Artificial Intelligence Cyber Challenge (AIxCC): AIxCC Semifinal Competition (ASC) Procedures and Scoring Guide

06/10/2024
Version 4

Defense Advanced Research Projects Agency
Information Innovation Office

675 North Randolph Street
Arlington, VA 22203-2114

Document Change Summary

The AIxCC competition guidelines will be updated throughout the competition period (Fall 2023 – August 2025). Please check for updates regularly and send any questions or feedback to aixcc@darpa.mil.

	Section	Description	Date
1	All	Initial Release: Request for Comments	12/13/2023
2	All	Final Release: Updated based on public feedback	03/29/2024
3	3.2.4, 3.2.5, 3.2.6, 3.2.7, 3.2.8	Additional information on CRS Specifications	04/19/2024
4	2.2.9, 2.2.10, 3.3, 6.0 – Appendix B.	AIxCC Runtime details, LLM Rate limits, additional examples in the Disqualification Guidelines and Scoring Algorithm explainer	06/10/2024

Table of Contents

1. AIxCC Semifinals Competition (ASC) Overview

After examining public feedback collected over the past few months, the AIxCC Organizers have made updates to the AIxCC Semifinals Competition (ASC). These revisions fulfill multiple objectives, all converging toward one goal: facilitating a fair, successful, and impactful ASC. The sections below summarize these changes.

This ASC Procedures and Scoring Guide will serve as the authoritative source of ASC format and procedures. This ASC Procedures and Scoring Guide does not supersede the AIxCC Rules, which can be found on the AIxCC website: https://aicyberchallenge.com/rules/.

1.1 Challenge Projects

The makeup of Challenge Projects (CPs) is being modified to include a Git repository of the CP's source code. The Git repository history will not reflect each CP base's history; instead, it will contain a large initial commit (i.e., a custom fork and reset of the project) followed by some number of commits created, curated, and refactored by the AIxCC Organizers. It is only these latter commits that are in scope for the ASC in terms of vulnerability discovery and patching. These commits will include benign code, custom or Introduced Vulnerabilities, and potentially real-world (0-day and n-day) vulnerabilities. The competitor Cyber Reasoning Systems (CRSs) will need to focus their efforts on changes introduced by the commits following the initial commit.

1.2 CRS Vulnerability Discovery and Patching

Given the above changes to CPs, the Vulnerability Discovery (VD) Submissions must now include the Git commit hash for the commit that is suspected to have introduced a Challenge Project Vulnerability (CPV), also referred to as the Bug-Inducing Commit (BIC). The AIxCC Organizers are introducing no more than one CPV per commit, so the VD Submission must call out the BIC that is source of the CPV.

1.3 Scoring and Evaluation

To focus competitor effort on vulnerability discovery and remediation, the AIxCC Organizers are eliminating the maintainability/linting score from the Program Repair Score calculation, detailed in the ASC Scoring Algorithm. The Generated Patch (GP) Submissions will not be evaluated based on code formatting during the ASC. The AIxCC Organizers reserve the right to add this evaluation/scoring factor back for the AIxCC Finals Competition (AFC).

To streamline the automated evaluation of GPs while empowering Teams to discover and fix real-world vulnerabilities if they are present in the CPs, the Scoring Server will no longer give an indication to the CRS if a GP submission successfully fixed a CPV and scored points. The Scoring Server will provide an indication if a GP was rejected or deemed invalid (e.g., improperly formatted, did not apply, or failed to build), but CRSs will not know if the CPV is fixed. This evaluation change is true for both AIxCC-introduced CPVs and real-world CPVs that may be present. Furthermore, the evaluation of GPs that target real-world CPVs will be conducted offline after the ASC.

1.4 Rules

With the inclusion of Git repository of CP source code, the AIxCC Organizers are eliminating the "No Diffing" rule. The AIxCC Organizers are modifying the CP bases' source code and histories in ways that likely diminish the effectiveness of many diffing strategies; however, diffing repositories across various states and points in time are valid and common analysis techniques are no longer outlawed.

One of the goals of the ASC is to assess what is possible with the foundational large language model (LLM) frameworks that the AIxCC Collaborators are offering to the CRS competitors. To address this research question while ensuring the availability of common computing resources to all CRS Teams during the ASC, the AIxCC Organizers are restricting LLM usage during the ASC to exclude any custom LLMs. All LLM usage for the ASC must go through a specified LLM interface proxy and must target only the limited set of foundational LLM providers and models that the AIxCC Organizers will specify by May 1, 2024.

1.5 Request for Comments for AIxCC Finals Competition (AFC) 2025

Please submit AFC feedback and ideas by emailing AIxCC@darpa.mil or via the AIxCC website FAQ page (https://aicyberchallenge.com/faqs/). One of the goals of the ASC is to foster innovations that surpass today's state-of-the-art advancements — extending beyond the scope of short term, incremental progress. Therefore, participants should not let the prospect of near-term, incremental enhancements constrain their creative vision for the AFC.

2.0 ASC Procedures: Challenge Projects

2.1 Challenge Project Overview

For the ASC, Teams will receive an identical corpus of Challenge Projects (CPs). Each CP has a basis, which is a real-world, open-source project that is widely used. CPs will contain, at a minimum, the following:

- A Git repository containing source code for the challenge with vulnerabilities introduced;
- One or more test harnesses that exercises CP functionality with CRS provided data;
- General functionality tests;
- Modifiable build process and build environment; and
- Standard scripted interface for building, patching, and running functionality tests on the CP.

2.1.1 ASC Bases

Each CP has a *basis* in a real-world, open-source project that is critical to industry, national security, and the public. The ASC will include up to seven (7) CPs, with one or more CPVs per CP. CPs will contain significant amounts of added and modified content, including but not limited to:

- An unspecified number of AIxCC-introduced vulnerabilities;
- An unspecified number of commits derived from historical commits;
- Additional features, behavior, and functionality;
- Test harnesses reaching various coverage amounts; and
- Functionality and security tests.

See Exemplar Challenges for more information about CPs that will be published prior to the ASC competition.

2.1.2 ASC Languages

The ASC will focus on vulnerabilities found in the following two (2) languages:

- C
- Java

While the CP bases and test harnesses may contain code in a variety of languages, only vulnerabilities found in C and Java are in scope for the ASC. A CP's harness or harnesses define which parts of a given CP's source code are relevant for that CP. Memory-corruption vulnerabilities will be the focus of a significant portion of the vulnerabilities.
Teams must not make any assumptions about the standard for the languages, such as ANSI C or C99, as each CP basis follows its own development process and coding standards.

2.1.3 ASC Harnesses

Each CP will come with one or more harnesses, which interact with a portion of the CP. Harnesses exercise functionality using data provided by the CRS. Harnesses will not exercise all functionality in a CP.

No harnesses used in the competition will be disclosed prior to the ASC. The harnesses in the exemplar challenges are for example only.

2.1.4 ASC Challenge Project Vulnerabilities

CPs will contain vulnerabilities that must be identified and secured, called Challenge Project Vulnerabilities (CPVs). The number of CPVs in a given CP will not be disclosed to competitor. Vulnerabilities discovered by the CRS will fall into one of two categories:

- **Introduced Vulnerabilities:** CPVs will be intentionally introduced into the CPs. These vulnerabilities will be fully synthetic or based on a real-world vulnerability, including publicly known vulnerabilities. Less than one-third (1/3) of introduced vulnerabilities will be based on pre-existing vulnerabilities. Each introduced vulnerability will be isolated to a single commit in the CP repository.
- **Real-world Vulnerabilities:** Since CPs are based on real-world software, vulnerabilities that were not intentionally introduced may be discovered by a CRS. Those vulnerabilities that exist, if present and discovered/fixed, will be worth the same points as **Introduced Vulnerabilities.**

2.1.5 Vulnerability Classes and Sanitizers

Vulnerabilities will be assessed based on their ability to trigger a specified AIxCC Sanitizer. For ASC, sanitizers will be based on KASAN, KFENCE, ASan, MemSan, UBSan, and Jazzer. The Common Weakness Enumerations (CWEs) that are in scope for ASC will be provided to competitors in the AIxCC GitHub infrastructure.

2.1.6 Challenge Project Tests

Competitor CRSs will only receive the functionality tests that are available in the CP bases.

2.1.7 Public Test Suite

The public test suite is released to the CRS as part of the CP. This suite tests the intended functionality of the CP after a patch has been applied to address the CPV.

2.1.8 Private Test Suite

These are tests run by the Scoring Server (not disclosed to the CRS or Teams), and include:

- additional functionality tests

- test cases that trigger known vulnerable code paths

2.1.10 Exemplar Challenges

The AIxCC GitHub infrastructure will contain the repository for the Linux Kernel exemplar CP, which is structured based on the current CP specification. This exemplar will serve as the best technical guide for competitors in developing their CRSs and interacting with CPs. Prior to the ASC, to facilitate the development and maturation of competitor CRSs, the AIxCC Organizers will release additional exemplar CPs. Please see the Master Schedule in the AIxCC Rules for additional information.

At a minimum, both the Linux Kernel and Jenkins CP bases will also appear in the ASC; therefore, Teams are strongly encouraged to use the exemplars when developing and exercising their competition CRSs. Furthermore, there will be an opportunity for Teams to exercise their in-development CRSs and the exemplars during the AIxCC Preliminary Events, which will involve the exemplar CP releases, the Challenge Project Sandbox, and the CRS Evaluation Window. Teams should monitor the AIxCC website (https://www.aicyberchallenge.com) for more details about the AIxCC Preliminary Events.

2.2 Challenge Project Specifications

During the ASC, each CP will be provided to a CRS as a `git` repository. The methods and interfaces to interact with CP from within a CRS are determined by the APIs and protocols that fall under the CRS Specification.

The AIxCC Organizers selected the oss-fuzz project's design as a reference when creating the structure of the CP repository. The ASC will build upon and modify the `oss-fuzz` process to suit the needs of the target CP bases and unique ASC structure. Do not assume that everything in the ASC and CP specifications will follow exactly the `oss-fuzz` reference. The CP specification and exemplar CPs are the ultimate source of truth for the ASC and CPs.

2.2.1 CP Repository Structure

A CP repository contains the following files and directories.

- `src/`
  - This directory contains the source code for the CP, which includes all injected vulnerabilities (CPVs) and testing-related files and modifications. The source code will contain its own git history that a CRS will use to discover vulnerabilities.
- `project.yaml`
  - This file will define many helpful details that a CRS will use to interact with the CP. To include names, target languages (e.g., C), testing harness information, etc.

This file includes identifiers that are relevant to fields expected in the Proof of Vulnerability and Proof of Understanding, such as the applicable test harness and sanitizers.

- `Dockerfile`
  - The CP-specific Docker build/configuration file. The other scripts in this repo will use this file to generate a Docker image that is used for building and testing the CP.
- `build.sh`
  - This file will build the CP, test harness binaries, and functionality test binaries (if needed). This file will be run within the Docker image used for dynamic testing.
  - Offers environment variable overrides to customize the build process for CRS analysis needs.
  - Examples of these include: `$CC $CXX $CCC ; $CFLAGS $CXXFLAGS`
- `run.sh`
  - Provides a standardized interface to interact with the CP and other scripts.

Note, the makeup of the CP repository is subject to change before the ASC. There is no expectation that a major overhaul will be necessary; instead, it is an acknowledgement that the needs and experiences of the competitors and the AIxCC Organizers will likely dictate some refinements as infrastructure evolves.

The exemplar CP repositories will contain additional files and resources to explain the usage of the CP and demonstrate both valid and invalid scenarios. Please consult the exemplar CP repositories for additional technical information and guidance.

CRSs will be free to modify the above files as desired for their analysis needs.

2.2.2 CP Usage

The `run.sh` script is the primary method to interact with the CP within the repository. All CPs will include a run.sh script that supports identical commands. The AIxCC Organizers will maintain an up-to-date version of run.sh in the CP Sandbox which is scheduled to be released on April 15, 2024. The Linux Exemplar Challenge README.md along has a version of the documentation that can be used until the CP Sandbox release. A CRS is free to interact with the CP by any means necessary, and with the `Dockerfile` and images as desired. By default, a CP can be built, tested, and patched using the `run.sh` script.

Each of these commands will use the Docker image that is built from the provided `Dockerfile` that is in the repository.

The `run.sh` script will apply the supported environment variables to the internal Docker commands if the variables are set when the script is invoked, (e.g., `CFLAGS=-Werror ./run.sh build`).

Additionally, there is nothing that limits a CRS from interacting with and/or modifying commands and options for the ASC; however, the burden to implement and thoroughly exercise any such modifications falls entirely on the ASC Competition Team. The AIxCC Organizers will not release all CP bases before the competition, and the AIxCC Organizers will not support or triage competitors' use of custom tooling and scripts.

3.0 ASC Procedures: Cyber Reasoning Systems (CRS)

3.1 Cyber Reasoning System Overview

Each Team will develop a CRS capable of automatically processing a set of ASC Challenge Projects (CPs). The aim is to find and fix Challenge Project Vulnerabilities (CPVs).
Given a CP, a CRS will perform:

1. Vulnerability Discovery
2. Patching

The design of a CRS must adhere to the CRS Rules. The specifications and requirements to which a CRS must adhere are described in CRS Specification.

3.1.1 Vulnerability Discovery (VD)

Throughout the competition, CRSs will analyze the software in a CP and find vulnerabilities, or CPVs.

To score points for a discovered vulnerability, the CRS must provide a Vulnerability Discovery (VD) Submission, which contains both a Proof of Vulnerability (PoV) and a Proof of Understanding (PoU).

3.1.2 Proof of Vulnerability (PoV)

CPs will include "test harnesses" that exercise CP functionality with CRS provided data. To demonstrate PoV, CRSs will submit a test case that demonstrates the identified vulnerability in the form of data passed to these harnesses.

PoVs will be evaluated by a private suite of sanitizers (AIxCC Sanitizers), which will be used to confirm the vulnerability exercised in the test case. If the submitted test case vulnerability cannot be confirmed by these means, the submission will be rejected.

Rejected submissions will negatively impact the Team score as outlined in Section 1.4.3: Accuracy Multiplier.

3.1.3 Proof of Understanding

CRS must submit the following with every PoV:

1. the Git commit hash for the bug-inducing commit
2. the AIxCC Sanitizer expected to catch this vulnerability

To be eligible for PoV points, the sanitizer called out in the PoU must match the sanitizer triggered in the PoV and the vulnerability must exist in the latest version of the CP (HEAD commit).

3.1.4 Vulnerability Discovery Submission

CRS will submit VD Submissions to the Scoring Server, and the Scoring Server will receive and attempt to parse submissions. On success, a unique identifier associated with the VD Submission (VD_UUID) will be returned to the CRS. On failure, the CRS will be notified but will not receive a VD_UUID.

A complete VD Submission to the Scoring Server must contain:

1. CP Identifier
2. PoV
3. PoU

Incomplete submissions will not be scored and will not impact the Team score. Incomplete submissions refer to submissions that don't include all the required information or are improperly formatted and will therefore be rejected by the API.

A CRS may request an update from the Scoring Server on the status of their VD Submission (using the VD_UUID provided). If a VD Submission properly discovers a vulnerability and scores, the CRS will receive a unique identifier representing the discovered vulnerability (CPV_UUID).

3.1.5 Patching

Over the course of the competition, CRSs will generate fixes for discovered CPVs. These will be modifications, or patches to the source code of the CP that remove the vulnerability while preserving the software's intended functionality.

The patches for CPVs that the CRS creates are referred to as Generated Patches (GPs).

3.1.6 Generated Patches

To demonstrate that a CRS can patch vulnerabilities, a CRS will submit a GP that will be applied to the source code's latest version.

A GP submission must specify a singular CPV that it aims to remediate by passing a CPV_UUID along with the patch. CRSs will submit source code patches in unified diff format, which are applied against the original program for validation. For ASC, individual patches will be validated independently against the original CP. Patches will not be validated in conjunction with any other submitted patch.

The patch evaluation process will be an extensive process aimed at ensuring patch correctness and quality. The assessment will primarily focus on the following criteria:

1. **GP's ability to retain desired program functionality**. Competitor patches must not only address vulnerabilities but also preserve the intended behavior of the software. Generate-and-validate approaches are prone to issues of patch overfitting. The AIxCC Organizers will mitigate this by providing a high-quality test suite, withholding test cases in the private test suite, and using additional static and runtime analyzers.
2. **GP's effectiveness in successfully remediating vulnerabilities**. The private test suite will exercise tests and vulnerability-confirming sanitizers, ensuring that vulnerabilities are being remediated. For ASC, the AIxCC Organizers will not measure vulnerabilities introduced by competitor-generated patches and reserve that metric for consideration in future competitions.

3.1.7 Generated Patch Submission

A CRS submits a Generated Patch (GP) Submission to the Scoring Server. The Scoring Server will provide a response to the CRS on whether the submission was received and successfully parsed by either providing a unique identifier associated with that GP submission (GP_UUID) or returning an indication of failure.

A complete GP Submission to the Scoring Server must contain:

1. Valid CPV Identifier (see CPV_UUID)
2. Unified Diff Patch File

Incomplete and invalid (e.g., no corresponding VD Submission) submissions will not be scored and will not impact the Team score. For specific submission format, refer to the CRS Specification.

A CRS may request an update from the Scoring Server on the status of their GP Submission (using the GP_UUID provided). The Scoring Server will return an error for two scenarios: 1) if the patch fails to properly apply against the CP, or 2) if the CP doesn't build with the patch applied. These errors will be the only indication that the GP failed and will negatively impact the

score. The Scoring Server will return a success code if the GP is applied and built, but there will be **NO** feedback regarding the GP's success or failure in remediating the CPV.

The full list of error codes generated in response to GP submissions will be provided to all verified AIxCC competitors in their private GitHub repository.

3.2 ASC CRS Specifications

3.2.1 Competition GitHub Repositories

Each competitor Team will be provided with a private GitHub repository. This repository will be copied from a reference repository, the `CRS Sandbox`. The reference CRS Sandbox should be treated as the single place of reference for how to implement a CRS. There will be a preliminary release of this repository during the March Kickoff Event so Teams can understand the structure; however, it will be subject to revisions leading up to the ASC. The upstream `CRS Sandbox` repository will follow a strict tagged version release process.

The reference `CRS Sandbox` repository will contain specifications and example code from which competition Teams can develop their CRSs. The reference `CRS Sandbox` is expected to go through revisions, but no further changes will be made to the upstream `CRS Sandbox` after July 1, 2024 at 9:59AM UTC. It is anticipated that changes will be required to stay current with the upstream AI feature sets in the accessible LLM APIs.

Teams will be notified of these changes and publishing of tagged releases, but the individual Teams are responsible for rebasing their private `CRS Sandbox` repositories against the upstream `CRS Sandbox` repository.

Prior to the CRS submission deadline on July 15, 2024, each Team must tag their `CRS Sandbox` repository with a designated version string (details are forthcoming) and push the tag to the remote GitHub repository. The AIxCC Organizers will obtain the tagged version of each competitor's `CRS Sandbox` repository by the CRS submission deadline.

3.2.2 CRS Sandbox Structure

The `CRS Sandbox` will contain a `README.md` file. This file will contain the versioned requirements and/or pointers to a specification file within the repository. Competitors should treat `README.md` as the single source of truth for the constraints and specifications needed to build a compatible CRS. Each `CRS Sandbox` will contain a `docker-compose.yaml` file, which will contain the configuration needed to run the CRS locally. The AIxCC Organizers will be automatically translating the `docker-compose.yaml` to Kubernetes resources via Kompose V3. The V3 variant of Kompose supports the `deploy` section which permits Teams to

specify replica sets. The CRS Sandbox will also contain a Mock CRS which competitors will replace with their own implementation.

The docker-compose.yaml file will specify a container for an API shim, referred to as the iAPI (internal API). The iAPI will be used for all inputs and outputs (I/O) to the CRS; however, it will only perform lightweight specification validation and pass data to an upstream API that will be available closer to the CRS submission deadline, July 15, 2024. The iAPI specification will be subject to change; however, all changes will be versioned and released through the upstream CRS Sandbox. The specific versions of Kubernetes and Kompose will be communicated in the release of the CRS Sandbox.

The CRS Sandbox will contain several environment variables which point to upstream resources needed at competition time. Environment variables are used so we can easily test this within the CRS Sandbox for local development, but then redirect to upstream resources as needed.

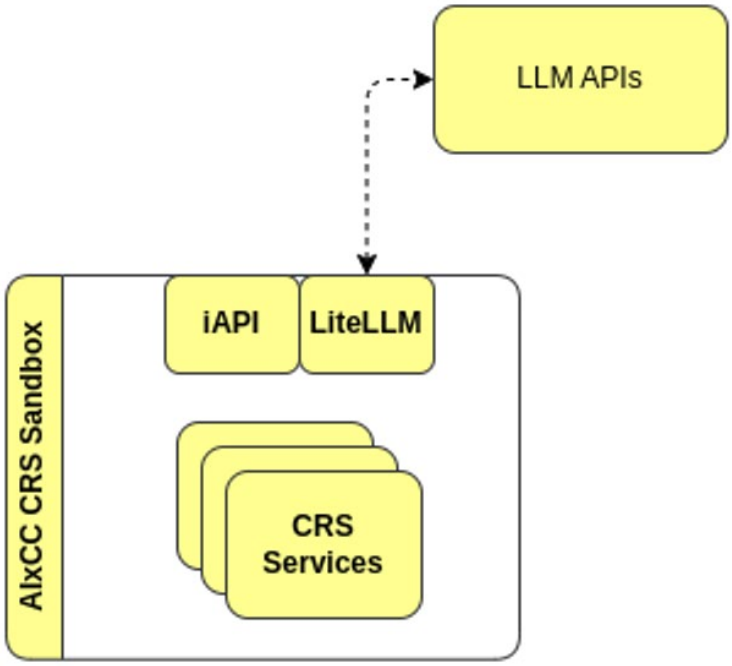
The two variables used for these resources will be:

AIXCC_LITELLM_HOSTNAME
AIXCC_API_HOSTNAME

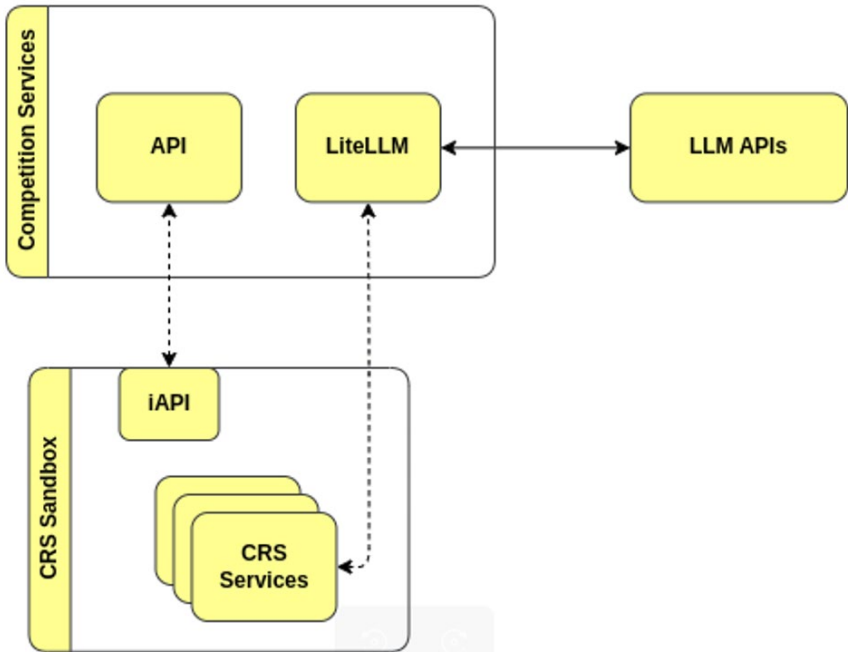
3.2.3 CRS Architecture

The following diagrams depict the CRS Sandbox during the development phase, and what it will look like during competition. The major difference is during development the iAPI will not be connected to the upstream API, and competitors will run a local version of LiteLLM. At competition the iAPI will be connected to the competition API, and the CRS will connect directly to LiteLLM. This ensures competitors can fully test and validate their CRS on a local development laptop or use a script which will be provided to run Kompose to convert their docker-compose script to Kubernetes manifests, which they can apply to larger infrastructure for testing.

Development Architecture:



Competition Architecture:



3.2.4 Compute Resources

Competitors must update the `docker-compose.yaml` file for their specific CRS. AIxCC staff will use logic to automatically convert CRSs to Kubernetes resources using Kompose V3.

At competition time, each CRS will have a dedicated Kubernetes cluster with **three (3) Kubernetes nodes**.

Each Kubernetes Node will possess or closely resemble the following specifications:

```
Processor: AMD EPYC 9004 (Genoa)
Number of Cores: 64 vCPU
Memory: 256 GB
Storage: 2.4TB NVMe
```

3.2.5 Large Language Model APIs

All interactions with LLMs must use the OpenAI API specification. AIxCC will be using the LiteLLM open-source project to serve as the central OpenAI API proxy for the competition. This is to support the level of transparency, visualization, and standardization required to have a fair competition. The version specified now (`LiteLLM OpenAI Proxy - v1.23.9`) is subject to change to stay up to date with the latest feature set of the project.

The use of `LiteLLM` as a proxy **does not** require that teams use `LiteLLM` clients or SDK. Instead, all LLM requests must go through the proxy to reach OpenAI, Anthropic, AzureML, and Google Gemini. This will require that teams use the OpenAI schema specified in the LiteLLM documentation.

3.2.6 Networking

The CRS **will not** have access to the internet at competition time. This constraint will enable the AIxCC team to maintain a more reliable competition framework as well as limit the attack surface and ensure fairness.

Teams are highly encouraged to test their CRS in a disconnected state to ensure the system can build and run correctly.

3.2.7 Build Artifacts

Teams should utilize their GitHub `CRS Sandbox` repository for (Open Container Initiative (OCI) images or other built artifacts. GitHub Large File Storage (LFS) will be available as well. No other container registry, image repository, or artifact storage will be permitted.

3.2.8 Constraints

This section will provide a high-level summary of the constraints that will be imposed on each CRS during competition.

- The ASC CRS competition environment **will NOT** have internet access.
- All LLM API requests **must** go through the AIxCC provisioned LiteLLM OpenAI proxy using the OpenAI schema. The proxy will be configured to allow access to OpenAI, Anthropic, AzureML, and Google Gemini.
- Teams must **NOT** incorporate custom trained/tuned LLMs in their CRSs for the ASC.
- Teams must **NOT** place any code into the `proprietary` folder without explicit written permission. If approved, teams must use the process outlined in `proprietary/REQUIREMENTS.md`.
- Teams should **ONLY** change code in areas as directed by the `README.md` documentation in the `CRS Sandbox` repo. Failure to comply may result in disqualification.

3.2.9 Competition Runtime

The ASC is designed to identify the most promising CRSs for the AFC by rigorously evaluating their performance across the AIxCC AoE. The ASC will be organized into a series of rounds and in each round a CRS will analyze a single CP. Each round will last four (4) hours. The AIxCC Organizers carefully designed this format to minimize gamesmanship and thoughtfully drew from feedback provided during the RFC period. This format is also intended to encourage developers to focus their efforts on creating robust and versatile systems capable of effectively identifying and fixing vulnerabilities across a diverse array of large-scale projects. By assessing the CRSs' performance in this controlled environment, the ASC serves as a critical cornerstone for demonstrating the most advanced and reliable cybersecurity solutions, setting the stage for impact-driven results at the AFC.

3.2.10 CRS LLM Query Capacities

Query Capacities

To fairly allocate available capacity across the competition, the ASC is providing each CRS with a maximum budget and the following query capacities:

1. Total Dollars per CP
2. Rate Limits
    i. Requests per Minute (RPM) or Queries per Minute (QPM) per Model
    ii. Tokens per Minutes (TPM) per Model

1.) Total Dollars per Challenge Project

The ASC will provide each CRS four (4) hours per Challenge Project (CP) to solve each CP in series. For each CP, the CRS will be able to spend up to the Total Dollars per CP. This budget will renew for each CP and can be used to query any of the allowed models.

The Total Dollars per CP is:

$100.00.

Model pricing during the competition will match the providers' publicly listed prices. Check each provider's pricing page for the most current information.

2.) Rate Limits

The ASC rate limits have two dimensions: (i) Requests per Minute (RPM/QPM) per Model and (ii) Tokens per minute per Model.

Rate limits are implemented using the LiteLLM-proxy repository. For more details about the proxy configuration please review the crs-sandbox repository[1]

i) Requests per Minute is the maximum number of queries that can be made to a specific model each minute.

ii) Tokens per Minute is the maximum number of tokens that can be sent and received from a specific model per minute.

[1] https://github.com/aixcc-sc/crs-sandbox/tree/main/sandbox/litellm

Company	Model	RPM/QPM	TPM
Anthropic	Claude 3 Haiku	1,000	100,000
Anthropic	Claude 3 Sonnet	1,000	80,000
Anthropic	Claude 3 Opus	1,000	40,000
OpenAI	GPT-4o	400	300,000
OpenAI	GPT-4-turbo	400	60,000
OpenAI	GPT-4	200	20,000
OpenAI	GPT-3.5-turbo	800	80,000
OpenAI	text-embedding-3-large	500	200,000
OpenAI	text-embedding-3-small	500	200,000
Google	Gemini 1.5 Pro	120	Pending
Google	Gemini 1.0 Pro	120	Pending
Google	text-embedding-004	Pending	Pending

3.3 CRS Disqualification Guidelines

To ensure that a fair and productive competition is conducted, the AIxCC Organizers reserve the right to introduce new competition rules at any time. ASC results and submissions will be audited by the AIxCC Organizers, ensuring that Teams comply with both the explicitly stated restrictions and the underlying ethos of the competition. To validate AIxCC results, the infrastructure Team will deploy various sensors and create auditable artifacts throughout the project's lifecycle.

The examples below constitute approaches, activities, and methods that will result in disqualification from the ASC – applicable both before and after the competition begins. Should a competition Team have concerns that their strategy may be viewed as contrary to the competition's ethos or received unfavorably, they are encouraged to contact the AIxCC Organizers for feedback. Likewise, Teams are encouraged to contact the AIxCC Organizers if they suspect that an external actor or another Team is targeting them or interfering with their AIxCC-related work.

3.3.1 No Superman Defenses

Attempts to resolve CPVs with unrealistic patches that hide an issue will be considered against the spirit of the competition. For example, resolving a double-free CPV by eliminating all calls to `free()` is not a valid or reasonable solution.

3.3.2 No Malicious Patches

The patches submitted during competition are meant to fix CPVs while preserving functionality. Any patches that introduce new functionality or attempt to score points without finding and

fixing vulnerabilities will be considered malicious. This includes patching techniques that cause the CP to avoid sanitizer detection.

In the case of the Linux Kernel, an example of a malicious patch would include disabling or subverting the kernel sanitizer checks/macros. For Java, a GP must not add/modify a SecurityManager to mitigate the vulnerability. Other violations would include forking a new process to avoid sanitizer detection or to isolate the vulnerability.

3.3.3 No Phoning Home

Per the CRS Specification, during the competition the ASC environment will only allow access to limited resources. Any attempts to circumvent the network restrictions will be considered malicious.

3.3.4 No Gaming the Scoring Algorithm

The ASC Scoring Algorithm is intended to be robust and motivating for specific research goals; however, the AIxCC Organizers cannot rule out loopholes in the design / implementation. It will be against the spirit of the competition if the AIxCC Organizers discover that a Team sought to exploit the scoring algorithm without addressing CPVs.

3.3.5 No Hacking the Infrastructure

Manipulating, tampering, or subverting the ASC infrastructure (before or during the competition) will be treated as malicious.

3.3.6 No Misuse of Collaborator Credits and Resources

AIxCC Collaborators are generously providing their resources to support this effort. Competitors must comply with Collaborators' applicable Terms of Service. Use of these resources or credits for activities unrelated to AIxCC is prohibited. Misuse may lead to disqualification from AIxCC and/or penalties under the terms and conditions of other vendors and services.

3.3.7 No Custom LLM Models

For the ASC, the AIxCC Organizers and Collaborators are focused on providing a fair competition that evaluates how foundational LLMs can augment the vulnerability discovery and remediation process. As such, competitor CRSs cannot include custom trained or tuned LLMs for the ASC. All LLM usage and access must go through the LiteLLM proxy, and only the designated LLM versions are allowed. The AIxCC Organizers have specified these designated LLMs in the crs-sandbox repository[2] on github .

[2] https://github.com/aixcc-sc/crs-sandbox/tree/main/sandbox/litellm

3.3.8 No Gaming the Challenge Project Code Base

Any attempt to distinguish between original CP code and ASC synthesized commits to look for authorship patterns or indicators that are unrelated to the functionality or security elements of the code changes is against the spirit of the competition and will likely result in disqualification. This includes using code-authorship detection techniques to identify git commits that were authored by AIxCC organizers to aid with vulnerability discovery; compare the CP's git history with your own copy of the public base-CP's git history to aid with vulnerability discovery; and/or comparing the CP's code with our own copy of the CP source code to aid with vulnerability discovery.

4.0 ASC Scoring Algorithm

Artificial Intelligence Cyber Challenge (AIxCC) Teams will be evaluated at the AIxCC Semifinal Competition (ASC) according to the scoring algorithm described in this document.

4.1 Scoring Algorithm Objectives (SAO)

Based on lessons learned from past DARPA Challenges,[3] the scoring algorithm described in this document is designed to meet the following objectives:

1. **Automated Evaluation**: AIxCC is an automated competition. Thus, scoring must be primarily automated, with manual review as needed.
2. **Metagaming**: Teams should focus on improvements in automated vulnerability discovery and program repair–not the analysis or defeat of the scoring algorithm.
3. **Neutrality:** The AIxCC competition will foster innovative research via a gamified environment that challenges participants to uncover how foundational models can advance the state of the art in automated program repair and vulnerability discovery. The scoring algorithm will uniformly weigh and evaluate submissions that conform to the rules of the environment.
4. **Real-World Relevance**: AIxCC aims to secure critical infrastructure and open-source software. To that end, the AIxCC scoring algorithm and challenges will be crafted such that the resulting CRSs can assess software on a scale that mirrors real-world software applications.

4.2 Cyber Reasoning System (CRS) Areas of Excellence (AoE)

Vulnerabilities are pervasive and ever-present in today's software development life cycles. In 2021, researchers estimated that "the average time taken to fix critical cybersecurity

[3] B. Price, M. Zhivich, M. Thompson and C. Eagle, "House Rules: Designing the Scoring Algorithm for Cyber Grand Challenge," in IEEE Security & Privacy, vol. 16, no. 2, pp. 23-31, March/April 2018, doi: 10.1109/MSP.2018.1870877.

vulnerabilities" had risen to 205 days.[4] The objective of AIxCC is to identify effective, integrated automation of cyber reasoning tasks as assessed by the AoE described below. These AoE, inspired by a detailed survey of the state of the art in automated program repair,[5] were selected to measure capabilities pertaining to the hardening of real-world, open-source projects via automated vulnerability discovery and program repair.

1. **Size of Software**: The challenge structure will reward CRSs that are able to find vulnerabilities in large code bases and generate multi-hunk patches.[6]
2. **Multi-Language Applicability**: The scoring algorithm will reward CRSs that can find and fix vulnerabilities in projects written in a variety of programming languages.
3. **Vulnerability Classes**: The scoring algorithm will reward CRSs that can find and fix a broad number of vulnerability classes.
4. **Vulnerability Discovery Accuracy**: The scoring algorithm will reward CRSs that are able to discover vulnerabilities with a high level of accuracy.
5. **Patch Effectiveness:** The scoring algorithm will reward CRSs that generate patches that effectively remediate vulnerabilities without deteriorating intended functionality.

In our review of public approaches, no single CRS is excellent in all five AoE; however, many approaches have shown promising results in a subset of the AoE. If successful, AIxCC Teams will significantly advance the state of the art.

4.2.1 Team Score

The Team Score represents the overall performance of a participating Team in the competition. This score will be the aggregate metric used to compute competition ranks. The Team Score will be computed for each Team using the following assessed metrics:

- Diversity Multiplier (DM),
- Accuracy Multiplier (AM),
- Vulnerability Discovery Score (VDS), and
- Program Repair Score (PRS).

Each Team Score will be calculated as follows:

$$Team\ Score = DM * AM * f\left(\sum_{CPV} VDS\ ,\ \sum_{CPV} PRS\right)$$

[4] https://www.zdnet.com/article/average-time-to-fix-critical-cybersecurity-vulnerabilities-is-205-days-report
[5] https://arxiv.org/pdf/2303.18184.pdf
[6] https://arxiv.org/pdf/2303.18184.pdf

To maintain scoring algorithm neutrality and mimic real-world evaluation of vulnerabilities, the AIxCC Organizers adopt a flat scoring scheme, where each CPV solution will be worth the same number of points.

4.2.2 Diversity Multiplier (DM)

The DM measures performance in AoE #2, Multi-Language Applicability and #3, Vulnerability Classes.

The DM will assess the ability of a CRS to complete a diverse set of cyber reasoning tasks. Specifically, the DM will reward systems capable of finding and patching a broad range of CWE's across multiple languages.

The DM will be calculated as follows using the aggregate of all submissions per Team:

$$DM = 1 + \frac{log_{10}(1 + CWE_{discovered})}{2 \times log_{10}(1 + CWE_{all})} + \frac{log_{10}(1 + CWE_{patched})}{2 \times log_{10}(1 + CWE_{all})}$$

where only a single instance of a CWE class needs to be discovered (and/or patched) to get credit for it in the DM calculation.

Figure 2 shows an example of a DM distribution in a competition with a total of seven unique CWEs to discover and patch. As the figure demonstrates, a CRS that can both patch and discover multiple unique CWEs will be rewarded with the greatest DM score.

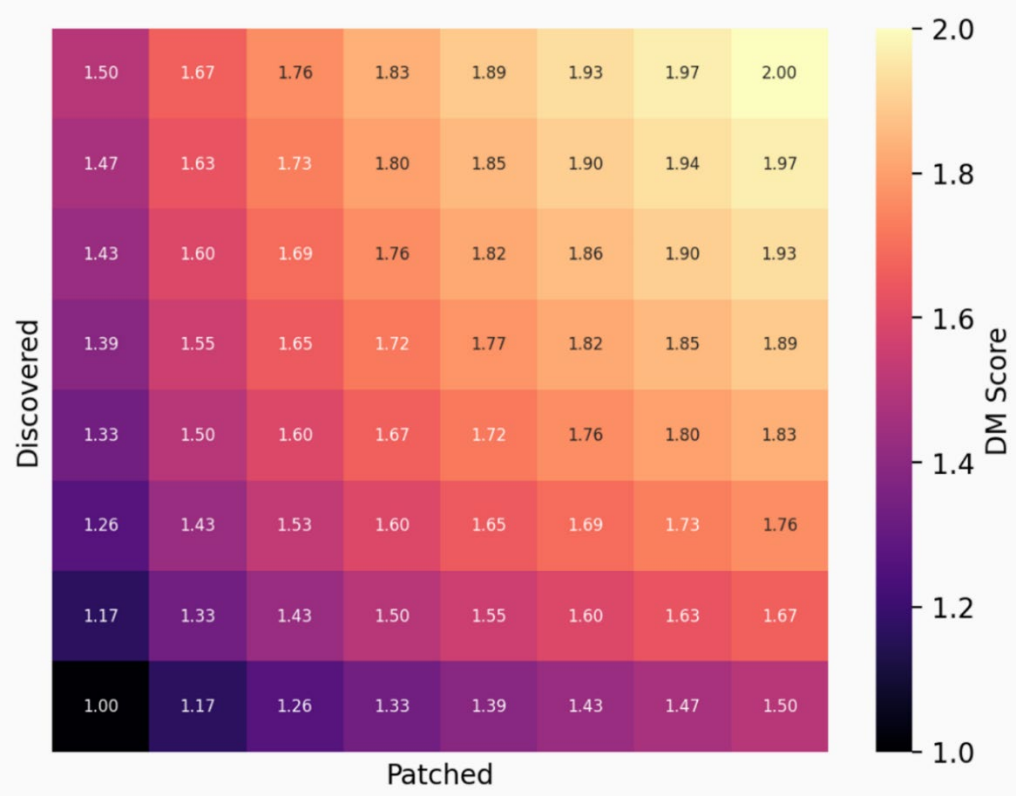


Figure 2: Heatmap showing the distribution of the DM for various CWE-discovered and CWE-patched counts, in a competition with a total possible CWE count of seven (7).

4.2.3 Accuracy Multiplier (AM)

The AM measures performance in AoE #4, Vulnerability Discovery Accuracy.

Over the course of the competition, the AM will keep track of submission inaccuracies. The AM serves as a penalty toward CRSs that attempt to guess and/or use the Scoring System as an oracle to extract challenge information. The following actions will increase the inaccurate submission count, thereby reducing the AM:

1. VD Submissions that receive a score of 0 due to no vulnerability found.
2. VD Submissions that receive a score of 0 due to a duplicate vulnerability found.
3. GP Submissions that receive a score of 0.
4. GP Submissions that target a CPV_UUID of which has already been targeted in a previous GP Submission.

This means a CRS may re-submit a GP for the same CPV_UUID to receive a higher PRS, but this will be at the cost of reducing the AM. Improperly formatted submissions that are rejected by the scoring system do not count as inaccurate submissions in the AM calculation.

The AM will be computed as follows using the aggregate of all submissions per Team:

$$AM = \phi \times \sigma\left(\frac{\delta - Submissions_{inacc.}}{\mu}\right)$$

where σ is the sigmoid or standard logistic function and φ, δ, and μ are constants whose values will be determined in advance of the competition. The significance of these constants are as follows:

- δ will be used to center the curve on a target number of rejected submissions; this decision will be dependent on the number of CPs and CPVs in play;
- μ will be used to control the slope, the rate in which the AM decreases; and
- φ will be used to scale the curve such that AM begins at 1.00 with 0 rejections.

4.2.4 Vulnerability Discovery Score (VDS)

The VDS measures performance in AoE #4, Vulnerability Discovery Accuracy. The VDS will be computed as follows per CPV:

$$VDS_{CPV} = \begin{cases} 0, & Submitted\ PoV\ does\ not\ trigger\ the\ sanitizer\ specified\ in\ the\ POU \\ 1, & Submitted\ PoV\ does\ trigger\ the\ sanitizer\ specified\ in\ the\ POU \end{cases}$$

Note that a Team may only score once for any given CPV. If a VD Submission discovers a CPV that has already been discovered by the Team in a previous VD Submission, the Scoring System will inform the Team that the submission is a duplicate.

4.2.5 Program Repair Score (PRS)

The PRS measures performance in AoE #5 and Patch Effectiveness. The PRS will be computed as follows per introduced CPV:

$$PRS_{CPV} = Functionality\ Score \times Security\ Score$$

4.2.5.1 Functionality Score

The CP test suites will be used to validate the correctness of the GPs.

$$Functionality\ Score = \begin{cases} 1, & zero\ functionality\ tests\ fail \\ 0, & some\ functionality\ tests\ fail \end{cases}$$

CP functionality test cases will be grouped by features exhibited in the software. The Functionality Score will be calculated based on how the submitted GP impacts the evaluation system's private tests.

4.2.5.2 Security Score

GPs will be assessed for their effectiveness in successfully remediating the target CPV.

$$Security\ Score = \begin{cases} 0, & test\ suite\ triggers\ sanitizer\ for\ GP \\ 1, & test\ suite\ does\ not\ trigger\ sanitizer \end{cases}$$

4.2.5.3 Real-World Vulnerabilities

For scenarios where a VD Submission was submitted and verified for non-introduced vulnerabilities, e.g., a suspect real-world (0-day and n-day) vulnerability in historically derived commits, a candidate GP submission will be evaluated after the ASC by the AIxCC Organizers. The potential PRS score for this type of GP submission is equivalent to the description above for the introduced vulnerability case, but the AIxCC Organizers cannot offer equivalent functionality scoring during the live ASC runtime for these potentially unknown CPVs.

4.2.5.4 Weighted VDS and PRS

Part of the Team Score is a function of a Team's total VDS and total PRS. To facilitate post-competition CRS acceptance, AIxCC places a heavier emphasis on program repair (versus vulnerability discovery); however, both metrics are key to pushing Teams past the edge-of-art.

The weighing function for the VDS and PRS will be calculated as follows using the aggregate of all submissions per Team:

$$f(VDS_{total}, PRS_{total}) = \alpha \cdot \log_{10}\left(1 + \frac{VDS_{total}}{\alpha}\right) + (\beta \cdot PRS_{total})$$

4.2.6 Tie Breaker

In the event of a tie where the results impact the qualifying Teams for the AFC, the AIxCC Organizers will use the following score components to break the tie.

- The Team with the highest number of scored GP submissions wins the tie, then
- The Team with the most identified and validated real-world vulnerabilities wins the tie, then
- The Team with the least number of invalid submissions (i.e., highest Accuracy Multiplier score) wins the tie.

If there is still a tie after comparing all those components, the CRS with the fastest competition execution time (i.e., the CRS that took the least time to achieve the score) wins the tie.

5.0 Appendix A: Acronyms

AM	Accuracy Multiplier
AIxCC	Artificial Intelligence Cyber Challenge
AoE	Areas of Excellence
API	Application Programming Interface
ASC	AIxCC Semifinal Competition
BIC	Bug-Inducing Commit
CP	Challenge Project
CPV	Challenge Project Vulnerability
CRS	Cyber Reasoning System
CWE	Common Weakness Enumerations
DARPA	Defense Advanced Research Project Agency
DM	Diversity Multiplier
GP	Generated Patch
JSON	JavaScript Object Notation
LFS	Large File Storage
LLM	Large Language Model
OCI	Open Container Initiative
PoV	Proof of Vulnerability
PRS	Program Repair Score
RFC	Request for Comments
SAO	Scoring Algorithm Objectives
VDS	Vulnerability Discovery Score

6.0 Appendix B: Scoring Algorithm Explainer

Understanding scoring is the first step in figuring out how to qualify for the AIxCC Final Competition (AFC). Section 4.0 of this document provides the general scoring algorithm, including several key constants. This explainer will provide values for those constants.

As a refresher, team score will be determined by the Diversity Multiplier (DM), the Accuracy Multiplier (AM), and a function of the number of vulnerabilities teams find and fix.

$$Team\ Score = DM * AM * f\left(\sum_{CPV} VDS\ ,\ \sum_{CPV} PRS\right)$$

The DM is designed to reward teams that find and fix vulnerabilities in as many CWE categories as possible. The list below shows all the in-scope CWE:

- C
  - CWE-125
  - CWE-787 + CWE119 (These will count as one CWE during scoring DM purposes)
  - CWE-416 + CWE-415 (These will count as one CWE during scoring DM purposes)
  - CWE-476
  - CWE-190
- Java
  - CWE-22
  - CWE-77
  - CWE-78
  - CWE-94
  - CWE-190
  - CWE-434
  - CWE-502
  - CWE-918

The AM function depends on the number of inaccurate submissions ($Submissions_{inacc}$) and is controlled by three parameters: Φ (phi), δ (delta), and μ (mu).

$$AM = \phi \times \sigma(\frac{\delta - Submissions_{inacc.}}{\mu})$$

The AM is designed with the following assumptions and goals:

- If a team makes zero (0) inaccurate submissions ($Submissions_{inacc}=0$), then AM = 1 and the score is not reduced.
- The AM, and the overall score, decreases as the number of inaccurate submissions increases.
- The AM has three major constants that influence its behavior:
  - Φ ensures that the AM equals 1 when there are no inaccurate submissions.
    - To accomplish this, we set Φ to 1.018, this is strictly a function of δ and μ.
  - δ reflects the AIxCC scoring algorithm's tolerance for mistakes and determines the midpoint of the AM curve, which is the point where the AM is roughly 0.5.
  - μ influences the steepness of the AM curve, controlling how quickly the AM drops as inaccurate submissions increase.

For the ASC, parameters were chosen to encourage experimentation but disincentivize use of the scoring system as an oracle. To that end, δ = 75 and μ = 18.75. Given μ and δ, Φ = 1.018 to ensure that the AM = 1 for teams with zero inaccurate submissions.

With that the AM for the ASC is:

$$AM \ = \ 1.018 \times \sigma(\frac{75 \ - \ Submissions_{inacc.}}{18.75})$$

The final function, which is influenced by the number of found and fixed vulnerabilities, has two constants: α (alpha) and β (beta).

$$f(VDS_{total}, PRS_{total}) \ = \ \alpha \cdot \log_{10}\left(1 \ + \ \frac{VDS_{total}}{\alpha}\right) + (\beta \cdot PRS_{total})$$

α scales the maximum component of the Vulnerability Discovery Score (VDS), while β is a multiplier for the Program Repair Score (PRS).

The VDS represents a team's performance in identifying vulnerabilities and the PRS quantifies their success in patching those identified vulnerabilities. As discussed in this document, the shape of this function was designed to reward teams that both find **and** fix vulnerabilities. However, the fact that finding a vulnerability is a strict prerequisite for fixing a vulnerability may bias the competition to rewarding "finding", placing teams and CRS's that prioritize patching at a strategic disadvantage. To mitigate this, α = 50 and β = 0.8, such that fixing vulnerabilities is worth roughly three times (3x) more than simply finding them.

$$f(VDS_{total}, PRS_{total}) \ = \ 50 \cdot \log_{10}\left(1 \ + \ \frac{VDS_{total}}{50}\right) + (0.8 \cdot PRS_{total})$$

As an additional bit of information, if the ratio between a team's weighted PRS score and team's weighted VDS score is approximately 3:1 there will be between 30 (2.35:1) and 125 (3.86:1) CPVs.