorum of Participants agree that a Vendor's products are affected even if the Vendor denies it, an opportunity exists to capture this information as part of the case.[11]

9.2.2 A Reward Function for Minimizing EM Strings

Similarly, the EM process also has the potential to generate arbitrarily long histories, as shown in §3.1.2.2. Again, reliance on humans to resolve this shortcoming may be acceptable for now; however, looking to the future, we can imagine a reward function to be optimized. The EM reward function might include the following:

- a preference for short paths
- a preference for quick agreement (i.e., the a transition appearing early in the EM history)
- a limit on how long an EM history can get without reaching $q^{em} \in A$ at all (i.e., How many proposal-rejection cycles are tolerable before giving up?)

9.3 Embargo Management Does Not Deliver Synchronized Publication

In our MPCVD protocol design, we were careful to focus the EM process on establishing when publication restrictions are lifted. That is not the same as actually scheduling publications following the embargo termination. Our experience at the CERT/CC shows that this distinction is rarely a significant problem since many case Participants simply publish at their own pace shortly after the embargo ends. However, at times, case Participants may find it necessary to coordinate even more closely on publication scheduling.

9.4 Ontology

Our proposed MPCVD protocol does not make its appearance in uncharted territory, where no existing CVD systems or processes exist. Rather, we propose it as an improvement to interactions among humans, systems, and business processes that already perform MPCVD around the world every day. Thus, for adoption to occur, it will be necessary to map existing systems and processes into the semantics (and eventually, the syntax) of whatever protocol emerges as a descendant of our proposal.

[11] In fact, this very problem is why the individual Vendor records in CERT/CC Vulnerability Notes contain a *CERT/CC Addendum* field.

Combined with the abstract case class model described in §8.1, an ontology (e.g., using Web Ontology Language (OWL)) could accelerate the semantic interoperability between independent Participant processes and tools that we set out to improve at the beginning of this report.

9.5 Modeling and Simulation

The protocol formalisms and Behavior Trees provided in this report combined with the CS model described in the Householder and Spring 2021 report [10] point the way toward improvements in MPCVD modeling and simulation. Given the complexity of the protocol state interactions described in Chapter 6 and the corresponding behaviors described in Chapter 7, we anticipate that modeling and simulation work will continue progressing toward a reference implementation of the protocol we describe in this report.

Furthermore, the reward functions outlined in §9.2 can—once fully realized—be used to evaluate the efficacy of future modifications to the protocol. This effort could, in turn, lead to future improvements and optimizations of the MPCVD process. The modularity of Behavior Trees provides ready ground for simulated experiments to determine what additional optimizations to the MPCVD process might be made in the future.

10 Conclusion

In this report, we described a proposal for an MPCVD protocol in the interest of improving the interoperability of the world's CVD processes. Our working name for this protocol is *Vultron*, intended to evoke the coordination and cooperation required to assemble five robot lions and their pilots into an ad hoc collective entity in defense against a shared adversary.

Our proposal is built on three primary processes, each modeled as DFAs:

1. the RM process model in §2
2. the EM process model in §3
3. the CS process model in §4, as originally described by Householder and Spring [10]

We discussed the interactions between these three DFAs in §5. Then, we combined them into a single formal MPCVD protocol in §6, which we concluded with an example of the proposed protocol in action.

We modeled the behavior of an individual MPCVD Participant in §7 as a nested set of Behavior Trees. The modularity of Behavior Trees allows for various Participants to be modeled as agents, which serves two purposes. First, it identifies increased automation of portions of the MPCVD process, improving the potential for human-machine hybrid CVD processes in the future. Second, it provides a means to model Participant behaviors in software, which can facilitate MPCVD process simulation and optimization at a scale previously unknown.

The implementation notes in §8 and the future work outlined in §9 set an agenda for MPCVD process improvement research and development for the near future.

Finally, the MPCVD protocol proposed in this report—in conjunction with the *CERT Guide to Coordinated Vulnerability Disclosure* [14, 13] and *SSVC* [27, 28, 29]—is intended to paint as complete a picture as possible of the CERT/CC's current understanding of how CVD should be performed.

Ready to form Vultron!

Request for Feedback

We intend for this report to be the start (not the end) of a conversation with the community. Although we made every effort to ensure the completeness and accuracy of the information contained in this report, there is always room for improvement. Please contact the author to provide recommendations, corrections, opinions, or requests for clarification:

Allen D. Householder

`https://www.sei.cmu.edu/contact-us/index.cfm?f=Allen&l=Householder`

A Interactions Between the MPCVD Protocol and SSVC

Once a report has been validated (i.e., it is in the RM $Valid$ state, $q^{rm} \in V$), it must be prioritized to determine what further effort, if any, is necessary. While any prioritization scheme might be used, in this appendix, we apply the Stakeholder-Specific Vulnerability Categorization (SSVC) model.

A.1 SSVC Supplier and Deployer Trees

The default outcomes for both the SSVC Supplier and Deployer Trees are $Defer$, $Scheduled$, $Out\ of\ Cycle$, and $Immediate$. The mapping from SSVC outcomes to RM states is straightforward, as shown in (A.1) for the Supplier Tree and (A.2) for the Deployer Tree. The SSVC $Defer$ output maps directly onto the RM $Deferred$ state. Otherwise, the three outputs that imply further action is necessary—$Scheduled$, $Out\ of\ Cycle$, and $Immediate$—all proceed to the RM *Accepted* state. The different categories imply different processes within the *Accepted* state. But because the RM model does not dictate internal organizational processes, further description of what those processes might look like is out of scope for this report.

$$q^{rm} \in \begin{cases} \xrightarrow{d} D & \text{when } SSVC(Supplier\ Tree) = Defer \\ \xrightarrow{a} A & \text{when } SSVC(Supplier\ Tree) \in \left\{ \begin{array}{c} Scheduled \\ Out\ of\ Cycle \\ Immediate \end{array} \right\} \end{cases} \tag{A.1}$$

$$q^{rm} \in \begin{cases} \xrightarrow{d} D & \text{when } SSVC(Deployer\ Tree) = Defer \\ \xrightarrow{a} A & \text{when } SSVC(Deployer\ Tree) \in \left\{ \begin{array}{c} Scheduled \\ Out\ of\ Cycle \\ Immediate \end{array} \right\} \end{cases} \tag{A.2}$$

We remind readers of a key takeaway from the protocol requirements in the main part of this report:

- Vendors SHOULD communicate their prioritization choices when making either a $defer$ ($\{V, A\} \xrightarrow{d} D$) or *accept* ($\{V, D\} \xrightarrow{a} A$) transition out of the $Valid$, $Deferred$, or *Accepted* states.

A.2 SSVC Coordinator Trees

SSVC version 2 offers two decision trees for Coordinators: A Coordinator Triage Tree and a Coordinator Publish Tree. The outputs for the Coordinator Triage Decision Tree are $Decline$, $Track$, and $Coordinate$. Similar to the Supplier Tree mapping in §A.1, the mapping here is simple, as shown in (A.3). Again, whereas the $Decline$ output maps directly to the RM $Deferred$ state, the remaining two states ($Track$ and $Coordinate$) imply the necessity for distinct processes within the Coordinator's RM *Accepted* state.

$$q^{rm} \in \begin{cases} \xrightarrow{d} D & \text{when } SSVC(Coord.\ Triage\ Tree) = Decline \\ \\ \xrightarrow{a} A & \text{when } SSVC(Coord.\ Triage\ Tree) \in \left\{ \begin{matrix} Track \\ Coordinate \end{matrix} \right\} \end{cases} \tag{A.3}$$

On the other hand, the SSVC Coordinator Publish tree falls entirely within the Coordinator's *Accepted* state, so its output does not directly induce any Coordinator RM state transitions. However, a number of its decision points *do* touch on the protocol models, which we cover in §A.3.

A.3 SSVC Decision Points and the MPCVD Protocol

Additional connections between the protocol and the SSVC decision trees are possible. We now examine how individual SSVC tree decision points can inform or be informed by Participant states in the MPCVD protocol.

A.3.1 Exploitation

The SSVC Exploitation decision point permits three possible values: $None$, PoC, and $Active$. These values map directly onto state subsets in the CS model, as shown in (A.4). A value of $None$ implies that no exploits have been made public, and no attacks have been observed (i.e., $q^{cs} \in \cdots\cdot xa$). The PoC value means that an exploit is public, but no attacks have been observed (i.e., $q^{cs} \in \cdots\cdot Xa$). Finally, the $Active$ value indicates attacks are occurring (i.e., $q^{cs} \in \cdots\cdot\cdot A$). These case states and SSVC values are equivalent in both directions, hence our use of the "if-and-only-if" ($\iff$) symbol.

$$SSVC(exploitation) = \begin{cases} None & \iff q^{cs} \in \cdots\cdot xa \\ PoC & \iff q^{cs} \in \cdots\cdot Xa \\ Active & \iff q^{cs} \in \cdots\cdot\cdot A \end{cases} \tag{A.4}$$

A.3.2 Report Public

The SSVC Report Public decision point also maps directly onto the CS model. A value of Yes means that the report is public, equivalent to $q^{cs} \in \cdots P\cdot\cdot$. On the other hand, a No value is the same as $q^{cs} \in \cdots p\cdot\cdot$. As above, " $\iff$ " indicates the bidirectional equivalence.

$$SSVC(report\ public) = \begin{cases} Yes & \iff q^{cs} \in \cdots P\cdot\cdot \\ No & \iff q^{cs} \in \cdots p\cdot\cdot \end{cases} \tag{A.5}$$

A.3.3 Supplier Contacted

If the Supplier (Vendor) has been notified (i.e., there is reason to believe they are at least in the RM *Received* state, equivalent to the $V\cdots\cdot$ CS state subset) the Supplier Contacted value should be Yes, otherwise it should be No.

$$SSVC(supp.\ contacted) = \begin{cases} Yes & \text{if } q^{rm}_{Vendor} \notin S \text{ or } q^{cs}_{Vendor} \in V\cdots\cdot \\ \\ No & \text{if } q^{rm}_{Vendor} \in S \text{ or } q^{cs}_{Vendor} \in vfd\cdots \end{cases} \tag{A.6}$$

A.3.4 Report Credibility

Unlike most of the other SSVC decision points covered here that form a part of a Participant's report prioritization process *after* report validation, the Report Credibility decision point forms an important step in the Coordinator's validation process. In fact, it is often the only validation step possible when the Coordinator lacks the ability to reproduce a vulnerability whether due to constraints of resources, time, or skill. Thus, a value of $Credible$ can be expected to lead to an RM transition to $Valid$ ($q^{rm} \in R \xrightarrow{v} V$), assuming any additional validation checks also pass. On the contrary, $Not\ Credible$ always implies the RM transition to $Invalid$ ($q^{rm} \in R \xrightarrow{i} I$) because "Valid-but-not-Credible" is a contradiction.

$$SSVC(report\ cred.) = \begin{cases} Credible & \text{implies } q^{rm} \xrightarrow{v} V \text{ (if validation also passes)} \\ Not\ Credible & \text{implies } q^{rm} \xrightarrow{i} I \end{cases} \tag{A.7}$$

A.3.5 Supplier Engagement

The possible values for the Supplier (Vendor) Engagement decision point are $Active$ or $Unresponsive$. From the Coordinator's perspective, if enough Suppliers in a CVD case have communicated their engagement in a case (i.e., enough Vendors are in the RM $Accepted$ state already or are expected to make it there soon from either the $Received$ or $Valid$ states), then the SSVC value would be $Active$.

Vendors in $Invalid$ or $Closed$ can be taken as disengaged, and it might be appropriate to select $Unresponsive$ for the SSVC Engagement decision point.

Vendors in either $Received$ or $Deferred$ might be either $Active$ or $Unresponsive$, depending on the specific report history.

This mapping is shown in (A.8) and on the left side of Figure A.1.

$$SSVC(supp.\ eng.) = \begin{cases} Active & \text{if } q^{rm} \in \{A, V\} \\ \left.\begin{matrix} Active \\ Unresponsive \end{matrix}\right\} & \text{if } q^{rm} \in \{R, D\} \\ Unresponsive & \text{if } q^{rm} \in \{I, C, S\} \end{cases} \tag{A.8}$$

A.3.6 Supplier Involvement

The Supplier Involvement decision point can take on the values $Fix\ Ready$, $Cooperative$, or $Uncooperative/Unresponsive$. We begin by noting the equivalence of the $Fix\ Ready$ value with the similarly named substate of the CS model.

$$SSVC(supp.\ inv.) = Fix\ Ready \iff q^{cs} \in VF\cdots \tag{A.9}$$

The Vendor RM states map onto these values as shown in (A.10) and on the right side of Figure A.1.

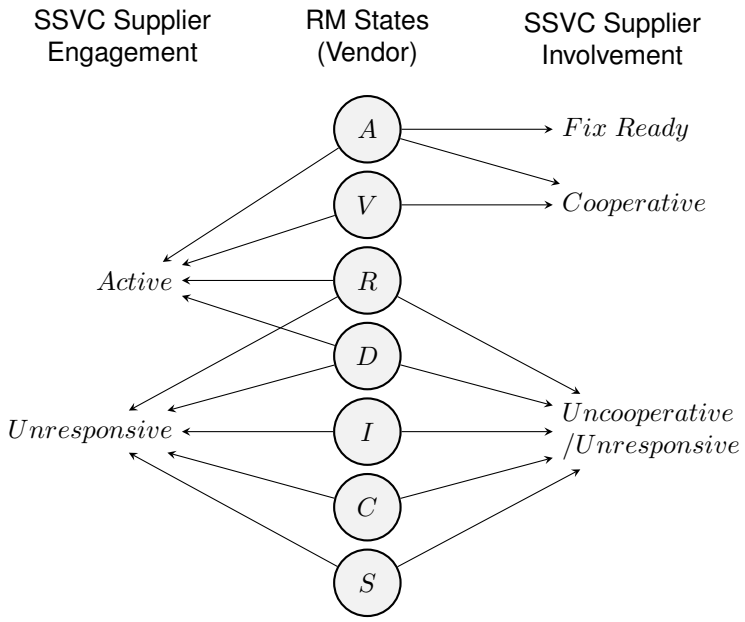


Figure A.1: Mapping Vendor RM States to the SSVC Supplier Engagement (Left) and Supplier Involvement (Right) Decision Point Values

$$SSVC(supp.\ inv.) = \begin{cases} \left.\begin{matrix} Fix\ Ready \\ Cooperative \end{matrix}\right\} & \text{if } q^{rm} \in A \\ Cooperative & \text{if } q^{rm} \in V \\ Uncoop./Unresp. & \text{if } q^{rm} \in \{R, I, D, C, S\} \end{cases} \tag{A.10}$$

A.3.7 Engagement vs. Involvement: What's the Difference?

Note the discrepancy between the mappings given for SSVC Supplier Engagement versus those for Supplier Involvement. This distinction is most prominent in the connections from the R and D RM states on the left and right sides of Figure A.1. These differences are the result of the relative timing of the two different decisions they support within a CVD case.

The decision to Coordinate (i.e., whether the Coordinator should move from RM $Valid$ to RM *Accept* ($q^{rm} \in V \xrightarrow{a} A$)) occurs early in the Coordinator's RM process. The SSVC Supplier Engagement decision point is an attempt to capture this information. This early in the process, allowances must be made for Vendors who may not have completed their own validation or prioritization processes. Hence, the mapping allows Vendors in any valid yet unclosed state ($q^{rm} \in \{R, V, A, D\}$) to be categorized as Active for this decision point.

On the other hand, the decision to Publish—a choice that falls entirely within the Coordinator's RM *Accepted* state—occurs later, at which time, more is known about each Vendor's level of involvement in the case to that point. By the time the publication decision is made, the Vendor(s) have had ample opportunity to engage in the CVD process. They might already have a $Fix\ Ready$ ($q^{cs} \in VF\cdots$), or they might be working toward it (i.e., SSVC $Cooperative$). However, if the Coordinator has reached the point where they are making a publication decision, and the Vendor has yet to actively engage in the case for whatever

reason—as indicated by their failure to reach the RM *Accepted* state or demonstrate progress toward it by at least getting to RM $Valid$ ($q^{rm} \in \{A, V\}$)—then they can be categorized as $Uncooperative/Unresponsive$.

B MPCVD Protocol and ISO/IEC Standards

A number of ISO/IEC documents are relevant to our proposed MPCVD protocol.

- ISO/IEC 30111:2019 *Information technology — Security techniques — Vulnerability handling processes*
- ISO/IEC 29147:2018 *Information technology — Security techniques — Vulnerability disclosure*
- ISO/IEC TR 5895:2022 *Cybersecurity — Multi-party coordinated vulnerability disclosure and handling*

We cover each in its own section below.

B.1 ISO/IEC 30111:2019

Clause 7 of ISO/IEC 30111:2019 closely relates to our proposed MPCVD protocol [17]. Table B.1 provides a mapping of ISO/IEC 30111:2019 onto the relevant concepts and sections of this report.

Table B.1: Mapping ISO/IEC 30111:2019 Onto the MPCVD Protocol

30111:2019 Clause	Sub-Clause	MPCVD Protocol Mapping
§7.1.1 General §7.1.2 Preparation	- -	See specific details below.
§7.1.3 Receipt	a) Internally Found Vulnerabilities	§2.1.1.2 The *Received* State (R) (p. 10) §6.6.1 RM Message Types (p. 59) §7.2 Vulnerability Discovery Behavior (p. 78) $q^{rm} \in S \xrightarrow{r} R$
	b) Externally Found Vulnerabilities	§2.1.1.2 The *Received* State (R) (p. 10) §6.6.1 RM Message Types (p. 59) §7.6.1 Process RM Messages Behavior (p. 95) $q^{rm} \in S \xrightarrow{r} R$
	c) Publicly Disclosed Vulnerabilities	§2.1.1.2 The *Received* State (R) (p. 10) §6.6.1 RM Message Types (p. 59) §6.6.3 CS Message Types (p. 60) §7.5.5 Monitor Threats Behavior (p. 92) §7.6.1 Process RM Messages Behavior (p. 95) §7.6.3 Process CS Messages Behavior (p. 97) $\left\{ \begin{array}{c} q^{rm} \in S \xrightarrow{r} R \\ q^{cs} \in \cdots p \cdot\cdot \xrightarrow{\mathbf{P}} \cdots P \cdot\cdot \end{array} \right\}$
§7.1.4 Verification	a) Initial investigation	§2.1.1.2 The *Received* State (R) (p. 10) §7.3.1 Report Validation Behavior (p. 80) $q^{rm} \in \begin{cases} R \xrightarrow{i} I & \text{if } \textit{invalid} \\ R \xrightarrow{v} V & \text{if } \textit{valid} \end{cases}$
	b) Possible Process Exit	
	1) Duplicate	attach to the original report

Continued on next page

Table B.1 – continued from previous page

30111:2019 Clause	Sub-Clause	MPCVD Protocol Mapping
	2) Obsolete product	$q^{rm} \in \begin{cases} I \xrightarrow{c} C & \text{if } invalid \\ V \xrightarrow{d} D \xrightarrow{c} C & \text{if } valid \end{cases}$
	3) Non-security	$q^{rm} \in I \xrightarrow{c} C$
	4) Other vendor	§7.5.3 Reporting Behavior (p. 88) $q^{rm} \in V \xrightarrow{a} A$
	c) Root Cause Analysis	§7.5 Do Work Behavior (p. 85) $q^{rm} \in A$
	d) Further investigation	§7.5 Do Work Behavior (p. 85) $q^{rm} \in A$
	e) Prioritization	§2.1.1.4 The *Valid* State (V) (p. 11) §7.3.2 Report Prioritization Behavior (p. 81) $q^{rm} \in \begin{cases} V \xrightarrow{d} D & \text{on } defer \\ V \xrightarrow{a} A & \text{on } accept \end{cases}$
	f) Inform reporter	§7.3.1 Report Validation Behavior (p. 80) §7.3.2 Report Prioritization Behavior (p. 81) Emit RV, RI, RA, RD as appropriate.
§7.1.5 Remediation development	all	§2.1.1.5 The *Accepted* State (A) (p. 11) §4.1.2 The *Fix Readiness* Substate (f, F) (p. 37) §5.2.2 Fix Ready (p. 47) §7.5.2 Fix Development Behavior (p. 88) $\left\{ \begin{array}{c} q^{rm} \in A \\ q^{cs} \in Vfd\cdots \xrightarrow{\mathbf{V}} VFd\cdots \end{array} \right\}$
§7.1.6 Release	-	§4.1.2 The *Fix Readiness* Substate (f, F) (p. 37) §7.5.4 Publication Behavior (p. 90) $q^{cs} \in VFdp\cdot\cdot \xrightarrow{\mathbf{P}} VFdP\cdot\cdot$
§7.1.7 Post-release	all	§7.3.3 Report Closure Behavior (p. 82) §7.5.1 Deployment Behavior (p. 86) $\left\{ \begin{array}{l} q^{cs} \in VF\cdot P\cdot\cdot \\ q^{rm} \in \{A, D\} \end{array} \right\}$
§7.2 Process monitoring	all	§7.5.1 Deployment Behavior (p. 86)
§7.3 Confidentiality	-	§7.4 Embargo Management Behavior Tree (p. 82)
§8 Supply chain considerations	-	§7.5.3 Reporting Behavior (p. 88)

B.2 ISO/IEC 29147:2018

Similarly, ISO/IEC 29147:2018 also overlaps with our proposed MPCVD protocol [16]. Our use of the following terms is consistent with ISO/IEC 29147:2018 §5.4 *Systems, components, and services*: Systems, Components, Products, Services, Vulnerability, and Product interdependency.

ISO/IEC 29147:2018 §5.5 *Stakeholder Roles* includes *User*, *Vendor*, *Reporter*, and *Coordinator*. We generally use *Deployer* instead of *User*, but the rest are consistent.

Perhaps unsurprisingly, clauses 5 through 9 of ISO/IEC 29147:2018 overlap significantly with our proposed MPCVD protocol. See Table B.2 for a thorough cross-reference.

Table B.2: Mapping ISO/IEC 29147:2018 Onto the MPCVD Protocol

29147:2018 Clause	Sub-Clause	MPCVD Protocol Mapping
§5.6 Vulnerability handling process summary	5.6.1 General 5.6.2 Preparation 5.6.3 Receipt 5.6.4 Verification 5.6.5 Remediation development 5.6.6 Release 5.6.7 Post-release	The first few subsections of ISO/IEC 29147:2018 §5.6 are recapitulated in ISO/IEC 30111:2019. Accordingly, see the corresponding rows of Table B.1
	5.6.8 Embargo period	§3.2 EM Discussion (p. 24)
§5.7 Information exchange during vulnerability disclosure	send-report-to	§6.6 Message Types (p. 58) §7.5.3 Reporting Behavior (p. 88) $q^{rm} \in \begin{cases} A + \overset{RS}{\searrow} & sender \\ S \xrightarrow{r} R + \overset{RS}{\swarrow} & receiver \end{cases}$
	release-advisory-to	§6.6 Message Types (p. 58) §7.5.4.1 Prepare Publication Behavior (p. 90) $q^{rm} \in \begin{cases} A + \overset{GI}{\searrow} & sender \\ \{R, V, A, D\} + \overset{GI}{\swarrow} & receiver \end{cases}$
§5.8 Confidentiality	all	§3 Embargo Management (EM) Model (p. 20) §8.3.2 Transport Protocol (p. 105)
§5.9 Vulnerability advisories	-	§4.1.4 The *Public Awareness* Substate (p, P) (p. 38) §7.5.4 Publication Behavior (p. 90)
§5.10 Vulnerability exploitation	-	§4.1.5 The *Exploit Public* Substate (x, X) (p. 38) §4.1.6 The *Attacks Observed* Substate (a, A) (p. 38) §7.5.5 Monitor Threats Behavior (p. 92)
§5.11 Vulnerabilities and risk	-	§A Interactions Between the MPCVD Protocol and SSVC (p. 112)
§6 Receiving vulnerability reports	6.1 General	§2 Report Management (RM) Model (p. 9) §4.1.1 The *Vendor Awareness* Substate (v, V) (p. 37) §7.6.1 Process RM Messages Behavior (p. 95)
§6.2 Vulnerability reports	6.2.1 General	§2 Report Management (RM) Model (p. 9) §7.5.3 Reporting Behavior (p. 88)
	6.2.2 Capability to receive reports	§2.1.1.2 The *Received* State (R) (p. 10) §7.6.1 Process RM Messages Behavior (p. 95)
	6.2.3 Monitoring	§7.6 Receiving and Processing Messages Behavior (p. 94)
	6.2.4 Report Tracking	§2 Report Management (RM) Model (p. 9) §7.3 Report Management Behavior Tree (p. 79) §8.1 An MPCVD Case Object (p. 100)
	6.2.5 Report Acknowledgement	§6.6.1 RM Message Types (p. 59) §7.6.1 Process RM Messages Behavior (p. 95)
§6.3 Initial assessment	-	§6.6.1 RM Message Types (p. 59) §7.3.2 Report Prioritization Behavior (p. 81)
§6.4 Further investigation	-	§6.6.1 RM Message Types (p. 59) §7.5 Do Work Behavior (p. 85)
§6.5 On-going communication	-	§6.6 Message Types (p. 58)

Continued on next page

Table B.2 – continued from previous page

29147:2018 Clause	Sub-Clause	MPCVD Protocol Mapping
§6.6 Coordinator involvement	-	§2.2.2 RM Interactions Between CVD Participants (p. 15) §3.2.10 Inviting Others to an Embargoed Case (p. 33) §6.9.4 Coordination With a Coordinator (p. 71) §7.5.3.2 Notify Others Behavior (p. 89)
§6.7 Operational security	-	§8.3.2 Transport Protocol (p. 105)
§7 Publishing vulnerability advisories	all	§7.5.4 Publication Behavior (p. 90) §4.1 CVD Case Substates (p. 37)
	7.3 Advisory publication timing	§3.2.1 Embargo Principles (p. 24)
	7.4 Advisory elements	§8.3.1 Message Formats (p. 104)
	7.5 Advisory communication	§7.5.4 Publication Behavior (p. 90)
	7.6 Advisory format	§8.3.1 Message Formats (p. 104)
	7.7 Advisory authenticity	§8.3.3 Identity Management (p. 105)
	7.8 Remediations	§7.5.1 Deployment Behavior (p. 86)
§8 Coordination	8.1 General	§2.1.1.2 The *Received* State (R) (p. 10) §2.1.1.5 The *Accepted* State (A) (p. 11) §2.2.2 RM Interactions Between CVD Participants (p. 15) §3.2.10 Inviting Others to an Embargoed Case (p. 33) §7 Modeling an MPCVD AI Using Behavior Trees (p. 76)
	8.2 Vendors playing multiple roles	§2.1.1.2 The *Received* State (R) (p. 10) §2.1.1.5 The *Accepted* State (A) (p. 11) §4.1.2 The *Fix Readiness* Substate (f, F) (p. 37) §7.5.3 Reporting Behavior (p. 88)
§9 Vulnerability disclosure policy	9.2.2 Preferred contact mechanism	§7.6 Receiving and Processing Messages Behavior (p. 94)
	9.3.2 Vulnerability report contents	§8.1 An MPCVD Case Object (p. 100)
	9.3.3 Secure communication options	§8.3.2 Transport Protocol (p. 105)
	9.3.4 Setting communication expectations	§3.2.6 Default Embargoes (p. 28) §6.7 Transition Functions (p. 62) §7.3 Report Management Behavior Tree (p. 79)
	9.3.6 Publication	§7.5.4 Publication Behavior (p. 90)
	9.4.3 Disclosure timeline	§3 Embargo Management (EM) Model (p. 20)

B.3 ISO/IEC TR 5895:2022

The recently released ISO/IEC TR 5895:2022 intersects most directly with our topic [18]. Table B.3 contains our mapping of relevant sections of that technical report to our protocol model.

Table B.3: Mapping ISO/IEC TR 5895 Onto the MPCVD Protocol

TR 5895 Clause	Sub-Clause	MPCVD Protocol Mapping
§4 Concepts	§4.2.3 Risk Reduction Effectiveness	§3.2.7 Early Termination (p. 31) §3.2.10 Inviting Others to an Embargoed Case (p. 33)
§5 MPCVD Scenarios in Scope	all	§3.2.10 Inviting Others to an Embargoed Case (p. 33)
§6 MPCVD Stakeholders	all	§1.5 Terms and Definitions (p. 7)
§7 MPCVD Lifecycle	§7.2 Policy Development	§2 Report Management (RM) Model (p. 9) §2.1.1.2 The *Received* State (R) (p. 10) §3.2.6 Default Embargoes (p. 28) §9.1 CVD Directory (p. 106)
	§7.3 Strategy development	§2.2.2 RM Interactions Between CVD Participants (p. 15) §3.2 EM Discussion (p. 24)
	7.4 Know your customers	§2.2.2 RM Interactions Between CVD Participants (p. 15) §3.2.10 Inviting Others to an Embargoed Case (p. 33)
	7.5 Encrypted Communication Methods and Conference Calls	§8.3.2 Transport Protocol (p. 105)
	7.6 Processes and Controls	See NDA note at the beginning of §3 Embargo Management (EM) Model (p. 20)
§8 MPCVD lifecycle for each product	all	§7.5.1 Deployment Behavior (p. 86) §7.5.2 Fix Development Behavior (p. 88) §7.5.3 Reporting Behavior (p. 88) §7.5.4 Publication Behavior (p. 90) $q^{rm} \in A$
§9 MPCVD lifecycle for each vulnerability	§9.1 Receipt	§2.1.1.1 The *Start* State (S) (p. 9) §2.1.1.2 The *Received* State (R) (p. 10) §4.1.1 The *Vendor Awareness* Substate (v, V) (p. 37) §7.6.1 Process RM Messages Behavior (p. 95) §7.5.3 Reporting Behavior (p. 88) $\begin{Bmatrix} q^{rm} \in & S \xrightarrow{r} R \\ q^{cs} \in & vfd\cdots \xrightarrow{\mathbf{V}} Vfd\cdots \end{Bmatrix}$
	§9.2 Verification	§2.1.1.2 The *Received* State (R) (p. 10) §2.1.1.4 The *Valid* State (V) (p. 11) §2.1.1.3 The *Invalid* State (I) (p. 10) §3.2.1 Embargo Principles (p. 24) §7.3.1 Report Validation Behavior (p. 80) §7.3.2 Report Prioritization Behavior (p. 81) §7.5.3 Reporting Behavior (p. 88) $q^{rm} \in R \begin{cases} \xrightarrow{v} V \\ \xrightarrow{i} I \end{cases}$ Emit RV, RI, RA, RD as appropriate.

Continued on next page

Table B.3 – continued from previous page

TR 5895 Clause	Sub-Clause	MPCVD Protocol Mapping
	§9.3 Remediation development	§2.1.1.5 The *Accepted* State (A) (p. 11) §4.1.2 The *Fix Readiness* Substate (f, F) (p. 37) §5.2.2 Fix Ready (p. 47) §7.5.2 Fix Development Behavior (p. 88) $\left\{\begin{array}{c} q^{rm} \in A \\ q^{cs} \in Vfd\cdots \xrightarrow{\mathbf{F}} VFd\cdots \end{array}\right\}$
	§9.4 Release	§4.1.2 The *Fix Readiness* Substate (f, F) (p. 37) §7.5.4 Publication Behavior (p. 90) $q^{cs} \in VFdp\cdot\cdot \xrightarrow{\mathbf{P}} VFdP\cdot\cdot$
	§9.5 Post Release	§7.3.3 Report Closure Behavior (p. 82) §7.5.1 Deployment Behavior (p. 86) $\left\{\begin{array}{c} q^{rm} \in \{A, D\} \\ q^{cs} \in VF\cdot P\cdot\cdot \end{array}\right\}$
	§9.6 Embargo Period	§2.2.2 RM Interactions Between CVD Participants (p. 15) §3.2 EM Discussion (p. 24) §5.1 Interactions Between the RM and EM Models (p. 45) $q^{em} \notin X$
§10 Information exchange		§2.2.2 RM Interactions Between CVD Participants (p. 15) §6.6 Message Types (p. 58) §7.5.3 Reporting Behavior (p. 88)
§11 Disclosure		§3.2.10 Inviting Others to an Embargoed Case (p. 33) §6.9.4 Coordination With a Coordinator (p. 71) §7.5.4 Publication Behavior (p. 90)
§12 Use case for hardware and further considerations		§3.2.10 Inviting Others to an Embargoed Case (p. 33) §5.1 Interactions Between the RM and EM Models (p. 45) §7.5.3 Reporting Behavior (p. 88)

C Embargo Management and the iCalendar Protocol

This appendix is not considered a core part of the *Vultron* MPCVD protocol as proposed in the main part of the report. We include it because the ideas outlined here were instrumental to the development of the more general EM process in the main protocol and may remain of use in future implementations.

The embargo negotiation process—in terms of the proposal, acceptance, rejection, etc.—is strikingly parallel to the process of scheduling a meeting in a calendaring system. To that end, we note the potential application of the `iCalendar` protocol specified in RFC 5445[12] to the EM process with the semantics described in this section. While we anticipate that future CVD APIs could adopt an `iCalendar`-compatible syntax like `jCal` (RFC 7265[13]), for this conceptual mapping, we use the basic `iCalendar` syntax from RFC 5445.

A CVD Case might have an associated `iCalendar` object. Embargo schedules can be represented as a single `VEVENT` object.

A mapping of EM concepts to `iCalendar` field mappings is provided in Table C.1.

- Reflecting the non-binding nature of embargoes, each `ATTENDEE` SHOULD be marked as `ROLE=OPT-PARTICIPANT` in the invitation.
- Vulnerability details MUST NOT appear in the iCalendar data.
- A case or vulnerability identifier SHOULD appear in the `VEVENT SUMMARY` along with the words "embargo expiration."
- Case or vulnerability identifiers SHOULD NOT carry any information that reveals potentially sensitive details about the vulnerability.
- An embargo proposal SHALL set `RSVP:True` for all attendees.

A Participant response with `ATTENDEE:partstat=TENTATIVE` serves as a basic acknowledgment that the embargo proposal has been received, but it does not represent agreement to the proposal.

The `iCalendar ATTENDEE:partstat=DELEGATED` value has no semantic equivalent in the EM process.

C.1 Proposing an Embargo

Following the model of inviting a group of attendees to a meeting, a proposed embargo can be achieved as follows:

1. An `ORGANIZER` sends an embargo invitation, represented by a `VEVENT` with `STATUS:TENTATIVE` listing Participants as `ATTENDEE`s ($q^{em} \in N \xrightarrow{p} P$).
2. Each `ATTENDEE` has `partstat=NEEDS-ACTION` set on the invitation, indicating that they have not yet accepted it.
3. Individual `ATTENDEE`s acknowledge (`partstat=TENTATIVE`), accept (`partstat=ACCEPTED`), or decline (`partstat=DECLINED`). Their response is sent to the `ORGANIZER`.

[12] RFC-5445 Internet Calendaring and Scheduling Core Object Specification (iCalendar) `https://datatracker.ietf.org/doc/html/rfc5545`

[13] RFC-7265 jCal: The JSON Format for iCalendar `https://datatracker.ietf.org/doc/html/rfc7265`

Table C.1: Mapping Embargo Information to `iCalendar` Semantics

Embargo Concept	`iCalendar` Mapping	EM Msg Type
Embargo object	`VEVENT`	-
Embargo ID	`SUMMARY:<case id> embargo expiration`	-
Embargo End Time and Date	`DTSTART` = `DTEND` (0 duration event)	-
Proposer	`ORGANIZER`	-
Participant (proposed)	`ATTENDEE; ROLE=OPT-PARTICIPANT; PARTSTAT=NEEDS-ACTION`	EP,EV
Participant (acknowledge proposal without acceptance)	`ATTENDEE; ROLE=OPT-PARTICIPANT; PARTSTAT=TENTATIVE`	EK
Participant (accepted)	`ATTENDEE; ROLE=OPT-PARTICIPANT; PARTSTAT=ACCEPTED`	EA, EC
Participant (rejected)	`ATTENDEE; ROLE=OPT-PARTICIPANT; PARTSTAT=DECLINED`	ER, EJ
Details (links to case trackers, etc.)	`DESCRIPTION`	-
Embargo status $q^{em} \in P$	`STATUS:TENTATIVE`	EP
Embargo status $q^{em} \in A$	`STATUS:CONFIRMED`	EA, EC
Embargo status $q^{em} \in X$ due to early termination	`STATUS:CANCELLED`	ET
Embargo status $q^{em} \in N$ due to lack of acceptance quorum	`STATUS:CANCELLED`	ER
Other	`CATEGORIES: EMBARGO RSVP: TRUE`	

4. If the ORGANIZER determines that there is a quorum of accepts, they mark the VEVENT as STATUS:CONFIRMED ($q^{em} \in P \xrightarrow{a} A$).
5. If the ORGANIZER determines that there is no sufficient quorum of accepts, they mark the new VEVENT as STATUS:CANCELLED ($q^{em} \in P \xrightarrow{r} N$).

C.2 Embargo Counterproposals

Counterproposals can be achieved in two ways:

1. by declining an initial invitation and then proposing a new one ($q^{em} \in P \xrightarrow{r} N \xrightarrow{p} P$)
2. by proposing a new embargo without declining the first one ($q^{em} \in P \xrightarrow{p} P$)

Either way, this is analogous to requesting a proposed meeting to be shifted to a different time or date prior to accepting the original proposed meeting time. However, following the argument from §3.2.6, we suggest that Participants start by (1) accepting the shortest proposed embargo and (2) proposing a revision to the new embargo instead, which we cover next.

C.3 Proposing a Change to an Existing Embargo

Similar to rescheduling an existing meeting, a proposed change to an existing embargo can be achieved as follows. (This process assumes that an existing embargo is represented by a VEVENT with STATUS:CONFIRMED.)

1. A new proposal is made as a VEVENT with STATUS:TENTATIVE and the same ATTENDEE list as the existing one ($q^{em} \in A \xrightarrow{p} R$).
2. Each ATTENDEE on the new invitation has partstat=NEEDS-ACTION set, indicating that they have not yet accepted the new invitation.
3. Individual ATTENDEEs acknowledge (partstat=TENTATIVE), accept (partstat=ACCEPTED), or decline (partstat=DECLINED). Their response is sent to the ORGANIZER.
4. If the ORGANIZER determines that there is a quorum of accepts ($q^{em} \in R \xrightarrow{a} A$), they
   (a) mark the new VEVENT as STATUS:CONFIRMED
   (b) mark the old VEVENT as STATUS:CANCELLED
5. If the ORGANIZER determines that there is no sufficient quorum of accepts ($q^{em} \in R \xrightarrow{r} A$), they
   (a) mark the new VEVENT as STATUS:CANCELLED
   (b) retain the old VEVENT as STATUS:CONFIRMED

C.4 Terminating an Existing Embargo

Terminating an existing embargo ($q^{em} \in \{A, R\} \xrightarrow{t} X$) can be triggered in one of two ways:

- A *normal* exit occurs when the planned embargo end time has expired.
- An *abnormal* exit occurs when some external event causes the embargo to fail, such as when the vulnerability or its exploit has been made public, attacks have been observed, etc., as outlined in §3.2.7.

Translating this into iCalendar semantics, we have the following, which assumes an existing embargo is represented by a VEVENT with STATUS:CONFIRMED.

1. *Normal termination*: The `VEVENT` retains its `STATUS:CONFIRMED` and passes quietly from the future through the present into the past.
2. *Abnormal termination*: The `ORGANIZER` sets the `VEVENT` to `STATUS:CANCELLED` and sends it out to the `ATTENDEE` list.

The above is consistent with our premise in §3.2.7: Embargo Termination (ET) messages are intended to have immediate effect.

References/Bibliography

URLs are valid as of the publication date of this document.

[1] William A. Arbaugh, William L. Fithen, and John McHugh. Windows of Vulnerability: A Case Study Analysis. *Computer*, 33(12):52–59, 2000.

[2] J. Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert, Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu, Nancy Pollard, et al. An Integrated System for Autonomous Robotics Manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962. IEEE, 2012.

[3] Leyla Bilge and Tudor Dumitraş. Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World. In *Computer and Communications Security*, pages 833–844. ACM, 2012.

[4] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.

[5] David Carney, David Fisher, Ed Morris, and Pat Place. Some Current Approaches to Interoperability. Technical Note CMU/SEI-2005-TN-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, August 2005.

[6] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2020.

[7] OASIS Common Security Advisory Framework Technical Committee. OASIS Common Security Advisory Framework (CSAF) TC. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=csaf`, 2020. Accessed: 2022-06-29.

[8] OASIS Common Security Advisory Framework Technical Committee. OASIS Common Security Advisory Framework (CSAF). `https://oasis-open.github.io/csaf-documentation/`, 2022. Accessed: 2022-03-28.

[9] Stefan Frei, Dominik Schatzmann, Bernhard Plattner, and Brian Trammell. Modeling the Security Ecosystem: The Dynamics of (In)Security. In *Economics of Information Security and Privacy*, pages 79–106. Springer, 2010.

[10] Allen Householder and Jonathan Spring. A State-Based Model for Multi-Party Coordinated Vulnerability Disclosure (MPCVD). Special Report CMU/SEI-2021-SR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, July 2021.

[11] Allen D Householder, Jeff Chrabaszcz, Trent Novelly, David Warren, and Jonathan M Spring. Historical Analysis of Exploit Availability Timelines. In *Workshop on Cyber Security Experimentation and Test (CSET)*. USENIX, 2020.

[12] Allen D. Householder and Jonathan Spring. Are We Skillful or Just Lucky? Interpreting the Possible Histories of Vulnerability Disclosures. *Digital Threats*, Jul 2021. Just Accepted.

[13] Allen D. Householder, Garret Wasserman, Art Manion, and Chris King. The CERT guide to coordinated vulnerability disclosure. `https://vuls.cert.org/confluence/display/CVD`. Accessed: 2022-03-01.

[14] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. The CERT Guide to Coordinated Vulnerability Disclosure. Special Report CMU/SEI-2017-SR-022, Software Engineering Institute, Carnegie Mellon University, 2017.

[15] Damian Isla. GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI. https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai, March 2005.

[16] ISO. Information technology — Security techniques — Vulnerability disclosure. Standard 29147:2018, International Organization for Standardization, Geneva, CH, October 2018.

[17] ISO. Information technology — Security techniques — Vulnerability handling processes. Standard 30111:2019, International Organization for Standardization, Geneva, CH, October 2019.

[18] ISO. Cybersecurity — Multi-party coordinated vulnerability disclosure and handling. Technical Report TR 5895:2022, International Organization for Standardization, Geneva, CH, June 2022.

[19] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Idris Adjerid, and Michael Roytman. Exploit Prediction Scoring System (EPSS). *Digital Threats*, 2(3), July 2021.

[20] Shyamalendu Kandar. *Introduction to Automata Theory, Formal Languages and Computation*. Always learning. Pearson, 1st edition, 2013.

[21] Lisa L. Brownsword, David J. Carney, David Fisher, Grace Lewis, Craig Meyers, Edwin J. Morris, Patrick R. H. Place, James Smith, and Lutz Wrage. Current Perspectives on Interoperability. Technical Report CMU/SEI-2004-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, March 2004.

[22] Art Manion. Apache Log4j allows insecure JNDI lookups. Vulnerability Note VU#930724, CERT Coordination Center (CERT/CC), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, December 2021.

[23] Michael Mateas and Andrew Stern. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4):39–47, 2002.

[24] Forum of Incident Response and Security Teams. Common Vulnerability Scoring System v3.1: Specification Document. https://www.first.org/cvss/v3.1/specification-document, 2019. Accessed: 2022-06-29.

[25] Petter Ögren. Increasing Modularity of UAF Control Systems Using Computer Game Behavior Trees. In *AIAA Guidance, Navigation, and Control Conference*, page 4458, 2012.

[26] Oxford English Dictionary Online. protocol, n. https://www-oed-com.cmu.idm.oclc.org/view/Entry/153243, December 2021. Accessed February 17, 2022.

[27] Jonathan M Spring, Eric Hatleback, Allen D. Householder, Art Manion, and Deana Shick. Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization. White paper, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2019.

[28] Jonathan M Spring, Eric Hatleback, Allen D. Householder, Art Manion, and Deana Shick. Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization. In *Workshop on the Economics of Information Security*, Brussels, Belgium, December 2020.

[29] Jonathan M Spring, Allen D. Householder, Eric Hatleback, Art Manion, Madison Oliver, Vijay Sarvapalli, Laurie Tyzenhaus, and Charles Yarbrough. Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization (Version 2.0). White paper, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2021.

[30] Chris Turner and David Waltermire. NIST Vulnerability Data Ontology. https://github.com/usnistgov/vulntology, 2022. Accessed: 2022-06-29.

[31] Rob Wright. Lessons learned from Meltdown and Spectre disclosure process. https://www.techtarget.com/searchsecurity/news/252446793/Lessons-learned-from-Meltdown-and-Spectre-disclosure-process, August 2018.

Acronym List

ACM	Association for Computing Machinery
AI	Artificial Intelligence
API	Application Programming Interface
CERT/CC	CERT Coordination Center at the Software Engineering Institute (SEI)
CISA	Cybersecurity and Infrastructure Security Agency, part of the United States of America (US) Department of Homeland Security (DHS)
CNA	Common Vulnerabilities and Exposures (CVE) Numbering Authority
CS	Coordinated Vulnerability Disclosure (CVD) Case State
CSAF	Common Security Advisory Framework by OASIS
CVD	Coordinated Vulnerability Disclosure
CVE	Common Vulnerabilities and Exposures by The MITRE Corporation (MITRE)
CVSS	Common Vulnerability Scoring System, maintained by Forum of Incident Response and Security Teams (FIRST)
DAG	Directed Acyclic Graph
DFA	Deterministic Finite Automaton
DHS	US Department of Homeland Security
DTRAP	Digital Threats: Research and Practice, an Association for Computing Machinery (ACM) journal
EM	Embargo Management
FFRDC	Federally Funded Research and Development Center
FIRST	Forum of Incident Response and Security Teams
FSM	Finite State Machine
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IPS	Intrusion Prevention System
ITSM	Information Technology Service Management
JSON	JavaScript Object Notation
MITRE	The MITRE Corporation
MPCVD	Multi-Party Coordinated Vulnerability Disclosure
NDA	Non-Disclosure Agreement
NIST	National Institute of Standards and Technology, part of the US Department of Commerce
NPC	Non-Player Character
OEM	Original Equipment Manufacturer
OT	Operational Technology
OWL	Web Ontology Language, a World Wide Web Consortium (W3C) standard
PoC	Proof of Concept
protobuf	Protocol Buffers
REST	Representational State Transfer
RM	Report Management
SAAS	Software-as-a-Service
SAML	Security Assertion Markup Language
SBOM	Software Bill of Materials
SEI	Software Engineering Institute, a Federally Funded Research and Development Center (FFRDC) operated by Carnegie Mellon University
SSVC	Stakeholder-Specific Vulnerability Categorization
UML	Unified Modeling Language
US	United States of America
VDP	Vulnerability Disclosure Program

W3C	World Wide Web Consortium
XML	Extensible Markup Language, a W3C standard
XMPP	Extensible Messaging and Presence Protocol, an Internet Engineering Task Force (IETF) standard
XSD	Extensible Markup Language (XML) Schema Definition, a W3C standard

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2022	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Designing Vultron: A Protocol for Multi-Party Coordinated Vulnerability Disclosure (MPCVD) (Vultron v0.4.0)	FA8702-15-D-0002

6. AUTHORS
Allen D. Householder

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2022-SR-012

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
SEI Administrative Agent AFLCMC/AZS 5 Elgin Street Hanscom AFB, MA 01731-2100	N/A

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT	12B. DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Coordinated Vulnerability Disclosure (CVD) process addresses a human coordination problem that spans individuals and organizations. In this report, we propose a formal protocol specification for Multi-Party Coordinated Vulnerability Disclosure (MPCVD) with the goal of improving the interoperability of both CVD and MPCVD processes. The *Vultron* protocol is composed of three interacting Deterministic Finite Automata (DFAs) for each CVD case Participant representing the Report Management (RM), Embargo Management (EM), and CVD Case State (CS) processes. Additionally, we provide guidance and commentary on the associated MPCVD Participant capabilities and behaviors necessary for this interoperability to be realized.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Coordinated Vulnerability Disclosure, CVD, Multi-party Coordinated Vulnerability Disclosure, MPCVD, Vulnerability Report, Embargo, Vulnerability Disclosure Program, VDP, Vultron, Protocol, Deterministic Finite Automata, DFA, Behavior Tree, Formal Protocol	145

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102