Continuous Certification of Non-Functional Properties Across System Changes: Supplement
Walkthrough

Anonymous for double-blind review

Our continuous certification process is composed by 5 phases (initial certification, adaptive system model generation, change detection, planning, and execution) described in Sections 3, 4 and 5 in the paper. In this section, we provide a complete walkthrough example of our process in execution, considering a target system (ToC) implemented as a microservice written in Java that manages applicable discounts in an e-commerce system. The microservice is composed of three components: *i)* c_{db} for database interaction, *ii)* c_{api} for serving HTTP endpoints, and *iii)* c_{cross} for horizontal functionalities such as logging and metrics.

Initial certification and adaptive system model generation. An important requirement for the target system ToC is to guarantee low response time to complete the checkout, for instance, at most 10ms with 1ms tolerance. We then consider property performance p_p=($\hat{p}_p$, {*lang*=Java, *max-time*=10ms, *tolerance*=1ms}), with $\hat{p}_p$=*Performance*, as the property to be certified. The purpose of the certification scheme is to continuously ensure that this property holds across service changes, possibly adapting certification model and certificate accordingly. At time t_0, CA prepares the first certification model $\mathcal{CM}_0$. It contains one test case tc_1 sending 50 concurrent requests to c_{api} with a varying interval. tc_1 is successful *iff* the response time is less than the threshold in p_p. On behalf of the CA, the accredited lab executes $\mathcal{CM}_0$ against ToC to collect evidence: this process is successful and evidence $\{ev\}_0$ contains the measured response times. The accredited lab also generates the system model $\mathcal{B}$ as a ML model trained on metrics collected during service execution (e.g., following the approach in [1]). Certificate $\mathcal{C}_0$=$\langle\mathcal{CM}_0$, $\{ev\}_0$, $\mathcal{B}$, $-$, *Valid*$\rangle$ is then awarded.

Change detection. At run time, metrics are continuously collected and analyzed according to $\mathcal{B}$. Source code is also monitored, as well as external vulnerability databases matching known vulnerabilities with the service and the target property performance (e.g., looking at the file `pom.xml`). Let us assume that the library `Log4j` v2.14.0 is at the basis of component c_{cross}. Vulnerability `CVE-2021-44228`[1] is retrieved according to the version of the library, hence detecting a change that possibly affects the certified property. Phase change detection returns $\langle\Delta_b$, Δ_c, $\Delta_v\rangle$, where Δ_b and Δ_c are equal to ($\bot$, $\emptyset$) and ($\bot$, $\emptyset$, $\emptyset$) respectively, and Δ_v=($\top$, {c_{cross}, `CVE-2021-44228`}). As already discussed in the paper, we take a conservative approach retrieving all relevant vulnerabilities (possibly not impacting on ToC according to p) and analyze their impact in

[1] `https://www.cve.org/CVERecord?id=CVE-2021-44228`

phase planning. The status of the certificate is therefore suspended pending the analysis of such vulnerability ($\texttt{state}(\mathcal{C}_0)$=*Suspended*).

Planning. The retrieved changes are analyzed to decide adaptive actions. According to the rules defined in phase planning, scenario *S2* is applied being (6a)$\vee$(6b)=$\top$ (the preconditions of *S0* and *S1* do not hold, and (6a) is the first evaluated to $\top$). As a consequence, the accredited lab creates a new set $\{tc_2\}_{\text{new}}$ of test cases, with tc_2 evaluating if the service can be compromised by exploiting the vulnerability: an exploit would in fact make the service unable to meet the expected response time hence affecting property p_p. The new certification model $\mathcal{CM}_1$ thus contains the same elements of $\mathcal{CM}_0$ with the addition of tc_2, while the set $\mathcal{T}$ of test cases to be executed contains tc_2 only.

Execution. The accredited lab executes the test case in $\mathcal{T}$ against ToC. Let us assume that the test case is successful and the `Log4j` vulnerability cannot be exploited. A new certificate $\mathcal{C}_1$=$\langle\mathcal{CM}_1, \{ev\}_1, \mathcal{B}, \mathcal{C}_0, \mathit{Valid}\rangle$ is awarded, where $\mathcal{B}$ is the updated system model, and $\{ev\}_1$ includes evidence $\{\overline{ev}\}$ collected according to tc_2 and evidence $\{ev\}_0$ not superseded by $\{\overline{ev}\}$.

At time t_2, the continuous certification process restarts from phase change detection, waiting for new changes to manage. Let us now assume that at time t_2, a new version of the microservice is released. Component c_{db} has been updated (code change) to improve its efficiency and, as a consequence, its behavior and the one of c_{api} as a cascading effect are affected.

Change Detection. Phase change detection identifies the above changes, and returns as output $\langle\Delta_b$=$(\top, \{c_{\text{api}}, c_{\text{db}}\})$, Δ_c=$(\top, \{c_{\text{db}}\}, \{1\})$, Δ_v=$(\bot, \emptyset)\rangle$, where $\{1\}$ is the metric value based on cyclomatic complexity of c_{db}. Given the detected changes, $\texttt{state}(\mathcal{C}_1)$=*Suspended*.

Planning. Phase planning analyzes the retrieved changes to define adaptive actions. The identified scenario is *S3* being the corresponding condition (8a)$\vee$(8b)$\vee$(8c)=$\top$ (the preconditions of *S0* and *S1* do not hold, the condition of *S2* does not hold, and (8a) is first evaluated to $\top$). A new certification model $\mathcal{CM}_2$ is defined where the maximum response time in property p_p is reduced 7ms following the performance improvement. The set $\mathcal{T}$ of test cases to be executed includes all test cases in $\mathcal{CM}_2$.

Execution. In phase execution, the accredited lab executes $\mathcal{T}$. Let us assume that the lab collects sufficient evidence following $\mathcal{T}$ and updates adaptive system model $\mathcal{B}$. A new certificate $\mathcal{C}_2$ can be released correctly representing the new version of the e-commerce microservice, such that $\texttt{state}(\mathcal{C}_2)$= *Valid*.

Again, at time t_3, the process restarts from phase change detection.

References

1. Le, V.H., Zhang, H.: Log-Based Anomaly Detection with Deep Learning: How Far Are We? In: Proc. of IEEE/ACM ICSE 2022. Pittsburgh, PA, USA (May 2022)