

The EVerest Dataset for Secure Software Engineering

Sophie Corallo✉
Karlsruhe Institute of Technology
Karlsruhe, Germany
sophie.corallo@kit.edu

Debora Grupp
Karlsruhe Institute of Technology
Karlsruhe, Germany
udimk@student.kit.edu

Dominik Fuchß
Karlsruhe Institute of Technology
Karlsruhe, Germany
dominik.fuchss@kit.edu

Jan Keim
Karlsruhe Institute of Technology
Karlsruhe, Germany
jan.keim@kit.edu

Frederik Reiche
Karlsruhe Institute of Technology
Karlsruhe, Germany
frederik.reiche@kit.edu

Tobias Hey
Karlsruhe Institute of Technology
Karlsruhe, Germany
hey@kit.edu

Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolek@kit.edu

Abstract

End-to-end security verification, from requirements through architecture to code, requires datasets that span all three artifact types with fine-grained security labels. No existing dataset provides this combination. We present the *EVerest dataset*, a multi-artifact resource based on EVerest, an industry-driven open-source software stack for electric vehicle charging stations. The dataset includes 84 manually elicited security requirements annotated with security objectives, 1,445 fine-grained security elements (components, entities, data, data flows, states, etc.), acceptance windows, coreferences, and architectural trace links, as well as the EVerest software architecture model, source code, and natural language documentation. It enables research on security requirements classification, named entity recognition, architectural trace linking, and design-time or code-level security verification. During dataset creation, a real security weakness (CWE-1295) was identified, disclosed to the project maintainers, and subsequently fixed. The dataset is publicly available [4]. A short video is available at https://youtu.be/pnn1uqpomvQ.

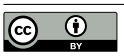
CCS Concepts

• **Software and its engineering** → **Requirements analysis**; *Software verification and validation*; • **Security and privacy** → *Software security engineering*.

Keywords

Dataset, Case Study, Security, Requirements, Architecture, Code

ACM Reference Format:
Sophie Corallo, Debora Grupp, Dominik Fuchß, Jan Keim, Frederik Reiche, Tobias Hey, and Anne Koziolek. 2026. The EVerest Dataset for Secure Software Engineering. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3832783.3834614



This work is licensed under a Creative Commons Attribution 4.0 International License.
ASE '26, Munich, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3834614

1 Introduction

Ensuring software security requires holistic verification across the development chain, from natural language requirements through software architecture to implementation. This includes (i) classifying security-relevant requirements and their objectives; (ii) recognizing security-relevant named entities in requirements and tracing them to architectural elements; and (iii) checking whether the implementation satisfies the resulting constraints. Omitting any of these steps can introduce security weaknesses. Though each step is well-studied, their combination is rarely explored due to missing datasets. Existing datasets either cover only requirements (e.g., PROMISE NFR) or include code but no architecture (e.g., DiverseVul). No dataset spans requirements, architecture, *and* code, preventing research on end-to-end security verification.

We present the *EVerest dataset*, a multi-artifact resource derived from EVerest, an industry-driven open-source software stack for Electric Vehicle (EV) charging stations. As shown in Figure 1, it contains (1) **84 security requirements** manually elicited from EVerest documentation and developer interviews, annotated with security objectives, 1,445 fine-grained security elements (components, entities, data, data flows, states, etc.), acceptance windows, references, and coreferences; (2) a **software architecture** model with requirement-to-architecture trace links; (3) the **source code** of EVerest; and (4) its **natural language documentation**. Available under the Apache License 2.0 [4], the dataset supports research on security requirements classification, named entity recognition, architectural trace linking, and design-time or code-level security verification. We also uncovered a real security weakness: a violation of the authentication token storage requirement (CWE-1295).

2 Related Datasets

To contextualize the EVerest dataset, we survey requirements engineering and software security datasets. We focus on datasets with textual artifacts and compare them by artifact coverage, security-label granularity, and availability. Table 1 summarizes the results.

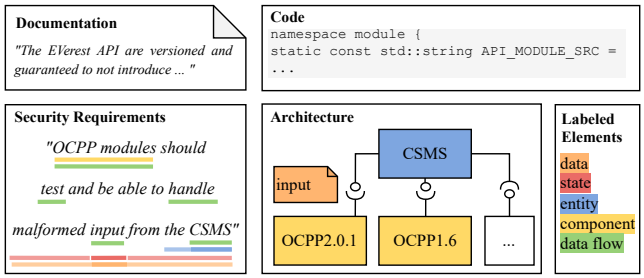


Figure 1: EVerest dataset: documentation, labeled security requirements, architecture, source code, and arch. trace links.

Dalpiaz [5] provides requirements from 22 web sources as user stories. QuRE [7] contains Mercedes-Benz specification requirements annotated with weak words and defects. PURE [8] collects requirements automatically extracted from 79 online documents and labels structural properties. Later studies re-labeled PURE for other tasks, such as distinguishing requirements from non-requirements [12]. However, some variants were not published [13, 17] or lack sufficient description [21]. PROMISE NFR [3] is a classification dataset that contains requirements from 15 projects, labeled with one functional and eleven non-functional classes, two of which are security-relevant. The dataset was further relabeled and extended by Dalpiaz et al. [6] and in NICE [18].

Other datasets better support security requirements classification. Slankas and Williams [22] labeled collected requirements with functional and 14 non-functional classes, including six security-related ones. Riaz et al. [20] explicitly label requirements by security objectives. Varenov and Gabdrahmanov [23] similarly provide security requirements from other datasets for multi-class labeling.

All datasets above are limited to requirements and lack complementary artifacts such as architecture or source code. Wang et al. [24] go further by linking security- and non-security-labeled requirements to source code. SecReq [14] labels requirements from three public specifications, whose design or architecture can partly be inferred; however, the exact source documents are undisclosed.

Security datasets with multiple artifacts usually exclude requirements. BADS [10] pairs code snippets with natural-language descriptions of intent, vulnerabilities, and severity. DiverseVul [2] provides vulnerability-related functions from security commits, annotated with commit messages and CWE identifiers.

No existing dataset covers requirements, architecture, and code while providing labels for both security-objective and generic requirements classification. EVerest addresses this gap.

3 About EVerest

EVerest is an open-source, modular software stack for EV charging stations, covering the full range from low-level hardware drivers to high-level charging protocols. Initiated by PIONIX GmbH and hosted by the Linux Foundation Energy, it is developed openly on GitHub [9] with an active community that meets regularly. *everest-core*, containing the central charge controller logic, was first released in December 2022. By June 2024, roughly 40 contributors had grown the codebase to approximately 50 kloc across more than 500 files, written in C++, C, JavaScript, Python, and Rust. While the project ships user-facing documentation, it provides neither formal requirements documents nor architectural models.

Table 1: Related datasets by artifact type. Parentheses denote partial coverage (✓) or derivable artifacts (-).

Artifact	Dalpiaz [5]	QuRE [7]	PURE [8]	PROMISE NFR [3]	Dalpiaz et al. [6]	NICE [18]	Slankas & Williams [22]	Riaz et al. [20]	Varenov [23]	Wang et al. [24]	SecReq [14]	BADS [10]	DiverseVul [2]
Req.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	(✓)
F/Q	-	-	(-)	✓	✓	✓	✓	-	-	-	-	-	-
Sec.	-	-	(-)	✓	-	(✓)	(✓)	✓	-	✓	✓	-	-
Sec.Obj.	-	-	-	-	-	-	(✓)	✓	✓	-	-	-	-
Arch.	-	-	-	-	-	-	-	-	-	-	(-)	-	-
Code	-	-	-	-	-	-	-	-	-	(-)	-	✓	✓

4 Dataset Construction

Since EVerest lacked both documented security requirements and a formal software architecture model, we constructed the dataset from scratch in cooperation with PIONIX in four steps: (1) We elicited coarse-grained security requirements from the EVerest community via an online questionnaire. (2) We conducted semi-structured interviews with EVerest developers to refine these requirements to the architectural level, yielding fine-grained requirements that explicitly reference specific EVerest components. (3) We derived a software architecture model from the EVerest source code. (4) Multiple annotators labeled acceptance windows, references, and coreferences of security elements in the requirements, as well as the contained trace links; disagreements were resolved through inter-annotator agreement.

To broaden the dataset's applicability, for example, for coarse-grained requirements classification, we additionally included architecture excerpts from the EVerest project documentation.

4.1 Questionnaire

Employing multiple elicitation techniques has been shown to improve the completeness and quality of requirements [25]. Thus, we designed an online questionnaire for step (1) to systematically elicit initial security requirements from the broader EVerest community. Given that participants may not have formal security expertise, we structured the questionnaire around four established security objectives (confidentiality, integrity, availability, and authentication).

The questionnaire comprises seven parts: an introduction, a demographics section, four objective-specific sections, and a closing section for uncategorized requirements. The introduction outlines the structure, defines security requirements, and provides writing guidelines. The demographics section captures participants' professional background and their self-assessed familiarity with EVerest and software security proficiency. For each security objective, participants are provided with the corresponding ISO 27000 definition [11], domain-specific elicitation prompts [15], and an example to anchor their responses.

Prior to distribution, we conducted a pilot study with two doctoral researchers with a security background, leading to minor refinements of the instrument. The questionnaire was then disseminated via the EVerest developer mailing list and announced at two consecutive weekly community meetings. Seven participants responded: three from Pionix and four from other organizations.

In total, participants submitted 67 security requirements. To ensure data quality, we applied a systematic cleanup procedure: removing responses that were overly general or unrelated to security, resolving coreferences, discarding non-requirement sentences, excluding answers that merely restated questionnaire examples, and correcting category misclassifications. After cleanup, 57 requirements were retained, 10 on authentication, 25 on confidentiality, 8 on integrity, 11 on availability, and 5 uncategorized, with two requirements carrying multiple labels. The left side of Figure 2 shows a representative requirement obtained from the questionnaire.

4.2 Interviews

As the questionnaire requirements were high-level and did not refer to EVerest components, we conducted semi-structured interviews with four EVerest developers in step (2) to refine the coarse-grained requirements to the architectural level (as depicted in Figure 2).

Each interview consisted of three parts. First, the interviewer gave a brief introduction, shared the study goals and consent information, and collected general background information from the participant. In the main part, the interviewer presented a coarse architecture diagram of EVerest and, using two examples in a shared interview-specific document, explained how coarse-grained requirements should be refined by explicitly naming responsible EVerest components. The document also contained a selection of 30 security requirements from the questionnaire. The interviewee was then asked to think aloud and write down fine-grained specifications for the given requirements; the interviewer supported them with writing-recommendation hints and reminders to reference specific components. Finally, where time constraints led to incomplete sentences, the interviewer revised them afterward and sent the result to the interviewee for confirmation or correction.

After a pilot with four doctoral researchers introduced to EVerest, we remotely interviewed four EVerest developers: three from Pionix and one from Chargebyte. Each interview lasted about 90 minutes.

In total, we retrieved 93 fine-grained requirements: 41 on confidentiality, 14 on integrity, 18 on availability, 10 on authentication, and 10 other security requirements.

4.3 Architecture Modeling

EVerest is built around loosely coupled modules communicating exclusively via MQTT; the framework manages their instantiation, communication, and dependency resolution. We therefore focused on the EVerest core, which encapsulates the central application logic and orchestrates the surrounding modules.

The architecture model was constructed as a Palladio component model [19] as part of a practical course in a computer science master's program, in which three students developed it under weekly supervision by three doctoral researchers in software engineering from the EVerest repository as of 3rd June 2024 (commit 177a8e6).

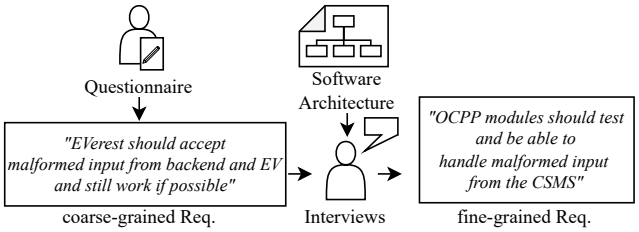


Figure 2: Req. Elicitation: Coarse-grained requirements are retrieved via a questionnaire and refined in interviews.

Each module was represented as a component with interface descriptions capturing the exchanged MQTT messages, including external modules interacting with internal ones. As the documentation lacked sufficient detail, all modeling decisions were grounded in the source code. For every contained method, the students created a service effect specification. service effect specifications describe the internal behavior of a component's service, capturing its control flow, resource demands, and interactions with other services. The resulting model comprises 29 components, 34 interfaces, and 144 service effect specifications. In the assembly model, each component is represented with its required and provided interfaces; the deployment scenario assumes a single charging station alongside external entities such as an update server and a charging station management system, though alternative deployments can be derived straightforwardly. Finally, the Palladio usage model captures 14 scenarios extracted from the existing documentation and source code, covering, among others, firmware updates, charger enable/disable operations, and limit configuration.

4.4 Labeling Requirements

To ensure that the elicited requirements were actually requirements, two annotators labeled the texts as requirements or not requirements. As a result, 84 security requirements remained.

The labels of the security objectives of the security requirements stem from the elicitation.

The labeling of security elements was performed by three annotators (one PhD researcher, two students). As element types, classes of SecLan [16] are used, which model common concepts in design-level security analyses/specifications and in security checks at the implementation level. The concepts, derived from an exploratory study, encompass and generalize all security elements of interest for security analyses. These include components, entities, data, states, hardware nodes, hardware connections, control flows, data flows, and internal activities. For this, the annotators initially started with the original definitions. Due to uncertainties, they reworked the definitions and added examples to reach a shared understanding. The resulting definitions and examples are documented in the dataset.

In shared annotation sessions, elements were labeled at two granularities. For explicitly mentioned components, entities, data, and nodes (e.g., "payment provider"), annotators marked acceptance windows: the *short sequence* captures the minimal text identifying the element, while the *long sequence* additionally encompasses articles, determiners, and modifying adjectives and adverbs (e.g.,

Table 2: Distribution of security elements in EVerest (n=1,445) over all 84 security requirements

	Components	Data	Nodes	Entities	States	Connections	Data flows	Activities	Control flows
named	88	59	35	119	587	33	35	196	56
↳ traced	77	32	–	3	–	–	–	–	–
referenced	96	33	4	16	5	2	15	10	12
coreferenced	11	13	2	8	5	1	1	1	2
total	195	105	41	143	597	36	51	207	70

"CSMS" vs. "the CSMS", as shown in Figure 1). Elements not unambiguously referring to an architectural element (e.g., "the provided interface") were marked as *references*; *coreferences* point back to the short sequence of the referenced element. For connections and data flows, annotators marked and linked the *source* and *target* elements; additional type-specific attributes were recorded where applicable, for instance, data flow labels capture the *data* transmitted and the *transmission* verb. All label definitions, descriptions, and supporting references are documented in the dataset's labeling guidelines.

In total, 1,445 elements were labeled across the 84 requirements (Table 2). The distribution is dominated by states (587), reflecting fine-grained use of that category, followed by activities (196), entities (119), components (88), data (59), nodes (35), data flows (35), and connections (33). Despite interviewees being asked to reference specific architectural locations, 99 occurrences were classified as non-specific references; coreferences were used sparingly.

Architectural trace links were annotated analogously, with conflicts resolved through inter-annotator agreement. The gold standard provides model element IDs for entity-like elements (e.g., components, data, nodes, and entities) as visualized by the color-coded architecture in Figure 1. Overall, the labeling process required approximately 100 person-hours across three annotators.

5 Summary and Outlook

This paper presented the EVerest dataset, a multi-artifact resource based on EVerest, an industry-driven open-source software stack for EV charging stations, spanning natural language documentation, security requirements, software architecture, and source code. The 84 manually elicited security requirements are annotated with security objectives, 1,445 fine-grained security elements, acceptance windows, coreferences, and architectural trace links. The dataset thereby addresses a recognized gap in the field, enabling research on security requirements classification, named entity recognition, architectural trace linking, and design-time as well as code-level security verification within a single resource.

During dataset creation, a concrete security weakness was identified using xDECAF [1]: Requirement 5 stipulates that "[…] tokens used for authentication should not be stored in plain text in log files or persistent storage […]". The dataset snapshot's source code, however, violates this requirement [1]. The weakness (CWE-1295) was disclosed to PIONIX GmbH and remedied shortly thereafter, confirming the dataset's real-world relevance.

Future work will explore further weaknesses or vulnerabilities in the EVerest codebase and automated end-to-end security verification approaches leveraging the dataset's multi-artifact structure.

[1]Line 13 of auth_token_providerImpl.cpp in the PN532TokenProvider module.

Data Availability Statement

We provide our dataset at [4]. This snapshot including all 84 manually elicited and annotated security requirements with security objectives, 1,521 fine-grained security elements (components, entities, data, data flows, states, etc.), acceptance windows, coreferences, and architectural trace links, as well as the EVerest software architecture model, source code, and natural language documentation.

Acknowledgments

This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF), supported by KASTEL Security Research Labs, Karlsruhe, by the pilot program Core Informatics at KIT (KiKIT) of the HGF, by the Deutsche Forschungsgemeinschaft (DFG) under the National Research Data Infrastructure – NFDI 52/1 – 501930651, and supported by the DFG - SFB 1608 - 501798263. We thank Pionix, all EVerest developers, and our students for their contributions. Generative AI tools were used for copy editing.

References

[1] Benjamin Arp, Felix Schwickerath, Alexander Vogt, Tom Hüller, Nils Niehues, and Nicolas Boltz. 2026. xDECAF: An Extensible Data Flow Diagram Analysis Framework for Information Security. In *IEEE/ACM ASE*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3832783.3834604

[2] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. 2023. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. Dataset: https://github.com/wagner-group/diversevul, version: commit 50a2c18. arXiv:2304.00409 [cs.CR]

[3] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, and Dan Port. 2007. *nfr*. doi:10.5281/zenodo.268542

[4] Sophie Corallo, Debora Grupp, Dominik Fuchß, Jan Keim, Frederik Reiche, Tobias Hey, and Anne Koziolek. 2026. *The EVerest Dataset for Secure Software Engineering*. doi:10.5281/zenodo.21157836

[5] Fabiano Dalpiaz. 2018. *Requirements data sets (user stories)*. doi:10.17632/7zbk8zsd8y.1

[6] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Başak Aydemir, and Sercan Çevikol. 2019. *explainable-re/RE-2019-Materials v0*. doi:10.5281/zenodo.3309669

[7] Henning Femmer, Frank Houdek, Max Unterbusch, and Andreas Vogelsang. 2025. *QuRE Dataset*. doi:10.5281/zenodo.16049822

[8] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. PURE: A Dataset of Public Requirements Documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 502–505. doi:10.1109/RE.2017.29

[9] LF Energy Foundation. 2024. EVerest. https://lfenergy.org/projects/everest/

[10] Yifan Huang, Weisong Sun, and Yubin Qu. 2025. A Vulnerability Code Intent Summary Dataset. Dataset version: commit 830b944. arXiv:2504.08180 [cs.SE] doi:10.48550/arXiv.2504.08180

[11] ISO/IEC. 2018. *Information technology — Security techniques — Information security management systems — Overview and vocabulary*. Technical Report ISO/IEC 27000:2018-02(E). ISO/IEC.

[12] Vladimir Ivanov, Andrey Sadovykh, Alexander Naumchev, Kirill Yakovlev, and Alessandra Bagnato. 2021. *ReqExp: BERT-based ML Model for Extracting Software Requirements*. doi:10.5281/zenodo.4630687

[13] Fatemeh Khayashi, Behnaz Jamasb, Reza Akbari, and Pirooz Shamsinejadbabaki. 2022. Deep Learning Methods for Software Requirement Classification: A Performance Study on the PURE dataset. arXiv:2211.05286 [cs.SE] doi:10.48550/arXiv.2211.05286

[14] Eric Knauss, Siv Hilde Houmb, Shareeful Islam, Jan Jürjens, and Kurt Schneider. 2021. *SecReq*. doi:10.5281/zenodo.4530183

[15] Roxanne E. Miller. 2009. *The Quest for Software Requirements*. MavenMark Books, Oconomowoc, WI, USA. https://dl.acm.org/doi/10.5555/1643607

[16] Sven Peldszus, Frederik Reiche, Kevin Hermann, Sophie Corallo, Thorsten Berger, and Robert Heinrich. 2026. Can I Check What I Designed? Mapping Security

Design DSLs to Code Analyzers. arXiv:2605.07814 [cs.CR] doi:10.48550/arXiv.2605.07814

[17] Kiramat Rahman, Anwar Ghani, Osama A. Khashan, Naif Alzahrani, Javed Rashid, and Arif Ur Rahman. 2025. Req2Vec: Learning Distributed Representation of Non-Functional Software Requirements. *IEEE Access* 13 (2025), 202906–202918. doi:10.1109/ACCESS.2025.3637025

[18] Gokul Rejithkumar and Preethu Rose Anish. 2025. *[Dataset] NICE: Non-Functional Requirements Identification, Classification, and Explanation Using Small Language Models.* doi:10.5281/zenodo.14590935

[19] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann (Eds.). 2016. *Modeling and Simulating Software Architectures – The Palladio Approach.* MIT Press. https://mitpress.mit.edu/9780262336796/modeling-and-simulating-software-architectures/

[20] Maria Riaz, Jason King, John Slankas, and Laurie Williams. 2014. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE).* 183–192. Original dataset no longer available, dataset: https://github.com/iambackend/Riaz-Dataset, version: commit 77cb510. doi:10.1109/RE.2014.6912260

[21] Computer Science3. 2023. last accessed: 04-10-2026. https://www.kaggle.com/datasets/computerscience3/public-requirementspure-dataset/data

[22] John Slankas. 2019. *NFRLocator.* Dataset version: commit 53de7dc. https://github.com/RealsearchGroup/NFRLocator

[23] Vasily Varenov and Aydar Gabdrahmanov. 2021. Security Requirements Classification into Groups Using NLP Transformers. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW).* 444–450. doi:10.1109/REW53955.2021.9714713

[24] Wentao Wang, Kavya Reddy Mahakala, Arushi Gupta, Nesrin Hussein, and Yinglin Wang. 2019. Data on security requirements in open-source software projects. *Data in Brief* 25 (2019), 103590. doi:10.1016/j.dib.2018.12.029

[25] Masooma Yousuf and M.Asger M.Asger. 2015. Comparison of Various Requirements Elicitation Techniques. *International Journal of Computer Applications* 116 (04 2015), 8–15. doi:10.5120/20322-2408