Exposing Library API Misuses via Mutation Analysis (ICSE 2019)

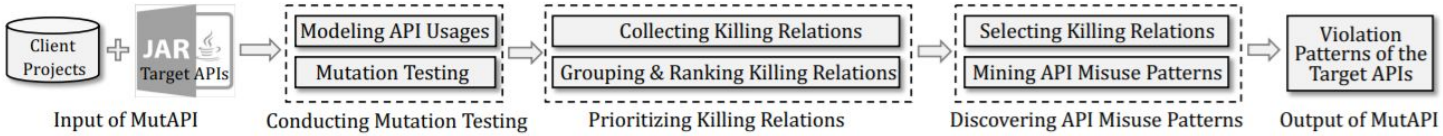

Fig. 4: Overview of MUTAPI

TABLE IV: Mutation Operators to Violate API Usages

Type	Designed Mutation Operator			Description
1	$sequence_1\ ;\ sequence_2$	$\Rightarrow$	$sequence_2\ ;\ sequence_1$	*Swapping API call sequences*
2	$sequence_1$	$\Rightarrow$	$sequence_1\ ;\ sequence_2$	*Adding an API call to a call sequence*
3	$sequence_1\ ;\ sequence_2$	$\Rightarrow$	$sequence_2$	*Deleting an API call from a call sequence*
4	$\texttt{if}\ (checker)\ \{\}\ ;\ sequence$	$\Rightarrow$	$sequence$	*Deleting the checker of receivers or parameters*
5	$structure\ \{\ sequence_1\ ;\}\ ;\ sequence_2$	$\Rightarrow$	$sequence_1\ ;\ sequence_2$	*Deleting the control structure of a sequence*
6	$structure\ \{\ sequence_1\ ;\}\ ;\ sequence_2$	$\Rightarrow$	$sequence_2$	*Deleting the control structure together with the enclosing APIs of a sequence*
7	$\texttt{if}\ (cond_1 \mid checker)$	$\Rightarrow$	$\texttt{if}\ (cond_2 \mid \texttt{true} \mid \texttt{false})$	*Changing the control condition to other conditions or boolean values*
8.1	$API(t_1, ..., t_i, ..., t_n)$	$\Rightarrow$	$API(t_1, ..., t_j, ..., t_n)$	*Replacing arguments (from t_i to t_j) of a method call*
8.2	$API(t_1, ..., t_i, ..., t_n)$	$\Rightarrow$	$API(t_1, ..., t_i, ..., t_{n+1})$	*Inserting arguments of a method call (changing API to an overloaded method)*
8.3	$API(t_1, ..., t_i, ..., t_n)$	$\Rightarrow$	$API(t_1, ..., t_i, ..., t_{n-1})$	*Deleting arguments of a method call (changing API to an overloaded method)*

MUTAPI generates mutants by applying these mutation operators on a set of client projects and collects mutant-killing tests as well as the associated stack traces. Misuse patterns are discovered from the killed mutants that are prioritized according to their likelihood of causing API misuses based on the collected information.

TABLE VII: Examples of Top-Ranked Discovered Violation Pairs

Rank	API	VIolation Pair	API Element	Description & Confirming References
1	`rcv = Line.intersection()`[1]	MISSING CHECKER	`if (rcv == null) {}`	The returning value could be null [6]
2	`Iterator<>.next()`	MISSING CALL	`Iterator<>.hasNext()`	Should check if there are sufficient tokens [4], [6]
3	`StringTokenizer.nextToken()`	MISSING CALL	`StringTokenizer.hasMoreTokens()`	Should check if there are sufficient tokens [4], [6]
4	`Integer.parseInt()`	MISSING EXCEPTION	`try {} catch(NumberFormatException)`	Might throw exceptions [6]
5	`Double.parseDouble()`	MISSING EXCEPTION	`try {} catch(NumberFormatException)`	Might throw exceptions [6]
6	`PdfArray.getPdfObject()`[2]	INCORRECT CONDITION	`if (!PdfArray.isEmpty()) {}`	Should check if the object is empty [6]
7	`rcv = SortedMap.firstKey()`	MISSING CHECKER	`if (rcv == null) {}`	The returning value could be null [4], [6]
8	`rcv = StrBuilder.getNullText()`[3]	MISSING CHECKER	`if (rcv == null) {}`	The returning value could be null [6]
10	`MessageDigest.getInstance()`	MISSING EXCEPTION	`try {} catch(NoSuchAlgorithmException)`	Might throw exceptions [56]
12	`Matcher.group()`	MISSING CALL	`Matcher.find()`	Required to be used together [57]
25	`Iterator.next()`	REDUNDANT CALL	`Iterator.remove()`	Shouldn't call `remove` during iteration [6]

1: from library `org.apache.commons.math`; 2: from library `com.itextpdf.text`; 3: from library `org.apache.commons.lang`; the others from Java

: to generate random perturbation /more faulty code examples.

An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation

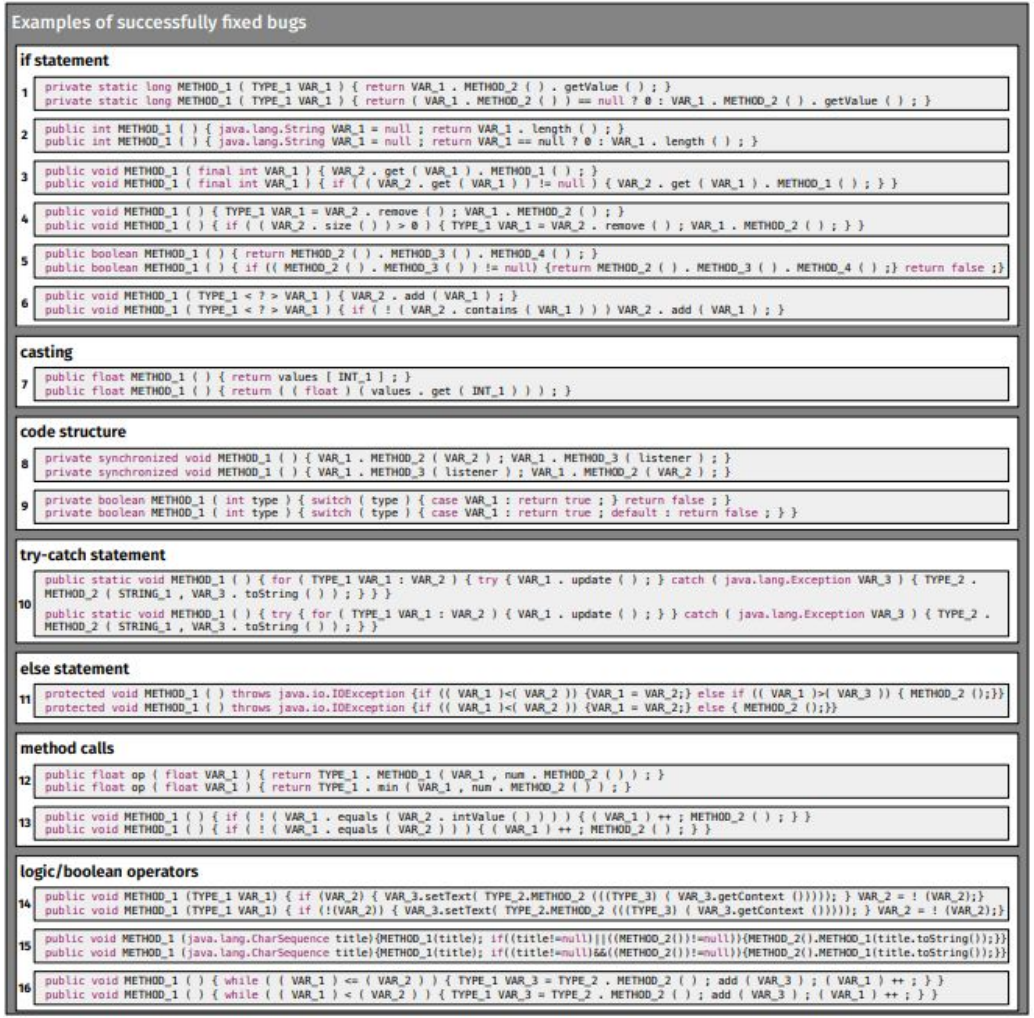

Fig. 6. Examples of successfully-generated patches.

A Systematic Evaluation of Static API-Misuse Detectors

S. Amann, H. A. Nguyen, S. Nadi, T. N. Nguyen and M. Mezini, "A Systematic Evaluation of Static API-Misuse Detectors," in *IEEE Transactions on Software Engineering Dec 2019*

TABLE 3
Capabilities of Surveyed API-Misuse Detectors

Detector	Method Calls		Conditions					Ex. Handl.		Iteration	
	Missing	Redundant	Missing `null`	Missing Val./State	Missing Sync.	Missing Context	Redundant	Missing	Redundant	Missing	Redundant
PR-MINER [11]	●	○	○	○	○	○	○	○	○	○	○
CHRONICLER [34]	●	○	○	○	○	○	○	○	○	○	○
COLIBRI/ML [12]	●	○	○	○	○	○	○	○	○	○	○
JADET [13]	●	○	○	○	○	○	○	○	○	◐	○
RGJ07 [14]	◐	○	●	●	○	○	○	○	○	○	○
ALATTIN [18]	◐	○	●	◐	○	○	○	○	○	○	○
AX09 [16]	◐	○	◐	◐	○	○	○	●	○	○	○
CAR-MINER [17]	◐	○	○	○	○	○	○	●	○	○	○
GROUMINER [15]	●	○	◐	◐	○	○	○	○	○	◐	○
DMMC [36]	●	○	○	○	○	○	○	○	○	○	○
TIKANGA [19]	●	○	○	○	○	○	○	○	○	◐	○
DROIDASSIST [20]	●	●	○	○	○	○	○	○	○	○	○

● denotes the capability to detect a violation. ◐ denotes the capability to detect a violation under special conditions. ○ denotes the inability to detect a violation.

TABLE 4
Summary of Empirical Evaluations of Surveyed API-Misuse Detectors

Detector	# of Target Projects	Eval. Setup	# of Reviewed Findings	Precision (Range)	
PR-MINER [11]	3	IP	Top 60	18.1%	(10-27%)
CHRONICLER [34]	5	IP	example-based		
COLIBRI/ML [12]	5	IP	example-based		
JADET [13]	5	IP	Top 10/project	6.5%	(0-13%)
JADET [37]	20	MP	Top 25% (50)	8.0%	(0-100%)
RGJ07 [14]	1	IP	example-based		
ALATTIN [18]	6	CP	Top 10/project	29.5%	(13-100%)
AX09 [16]	3	IP	All (292)	90.4%	(50-94%)
CAR-MINER [17]	5	CP	Top 10/project	60.1%	(41-82%)
GROUMINER [15]	9	IP	Top 10/project	5.4%	(0-8%)
DMMC [36]	1	IP	All (19)	73.7%	
DMMC [1]	3	IP	Top 30	56.7%	
TIKANGA [19]	6	IP	Top 25% (121)	9.9%	(0-33%)
DROIDASSIST [20]	not evaluated				

For the evaluation setup, IP denotes that detectors mine and detect on individual projects, MP denotes that they mine and detect on multiple projects at once, and CP denotes that they mine on 3rd-party examples before detection (cross-project).

API-Misuse Detection Driven by Fine-Grained API-Constraint Knowledge Graph *(ASE)*, 2020

It empowers the detection of three types of frequent API misuses missing calls, missing condition checking and missing exception handling.

API-constraint knowledge graph contains call-order and condition-checking relations between related APIs, and constraint-enriched return and throw relation

Table 2: Accuracy of the API-Constraint Relations

Relations	Check Points	Acc1	Acc2	AccF	Kap.
Conditioned Return	*value-literal*	98.7%	99.2%	**99.0%**	0.60
	condition clause	99.7%	99.7%	**99.7%**	1.00
Trigger on Throw	*trigger clause*	100.0%	100.0%	**100.0%**	1.00
	should not be trigger?	88.0%	85.7%	85.9%	0.87
Call-order	*API linking*	95.1%	96.4%	95.3%	0.84
	attribute	97.1%	98.7%	**98.2%**	0.62
	Should be call-order?	90.4%	91.7%	90.9%	0.86
Condition-checking	*API linking*	96.4%	94.3%	94.3%	0.77
	attribute	99.5%	99.2%	**99.5%**	0.80
	Should be cond-checking?	95.3%	96.4%	95.6%	0.74

Table 3: Six API Misuse Scenarios in Our User Study

Task	Involved API	Misuse reason	Difficulty
T-1	java.util.Arrays	Condition-checking	Easy
T-2	java.util.List, java.util.ArrayList	Condition-checking	Easy
T-3	java.io.FileReader, java.io.File, java.util.Scanner	Condition-checking	Medium
T-4	javax.swing.JFrame, javax.swing.JButton, javax.swing.JPanel	Call-order	Difficult
T-5	java.util.ArrayList, java.util.Iterator, java.util.List	Missing call	Medium
T-6	javax.swing.JFrame, java.awt.Dimension	Missing call	Difficult

Are Code Examples on an Online Q&A Forum Reliable, ICSE 2018

API misuse is caused by three main reasons—missing control constructs, missing or incorrect order of API calls, and incorrect guard conditions.

ExampleCheck efficiently searches over GitHub and retrieves an average of 55144 code snippets for a given API within 10 minutes

(chrome extension exists but still to find actual code)

Demystify official API usage directives with crowdsourced API misuse scenarios, erroneous code examples and patches

No open source

Assisting Example-based API Misuse Detection via Complementary Artificial Examples (TSE 2021)

Pipeline

Alternative iterators

Complementary imports

Inverted conditions

intermediates

A Large-scale Study on API Misuses in the Wild(ICST 2021)

Categorization of API Misuses:(9 categories)

Conditional: null checks, return value, object state

Exception: try, catch

Synchronization:

API call:

Missing,redundant calls,replaced arguments, replaced name,replaced receiver,

API Misuse Detection An Immune System inspired Approach

Table VII: Comparison of different misuse detection tools

Tools	Mono API detection	Input to learning step	Type of detected misuse
Alattin [39]	No	Code examples extracted using code search engines	Missing null checks Missing condition value or state
AX09 [41]	No	Client systems source code	Incorrect error handling
CAR-Miner [40]	No	Code examples extracted using code search engines	Incorrect error handling
Chronicler [43]	No	Client systems source code	Missing method call
DMMC [32], [38]	No	Java byte code of Client systems	Missing method call
DroidAssist [44]	Yes	Java byte code of Client systems	Missing method call
Jadet [33]	No	Client systems source code	Missing method calls Missing loops
PR-Miner [42]	Yes	Client systems source code	Missing method calls
Tikanga [34]	No	Client systems source code	Missing condition value or state
GrouMiner [28]	No	Client systems source code	Missing API elements

outliers

- Less frequent codes:
  - False positives of static detectors (based on pattern mining)
- API misuses:
  - Runtime errors
    - Conditional: null checks, return value, object state, inverted conditions
    - Exception: try, catch
    - Synchronization:
    - API call:
      - Missing/redundant calls
      - replaced arguments
      - replaced name
      - replaced receiver
      - Call sequence order
  - Vulnerable codes

APi Miners

Migration mapper

Exemplore

Prob api miner

Clams

fase

Inferring crypto API rules from code changes. ACM SIGPLAN Conference on Programming Language Design and Implementation 2018

Rumen Paletov, Petar Tsankov, Veselin Raychev, and Martin Vechev

API Class	Description
`Cipher`	A cryptographic cipher used for encryption and decryption
`IvParameterSpec`	An initialization vector (IV) used in ciphers that operate in feedback mode (e.g. CBC)
`MessageDigest`	An engine class designed to provide the functionality of cryptographically secure message digests such as SHA-1 or MD5
`SecretKeySpec`	A constructor for secret keys a byte array
`SecureRandom`	An engine class that provides the functionality of a Random Number Generator (RNG)
`PBEKeySpec`	A user-chosen password that can be used with password-based encryption

Figure 5. Target classes for learning semantic usage changes.

CRYSL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs

S. Krüger, J. Späth, K. Ali, E. Bodden and M. Mezini, in IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2382-2400, 1 Nov. 2021, doi: 10.1109/TSE.2019.2948910.

CrySL, a specification language for correct usages of cryptographic APIs. Static Bug finding using user predefined crypto rules (opensource) for android apps and maven projects.

TABLE 3
Types of API Misuses Reported by CogniCrypt$_{SAST}$ for Android Apps That use the JCA

API Misuse Type	# Warnings	# Apps
Incorrect calling sequences	4,708 (23.0%)	2,896
Incorrect parameter values	11,178 (54.7%)	3,955
Calls to forbidden methods	97 (0.5%)	62
Insecure compositions	4,443 (21.8%)	1,367
Total	20,426	4,143

https://ieeexplore.ieee.org/abstract/document/8901573

CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses

S. Afrose, S. Rahaman and D. Yao, "CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses," *2019 IEEE Cybersecurity Development (SecDev)*, 2019

ARBITRAR: User-Guided API Misuse Detection

Z. Li, A. Machiry, B. Chen, M. Naik, K. Wang and L. Song, "ARBITRAR: User-Guided API Misuse Detection," *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1400-1415, doi: 10.1109/SP40001.2021.00090.

active learning algorithm that ranks API usages by their likelihood of being invalid.

Dataset: c/c++ (openSSL, linux kernel)

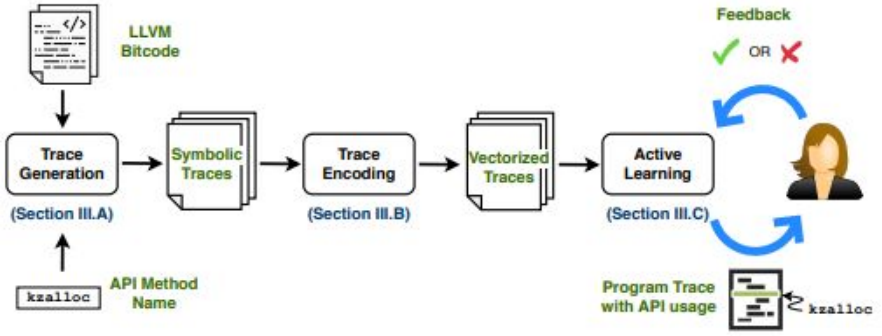

Fig. 1: Overview of ARBITRAR.