2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

A Dataset of Parametric Cryptographic Misuses

Anna-Katharina Wickert, Michael Reif, Michael Eichberg, Anam Dodhy, and Mira Mezini
Software Technology Group
Technische Universität Darmstadt
Darmstadt, Germany
⟨wickert,reif,eichberg,mezini⟩@cs.tu-darmstadt.de, a.dodhy@gmail.com

***Abstract*—Cryptographic APIs (Crypto APIs) provide the foundations for the development of secure applications. Unfortunately, most applications do not use Crypto APIs securely and end up being insecure, e.g., by the usage of an outdated algorithm, a constant initialization vector, or an inappropriate hashing algorithm. Two different studies [1], [2] have recently shown that 88% to 95% of those applications using Crypto APIs are insecure due to misuses. To facilitate further research on these kinds of misuses, we created a collection of 201 misuses found in real-world applications along with a classification of those misuses. In the provided dataset, each misuse consists of the corresponding open-source project, the project's build information, a description of the misuse, and the misuse's location. Further, we integrated our dataset into MUBench [3], a benchmark for API misuse detection. Our dataset provides a foundation for research on Crypto API misuses. For example, it can be used to evaluate the precision and recall of detection tools, as a foundation for studies related to Crypto API misuses, or as a training set.**

***Index Terms*—Cryptographic misuse, API-misuse, dataset, benchmark**

I. Introduction

Today, many applications handle or store sensitive information and this information is protected using cryptography. However, recent research [1], [2], [4], [5] has shown that developers struggle with the correct and secure usage of cryptographic APIs (Crypto APIs) and, therefore, end up with insecure crypto which can cause vulnerabilities in their programs. An example of an insecure usage is an outdated hash algorithm, e.g., *SHA-1*, or a weak encryption algorithm, e.g., *DES*. Recent research on the security of Android apps [1], [2] has revealed that 88% to 95% of the apps using Crypto APIs include at least one usage of a Crypto API, which is unsafe. In another study, Ma et al. identified that 65% of all analyzed Android apps use a broken hash function [6].

Besides analyzing source code, other research focuses on user studies to identify the reason for misuses, e.g., insufficient documentation [5], [7]. A study by Meng et al. concentrated on Stack Overflow posts and showed that developers struggle to understand the implications of their security decisions [8]. Lazar et al. [4] analyzed entries on vulnerabilities and identified that 83% of these entries are due to API misues rather than an insecure implementation within the library.

A crypto misuse, in the following referred to as a *misuse*, is some code that uses a Crypto API such that it is considered insecure by experts, such as the usage of *SHA-1* as a hashing algorithm [9]. Thus, an application using *SHA-1* for sensitive information is insecure.

To address the problem with the prevalence of Crypto API misuses, static analysis tools to detect Crypto API misuse have been proposed, such as *CryptoLint* [1] and *CogniCrypt*$_{SAST}$ [2]. However, currently, we lack labeled datasets of real-world crypto misuses. In lack of such labeled datasets, current tools check source code with manually created rule sets—an error-prone, tedious, and time-consuming process—and the validity of their results is assessed in a non-systematic way.

A labeled real-world dataset can be used both as the ground truth to evaluate the precision and recall of such tools and as a learning basis for approaches that semi-automate the rule creation process, i.e., by inferring basic API templates and method call protocols.

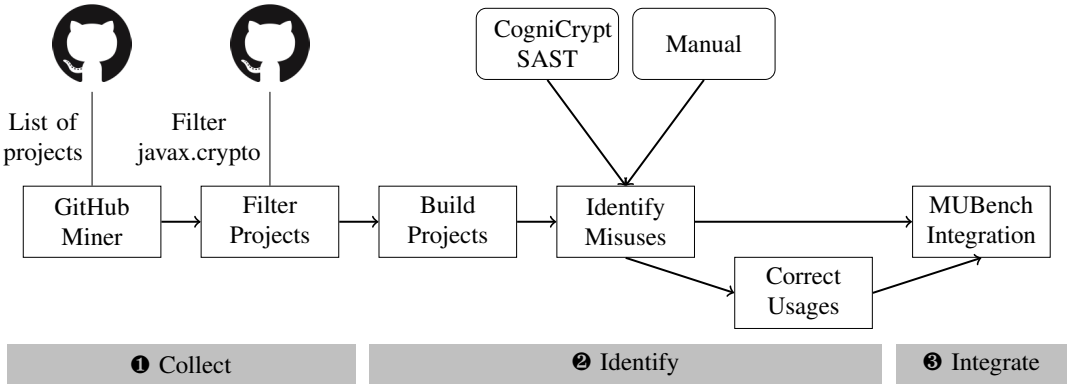
To address this need, we have created a real-world dataset of Crypto API misuses focused on parametric misuses as one of the most severe misuse problems [2]. A parametric Crypto API misuse occurs due to passing an insecure configuration as a parameter. For example, in the Java Crypto API, the en-/decryption algorithm and the padding mode are specified using a single parameter. Our dataset describes each misuse and specifies where the code was found. We provide a GitHub URL, a GitHub commit id of a version with a misuse, a misuse description, and an exact location (source file and method) of the misuse. To generate the dataset, we mined projects from GitHub, identified respective misuses, and collected in total 201 misuses. Overall, the paper makes the following contributions:

- The parametric crypto misuse dataset, which contains real-world crypto misuses collected from GitHub.
- A set of secure solutions for identified misuses.
- The integration of our dataset in MUBench, a benchmark for API-misuse detection, which is available on GitHub: https://github.com/stg-tud/MUBench/pull/427.
- An evaluation of precision and recall of *Find Security Bugs* [10] upon our dataset which we publish on our review site: http://mubenchmsr.akwickert.de.

In the remainder of the paper, we present our methodology for creating the dataset, how we store our data, details about our dataset, an evaluation case-study based on our dataset, and possible future work based on our dataset.

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00023

Fig. 1. Overview of our methodology to create the dataset.



II. Creating a dataset of Parametric Cryptographic Misuses

In this section, we describe our methodology to create a dataset of parametric Crypto API misuses. To build this dataset we:

1) mined GitHub projects which use the standard Java Cryptography Architecture (JCA) (Section II-A),
2) identified parametric Crypto API misuses within those projects, (Section II-B),
3) and added these projects to MUBench [3] (Section II-C).

Figure 1 illustrates these steps.

A. Collecting Projects Using Java's Standard Cryptographic Library

Our aim was to collect real Java projects which use the JCA library and in particular *javax.crypto* [11]. In the remainder of this section, we will use the term Crypto API to represent this API. For that, we used GitHub's API to find Java projects. Unfortunately, a limitation of GitHub's API is that a combined search of repositories and code within each of these repositories is not possible, and, thus we collected our projects in two steps.

The first step towards a dataset of real-world projects was to collect *meaningful* Java GitHub projects. To ensure that the projects are known, recent, and non-toy projects, we selected only projects with more than 100 stars that were created between the 1st of July 2015 and the 1st of August 2018, and which have more than 100 commits. We ended up with 1369 projects.

In step two, we removed those previously identified projects that do not use Crypto API functionality. As a Crypto API usage, we considered all usages of the Crypto API. In total, only 134 of the 1369 Java projects use a Crypto API.

For both steps, we developed a Python script which uses the *github3.py* library; the implementation and the resulting files are available on GitHub: https://github.com/stg-tud/github-query-script [12].

B. Identify Cryptographic API Misuses

After our first step, in which we collected Java projects using a Crypto API, we run a static analysis to identify misuses of the Crypto API. We identified misuses with the help of $CogniCrypt_{SAST}$, a state-of-the-art static analysis tool for finding Crypto API misuses [2]. Given that $CogniCrypt_{SAST}$ only analyzes binary code, we had to build all projects. This reduced the number of projects to 53, because we were not able to build the rest of the projects.

With the help of $CogniCrypt_{SAST}$, we analyzed each of the 53 projects; $CogniCrypt_{SAST}$ generated reports for 39 projects with detailed information about each identified misuse. Unfortunately, $CogniCrypt_{SAST}$ failed for the remaining 15 projects which we therefore analyzed manually. For our manual review process, we looked up and used the rules which are used internally by $CogniCrypt_{SAST}$. After combining the results of both steps, we found 43 projects which have in total 201 parametric Crypto API misuses.

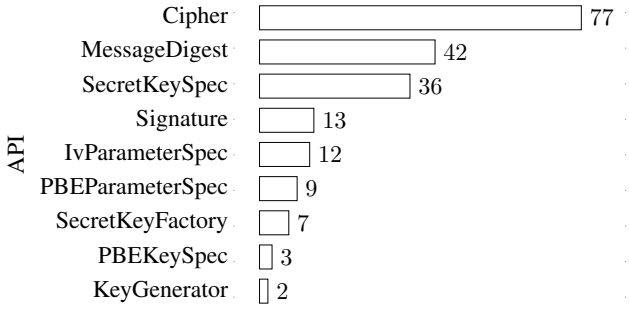
To further enrich our dataset, we hand-crafted secure solutions for 7 projects originally having 44 misuses. For each misuse, we created a corrected version and afterwards verified the correctness with $CogniCrypt_{SAST}$. We selected our projects such that each category is represented in at least two projects and ended up with the following projects: *dragonite-java, habridge, instagram4j, jeesuite-libs, nettygameserver, smart, and whatmars*.

C. Store Data in MUBench

After we identified our misuses, we stored the misuses in MUBench, a benchmark for API-misuses. MUBench uses the following three different *yaml* files to represent a misuse [13]:

- *project.yml* - Describes information about the project in general including the URL to its source code repository.
- *version.yml* - Describes information about a version of a project through a commit id, the identified misuses within this version, and commands to build this version.

Fig. 2. Details on the misuses found in the dataset.



- *misuse.yml* - Describes information about the misuse including the location of the misuses, and the kind of misuse.

Besides storing misuses, MUBench allows to include Java files in the *correct-usages* folder where each file represents a misuse's fix.

III. Identified Misuses in Detail

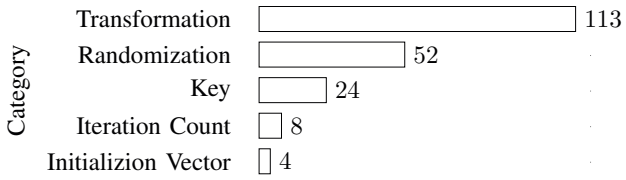
In total, we identified 201 misuses within 43 different projects. The two most misused classes *Cipher* and *MessageDigest* make up more than 59% of the identified misuses. It is important to note that we include all identified and verified misuses of each project to label all misuses which we are aware of. Thus, certain misuses occur more often than others in our dataset. We provide more details on the included classes and the number of misuses in Figure 2.

While analyzing our dataset, we observed that developers struggle with the API due to a few underlying crypto concepts. These concepts, like a key, are used by crypto algorithms and often influence the algorithm's security directly. From now on, we refer to these concepts as categories and identified the following five in our dataset: Transformation, Key, Initialization Vector, Randomization, and Iteration Count.

The *Transformation* category comprises misuses that transform information with considerably insecure algorithms, e.g., an insecure encryption algorithm or an insecure message digest. The *Key* category includes misuses that pass an insecure key to a function. Misuses of the *Initialization Vector* category pass an insecure initialization vector to a Crypto API function, e.g., due to constant value used for the creation of an initialization vector. When a misuse's cause is a constant or a not properly randomized parameter that is then passed to a API function, it belongs to the *Randomization* category. The last category, *Iteration Count*, describes misuses where an algorithm must be executed iteratively for a number of rounds to be secure but it is only performed a few times.

We enriched our dataset with these categories so that we can differentiate between the misuses. Figure 3 describes the distribution of our misuses among our categories.

Fig. 3. Misuses per each category



IV. Dataset Usages

Our dataset can be used as a foundation for further Crypto API misuse research. Next, we describe three different fields of applications: Evaluation of static analysis tools, research focused on Crypto APIs, and foundation for learning approaches. To illustrate one use case, we will present an evaluation using our data set targeting the static analysis tool *Find Security Bugs*.

A. Evaluation of Static Analysis Tools

First, our dataset facilitates the evaluation of the precision and recall of static analysis and bug detection tools because we integrated our dataset in MUBench [14].

To evaluate a tool with MUBench, one has to execute a Docker command to start an interactive Shell, run the experiments, and publish the data to a review site [13]. For this evaluation, we chose the static analysis tool *Find Security Bugs* [10], selected ten projects of our dataset, evaluated precision and recall, and published our evaluation on our review site: http://mubenchmsr.akwickert.de.

Our analysis reveals that *Find Security Bugs* is usable in practice [15] due to a precision of 100%. The recall is, however, only 42,7%. In both experiments, we observe most of the true positives for our category *Transformation* with 94,37% and 96,88% for precision and recall.

Besides the excellent results for the category *Transformation*, *Find Security Bugs* only identifies true positives in the category *Randomization*. These results indicate possible future improvements to detect more parametric crypto misuses.

B. Research on Crypto APIs

Second, future research on Crypto APIs can use our dataset as a foundation and rely on the detailed information for each misuse which we provide. Each misuse comes with a GitHub URL, build information, and the exact location, and, thus, it is possible to gather further information, e.g., a commit message which introduces a misuse. Based on this information, researchers can explore many further aspects of crypto misuses:

- How many tangled commits can we observe for crypto changes? How do these numbers compare to the overall project?
- Which part of the project co-evolve with crypto changes?
- Is there a connection between the number of misuses in a project and the code quality of the project?

- Using crypto is challenging. Does that mean that we can observe more documentation in crypto implementations than in the code?
- Does a correlation between authors and crypto misuses exist?

C. Training Set for Learning Algorithms

Third, our dataset can be used as a training set for learning algorithms. We provide the first labeled dataset for parametric crypto misuses. Thus, we build a starting point for future research on this topic, e.g., by syntactically generating more labeled data [16].

V. Future Improvements

Our dataset concentrates on parametric Crypto API misuses. They are one of the most significant sub problems of crypto misuses. Therefore, we did not add further crypto misuses, e.g., the correct call order of crypto functions. Those crypto misuses are a possible improvement of our dataset for future work. Another improvement that will increase the overall number of included misuses is to add other projects with misuses. However, this process is time-consuming and requires much manual work due to identifying the misuses, building the project, and including the misuses.

Another possible improvement for the future is to include more corrected usages. Even if we have ensured that for each misuse category at least two corrected versions exist, we do not provide correct usages for each project as it requires a high manual effort.

VI. Challenges and Limitations

While creating our dataset, we observed two main challenges. First, we needed to build the downloaded repositories. Despite build systems exist, this is still a challenging task due to many different possible configurations. Therefore, we only include the repositories which we managed to build.

Second, we had to determine whether a usage of the Crypto API is a misuse or not. For most of the projects, we used *CogniCrypt*$_{SAST}$ which is a state-of-the-art static analysis tool for detecting crypto misuses. To avoid false positives in our dataset, we manually verified the findings and excluded 49 reports. Further, *CogniCrypt*$_{SAST}$ did not succeed in running on all projects. Therefore, we manually identified misuses for these projects.

Due to our two main challenges, we face two limitations. First, our selection of projects is biased as we only included projects which we managed to build. Second, we can not guarantee that we identified all parametric crypto misuses in the projects and that all of our corrected versions are secure. However, we relied on a state-of-the-art tool and verified the code manually to mitigate the risk by using this tool.

The risk of using *CogniCrypt*$_{SAST}$ is that bugs in the implementation or rules can possibly cause falsely classified or unreported misuses. Implementation problems, e.g., an imprecise call graph of the static analysis, can lead to cases where *CogniCrypt*$_{SAST}$ did not manage to identify a misuse. We tried to mitigate this risk by manually verifying our results and checking the identified usages of the Crypto API. For this verification we relied on the rules of *CogniCrypt*$_{SAST}$, and, thus we rely on the correctness of these rules.

Another limitation of our dataset is that it only includes Crypto misuses of the *JCA* library. Thus, Crypto misuses of other libraries, e.g., *Bouncy Castle* [17] are not represented.

Besides these limitations, users of our dataset should be aware that security assessments change over time. Thus, it is possible that some usages, which we defined as secure, may be considered insecure in the future.

VII. Related Datasets and Work

The initial dataset provided by Amann et al. [3] includes 16 misuses of JCA which have been identified through code changes of JCA usages found on SourceForge and GitHub. Later [14], the authors extended their dataset which resulted in a total of 39 misuses of the JCA. In contrast to the existing work, we concentrated only on parametric misuses being one of the major issues [2] and included more APIs, e.g., *javax.crypto.MessageDigest*. In addition, we extended the number of misuses by a factor of five.

For a more general dimension of vulnerabilities, researchers have created different datasets. Gkortzis et al. built a dataset which maps vulnerability entries to open-source projects which can be further analyzed [18]. In contrast to our work, the dataset does not identify the vulnerability's exact location.

Several recent studies [1], [2] focus on identifying the number of cryptographic misuses. For this, the authors did not systematically describe in which method a misuse occurs. Thus, our dataset provides more details for future research about misuses; the provided information, e.g., the projects which contain the misuses, can be a starting point for extending our dataset.

VIII. Conclusion

This work presents a dataset of 201 parametric Crypto API misuses collected from GitHub projects. We integrated all misuses in an existing benchmark for API-misuses called MUBench. Besides storing misuses in MUBench, we integrated 44 corrected usages of identified misuses for 7 projects. With these misuses we benchmarked *Find Security Bugs*. This evaluation has shown that our dataset can be used for further studies on precision and recall of tools which can identify parametric Crypto API misuses. Besides benchmarking, our dataset can improve the understanding of crypto misuses, and enable learning-based algorithms to operate on a labeled set.

Acknowledgment

This work was supported by the DFG as part of CRC 1119 CROSSING, by the German Federal Ministry of Education and Research (BMBF) as well as by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP.

REFERENCES

[1] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 73–84.

[2] S. Krüger, J. Späth, K. Ali, E. Bodden, and M. Mezini, "CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs," p. 27, 2018.

[3] S. Amann, S. Nadi, H. A. Nguyen, T. N. Nguyen, and M. Mezini, "MUBench: a benchmark for API-misuse detectors," in *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*. Austin, Texas: ACM Press, 2016, pp. 464–467.

[4] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, "Why Does Cryptographic Software Fail?: A Case Study and Open Problems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14. New York, NY, USA: ACM, 2014, pp. 7:1–7:7.

[5] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, ""Jumping Through Hoops": Why do Java Developers Struggle with Cryptography APIs?" in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 935–946.

[6] S. Ma, D. Lo, T. Li, and R. H. Deng, "CDRep: Automatic Repair of Cryptographic Misuses in Android Applications," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security - ASIA CCS '16*. Xi'an, China: ACM Press, 2016, pp. 711–722.

[7] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the Usability of Cryptographic APIs," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 154–171.

[8] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty, "Secure coding practices in java: Challenges and vulnerabilities," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 372–383.

[9] G. F. O. for Information Security (BSI), "Cryptographic mechanisms: Recommendations and key lengths," Tech. Rep., May 2018.

[10] A. Philippe, "Find Security Bugs." [Online]. Available: https://find-sec-bugs.github.io/

[11] Oracle, "Java Cryptography Architecture (JCA) Reference Guide." [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

[12] A.-K. Wickert and A. Dodhy, "stg-tud/github-query-script: MSR19 Alpha," Feb. 2019. [Online]. Available: https://zenodo.org/record/2558580

[13] S. Amann, "MUBench - GitHub repository," Jan. 2019, original-date: 2016-02-12T15:26:49Z. [Online]. Available: https://github.com/stg-tud/MUBench

[14] S. Amann, H. A. Nguyen, S. Nadi, T. N. Nguyen, and M. Mezini, "A Systematic Evaluation of Static API-Misuse Detectors," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[15] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*. San Francisco, CA, USA: IEEE, May 2013, pp. 672–681.

[16] M. Pradel and K. Sen, "DeepBugs: A Learning Approach to Name-based Bug Detection," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 147:1–147:25, Oct. 2018.

[17] Legion of the Bouncy Castle Inc, "Bouncy Castle." [Online]. Available: http://bouncycastle.org/

[18] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "VulinOSS: a dataset of security vulnerabilities in open-source systems," in *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. Gothenburg, Sweden: ACM Press, 2018, pp. 18–21.