Artificial Intelligence Cyber Challenge (AIxCC) Request for Comments (RFC)

12/13/2023

Version 1

Defense Advanced Research Projects Agency
Information Innovation Office

675 North Randolph Street
Arlington, VA 22203-2114

Table of Contents

Introduction

Establishing a fair and effective scoring system is critical to AIxCC's success, and we can't do that alone. As such, this document is formatted as a Request for Comments (RFC) and it is split into two parts:

1. AIxCC Scoring Algorithm
2. AIxCC Exemplar Challenge Preview

We want your input. **Please submit your feedback by emailing AIxCC@darpa.mil by or before 11:59PM EST on Monday, January 15, 2024**. We ask that you limit your feedback to 400 words or less. Longer submissions will not be considered.

1 AIxCC 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.

To help focus community feedback, we have included an outline of the vision for the competition, the Scoring Algorithm Objectives, and the technical Areas of Excellence against which we seek to evaluate AIxCC Cyber Reasoning Systems (CRSs). Further refinements and clarifications to this document will be released following RFC period (December 13, 2023 – January 15, 2024).

1.1 Scoring Algorithm Objectives (SAO)

Based on lessons learned from past DARPA challenges,[1] 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 also be automated.
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 outperform the state of the art in automated program repair and vulnerability discovery. The scoring algorithm will not incentivize any particular approach or methodology.
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.

1.2 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 vulnerabilities" had risen to 205 days.[2] The objective of the AIxCC competition 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

[1] 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.

[2] https://www.zdnet.com/article/average-time-to-fix-critical-cybersecurity-vulnerabilities-is-205-days-report

repair,[3] 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.[4]
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.
6. **Patch Acceptability:** The scoring algorithm will reward CRSs that generate patches likely to be accepted by human maintainers.

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

1.3 Game Format

Challenges. Teams will receive an identical corpus of challenges. These Challenge Projects (CPs) are modeled on real-world open-source projects. CPs for the ASC will consist of source code, as well as a modifiable build process and build environment that can be interacted with via standardized Application Programming Interface (API).

Vulnerabilities. The CPs will contain a number of vulnerabilities that must be identified and secured, called Challenge Project Vulnerabilities (CPVs). During the competition, Teams will not know the number of vulnerabilities that comprise each challenge. Only CPVs categorized under a competition-specific set of allowed software weaknesses (CWEs) will be eligible for scoring. The sets of software weaknesses will be announced prior to each competition; they will include CWEs that fall within the 25 classes of software weaknesses identified in MITRE's 2023 Top 25 Most Dangerous Software Weakness report.[5]

Autonomy. Teams will field CRSs that autonomously find and fix CPVs. Teams will be awarded points based on their ability to correctly find and remediate CPVs. A team's CRS must be entirely autonomous; during the competition, Teams will not be allowed to intervene with CRS execution.

[3] https://arxiv.org/pdf/2303.18184.pdf
[4] https://arxiv.org/pdf/2303.18184.pdf
[5] https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

Evaluation. Each CP will be accompanied by a public test suite and a private test suite. The public test suite will exercise expected program behavior and will be used to validate the functional correctness of competitor-generated patches. The private test suite will consist of test cases that trigger known-vulnerable code paths. This test suite will validate whether or not vulnerabilities have been successfully remediated. The private suite will also include test cases submitted by Teams as Proof of Vulnerability.

The competitor-submitted test cases will be unique to each competitor and will not be used to assess other teams.

1.3.1 Vulnerability Discovery Submissions

Over the course of the competition, CRSs will analyze CP software, submitting discovered vulnerabilities for scoring. CRSs are responsible for submitting both Proof of Vulnerability and Proof of Understanding for each vulnerability discovery submission.

1.3.1.1 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.

1.3.1.2 Proof of Understanding

In addition to a PoV, CRSs must submit the following evidence to prove their understanding of the discovered vulnerability in the original program:

1. A JavaScript Object Notation (JSON) document specifying which AIxCC Sanitizer and harness is expected to catch this vulnerability.
2. The lines of code that contributed to the vulnerability.

To be eligible for PoV points, the sanitizer called out in the Proof of Understanding must match the sanitizer triggered in the PoV. The lines of code will be used to validate the CRSs' understanding, but will not affect the score for ASC.

Exact specifications for submission format will be released prior to the competition.

1.3.2 Patch Submissions

Over the course of the competition, CRSs will generate source code patches – hereafter referred to as Generated Patches (GPs) – to repair CPVs. GPs will be evaluated against the PoVs previously submitted by the CRS and a suite of private PoVs to validate patch robustness.

GP submissions will only be eligible to score against CPVs that have previously been discovered by the competing CRS. GPs will specify the singular CPV that they aim to patch via a competition-provided unique identifier for each discovered CPV.

CRSs will submit source code patches in unified diff format,[6] which are applied against the original program for validation. For ASC, the patches will be validated independently against the original CP.

Evaluating patches that are submitted by Teams will be an extensive process aimed at ensuring patch correctness and quality. The assessment will primarily focus on three key 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. We attempt to mitigate this by providing a high-quality test suite, hold-out test cases in the private test suite, and additional static and runtime analyzers such as the AIxCC sanitizers mentioned in the previous section.
2. **GP's effectiveness in successfully remediating vulnerabilities**. The private test suite will exercise tests and vulnerability-confirming sanitizers, ensuring that vulnerabilities are being mitigated. For ASC, we do not measure vulnerabilities introduced by competitor-generated patches and reserve that metric for consideration in future competitions.
3. **The extent to which developers would accept the GP under normal circumstances**. To facilitate future maintenance and understandability by developers, it is crucial that submitted fixes are effective—but *also* adhere to best coding practices and established coding style.

1.3.3 Feedback to Teams

During ASC, each CRS will receive feedback responses from the Scoring API for its own vulnerability discovery and patch submissions. These responses will consider submissions individually and will not provide feedback on overall completion statistics, such as *remaining CPVs to be discovered*.

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

1.4 ASC Scoring Method

1.4.1 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:

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

We calculate part of the team score as a function of a team's total VDS and total PRS. These scores are handled separately to reflect the desired AIxCC research focus and allow for flexibility in how unknown vulnerabilities are handled. 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 VDS and PRS will be calculated as follows:

$$f(VDS_{total}\ , PRS_{total}) = \alpha \times log\ (1 + \frac{VDS_{total}}{\alpha}) + (\beta \times PRS_{total})$$

where α and β are constants whose values will be determined in advance of the competition. Figure 1 visualizes the relationship between VDS and PRS totals and how they may be weighed.

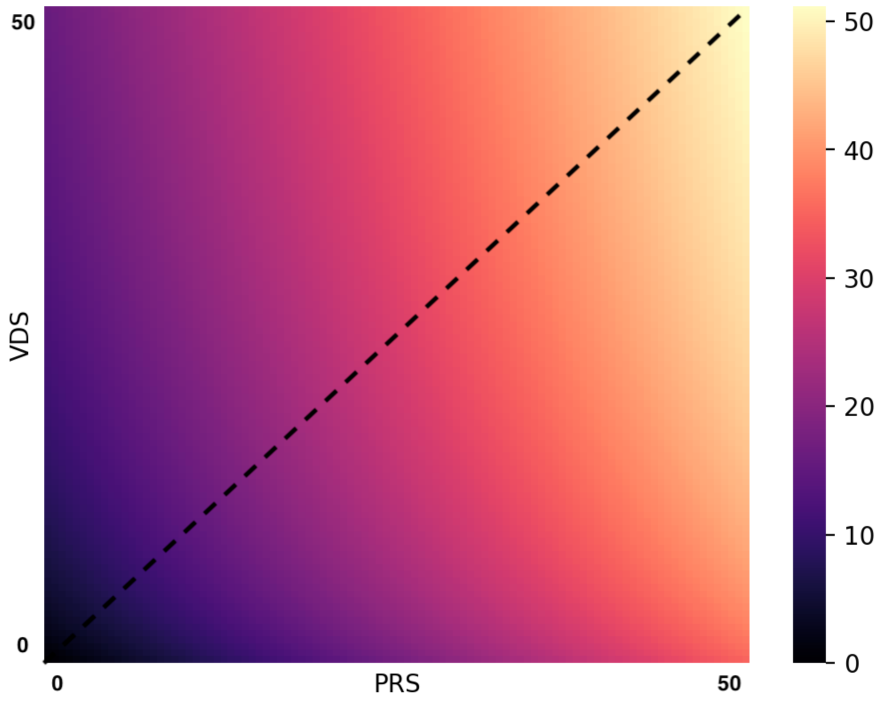

Figure 1: Heatmap showing the results of an example weighing function for various VDS and PRS totals.

As the figure demonstrates, a CRS which primarily scores in VDS, but not PRS, will receive a relatively low score due to the diminishing returns of VDS's impact on the resulting team score. On the other hand, a CRS that balances VDS and PRS scoring will outscore the aforementioned VDS-heavy CRS. In the example above, out of a hypothetical maximum of 50 points, a VDS-only CRS that discovers every CPV may score a maximum of 16.6 points, while a balanced CRS that discovers and patches even a quarter of the CPVs will receive a higher score. Thus, successful CRSs will employ a combination of both vulnerability discovery and program repair capabilities.

For the ASC, patches must target previously-discovered vulnerabilities. This means that in general, a CRS's VDS will be greater than or roughly equal to its PRS during the qualifying competition. Thus, a CRS competing in the ASC will always score above the diagonal line provided in the figure above.

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

1.4.2 Diversity Multiplier (DM)

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

The DM will assess a team’s ability 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. We do not explicitly account for a multi-language metric and instead intuit that CWEs cluster together differently by language. For example, CWE-79 ‘Cross-site Scripting’ may appear more frequently for web-based languages like PHP and less so for low-level languages like C/C++.

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

Figure 2 shows an example of a DM distribution in a competition with a total of five 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. As noted in Section 1.3, for the ASC, CRSs must discover CPVs before patching them, thus, a CRS will always score above the diagonal line provided in the figure below.

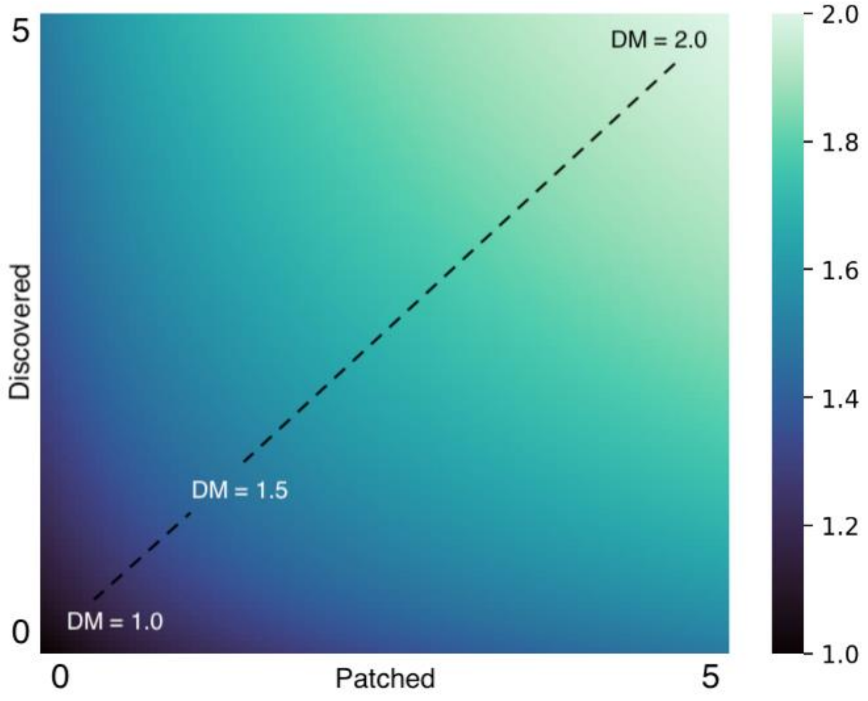

Figure 2: Heatmap showing the distribution of the DM for various CWE-discovered and CWE-patched counts, in a competition with a total possible CWE count of 5.

1.4.3 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 a handful of submission inaccuracies. The more invalid or rejected submissions that a team has, the lower the AM will be. Some example submission inaccuracies include non-scoring vulnerability discovery submissions and program repair submissions that target CPVs of which the CRS has already submitted patches for. This means a CRS may re-submit a GP for the same CPV in an attempt to receive a higher PRS, but this will be at the cost of reducing the AM.

AM will be calculated as follows:

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

where σ is the sigmoid 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 accuracy decreases.
- φ will be used to scale the curve such that AM begins at 1.00 with 0 rejections.

1.4.4 Vulnerability Discovery Score (VDS)

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

VDS will be computed as follows:

$$VDS_{CPV} = \begin{cases} 0, & Submitted\ PoV\ did\ not\ trigger\ sanitizers \\ 1, & Submitted\ PoV\ trigger\ sanitizers \end{cases}$$

1.4.5 Program Repair Score (PRS)

The PRS measures performance in AoE #5, Patch Effectiveness, and #6, Patch Acceptability.

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

1.4.5.1 Functionality Score

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

$$functionality\_score(features\_broken) = \begin{cases} 1.0, & features\_broken = 0 \\ 0.5, & features\_broken = 1 \\ 0.2, & features\_broken = 2 \\ 0.0, & features\_broken \geq 3 \end{cases}$$

CP functionality test cases will be grouped by features exhibited in the software. The Functionality Score will be determined by the number of features negatively affected by the submitted patch.

1.4.5.2 Security Score

The CRS-submitted patches will be assessed for their effectiveness in successfully remediating the target CPV.

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

1.4.5.3 Maintainability Score

AoE #6, Patch Acceptability, is a metric that is difficult to objectively measure. Presently, no widely accepted measurement methodologies exist [5]. To measure CRSs progress along AoE #6, we use static code quality analyzers, such as linters, as a proxy for this aspirational metric:

$$Maintainability\ Score = \begin{cases} 1, & code\ linters\ errors\ and\ warnings\ are\ thrown \\ 1.05, & code\ linters\ pass\ without\ errors\ or\ warnings \end{cases}$$

1.4.6 Tie Breaker

In advance of the ASC, DARPA will release a tie-breaker algorithm to be used in the unlikely event of a tie.

2 AIxCC Exemplar Challenges Preview

Most critical infrastructure in the U.S. is built upon open-source software, which ultimately means that with the right tools, anyone can inspect, modify, and enhance the way this software operates. As malicious cyber actors continue to target U.S. water systems, healthcare facilities, energy grids, and the banking sector, their goal is often to destabilize our economy and compromise the safety of Americans.

That's where the Artificial Intelligence Cyber Challenge (AIxCC) comes in: AIxCC aims to inspire technical experts to collaborate across industries and to identify solutions to some of the most critical vulnerabilities facing our infrastructure using all of the tools at our disposal. Participants are encouraged to use any combination of autonomous methodologies (such as Large Language Models [LLMs], fuzz testing and/or program analysis) to understand and resolve vulnerabilities in open-source software. Before we release the exemplar challenges to the public, we are thrilled to share with you a sneak preview of the Challenge Projects. Read on for more details.

2.1 What's in a Challenge Project (CP)?

2.1.1 Challenge Project (CP)

During the AIxCC Semifinal Competition (ASC), competing Cyber Reasoning Systems (CRSs) will be provided with multiple CPs used to demonstrate their capabilities. The CPs are modified versions of existing open-source software that are critical to our modern lifestyles and infrastructures.

These open-source software projects are hand-selected and modified specifically to challenge state-of-the-art artificial intelligence and program analysis techniques. As a result, the projects are large in size and have many complex features.

The AIxCC CPs will contain significant amounts of added and modified content, including but not limited to:

- An unspecified number of injected vulnerabilities;
- Features, behavior, and functionality; and
- Functional and security tests.

During the competition, participating AIxCC teams will be responsible for downloading and analyzing the CPs to find and fix the embedded vulnerabilities. Details on how a competing CRS will be assessed and scored can be found in the Scoring Algorithm, included above in Section 1.

2.1.2 Challenge Project Vulnerability (CPV)

The CPs will contain a number of vulnerabilities that must be identified and secured, called Challenge Project Vulnerabilities (CPVs). During the competition, participating teams will not know the number of vulnerabilities that comprise each challenge. Only CPVs categorized under a competition-specific set of allowed software weaknesses will be eligible for scoring. The sets of software weaknesses will be announced prior to each competition; they will include Common Weakness Enumerations (CWEs) that fall within the 25 classes of software weaknesses identified in MITRE's 2023 Top 25 Most Dangerous Software Weakness report[7].

These CPVs are designed to mimic real-world vulnerabilities and provide an adequate challenge for CRSs to demonstrate both their discovery and patching capabilities.

2.2 The Challenge Project Exemplars

Ahead of the ASC, DARPA will release a number of Challenge Project Exemplars in the form of software projects with injected vulnerabilities. These Exemplars may include any combination of the following artifacts: detailed build environment and instructions for building, running, and testing the projects; example discovery and patch submissions that score; and further information.

The open-source software projects that the Challenge Project Exemplars are built from will be the same open-source software projects used in the AIxCC semifinal competition. However, the vulnerabilities showcased in these exemplar releases will not appear in the semifinal competition. Rather, the competition CPs will contain many more vulnerabilities, all of which will not be publicized before the competition takes place.

2.2.1 How to Use an Exemplar CP

Competing teams and other researchers are encouraged to utilize the CPs in the testing of their CRS solutions. The format of the CP can help prepare CRSs on what to expect at the time of the competition. The solutions to any publicly released exemplars may be used against the live submission system detailed in Section 2.4.2, below. Only solutions for released exemplar vulnerabilities will receive feedback or scores from the submission systems.

[7] https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

2.3 Challenge Project Exemplar Preview

2.3.1 The Linux Kernel Challenge Project

We are excited to announce that the first Challenge Project Exemplar is the Linux Kernel.

According to security usage statistics, most of the world's personal computers, servers, Android mobile devices, and cloud environments are built on Linux Operating Systems (OS). This includes over 81% of websites and 90% of public cloud workloads,[8] including Amazon Web Service, Google Cloud, and Microsoft Azure. U.S. critical infrastructure is no exception to this rule.

At the heart of Linux OS is the Linux kernel, which serves as the core interface between a computer's hardware and its processes.[9] The Linux kernel can be found in a variety of critical infrastructure services, such as medical equipment, autonomous vehicles, spacecraft, and so much more.[10]

As the developer community contributing to Linux-based improvements has grown exponentially over the years, so has the number of attacks targeting and impacting Linux products. We need your help to redefine how we secure widely used, critical code by automatically finding and fixing software vulnerabilities at the necessary speed and scale for real-world applicability.

2.3.2 The Vulnerability

The first Linux exemplar CPV is a reintroduction of the published vulnerability CVE-2021-43267.[11] It is located in the Transparent Inter Process Communication (TIPC) subsystem within the Linux Kernel. This protocol allows communication across clusters on a network via UDP or directly via Ethernet. It behaves much like Unix Domain sockets through between cluster nodes.[12]

The vulnerability is heap-based buffer overflow in the processing of a MSG_CRYPT packet type due to a failure to validate the size field of the included key. The kernel allocates a buffer based upon the size indicated in the packet but copies a length base upon the actual size of the provided

8 https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-linux-threat-landscape-report
9 https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel#:~:text=The%20Linux%C2%AE%20kernel%20is,resources%20as%20efficiently%20as%20possible
10 https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-linux-threat-landscape-report
11 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43267
12 https://www.kernel.org/doc/html/latest/networking/tipc.html

key material. This disparity allows an attacker to specify a small size while providing a larger key. The public patch[13] injects additional size checks and subsequently returns an error if the size field is invalid. Additionally, there is a publicly available exploit along with a more detailed analysis.[14,15]

The vulnerable function along with a diff of the vulnerable and non-vulnerable kernel code can be viewed in **Appendix A**.

If this were a competition-CPV, during the event, competing CRSs would be responsible for locating this vulnerability, identifying the vulnerability class and AIxCC sanitizer that maps to that class, and demonstrating it via a proof-of-vulnerability that triggers the targeted sanitizer. The CRSs would then be able to supply a Generated Patch (GP) to remediate the vulnerability. Specific details on competition scoring and sanitizers can be found in the Scoring Algorithm, detailed in **Part One**.

A relatively simple CodeQL query identified and returned CVE-2021-43267 as one of 60 potential vulnerabilities in Linux.[16] AIxCC is designed to encourage teams to improve these vulnerability discovery tools while minimizing the need for manual intervention for quality control and subsequent evaluations.

2.4 Future Challenges

2.4.1 Future Challenge Project Exemplars

In early 2024, the AIxCC team will begin to release full Challenge Project Exemplars to the public. The first exemplar will be the Linux Kernel with the exemplar vulnerability described above. The second exemplar will be the open-source automation server and CI/CD solution, Jenkins, with an exemplar vulnerability of its own.

2.4.2 Submission Systems

In the months leading up to the competition, DARPA will release two distinct submission systems that teams may use to help design and assess their CRS ahead of competition.

First, a demo scoring submission system will be released through which a team may submit solutions for the exemplar CPVs against an online scoring system Application Programming Interface (API). This API will be designed to assess and score submissions in a way identical to

13 https://github.com/torvalds/linux/commit/fa40d9734a57bcbfa79a280189799f76c88f7bb0
14 https://haxx.in/posts/pwning-tipc/
15 https://www.sentinelone.com/labs/tipc-remote-linux-kernel-heap-overflow-allows-arbitrary-code-execution/
16 https://www.sentinelone.com/labs/tipc-remote-linux-kernel-heap-overflow-allows-arbitrary-code-execution/

the scoring system during the semifinal competition, but will be limited exclusively to the exemplar vulnerabilities that have been publicly released. Teams are encouraged to use this demo submission system to ensure their CRS is adequately able to solve challenges and submit solutions.

Second, a CRS submission portal will be opened through which a team may submit their CRS in full, to be tested within the AIxCC competition infrastructure. The submitted CRS will be run in the same way as the competition, but it will be run exclusively against the exemplar CPs already publicly released. Teams are encouraged to use this submission portal to ensure their CRS can properly perform within the competition infrastructure.

During the semifinal competition, CRS solutions will be run in the cloud. Teams will be allotted a fixed budget for cloud infrastructure specifications that they may allocate to best fit the needs of their CRS. To provide these offerings, AIxCC is partnering with Microsoft Azure and Google Cloud Computing Services for the competition. Further details on Collaborator Resource Access will be released in March 2024.

3.0 Appendices

3.1 Appendix A - Vulnerability Code

The vulnerable function in `net/tipc/crypto.c`.

```
/**
 * tipc_crypto_key_rcv - Receive a session key
 * @rx: the RX crypto
 * @hdr: the TIPC v2 message incl. the receiving session key in its data
 *
 * This function retrieves the session key in the message from peer, then
 * schedules a RX work to attach the key to the corresponding RX crypto.
 *
 * Return: "true" if the key has been scheduled for attaching, otherwise
 * "false".
 */
static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr)
{
	struct tipc_crypto *tx = tipc_net(rx->net)->crypto_tx;
	struct tipc_aead_key *skey = NULL;
	u16 key_gen = msg_key_gen(hdr);
	u32 size = msg_data_sz(hdr);
	u8 *data = msg_data(hdr);
	unsigned int keylen;

	keylen = ntohl(*((__be32 *)(data + TIPC_AEAD_ALG_NAME)));

	spin_lock(&rx->lock);
	if (unlikely(rx->skey || (key_gen == rx->key_gen && rx->key.keys))) {
```

```
                pr_err("%s: key existed <%p>, gen %d vs %d\\n", rx->name,
                    rx->skey, key_gen, rx->key_gen);
                goto exit;
        }

        /* Allocate memory for the key */
        skey = kmalloc(size, GFP_ATOMIC);
        if (unlikely(!skey)) {
                pr_err("%s: unable to allocate memory for skey\\n", rx->name);
                goto exit;
        }

        /* Copy key from msg data */
        skey->keylen = keylen;
        memcpy(skey->alg_name, data, TIPC_AEAD_ALG_NAME);
        memcpy(skey->key, data + TIPC_AEAD_ALG_NAME + sizeof(__be32),
            skey->keylen);

        rx->key_gen = key_gen;
        rx->skey_mode = msg_key_mode(hdr);
        rx->skey = skey;
        rx->nokey = 0;
        mb(); /* for nokey flag */

exit:
        spin_unlock(&rx->lock);
        /* Schedule the key attaching on this crypto */
        if (likely(skey && queue_delayed_work(tx->wq, &rx->work, 0)))
                return true;

        return false;
}
```

The reverse patch diff that reintroduces the vulnerability.

```
diff --git a/net/tipc/crypto.c b/net/tipc/crypto.c
index 2b236d95a..8a71a1be9 100644
--- a/net/tipc/crypto.c
+++ b/net/tipc/crypto.c
@@ -2284,33 +2284,20 @@ static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr)
        u8 *data = msg_data(hdr);
        unsigned int keylen;

-       /* Verify whether the size can exist in the packet */
-       if (unlikely(size < sizeof(struct tipc_aead_key) + TIPC_AEAD_KEYLEN_MIN)) {
-               pr_debug("%s: message data size is too small\\n", rx->name);
-               goto exit;
-       }
-
        keylen = ntohl(*((__be32 *)(data + TIPC_AEAD_ALG_NAME)));

-       /* Verify the supplied size values */
-       if (unlikely(size != keylen + sizeof(struct tipc_aead_key) ||
-                    keylen > TIPC_AEAD_KEY_SIZE_MAX)) {
-               pr_debug("%s: invalid MSG_CRYPTO key size\\n", rx->name);
-               goto exit;
-           }
-
            spin_lock(&rx->lock);
            if (unlikely(rx->skey || (key_gen == rx->key_gen && rx->key.keys))) {
                        pr_err("%s: key existed <%p>, gen %d vs %d\\n", rx->name,
                               rx->skey, key_gen, rx->key_gen);
-                       goto exit_unlock;
+                       goto exit;
            }

            /* Allocate memory for the key */
            skey = kmalloc(size, GFP_ATOMIC);
            if (unlikely(!skey)) {
                        pr_err("%s: unable to allocate memory for skey\\n", rx->name);
-                       goto exit_unlock;
+                       goto exit;
            }

            /* Copy key from msg data */
@@ -2325,10 +2312,8 @@ static bool tipc_crypto_key_rcv(struct tipc_crypto *rx, struct tipc_msg *hdr)
            rx->nokey = 0;
            mb(); /* for nokey flag */

-exit_unlock:
-           spin_unlock(&rx->lock);
-
 exit:
+           spin_unlock(&rx->lock);
            /* Schedule the key attaching on this crypto */
            if (likely(skey && queue_delayed_work(tx->wq, &rx->work, 0)))
                        return true;
```

3.2 Appendix B: Acronyms

AM	Accuracy Multiplier
AIxCC	Artificial Intelligence Cyber Challenge
AoE	Areas of Excellence
API	Application Programming Interface
ASC	AIxCC Semifinal Competition
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
LLM	Large Language Model
PoV	Proof of Vulnerability
PRS	Program Repair Score
RFC	Request for Comments
SAO	Scoring Algorithm Objectives
TIPC	Transparent Inter Process Communication
VDS	Vulnerability Discovery Score