AIxCC Final Competition Procedures and Scoring Guide

Release Date 06/06/2025
Version 2

Defense Advanced Research Projects Agency
Information Innovation Office

675 North Randolph Street
Arlington, VA 22203-2114

Table of Contents

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.

Version	Section(s)	Change Description	Date
1.0	All	Release 1.0 (AFC)	03/12/2025
2.0	2.6.2; 3.2; 3.3; 3.4; 3.4.1; 3.7.3; 3.7.3.1; 3.10.3; 3.10.5; 4.2.2; 4.3.1; 4.3.3; 4.3.4; 4.3.5	Duplicate and variant submission clarifications; PoV, Patch, and SARIF submission rule clarifications; Custom model clarifications and pre-approval timeline; Open-sourcing Custom Model License; LLM Budgets; Scoring updates and clarifications; Added Section 4.3.5	06/06/2025

1 Overview

1.1 Document Purpose

This AIxCC Final Competition Procedures and Scoring Guide describes the format, procedures, and scoring for the AIxCC Final Competition (AFC), which will take place over a series of rounds in 2025. The purpose of this document is to inform teams of what to expect in the AFC by describing the AFC format, challenges, Cyber Reasoning System ("CRS") evaluation methods, and scoring.

This guide replaces the "AIxCC Semifinal Competition (ASC) Procedures and Scoring Guide" and formally codifies any "Technical Notes" previously published via Slack. This document is intended to be completely consistent with previous AFC information, but to the extent that discrepancies exist, this document is controlling. Importantly, this document does not in any way supersede the **AIxCC Rules**, which can be found on the AIxCC website: https://aicyberchallenge.com/rules/. Teams and their respective CRSs must adhere to all rules and requirements stated in the AIxCC Rules.

1.2 Document Terminology

The AFC is executed in a series of **"rounds."** In each round, CRSs will be presented with **"challenges"** to solve. This is accomplished as request/response dialogs between the **"CRS"** and the **"competition framework."**

Teams must implement a set of services referred to as the **"CRS API"** for receiving tasks from the competition framework. Likewise, AIxCC Organizers will create a set of services for receiving submissions from CRSs referred to as the **"competition API."**

During round execution, the competition framework interacts concurrently with all CRSs; it sends challenges and evaluates submissions sent by CRSs. Each CRS solves challenges and sends submissions.

"API Details and Documentation" refers to the version-controlled specifications and documentation contained in AFC GitHub repositories. This is where technical details such as endpoint URLs, field level messages, and API and telemetry documentation are located. "**CRS Specification"** refers to the Cyber Reasoning System section (Section 3) of this document. The AFC repository can be found at: https://github.com/aixcc-finals/example-crs-architecture.

1.3 AFC Rounds Format

The AFC will take place over three (3) unscored exhibition rounds and one (1) scored round. For each round, each CRS will receive a series of challenges. CRSs may receive multiple tasks at once. For each challenge, the CRS has a limited amount of time to find and fix vulnerabilities by sending submissions to the competition API for evaluation. At the same time, the CRS will have additional scoring opportunities by demonstrating a capability to assess static descriptions of potential vulnerabilities formatted as Static Analysis Results Interchange Format (SARIF) reports and to bundle its vulnerability discoveries, patches, and assessments.[1]

[1] https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html

When a round is complete, organizers will provide the competitors with access to their respective CRS data collected during round execution for the respective team's CRS. Teams should use this feedback to improve their CRS for the next round, as applicable.

1.4 Schedule

A series of three mandatory but unscored exhibition rounds, designed to ensure that all APIs operate as expected and to provide teams with useful feedback, will open on April 1, 2025. The scored round will open on June 26, 2025.

Table 1 outlines the round schedule although these dates and times may be subject to change. The round open date specifies the time at which all competitors are expected to have their CRS in a healthy state and ready for tasking within their provided round-specific Azure subscription.

This information should be considered informative in nature. The authoritative document outlining the dates and times for round openings is the AIxCC Rules document. Should there be any future changes to the schedule, they will be reflected in the AIxCC Rules document.

Table 1: Round Schedule

Round Name	Scoring Status	Round Open Date/Time
Exhibition Round 1	Unscored	2025-04-01 15:00:00 UTC
Exhibition Round 2	Unscored	2025-05-06 15:00:00 UTC
Exhibition Round 3	Unscored	2025-06-05 15:00:00 UTC
Final Round	**Scored**	**2025-06-26 15:00:00 UTC**

1.5 Round Details

To assess a wider spectrum of CRS capabilities and focus on certain aspects of the CRS, round details will vary for each round. For example, a round may be focused solely on a particular challenge type (see Section 2.6.2).

Round details will include, but are not limited to:

- Round open date/time
- The approximate duration for each challenge
- The maximum number of concurrent challenges
- The total number of challenges (by challenge type)
- The maximum LLM budget
- The maximum Azure budget
- If unharnessed challenges will be included in the round (see Section 5.1)

AIxCC Organizers will provide concrete details no less than thirty (30) days prior to the round.

Other details, such as unplanned changes to competition APIs or other third-party software dependencies, will be released on a case-by-case basis as necessary.

If there are any questions about the details, please use the appropriate Slack channel for communication.

1.6 AFC Objectives

Objectives of the AFC include the following:

- Seed the next generation of software companies that will be able to address the growing need for remediating software security issues at scale.
- Inspire and cultivate cybersecurity innovators to steer bright new minds toward an AI-aligned cybersecurity career path.
- Promote and facilitate adoption of AI-driven security analysis tools across open source software projects.
- Expand security analysis of open source software projects aligned to critical infrastructure sectors.
- Foster growth and adaptation of existing and potentially new foundational AI models toward security-centric use cases.
- Generate one or more software applications that can be transitioned for real-world use to assess, find, and fix software bugs.

1.7 Document Status

The information provided in this document is intended to be an accurate representation of the current design for the AFC. While Organizers do not anticipate major changes, the information herein is subject to change at the sole discretion of DARPA.

1.8 Competition Archive

Organizers intend to publicly release salient AIxCC artifacts upon completion of the AFC. The format of the archive is under development.

2 Challenges

This section defines and describes an AFC challenge, its constituent parts, and a description about how to build and run security tests. Note, details on the gameplay with respect to challenges are introduced in Section 3.1 and described in subsections of Section 3. SARIF, which is a scorable aspect of the AFC, is described starting in Section 3.1.

2.1 Overview

During a round, several sets of challenges will be sent to each CRS for analysis. These challenges represent realistic scenarios in which a CRS could provide significant value to source code maintainers and contributors.

There are two types of challenges in the AFC: "full-scan" and "delta-scan." Both are described in Section 2.6.2. Details on differences in scoring between the two are described in Section 4.

Each challenge contains the following:

- Source code for the challenge. For "delta-scan" challenges, this includes additional data in unified diff format containing proposed changes to the source code,[2]
- Tasking parameters for the challenge, such as challenge type (delta-scan or full-scan) and a deadline timestamp which specifies a time limit for the challenge, and
- Analysis tooling to provide a standardized method of building, running, and testing the challenge source code.

2.2 Challenge Basis

Each challenge has a basis, which is a real-world project that may be critical to industry, national security, and the public. No challenge or basis will be disclosed prior to the start of AFC round execution.

For some challenges, the basis has been modified to contain:

- An unspecified number of challenge-introduced vulnerabilities.
- Additional features and functionality.

In addition, due to the nature of the competition, any challenge basis may include an unknown number of pre-existing (zero-day) vulnerabilities which can be scorable.

During the AFC, any number of challenges may share the same challenge basis, and the challenge source code will differ between challenges, even those sharing the same basis.

Note that a challenge basis may be referred to as a "challenge repository."

2.3 Challenge Harness

Each challenge base is paired with one or more challenge harnesses which, when built, are binary entry-points for CRS-generated data to be used to exercise vulnerabilities. The source code for challenge harnesses may exist in the challenge source code or be accessible from the provided analysis tooling.

[2] https://www.gnu.org/software/diffutils/manual/html_node/Unified-Format.html

The challenge harnesses for AFC are in the style of libFuzzer fuzz targets and include both C and Java harnesses. [3,4] The analysis tooling supplied with each challenge provides a standardized way to build all challenge harnesses.

2.3.1 Analysis Tooling

Each challenge includes analysis tooling that provides a standard method of building the challenge harnesses with various sanitizers, architectures, and engines, as well as executing the challenge harnesses with supplied input test data. The analysis tooling may also provide additional information related to the challenge source code, harnesses, build, and test processes.

The AFC will use analysis tools compliant with The Open Application Security Testing[5] (TOAST) Specification, which is being developed for AIxCC and made available to competitors to view and work with ahead of the AFC. Although under iterative development, the exact analysis tooling version that will be used for each round will be included in the details released ahead of each round. Any modifications to the AFC analysis tooling between rounds will be minimal and will not break backwards compatibility with prior round usage.

2.4 Languages

The AFC will focus on vulnerabilities found in:

- The C programming language
- The Java programming language

While each challenge basis and challenge harnesses may contain code in a variety of languages, only vulnerabilities found in C and Java are in scope.

2.5 Challenge Vulnerabilities

Challenges contain an unspecified number of vulnerabilities that can be discovered and patched. The origin of any given vulnerability is either challenge-introduced or pre-existing (zero-day).

An AFC vulnerability is defined as one which enables a harness-reachable crash and will be evaluated and scored as such (see Section 3.2.1).

For scoring and evaluation purposes, challenge vulnerabilities are defined in an inductive process based on a combination of the results from the round and the competition-designed vulnerability PoVs and patches, rather than explicitly defined ahead of time. See Section 3.2 for detail.

2.5.1 Challenge-Introduced Vulnerabilities

Challenges may include any number of synthetic vulnerabilities introduced by the AIxCC Organizers. These vulnerabilities will be designed to mimic real-world issues, and all will be scorable.

[3] https://llvm.org/docs/LibFuzzer.html#fuzz-targe
[4] https://www.code-intelligence.com/blog/how-to-write-fuzz-targets-for-java-applications
[5] https://github.com/aixcc-finals/toast

2.5.2 Zero-Day Vulnerabilities

Since challenges are based on real-world software, vulnerabilities that were not intentionally introduced may be discovered by a CRS. Those vulnerabilities are scorable if they can be demonstrated by a harness-reachable crash.

2.6 Challenge Parameters

A component of the challenge includes parameters that constrain the CRS while processing the task. Examples include time limits and types; both parameters are described below.

2.6.1 Challenge Deadline

Challenges are designed to enable time constraining the CRS with respect to vulnerability discovery and patch generation. Any submissions that occur after the deadline specified in the task will be rejected by the competition API.

2.6.2 Challenge Types

Challenges will be one of two types: “full-scan” and “delta-scan.”

- In a **full-scan** challenge, the challenge source code is the modified basis.
- In a **delta-scan** challenge, the challenge “base state” is the modified basis, and the “delta state” is the base state with an additional change (“diff”) applied that represents a delta from the base state.

In a full-scan challenge, the CRS must find and fix vulnerabilities anywhere in the source code, reachable and crashable via the challenge harnesses.

In a delta-scan challenge, the CRS must find and fix vulnerabilities that the delta has explicitly introduced or revealed. For any given delta-scan challenge, the vulnerabilities themselves may exist in either the base state or the delta, but the harness-reachable crash(es) can only be caused after the delta is applied.

For example, consider an existing vulnerability that is not “enabled” in the base state. If the delta-scan diff changes configuration settings which now enable that code, this vulnerability is scorable for the delta-scan challenge because it is now reachable due to the delta, whereas before it was unreachable.

2.7 Functional Tests

The base source code for a given challenge will have one or more functional tests used to assess patch quality.

- Tests for a given challenge may include all pre-existing public tests, as well as organizer-created tests specific to the challenges.
- For any given challenge, the CRS may or may not be provided a standardized method of running functional tests, and as such, should be able to handle both cases.

2.8 Challenge Examples

Organizers have provided a tool to generate a challenge based on a given set of parameters. The resulting challenge is encapsulated in a JavaScript Object Notation (JSON) blob that matches the description in the API Details and Documentation. The tool supports generation of both “delta-scan” and “full-scan” challenge types with example vulnerabilities introduced.

The tool, documentation, and examples can be found here:

https://github.com/aixcc-finals/generate-challenge-task

3 Cyber Reasoning System (CRS)

This section provides a high-level overview of how a round is executed and defines vulnerability discovery, patch generation, and SARIF assessment submissions. It also contains:

- Requirements and specifications for both CRS development and CRS runtime (round execution), including how and when to submit CRS solutions;
- References to examples and detailed API specifications and documentation; and
- Disqualification guidelines for a CRS.

3.1 Overview

Each team will develop a CRS capable of automatically processing a set of AFC challenges while conforming to all rules and constraints of the **AIxCC Rules** and this document. Challenges are fully defined in Section 2. Elements of the AFC also involve SARIF reports. A SARIF report can be used as a structured way to represent and convey vulnerability information. The AFC uses SARIF in two primary ways. The first, "SARIF generation" is when the CRS generates a SARIF report that represents a vulnerability it has found. The second, "SARIF assessment" is the CRS assessing a SARIF report sent to it by the competition framework ("SARIF broadcast").

In successive rounds, CRSs will be presented with sets of challenges. The aim of each CRS is to find and fix vulnerabilities contained in the challenges. To place an upper bound on the number of challenges a CRS processes concurrently, the total number of challenges for the round is spread across multiple sets.

The following is a summary of ways in which CRSs can demonstrate capabilities. Each one has a corresponding "submission" message. All of these are described in the identified section. Each section contains requisite definitions, a description of the submission requirements, and notes on how the submission is evaluated. For each capability, the CRS may send a submission before the challenge deadline. Scoring details for each are provided in Section 4: AFC Scoring Algorithm.

Table 2: Summary of Submissions

Capability	Section	Description
Vulnerability Discovery	3.2	The CRS may send PoV submissions to the Competition API.
Patch Generation	3.3	The CRS may send generated patch submissions to the competition API.
SARIF Assessment	3.4	The CRS may receive zero or more SARIF reports related to active challenge tasking via SARIF broadcasts. The CRS may score points by assessing the SARIF's correctness and submitting a SARIF assessment.
SARIF Generation	3.5	The CRS may generate its own SARIF reports to describe its findings. *(Not scorable, see Section 4.1.2)*

Capability	Section	Description
Bundling	3.6	The CRS may send challenge bundles to the competition API, detailing explicit pairings of its findings and/or SARIF broadcasts.

3.1.1 Competition API Submission Evaluation

To balance timely response with detailed evaluation for submissions, evaluation occurs in phases: completeness checks, automated verification, and post-round analysis and audits. Each phase generally takes more time than the previous and is described below.

3.1.1.1 Completeness Checks

Completeness checks are performed at submission time and synchronously return a response. They are designed to pass or fail relatively quickly and use the following logic:

- The submission is considered complete if all inputs are well-formed, required fields are provided, and all fields pass applicable range checks. Otherwise, the submission is incomplete.
- All submissions receive a response code; complete submissions receive a tracking identifier that can be used by the CRS to request evaluation status.

3.1.1.2 Automated Verification

These are long-running tasks that take a variable amount of time depending on the type of submission and complexity of the challenge. Automated verification is performed only for complete submissions.

Submission status can be requested by the CRS to discover if verification is in progress or complete, negative or positive.

3.1.1.3 Post-Round Analysis and Audits

Post-round analysis and audits are performed to verify evaluation, such as correctness of submissions. This may include but is not limited to additional automated and human review.

3.2 Vulnerability Discovery

To score points for a discovered vulnerability, the CRS must provide a Proof of Vulnerability submission.

3.2.1 Proof of Vulnerability (PoV)

Challenges contain “challenge harnesses” that exercise challenge functionality with CRS-provided data. To demonstrate a vulnerability discovery, CRSs will submit information sufficient for the evaluation system to reproduce the identified vulnerability in the form of data passed to these harnesses.

3.2.1.1 Variant PoVs

The concept of a “Variant PoV” is new in AFC. A challenge vulnerability is implicitly defined by a set of one or more PoVs that exercise that vulnerability. When a CRS submits a valid PoV, that PoV is considered one of that CRS’s Variant PoVs for the underlying vulnerability.

A challenge vulnerability may have any number of Variant PoVs associated with it. The set of variants is generated post-round, determined by the deduplication methodology in Section 4.3.3, and includes PoVs from all CRSs and competition-designed PoVs. Cross-team Variant PoVs are used to improve patch quality assessment. More details about how these are used in scoring can be found in Section 4: AFC Scoring.

3.2.2 Proof of Vulnerability Submission

A CRS will send submissions to the competition API. Complete submissions include the following and will receive a tracking identifier that can be used to request status:

- Challenge identifier
- PoV challenge harness name
- PoV challenge harness build option: sanitizer (if applicable)
- PoV challenge harness build option: engine ("libfuzzer")
- PoV challenge harness build option: architecture ("x86_64")
- PoV binary data that represents input bytes to challenge harness

The set of valid PoV build options for any given challenge will be communicated to the CRS in the challenge task and may differ between challenges. PoVs with non-valid build options will be rejected.

Note, for the AFC only "x86_64" architecture will be supported, and thus the CRS must supply that value in its PoV submission. The field is included to future-proof the specification for additional architecture support. All PoV submissions will be evaluated with the "libfuzzer" fuzzing engine.

3.2.2.1 Duplicate PoVs

For a given challenge task, a variety of PoVs may cause different crashes for the same underlying vulnerability. In the AFC, these are often referred to as "duplicate" PoVs. The scoring system will use the PoV deduplication methodology described in Section 4.3.3 to determine if a set of PoVs are considered duplicates. Thus, the set of "duplicate" PoVs and the set of "variant" PoVs are equivalent.

Duplicate PoV submissions will *not* negatively affect a team's accuracy, as previous rulings stated. However, a CRS will not score additional points for duplicate PoV submissions. Further details on scoring duplicate vulnerabilities can be found in Section 4: AFC Scoring.

3.2.3 Proof of Vulnerability Evaluation

PoVs must reliably reproduce the crash. If the submitted PoV cannot be confirmed by these means, the submission will be rejected. Unlike in the AIxCC Semifinal Competition (ASC), a scorable crash does not need to be explicitly caused by a sanitizer. Rejected submissions will negatively impact the team score (see Scoring Algorithm). Further details on PoV scoring can be seen in Section 4: AFC Scoring.

To evaluate PoV submissions, the competition infrastructure will attempt to reproduce the PoV using the commands in the TOAST specification described in Section 2.3.1.

PoVs must adhere to the following rule:

1. They must demonstrate a problem in the challenge source code, not simply a problem in the harness.

PoVs which are reproducible but fail to adhere to the above rules will not score but will not negatively affect a team's accuracy. As with all submissions, PoVs are subject to post-round review.

3.3 Patching

A CRS may generate patches independent of its PoVs and SARIF broadcast assessments. Patches are modifications to the source code of the challenge that removes the vulnerability while preserving the software's intended functionality.

The patches that the CRS creates to fix vulnerabilities are referred to as generated patches.

3.3.1 Generated Patches

A generated patch submission must include source code modifications that only affect C or Java source code, depending on the challenge language, and must be submitted in unified diff[6] format. The patch is applied against the specified challenge source code for validation. Individual patches will be validated independently against the original challenge. Patches will not be validated in conjunction with any other submitted patch.

Unlike in the ASC, patches do not require ties to PoVs. Instead, patches may be submitted on their own, and may receive points without PoVs, detailed in later sections.

3.3.2 Generated Patch Submission

A CRS will send submissions to the competition API. Complete submissions include the following and will receive a tracking identifier that can be used to request status:

- Challenge identifier
- Patch content as a unified diff

3.3.2.1 Duplicate Patches

For a given challenge task, a variety of patches may functionally remediate the same set of challenge vulnerabilities. These are often referred to as "duplicate patches". The scoring system processes all CRS patch submissions for a challenge to ultimately determine which patches should be used for scoring. Since patches may remediate one *or more* challenge vulnerabilities, the patch selection methodology is non-trivial, and thus does not directly use the term "duplicate patch".

Generally, however, a CRS will not score additional points for duplicate patch submissions. And non-scoring patch submissions (such as duplicates) *will* have a negative effect on a team's challenge accuracy. Further details on scoring patches and patch accuracy can be found in Section 4: AFC Scoring.

3.3.3 Generated Patch Evaluation

Patch evaluation is a multi-phase asynchronous process. During evaluation, a CRS can obtain submission status using the competition API. The basic steps are:

[6] https://www.gnu.org/software/diffutils/manual/html_node/Unified-Format.html

1. The patch is applied to the challenge source code. Note, for a "delta-scan" challenge, the "diff" portion of the challenge is applied to the challenge's source code base, then the patch is applied to the result. For a "full-scan" challenge, the patch is applied to the challenge's source code base.
2. After applying the patch, the challenge source and applicable harness are built.
3. To verify that program functionality is preserved, all available functional tests are executed.

Patches that pass the above validation tests are then scored after the challenge deadline ends. Unlike in the ASC, patches are not scored against a singular PoV. Instead, patches will be scored on their ability to remediate vulnerabilities that were discovered by all competing CRSs and the competition creators. Further details on patch scoring can be found in Section 4: AFC Scoring.

Generated Patches must adhere to the following rules:

- They must not modify source code outside of the target language for the project.
- They must not remediate the vulnerability via modifications to the harness.
- They must not pass functional tests via modifications to the functional test source code.

All patches which fail to adhere to the above rules will not score and will negatively affect a team's accuracy. After the round, patches will go through additional reviews to assess quality and integrity.

3.4 SARIF Assessment

During round execution, the competition framework may send SARIF broadcasts (reports). Each report describes a potential vulnerability in an active challenge (delta or full). The reports will not contain PoV information, and they are not guaranteed to be accurate in their description of a real problem in the challenge code.

There are two ways for a CRS to score from a SARIF broadcast:

1. Correctly assess the SARIF (See Section 3.4.1 SARIF Assessment Criteria)
2. Broadcast IDs from the report may be included in a Bundle Submission (see Section 3.6.1)

3.4.1 SARIF Assessment Criteria

A SARIF report is "correct" if the problem it describes is in fact a problem in the source code of the associated task. The content of the SARIF report may vary, but all fields included in the SARIF report should be considered when assessing as correct or incorrect.

For example, suppose a SARIF report asserts that source code in a specified location is vulnerable to "Buffer Overflow" (CWE-120). If that area of source code is vulnerable to a buffer overflow, the report should be assessed as "correct"; otherwise, it should be assessed as "incorrect". The assessment should be focused solely on the specified vulnerability in the specified source code location.

SARIF report contents and additional examples can be found in the API Details and Documentation.

Due to the nature of the challenges, correct SARIF reports will describe harness-reachable, sanitizer-triggered crashes.

3.4.2 SARIF Assessment Submission

A CRS will send submissions to the competition API. Complete submissions include the following:

- SARIF identifier - included as part of the endpoint URL path
- Assessment - correct or incorrect
- Description - plain text that justifies the given assessment

For the SARIF Assessment submission to be accepted, it must have a non-empty description.

3.4.3 SARIF Assessment Evaluation

The methodology for SARIF assessment evaluation is detailed in Section 4: AFC Scoring.

3.5 SARIF Generation

In addition to the above submissions, a CRS may generate a static description of the vulnerability in SARIF format. This generated SARIF submission may be created by the CRS to demonstrate its understanding of the vulnerability in a static way, apart from the PoV or patch. This generated SARIF report will not affect scoring but may be associated with the PoVs and patches (see Section 4: Scoring Algorithm for more details).

3.5.1 SARIF Report Submission

The competition has one additional constraint compared to the public SARIF format:

- CRS-generated SARIFs must contain rules, and all results must specify a rule ID.

3.6 Bundling

3.6.1 Bundle Submission

CRSs may indicate that other submissions are related by adding them to a bundle. For example, a CRS can indicate that a patch they generated fixes a vulnerability they found by submitting a bundle containing both.

CRSs may add and remove parts of a bundle after submission. CRSs may also delete bundles.

Bundles contain the following fields. All fields are optional, but a bundle should contain at least two to be meaningful.

- A reference to a previously submitted PoV
- A reference to a previously submitted patch
- A reference to a previously submitted SARIF report
- A reference to a broadcast SARIF report
- A plaintext description containing any additional information to explain the CRS's findings

3.7 CRS Development Constraints

3.7.1 Azure Subscription

Teams will be provided with an Azure subscription that can be used for development and a unique subscription to be used during each round execution. Resources do not need to be duplicated in each subscription and can be moved between the subscriptions, if desired, as long as they exist in the current round subscription by the round open date in a healthy state.

3.7.1.1 Azure Development Budget

For development, teams are allotted **$100,000** to be used during development for the duration of the AFC. See the Round Execution Budgets and Limits section for round execution budgets.

3.7.2 GitHub Repositories

AIxCC organizers will provide competitors with private GitHub repositories for CRS submissions. All versions of CRSs used during round execution must exist in the provided repository. For AFC, the AIxCC organizers will be using a GitHub organization instead of GitHub enterprise, so competitors will be required to submit the public GitHub usernames of their team members.

The AIxCC organizers will provide GitHub workflows and/or actions for continuous evaluation of competitor CRS repositories to ensure the code and interfaces conform to the published specifications. The AIxCC organizers will use the **#announcements** channel in the "AIxCC Finalists" Slack channel to notify competitors of any version changes related to the evaluation actions.

3.7.2.1 LICENSE File

To comply with the AIxCC Open Source Requirement as specified in Section 3.4 of the AIxCC Rules, all teams must include a license file in the top level of their CRS GitHub repository. An example license is provided in Appendix B and is also available in the API Details and Documentation**.** Name the file "LICENSE" and place it in the root CRS source code repository.

3.7.3 Custom Models

For AFC, custom models are not prohibited. Models may be trained offline but must be hosted within Azure compute infrastructure.

Custom models that are part of a competitor's CRS solution **must be pre-approved** no less than seven (7) days prior to round open and must be reproducible by AIxCC Organizers. Model architectures, weights, and other configuration data must be released according to the open source requirement. Training data is not required to be open sourced but must be made available to AIxCC Organizers. Open sourced training data is highly encouraged.

All the following must be provided for model reproducibility:

1. License compliance and attribution - As required by the source, all licenses and attribution must be described, provided (if applicable), and included in the CRS source submission. This includes, but is not limited to, any required software or data used to reproduce the model.

2. Hardware and/or compute requirements - A description of the compute resources required to reproduce the model must be provided to organizers.
3. Environment - Instructions and/or scripts to set up the environment, run data processing, train, and evaluate the model, along with any expected outputs or checkpoints.
4. Data processing – Competitors must provide a script that handles all data pre-processing steps (e.g., cleaning, tokenization). The script should have clear, reproducible steps and be executable without additional modifications.
5. Training data - All source data must be accessible to AIxCC Organizers. Data is accessible if one or more of the following is true:
    - The data is included in the CRS submission; OR
    - The data source is fully documented, and the data is retrievable by organizers (e.g., URLs, APIs, or datasets). Version numbers or timestamps must be included if the data or access methods change over time; OR
    - A script is included in the CRS source submission to fetch the data outside the competition environment.

3.7.3.1 Custom Model License

As stated in the AIxCC Rules document, section 3.4, "*The open-source requirement applies to all source code and artifacts, including, but not limited to, models, for the Cyber Reasoning System (CRS) generated for, and used as an entry in AIxCC.*"

If using a custom model, a license form that allows for others to use/reproduce the model must be chosen. One example is the OpenMDW license (https://openmdw.ai/).

3.8 Framework APIs and Specifications

AIxCC Organizers will implement the competition API to enable a CRS to send all submissions and to request status on submissions.

Likewise, the CRS must implement the CRS API to enable CRS receipt of challenges and broadcast vulnerabilities. The CRS API must also include a method for obtaining health and metrics information.

All endpoints for the CRS API and competition API must have a uniform implementation:

- Endpoints will be HTTP-based with JavaScript Object Notation (JSON) inputs, JSON outputs, and return HTTP status codes
- All endpoints must support key/token authentication using HTTP Auth
- All endpoints must use HTTPS signed by a public Certificate Authority

3.8.1 Telemetry Specification

To address requirements for analysis of activities and metrics, AFC will use OpenTelemetry (OTel). OpenTelemetry is an observability framework and toolkit focused on the generation, collection, management, and export of metrics, traces, and logs.[7,8]

For AFC, CRSs must comply with the telemetry specifications contained in the API Details and Documentation which includes requirements and guidelines for using OpenTelemetry-compatible libraries for tracking detailed LLM usage and other metrics.

Teams are encouraged to go beyond the minimally required and optional items in the telemetry specification. Leveraging OTel can be used to gain insights into CRS behavior during development to inform improvements between rounds, help with competition visualizations, and prepare for post-competition artifact publication

Teams are encouraged to forward unstructured logs but include the required metadata which will be captured in the specification.

CRS-generated telemetry data will be made available to respective teams after each round. Providing telemetry and logs will help teams more quickly be able to adapt to this feedback as it will be made available prior to restoring access to a team's Azure subscription.

3.9 CRS Solution Deadlines

At round open, a CRS must be running and in a healthy state. The status endpoint information must correlate with the tagged release in the competitor's GitHub repository.

3.10 Round Execution Constraints

3.10.1 CRS Provisioning and Startup

Prior to the round open:

- Competitors must provision their CRS within an Azure subscription using Terraform. Use of a Makefile is permitted.
- After the "terraform apply" or "make up" command completes, the CRS must be fully functional and ready to receive tasks. README documentation shall be made available to AIxCC organizers of any additional setup/environment variable requirements that are needed for purposes of reproducing the CRS for validation purposes.
- A successful health check signals that the CRS is ready to receive tasks.

Please refer to the API Details and Documentation for examples.

3.10.2 Post Round

For each round, access to all teams' round-specific Azure subscriptions will be revoked at the round open date/time. After the round is completed and scored, AIxCC organizers will provide API interactions, and the telemetry data collected, to each team. This data or derivatives are NOT to be shared in any form with

[7] https://opentelemetry.io/docs
[8] https://opentelemetry.io/docs/concepts/observability-primer/

other teams or be made public until after the conclusion of the competition and the full and final competition results are announced publicly by DARPA. Doing so may be grounds for disqualification. Within seven days of the close of a round, access to Azure subscriptions will be restored so that teams can collect data, as needed. This delay is to encourage teams to use the provided telemetry and log forwarding mechanisms as their primary means of feedback as they will receive access to this data first.

3.10.3 Round Execution Budgets and Limits

During round execution, monetary budgets apply to both Azure subscription and commercial LLM utilization. Each budget is specific to the round and includes two items:

- Azure Subscription Budget - Azure subscriptions are supplied by and paid for by AIxCC Organizers.
- LLM Utilization Budget - Credentials (API Keys) are supplied by the AIxCC Organizers.

If either round budget is exceeded during round execution:

- Any competition scorable submission received by the competition API after budget exhaustion will not be reflected in the final score.

Organizer-provided round budgets will vary based on the volume and types of challenges expected in the round.

- Organizers will provide the round budgets for Azure subscription and LLM utilization for each round as part of the Round Details (see Section 1.4.1).
- Competitors should use this information to scale and allocate resources accordingly.

3.10.3.1 Azure and LLM Budget Items

For the Azure subscription budget, budget items include any costs incurred through the round-specific Azure subscription.

For the LLM utilization budget, budget items include any costs incurred accessing AIxCC organizer-approved commercially available LLM services. Utilization is calculated based on telemetry provided by CRSs and collected by AIxCC organizers (see Section 3.8.1).

3.10.3.2 LLM Round Execution Query Capacities

LLM rate limits are set by providers on a per-model basis.

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

3.10.4 Large Language Model API Telemetry

Capturing LLM usage (e.g., inputs, outputs, durations, counts, etc.) is essential for both round execution and post AFC research and analysis. The AFC leverages OpenTelemetry (OTel) to capture LLM usage.

CRSs must conform to the telemetry specifications in Section 3.8.1 for model access for both public and custom models (as applicable).

3.10.5 Networking

During round execution, CRSs have VPN access to the competition API and vice versa.

The CRS will have restricted access to the internet during round execution. Teams may not use third-party compute resources or non-public APIs unless the resource is on an approved list curated by AIxCC organizers.

Access is limited to the following:

- The competition API
- AIxCC organizer-approved commercial LLM access
- Resources needed to build and run harnesses contained within challenges

3.11 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. AFC 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 AFC and are applicable both before and after the AFC 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 interfering with their AIxCC-related work.

The guidelines that follow are meant to be informative in nature. The document governing competition disqualification is the AIxCC Rules document.

3.11.1 No Superman Defenses

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

3.11.2 No Malicious Patches

The patches submitted during competition are meant to fix vulnerabilities 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 challenge to avoid sanitizer detection. For Java, a generated patch must not add/modify a Security Manager to mitigate the vulnerability. Other violations would include forking a new process to avoid sanitizer detection or to isolate the vulnerability.

3.11.3 No Phoning Home

Any attempts to circumvent the network restrictions described in the CRS Specification (see Section 1.7 and Section 3) will be considered malicious.

3.11.4 No Gaming the Scoring Algorithm

The AFC Scoring Algorithm is intended to be robust and motivating for specific research goals; however, the AIxCC Organizers cannot rule out loopholes in the design or 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 vulnerabilities.

3.11.5 No Hacking the Infrastructure

Manipulating, tampering, or subverting the AFC infrastructure (before or during the competition) will result in disqualification of the offending team.

3.11.6 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. Use or attempts to use others' resources, including other competitors' resources, is also prohibited. Misuse may lead to disqualification from AIxCC and/or penalties under the terms and conditions of other vendors and services.

3.11.7 No Obfuscation Tactics in Custom Models

Custom trained/tuned models are permitted as specified in the CRS Specification (see Section 1.7 and Section 3); however, model obfuscation tactics of any form could lead to disqualification. Custom models that do not conform to open source or reproducibility requirements could lead to disqualification.

3.11.8 No Pre-baking Models

Generating models that resemble a lookup table of challenge basis or challenge repositories to vulnerabilities and then directly attempting to brute force submissions would be considered against the spirit of the competition.

3.11.9 No Gaming the Challenge Code Basis

Any attempt to distinguish between the real-world public basis for a challenge and the challenge source code 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 code changes that were authored by AIxCC Organizers to aid with vulnerability discovery; and/or comparing the challenge source code with our own copy of the challenge source code to aid with vulnerability discovery.

4 AFC Scoring

4.1 Scoring Design Objectives

The scoring algorithm for the AFC described in this document is designed to measure, incentivize, and reward competing CRSs for their potential real-world value and positive impact on open source security.

In this competition, we aim to incentivize CRS impact for the following stakeholders:

1. Open source maintainers
2. Open source contributors

Because of this, the scoring algorithm described in this document is designed to meet the following objectives:

1. Encourage and reward the key areas of excellence we want to see from a CRS.
2. Encourage and reward additional areas of exceptional quality of a CRS.

We believe that by achieving these scoring objectives, we can use the scoring algorithm to direct the competition and competitors to have the greatest impact on open source security.

4.1.1 Areas of Excellence

The following areas of excellence, if demonstrated by a CRS, will result in direct score value, affecting the team's final standings for the competition and winnings. These areas of excellence are designed to reflect real-world value that a CRS may provide, rather than competition-specific achievements.

1. **Size and Variety of Software:** A CRS will be tasked with numerous challenges across many software repositories of varying size, complexity, and behavior.
2. **Multi-language Applicability:** A CRS will be tasked with challenges in both C and Java projects.
3. **Analysis Scope Flexibility:** A CRS will be tasked with both full-scan and delta-scan challenges and will be rewarded by finding vulnerabilities in a limited scope of software.
4. **Vulnerability Discovery:** A CRS will be incentivized to discover vulnerabilities and submit PoVs.
5. **Quality Patch Generation:** A CRS will be incentivized to generate viable patches that do not break functionality. Patches may be submitted for vulnerabilities that the CRS has or has not submitted PoVs for.
6. **Proof and Patch Correlation:** While discovery and patching can be performed independently, a CRS will be incentivized to perform both, explicitly stating the correlation between patch and PoV.
7. **Submission Accuracy:** A CRS will be disincentivized to submit incorrect or duplicate submissions in areas where it would be considered negative value to users.

8. **SARIF Assessment:** A CRS will be asked to evaluate a SARIF broadcast to determine its correctness as an initial step toward reasoning over SARIF broadcasts in the real-world.

4.2 Scoring Algorithm

The scoring algorithm is broken down into several sub-components. This section details the high-level overview of each of the scoring algorithm components. Further details on how exactly the scores are calculated can be found in Section 4.3: Further Details.

4.2.1 Team Score

The team score represents the overall performance of a participating team in the AFC. The team score will be an aggregate of the challenge scores from the final scored round, and will be the determining factor for team ranking, standings, and winners.

$$Team\ Score = \sum Challenge\ Scores$$

4.2.2 Challenge Score

The challenge score represents the performance of a CRS on one individual challenge. The challenge score is a weighted sum of CRS performance in vulnerability discovery, program repair, SARIF broadcast assessment, bundling, and accuracy across all vulnerabilities in the challenge.

$$Challenge\ Score = AM * (VDS + PRS + SAS + BDL)$$

The challenge score is based on five key scoring metrics, defined in the following sections:

- Accuracy Multiplier (AM)
- Vulnerability Discovery Score (VDS)
- Program Repair Score (PRS)
- SARIF Assessment Score (SAS)
- Bundle Score (BDL)

The challenge score will be calculated the same, regardless of the challenge type: full-scan or delta-scan, but the scorable elements within a challenge may differ depending on the challenge type.

4.2.2.1 Accuracy Multiplier

The Accuracy Multiplier (AM) measures CRS accuracy within an individual challenge. A CRS is responsible for producing accurate and quality submissions to the game infrastructure and thus will be negatively impacted by inaccurate and duplicate submissions.

The intent of the AM is two-fold: First, to represent and incentivize CRS accuracy. Second, to counter any potential point value a CRS may gain from its inaccurate submissions, disincentivizing any brute-force or oracle approaches to the competition.

Because of the variable number of scorable events for any given challenge, the accuracy multiplier is based on a ratio of accurate submissions vs. cumulative (accurate and inaccurate) submissions within a challenge. Section 4.3.1 enumerates which submissions increase the accurate count, increase the inaccurate count, or have no effect on either.

The AM will be computed as follows:

$$AM = 1 - (1 - r)^4$$

Where r is a measured ratio of accurate submissions over the sum of accurate and inaccurate submissions, computed as follows:

$$r = \frac{submissions_{accurate}}{submissions_{accurate} + submissions_{inaccurate}}$$

The resulting AM value in relationship to the accuracy ratio is shown in the following figure.

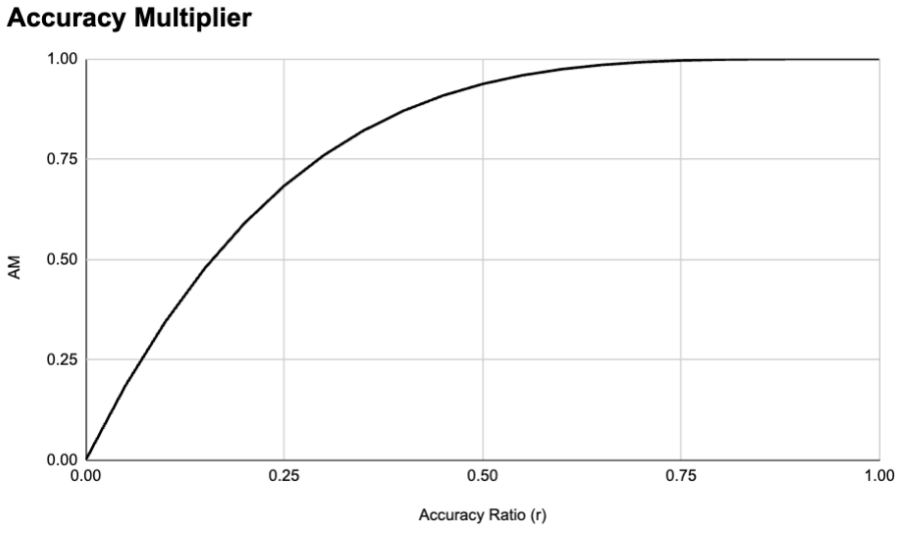

Figure 1: Accuracy Multiplier charted over accuracy ratio, r.

See Section 4.3.1: Accuracy Multiplier Calculations for further details and examples on how the accurate and inaccurate submission counts are calculated.

4.2.2.2 Vulnerability Discovery Score

The Vulnerability Discovery Score (VDS) represents the performance of a CRS in discovering and proving vulnerabilities for a given challenge. The VDS is calculated as the sum of all scoring PoV values for the challenge.

$$VDS = \sum value_{PoV}, for\ PoV \in Scoring\ PoVs$$

where the set of "Scoring PoVs" is determined by the scoring selection criteria defined at the end of this section. PoV values are calculated as follows:

$$value_{PoV} = \begin{cases} 2 \times \tau_{PoV}, & Submitted\ PoV\ does\ cause\ reproducible\ crash \\ 0, & Submitted\ PoV\ does\ not\ cause\ reproducible\ crash \end{cases}$$

where τ_{PoV} is the PoV's time multiplier. The available point score for each PoV decreases over the course of the challenge window, to a minimum of 50%. This is enforced by the time multiplier, **τ**.

$$\tau_{PoV} = 0.5 + \frac{time\ remaining\ in\ challenge\ window}{2\ *\ total\ challenge\ window\ time}$$

PoV submissions will go through a deduplication process to ensure a CRS only scores one PoV per unique vulnerability discovered (see Section 4.3.3 for details). The set of scoring PoVs is composed of all the last-submitted variant PoVs for all unique vulnerabilities discovered by the CRS, for that challenge. Another way to consider this: For any set of PoV submissions that are considered duplicates of each other, only the last submission is selected as a scoring PoV.

Reproducible, non-scoring PoVs will not earn points directly, but may be used in bundles. They will also be used in patch evaluation along with the other variants and will *not* affect the Accuracy Multiplier. Non-reproducible or erroneous PoVs will still negatively affect the Accuracy Multiplier (see Section 4.3.1 for details).

4.2.2.3 Program Repair Score

The Program Repair Score (PRS) represents the performance of a CRS in generating patches for a given challenge. The score is calculated as the sum of all scoring patch values for the challenge.

$$PRS = \sum value_{patch}, for\ patch \in Scoring\ Patche$$

where the set of "Scoring Patches" is determined by the scoring selection criteria defined at the end of this section. Patches must adhere to a set of rules defined in Section 3.3.3 above. Patch values are calculated as follows:

$$value_{patch} = \begin{cases} 6 \times \tau_{patch}, & \textit{Submitted patch adheres to the rules, remediates one or more Challenge Vulns for this challenge, AND passes all functionality tests.} \\ 0, & \textit{Submitted patch does not remediate any known Challenge Vulns OR fails to apply, build, or pass functionality tests, or fails to adhere to the rules.} \end{cases}$$

Where τ_{patch} is the patch's time multiplier. The available point score for each patch decreases over the course of the challenge window, to a minimum of 50%. This is determined by the time multiplier, $\boldsymbol{\tau}$.

$$\tau_{patch} = 0.5 + \frac{time\ remaining\ in\ challenge\ window}{2\ *\ total\ challenge\ window\ time}$$

Patch submissions will go through a selection process to determine which of the CRS-submitted patches should be used for scoring. This selection process is non-trivial due to the complex and subjective nature of selecting between patches that may remediate multiple overlapping vulnerabilities. The selection process is outlined in detail in Section 4.3.4 and should be understood thoroughly.

Patches not selected for scoring will not earn points directly but may be used in bundles. These patches *will* negatively affect the Accuracy Multiplier: see Section 4.3.1 for details.

4.2.2.4 SARIF Assessment Score

The SARIF Assessment Score (SAS) represents the ability of a CRS to accurately assess a SARIF broadcast against the challenge codebase.

$$SAS = \sum value_{assessment}, for\ assessment \in Scoring\ Assessments$$

where the set of "Scoring Assessments" is determined by the scoring selection criteria defined at the end of this section. Assessment values are calculated as follows:

$$1 \times \tau_{assessment}, \quad \textit{Assessment correctly identifies report as correct or incorrect.}$$

$$value_{assessment} = \begin{cases} 0, & \textit{Assessment incorrectly identifies report correctness.} \end{cases}$$

where $\tau_{assessment}$ is the assessment's time multiplier. The available point score for each assessment decreases over the remainder of the challenge window, to a minimum of 50%. This is determined by the time multiplier, $\boldsymbol{\tau}$.

$$\tau_{assessment} = 0.5 + \frac{time\ remaining\ in\ challenge\ window}{2\ *\ time\ remaining\ in\ challenge\ window\ when\ SARIF\ was\ broadcast}$$

SARIF Assessment Scoring Selection Criteria

Due to the binary nature of the assessment, only one SARIF Assessment will be evaluated per broadcast. The last assessment submission will be used for scoring. Assessments prior to the last will negatively affect scoring. See Section 4.3.1 for details.

4.2.2.5 Bundle Score

The Bundle Score (BDL) represents the CRS's ability to pair PoV, patch, and/or SARIF broadcast Universally Unique Identifiers (UUIDs) together. This score is used to incentivize a CRS to reason over how its findings and broadcasts are related and correctly associate them together.

In addition to PoVs, patches, and broadcast UUIDs, a CRS may additionally include a CRS-generated SARIF report UUID, and/or a plaintext description detailing any additional information to explain its reasoning or findings. These fields will not have any direct effect on the Bundle Score but will be used when calculating the unscored CRS excellence metrics (see Section 4.2.3).

The Bundle Score is calculated as the sum of select bundle scores that remain at the end of the challenge window.

$$BDL = \sum score_{bundle}, for\ bundle \in Scoring\ Bundles$$

where the set of "Scoring Bundles" is determined by the scoring selection criteria defined at the end of this section. The score for each individual bundle depends on the contents of the bundle, detailed below.

For bundles that include *only a single item from the set:* {PoV, patch, Broadcast UUID}, the bundle score will be zero and that bundle will not be included in the set of scoring bundles. For example, a PoV bundled with a CRS-generated SARIF and plaintext-description will be collected and used for excellence metrics but will not earn additional points.

For bundles that include *more than one from the following set*: {PoV, patch, broadcast UUID}, the bundle score is calculated as follows:

$$score_{bundle} = \begin{cases} value_{bundle}, & \textit{The bundle contains all correct pairings} \end{cases}$$

$-value_{bundle}$, *The bundle contains incorrect pairings*

The bundle score acts as a higher risk or reward than simply generating PoVs and patches, as incorrect bundling will negatively affect the score, however, a CRS may gain significant point advantage from successful bundling. The value for non-zero bundles is calculated as follows:

$$value_{bundle} = \begin{cases} 0.5 * (value_{PoV} + value_{patch}) + b, & \textit{if PoV and Patch are bundled.} \\ b, & \textit{if PoV and Patch are not both supplied.} \end{cases}$$

where the PoV and patch values are the values that the respective PoV and patch UUIDS receive, as detailed in previous sections. If a PoV or patch UUIDs are *not* included in the bundle, their PoV and patch values in the above formula will be set to zero, respectively.

The broadcast pairing value, *b*, is calculated as follows:

$$b = \begin{cases} 0, & \textit{No Broadcast UUID included in bundle, or the bundle contains incorrect pairings.} \\ 1, & \textit{The bundle contains correct pairings of Broadcast UUID and PoV, but no Patch.} \\ 2, & \textit{The bundle contains correct pairings of Broadcast UUID and Patch, but no PoV.} \\ 3, & \textit{The bundle contains correct pairings of Broadcast UUID, PoV, and Patch.} \end{cases}$$

The requirements for correct bundle pairings are defined as follows:

- For a PoV & patch pairing: the patch must remediate the PoV crash it is paired with.
- For a PoV & broadcast UUID pairing: the PoV must crash the vulnerability defined in the SARIF broadcast. Specifically, the CRS-supplied PoV and withheld PoV(s) associated with the SARIF broadcast must exercise the same vulnerability (see Section 4.3.3).
- For a patch & broadcast UUID pairing: the patch must fix the problem defined in the SARIF broadcast. Specifically, the patch must remediate the challenge vulnerability associated with the withheld PoV(s) associated with the SARIF broadcast.
- For a PoV, patch, and broadcast UUID pairing all the above must be true.

The set of "Scoring Bundles" includes a maximum of one bundle per challenge vulnerability. Bundles are associated with their target challenge vulnerabilities by their PoV and/or their SARIF Broadcast vulnerability associations. Any bundles that are deleted via the competition API will not be considered for selection. For the remaining bundles at the end of a challenge deadline, the last-submitted bundle per challenge vulnerability is selected for scoring, with a maximum of one bundle per patch UUID.

4.2.3 Non-Scoring Excellence Recognition

There are highly exceptional characteristics that a CRS may exhibit, however scoring those abilities is subjective or otherwise beyond the focus of the AFC. To highlight exceptional capabilities, AIxCC Organizers are developing metrics that will result in achievements, accolades, and other forms of recognition, but in no way affect final scores.

Excellence metrics being considered include capabilities such as the ability to produce patches that have minimal performance impact on the running software; and the ability of a CRS to effectively and rapidly respond to the various types of tasking while making efficient use of its resources while still retaining a high degree of effectiveness and accuracy. As they are finalized, details will be communicated to competitors.

4.3 Further Details

These sections go into further detail about the rationales and approaches of the scoring algorithm components.

4.3.1 Accuracy Multiplier Calculations

The Accuracy Multiplier (AM) is calculated based on two metrics: an accurate submission count and an inaccurate submission count. The following details what exactly affects those two counts, and what does not.

The following submissions will increase the accurate submission count:

- Non-duplicate, scoring PoV submissions that demonstrate reproducible crashes.
- Patch submissions chosen to score by the patch scoring selection process (see Section 4.2.2.3 and further detail in Section 4.3.4).
- Scoring SARIF assessment submissions (the last-submitted assessment, if it is correct).
- Non-duplicate, correct scoring bundle submissions.

The following submissions will increase the inaccurate submission count:

- PoV submissions that do not demonstrate reproducible crashes.
- Patch submissions that fail to apply to the codebase or result in harness build failures.
- Patch submissions that fail to remediate at least one challenge vulnerability.
- Patch submissions that otherwise are not chosen to score by the patch scoring selection process (see Section 4.2.2.3 and further detail in Section 4.3.4).

- Non-scoring SARIF assessment submissions (incorrect, or non-last-submitted)
- Bundle submissions that are deemed incorrect or duplicate after the close of a challenge.

The following submissions will not affect either accurate or inaccurate submission counts:

- Duplicate, reproducible PoV submissions.
- Patches that apply and build but fail functionality testing. In the real world, it is normal to iterate on patches based on continuous integration (CI) pass/fail status within a pull request (PR). Even further, because a CRS may not have the direct ability to run functionality testing on its own, re-submitting an updated patch based on functionality test information is acceptable.
- Server-side erroneous handling of submissions. Specifically, any time it is suggested that a CRS should resubmit will not affect the inaccurate submission count (see competition API for details).
- Submissions that contain schema mismatch. Although they add noise, mismatched schema submissions cannot be used to gain value for a CRS and thus will not affect inaccurate submission count.

4.3.2 PoV Crash Evaluation Methodologies

The AFC will incorporate state of the art crash evaluation methodologies to assess the score of submitted PoVs. These methodologies are in development and will be made available to competitors in advance of the competition for usage and review.

4.3.3 Deduplication Methodologies

The AFC will incorporate state of the art crash deduplication methodologies to refine and uniquify the set of PoVs submitted by both CRSs and challenge development. Two PoVs are considered "duplicates" if it is determined that they exercise the same underlying vulnerability. These methodologies will be made available to competitors in advance of the competition for review[9].

PoV deduplication is a multi-part process that incorporates both crash-based deduplication and designed-patch deduplication, along with other deduplication methodologies. The full methodology is documented and shared on github[9].

4.3.4 Patch Scoring Extended

In the AFC, patches are evaluated after the challenge deadline has passed. As required for all submissions, patches must be submitted before the challenge deadline to be evaluated. Unlike ASC, in which patches were scored on individual PoVs, in the AFC, patches are scored based on their ability to remediate known challenge vulnerabilities, which may involve testing against one or more variant PoVs for that known vulnerability.

For a patch to remediate a known vulnerability, it must remediate all crashes caused by all variant PoVs collected for that given vulnerability. This includes all variant PoVs submitted from all CRSs and includes all PoVs created by the challenge authors for the given vulnerability, for the given challenge.

[9] See https://github.com/aixcc-finals/example-crs-architecture for shared scripts and methodologies.

The set of variant PoVs for a challenge vulnerability is defined by the PoV deduplication methodology previously described in Section 4.3.3.

The concept of Variant PoVs adds an element of CRS consensus to the AFC that helps improve and ensure patch quality. The following table helps explain this concept of patch scoring. Note, all the information in the following table is assumed to be for a singular challenge task.

CRS	PoV	Variant of
CRS A	A-1	Vuln-1
CRS A	A-2	Vuln-2
CRS B	B-1	Vuln-1
CRS B	B-2	Vuln-1
CRS C	C-1	Vuln-4
CRS D	D-1	Vuln-1
CRS D	D-2	Vuln-3
Challenge Author	CA-1	Vuln-1
Challenge Author	CA-2	Vuln-2
Challenge Author	CA-3	Vuln-3

Challenge Vulnerability	Variant PoVs
Vuln-1	A-1, B-1, B-2, D-1, CA-1
Vuln-2	A-2, CA-2
Vuln-3	D-2, CA-3
Vuln-4	C-1

Table 3: Variant PoVs for vulnerabilities from various sources.

In the above example, seven PoVs were submitted by four teams, and three PoVs were from challenge authors. The resulting vulnerability-to-variant mapping is shown on the table to the right. For a patch to successfully remediate Vuln-2, it must resolve all crashes from PoVs A-2 and CA-2.

4.3.5 Scoring Patch Selection Process

After a challenge task has ended, a CRS's patch submissions for the challenge will go through a patch selection process to determine which will be used for scoring.

A patch selection process is necessary due to the complexities added by patches being able to remediate multiple challenge vulnerabilities at a time. This process resolves several issues that arise when a CRS submits several patches which remediate different overlapping subsets of challenge vulnerabilities.

The purpose of this patch selection process is three-fold:

1. To align patch scoring with real-world value, not simply competition strategy.

2. To enable a CRS to improve its submissions over the course of the challenge window.
3. To remove opportunities for cheating and other negative behaviors.

Align with Real-world Value

The patch selection process aims to select the "best set" of patches among all submitted patches by a CRS, for a given challenge task. This selection prioritizes rewarding patch specificity, and then prioritizes choosing a minimal set that remediates the highest number of challenge vulnerabilities.

Enable Patch Improvement

Next, the selection process prioritizes the last-submitted patch by a CRS. This allows a CRS to submit updated patches for the same challenge vulnerabilities with changes or improvements (note, however, this will affect accuracy).

In the simplest case of a CRS submitting patches which remediate individual challenge vulnerabilities, this selection process reduces to simply scoring the last-submitted patch for each vulnerability.

Remove Opportunities for Cheating and Negative Behaviors

The patch selection process aims to remove any opportunity for a CRS to gain unearned points through composing submission sets that game the scoring system. The number of patches selected for scoring will be no more than the total number of remediated vulnerabilities for the challenge.

Further details and specifics on the patch selection process can be reviewed within the competition github[10].

[10] See https://github.com/aixcc-finals/example-crs-architecture for more details on patch selection.

5 Unscored Research

Competitors will be encouraged to perform certain tasks or provide artifacts within the competition that will not impact their score but are focused on highlighting the potential for innovation.

All aspects described within this section do not impact score. The purpose of this is to generate artifacts which can help further innovation within the space and facilitate mechanisms for teams to submit elements that they can obtain after a round to help improve their CRS.

5.1 Unharnessed Challenges

One area of unscored research is answering challenges that do not use the existing harnesses provided within a challenge repository. During competition rounds, teams may be tasked with specific challenges that focus on instrumenting unharnessed challenges which will be indicated by a task field which will be captured in the repository mentioned in section 1.7.1 regarding the API details. AIxCC Organizers will communicate more details about these either as a document update, a message in the #announcement Slack channel, within the round details or some combination thereof. No submissions against unharnessed challenges will impact the team score, and unharnessed challenges will never be tasked concurrently with challenges that do impact the team score.

Unharnessed challenges will take advantage of the unharnessed artifact endpoint described in section 5.4.

5.2 Unscored Research Budgets

Participation in unscored research areas are voluntary but allow teams further opportunities to obtain additional feedback from their CRS between rounds. Teams will be notified in the round details if AIxCC Organizers intend to task CRSs with unharnessed challenges during a round.

Engagement in unscored research as written in Section 5 is optional, and there **will not** be dedicated budgets for this tasking.

5.3 Unscored Research Outputs

Teams will receive artifacts from this unscored research. The intention is that artifacts from unscored research will be included as part of the competition archive format outlined in section 1.10. Thus, team performance on unscored research will be made publicly available as part of the competition archive.

5.4 Unscored Research Recognition

As per the section 5 summary, these focus areas will not impact competitor scores in the final round. However, it is possible that teams could receive recognition for certain achievements as part of this focus area during or after the completion of the competition.

5.5 Unharnessed Artifact Endpoint

The AIxCC organizers will provide a dedicated endpoint where a CRS can submit artifacts related to unharnessed discoveries. Teams will be notified when this endpoint is available in round detail updates. Teams may utilize this endpoint for any challenge in cases where they want to explore harness expansion or unharnessed code to highlight CRS capabilities, but no submissions to this endpoint for any challenge will impact the team score.

Appendix A - Acronyms

Acronym	Description
AFC	AIxCC Final Competition
AIxCC	Artificial Intelligence Cyber Challenge
AM	Accuracy Multiplier
API	Application Programming Interface
ASC	AIxCC Semifinal Competition
BDL	Bundle Score
CRS	Cyber Reasoning System
DARPA	Defense Advanced Research Projects Agency
JSON	JavaScript Object Notation
LLM	Large Language Model
OSI	Open Source Initiative
PRS	Program Repair Score
SAS	SARIF Assessment Score
VDS	Vulnerability Discovery Score

Appendix B - MIT License File

The MIT License below is an example of a license approved by the Open Source Initiative (OSI). Teams may use others in compliance with the Open Source Requirement as specified in Section 3.4 of the AIxCC Rules.

Instructions

1. Copy the text of the license below into the file.
2. Create a text file named LICENSE in the root of your CRS source code repository.
3. Perform initial commit to your CRS source code repository.

Appendix C - Round Execution Arbitration Process

During round execution, AIxCC Organizers may be confronted with an issue or concern which is best resolved with feedback from DARPA leadership and/or competitors.

The arbitration process provides a mechanism to resolve these issues in a fair way that removes the appearance of AIxCC Organizer bias that could result from decisions made without competitor feedback. It provides a pathway to solicit and receive anonymous feedback from competitors in these situations.

Arbitration Process Framework

If, during execution of a round, AIxCC Organizers are confronted with an issue they cannot resolve or determine that competitor feedback is warranted, the issue will be communicated to DARPA leadership along with possible courses of action. If DARPA leadership determines a course of action, the action will be taken.

If however, DARPA leadership determines the issue or concern should be presented to competitors, they may invoke the arbitration, and the following will take place:

1. AIxCC Organizers will post a message in the AIxCC Finalists Slack channel #arbitration. The message will describe the issue and provide a finite list of potential courses of action.
2. Competitor teams will be provided 24 hours to review the issue.
3. Competitor team leads will vote on the issue by sending their team vote to DARPA leadership at aixcc@darpa.mil.
4. DARPA leadership will tally the vote. In the event of a tie, the DARPA program manager will serve as the tie-breaking vote.
5. Competitors and AIxCC Organizers will be notified of the decision and path forward.