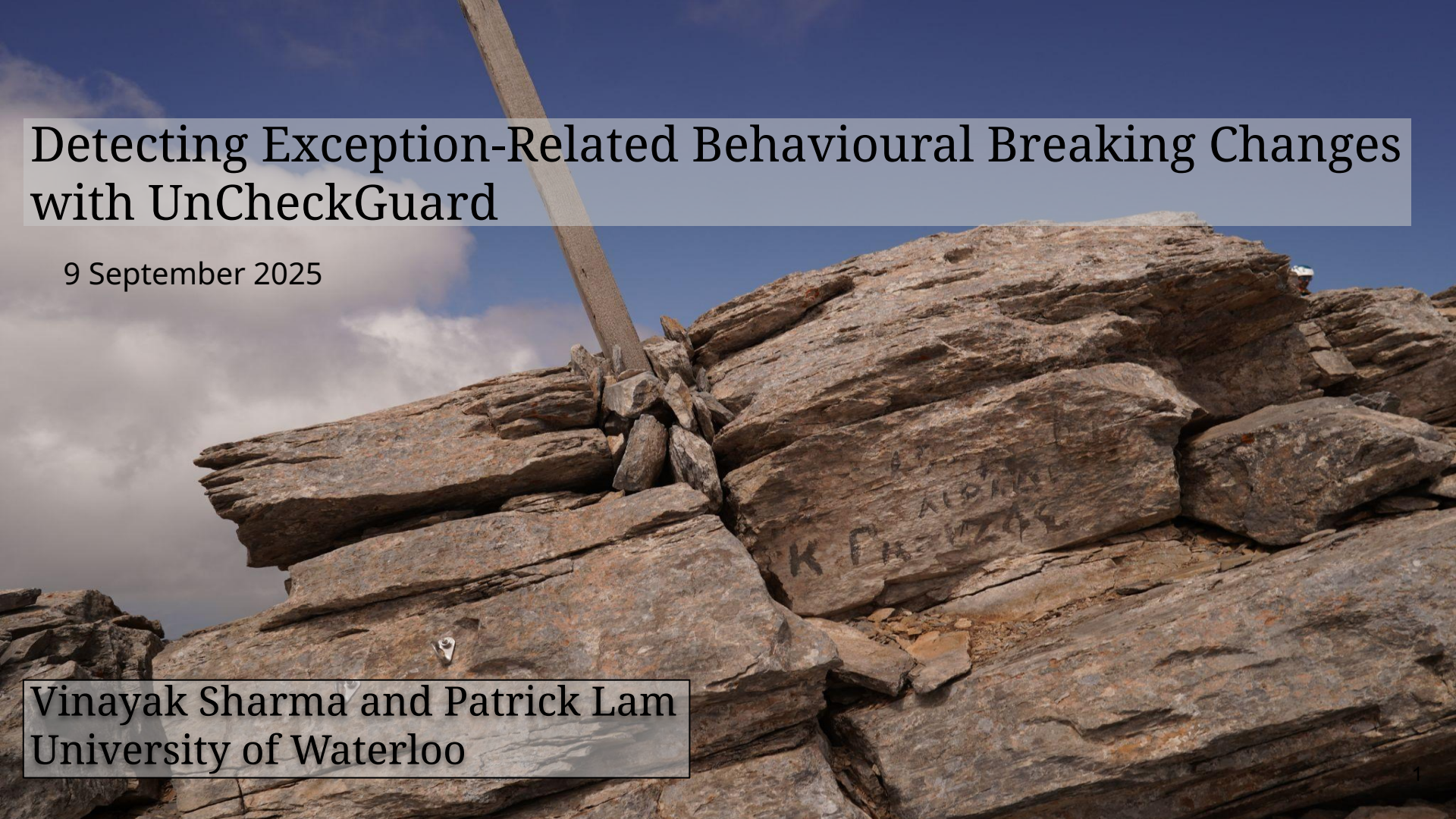
Detecting Exception-Related Behavioural Breaking Changes with UnCheckGuard

9 September 2025



Vinayak Sharma and Patrick Lam
University of Waterloo

Goal

Detect Behavioural Breaking Changes caused by newly-added exceptions

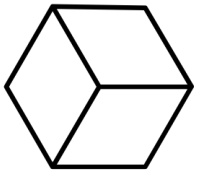
Context: Third-Party Libraries are Ubiquitous

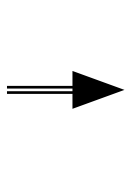
But: Library Upgrades are Expensive

“Brown metal pipe with padlock” Image: Mike Hindle, Unsplash

Types of Breaking Changes

Syntactic





Semantic
aka Behavioural

“Good Fit” Image: ochre7, Noun Project

Syntactic Breaking Changes

Handled by existing tools, for instance:

- japicmp
- Revapi

We won’t talk about them further.

Semantic/Behavioural Breaking Changes

```java
public class Account { // BEFORE
    public void withdraw(double amount) {
        System.out.println("Withdrew $" + amount);
    }
}
```

```java
public class Account { // AFTER
    public void withdraw(double amount) {
        if (amount < 0) {
            throw new IllegalArgumentException("Amount must be positive");
        }
        System.out.println("Withdrew $" + amount);
    }
}
```

Breaking change 1 (our focus): Adding new unchecked exceptions

Aside: Checked vs Unchecked Exceptions

Java defines checked exceptions:

```
void foo() throws IOException { // … }
```

and also unchecked exceptions:

```
class IllegalArgumentException
       extends RuntimeException {}
class OutOfMemoryError
       extends Error {}
```

Unchecked exceptions are not visible in method signatures.

Semantic/Behavioural Breaking Changes

```java
public class DiscountService { // BEFORE
    /**
     * Applies a discount if customer is a premium user.
     */
    public double applyDiscount(double price, boolean isPremium) {
        if (isPremium) {
            return price * 0.9; // 10% discount
        }
        return price;
    }
}
public class DiscountService { // AFTER
    /**
     * Changed: Now applies discount only for orders above $100.
     */
    public double applyDiscount(double price, boolean isPremium) {
        if (isPremium && price > 100) {
            return price * 0.9; // 10% discount
        }
        return price;
    }
}
```

Breaking change 2: User-visible change in internal logic

Detecting Behavioural Breaking Changes

e.g. [Mujahid et al 2020] and CompCheck [Zhu et al 2023]: dynamic approaches using client test cases

What if we could **statically** warn client developers about potential behavioural breaking changes caused by newly added unchecked exceptions?

Can client developers depend on semantic versioning?

Motivating Example

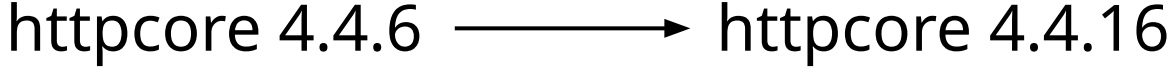
Real Example: HttpAsyncClientUtils & library httpcore

```
1  if (proxy) {
2   return HttpAsyncClients.custom()
3    .setConnectionManager(conMgr)
4    .setDefaultCredentialsProvider(credentialsProvider)
5    .setDefaultAuthSchemeRegistry(authSchemeRegistry)
6    .setProxy(new HttpHost(host, port))
7    .setDefaultCookieStore(new BasicCookieStore())
8    .setDefaultRequestConfig(requestConfig).build();
9  } else {
10   // ...
```

Client uses constructor <init>(String, int) for org.apache.http.HttpHost, with the following values:

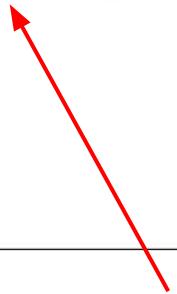
```
1     private String host = "";
2     private int port = 0;
3    // ...
```

Library update!



httpcore library: a breaking change

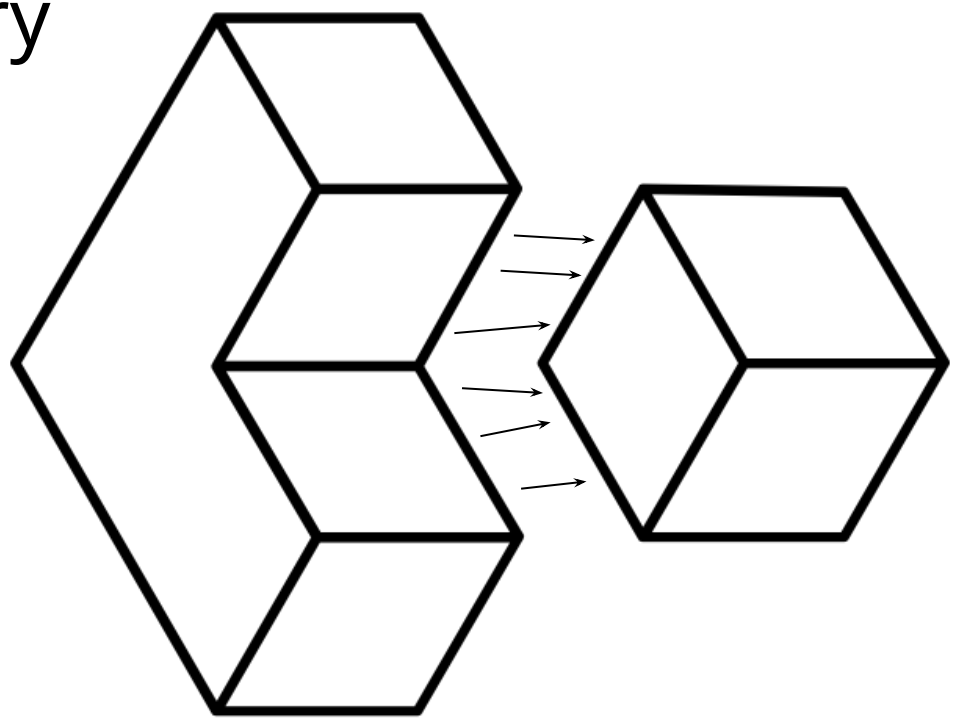
```
1  public static <T extends CharSequence> T containsNoBlanks(
     final T argument, final String name) {
2        ...
3        if (argument.length() == 0) {
4            throw new IllegalArgumentException(name + " may
                 not be empty");
5        }
6        ...
7        return argument;
8    }
```



new unchecked exception in 4.4.16

Detecting client-relevant changes

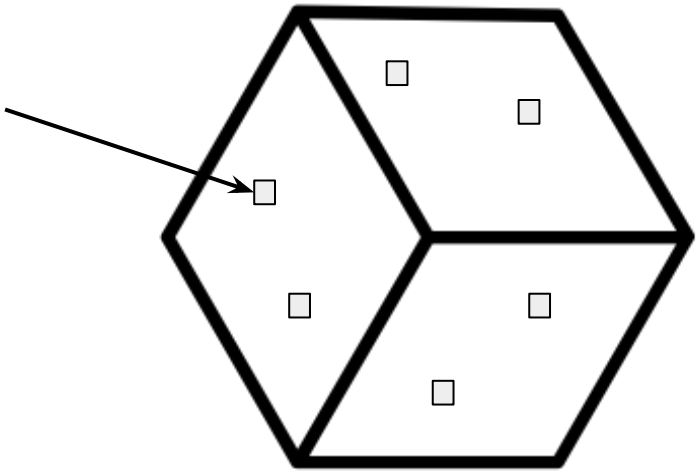
1. Find all client-library call sites



“Good Fit” Image: ochre7, Noun Project

Detecting client-relevant changes

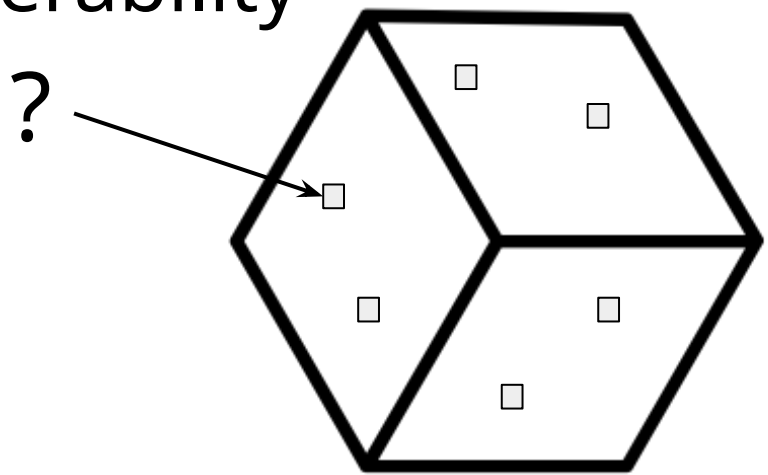
2. Find reachable new exceptions



“Good Fit” Image: ochre7, Noun Project

Detecting client-relevant changes

2a. Use taint analysis (FlowDroid) to ensure triggerability



“Good Fit” Image: ochre7, Noun Project

Why Taint Analysis?

Clearly this method throws an exception…

```
void libraryMethodAlwaysException() {
  throw new RuntimeException();
}
```

… and this one doesn’t:

```
void libraryMethodNeverException() {
    if (false) {
        throw new RuntimeException();
    }
}
```

Why Taint Analysis?

We want to know when the client can trigger the exception:

```
void libraryMethod(boolean b) {
    if (b) {
        throw new RuntimeException();
    }
}
```

This is a problem for taint analysis!

Taint Analysis Example

```
// library method
void applicableTo(PCollection<?> input) {
    if (... && input.isBounded() != IsBounded.BOUNDED) {
        throw new IllegalStateException(“...”);
    }
}
```

Library method `applicableTo` is called by the client.

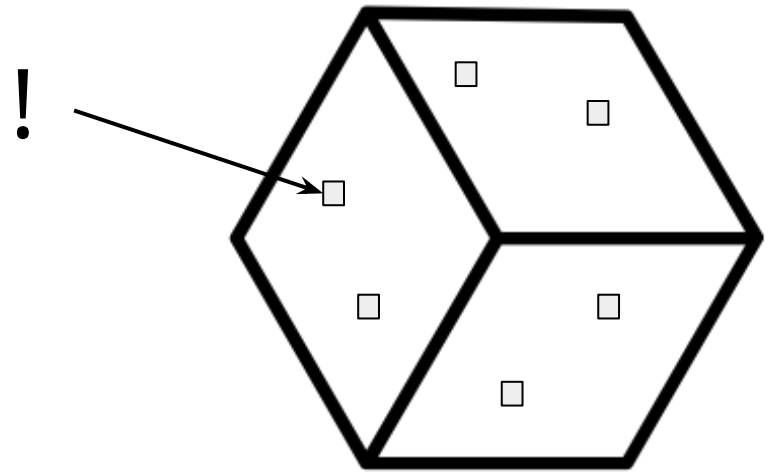
Source: parameter “`input`”

Sink: constructor for `IllegalStateException`

Taint analysis says yes.

Detecting client-relevant changes

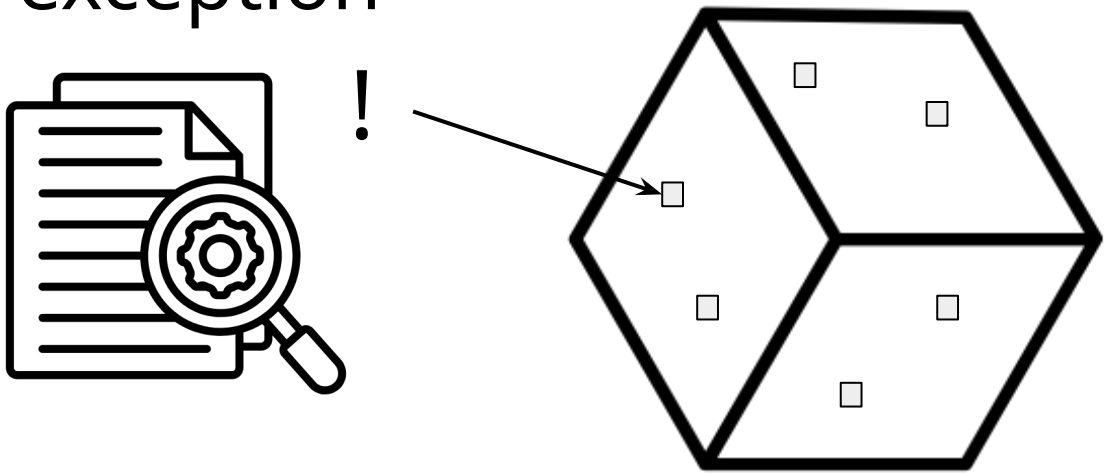
2b. Report a breaking change



“Good Fit” Image: ochre7, Noun Project

Validating client-relevant changes

3. Write a client-focussed test case triggering the exception



Research Questions (and some Answers)

Research Questions

RQ1a. Do library clients call methods with new added exceptions?
RQ1b. Is it possible for the clients to trigger these exceptions?
RQ1c. Is it possible to write client-focussed test cases triggering these exceptions?

RQ2. Under what circumstances (major/minor/patch) do new added exceptions occur?

RQ1a: Calling new exceptions

Do library clients call methods with new added exceptions?

Answer:

120/302 libraries have newly-added exceptions

Clients called methods reaching new exceptions

at 15678 callsites.

RQ1b: Triggerable exceptions

Is it possible for the clients to trigger these exceptions?

Answer:

Out of the 15678 callsites,
1708 appear to be potentially affected by new exceptions.

RQ1c: Test cases for new exceptions

Is it possible to write client-focussed test cases triggering these exceptions?

Answer:

We tried to write tests in 21 cases and succeeded for 3 of them.

RQ2: When do new exceptions occur?

We just saw a patch (x.y.**Z**) upgrade with a new exception:

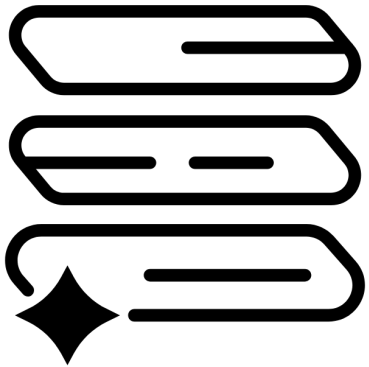
httpcore 4.4.6 ⟶ httpcore 4.4.16

RQ2: When do new exceptions occur?

Major (**X**.y.z)	50
Minor (x.**Y**.z)	57
Patch (x.y.**Z**)	14
Total	120

Data Collection

Gathering Java Client/Library Pairs



DUETS Dataset

5 stars on GitHub, with pom.xml file, single-module Maven projects

“Dataset” Image: Jubayer Islam, Noun Project

Filtering DUETS

Total clients	147,991
single-module, passing tests	34,280
10+ stars	19,290
Random sample	1,011

Methodology

Methodology

1. **Get libraries used by clients (old + new)**
2. Find external library calls from client
3. Find library methods with new exceptions
4. Filter new exceptions by taintedness
5. Manually write test cases for flagged library methods

1. Getting the Client

We obtain the client application from GitHub

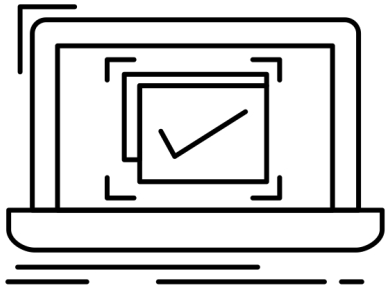
We compile the client application

We fetch the current and latest versions of all the libraries on which the client depends

“system check” Image: Flatart, Noun Project
“Library list” Image: Vittorio Scaperrotta, Noun Project

1b. Compiling the Client

We obtain the client application from GitHub



We compile the client application

We fetch the current and latest versions of all the libraries on which the client depends

“system check” Image: Flatart, Noun Project
“Library list” Image: Vittorio Scaperrotta, Noun Project

1c. Getting the Libraries

We obtain the client application from GitHub

We compile the client application

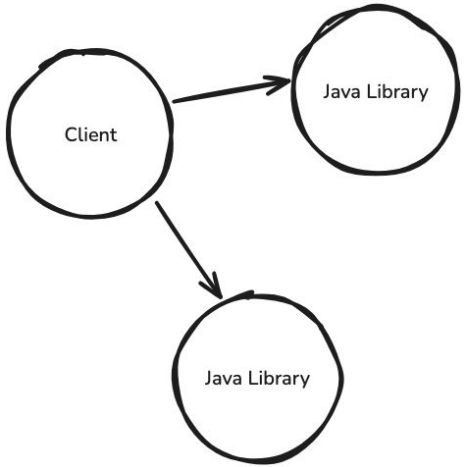
We fetch the current and latest versions of all the libraries on which the client depends

“system check” Image: Flatart, Noun Project
“Library list” Image: Vittorio Scaperrotta, Noun Project

Methodology

1. Get libraries used by clients (old + new)
2. **Find external library calls from client**
3. Find library methods with new exceptions
4. Filter new exceptions by taintedness
5. Manually write test cases for flagged library methods

2. Finding External Method Invocations



SootUp

We identify all external method invocations in the client:

- for every method call,
  - is the target's declaring class in the client, Java stdlib, or elsewhere (i.e. library)?

2b. Matching Client & Library Calls

We currently match fully-qualified method signatures between the client and the library.

(This misses some dynamic calls across the client/library boundary, causing potential underreporting.)

Methodology

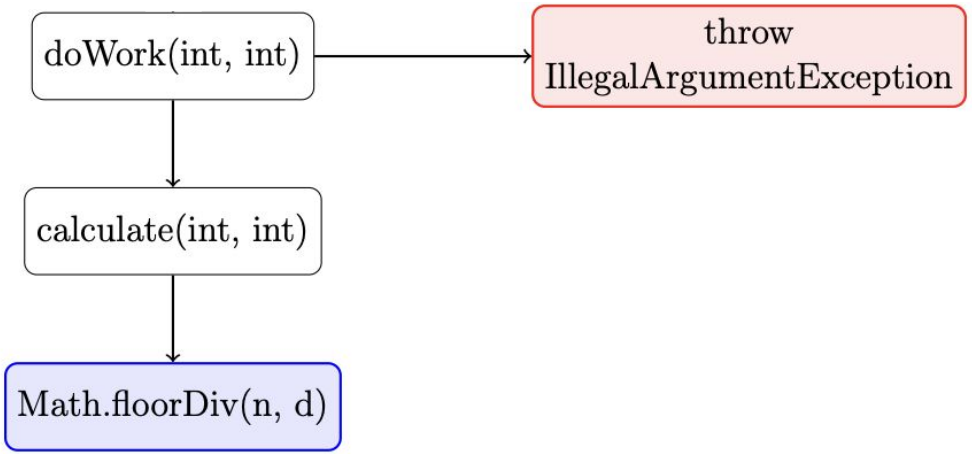
1. Get libraries used by clients (old + new)
2. Find external library calls from client
3. **Find library methods with new exceptions**
4. Filter new exceptions by taintedness
5. Manually write test cases for flagged library methods

Running Example

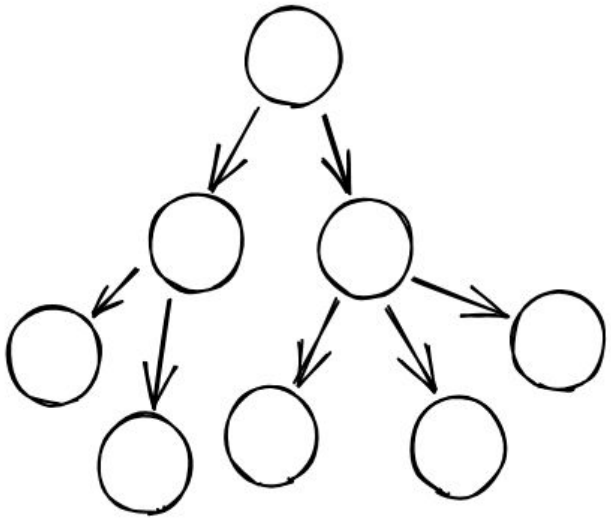
```
public class Calculator {                                                    // Library
    public static void doWork(int n, int d) {
        if (d == 0) {
            throw new IllegalArgumentException("denominator cannot be zero");
        }
        calculate(n, d);
    }

    public static void calculate(int n, int d) {
        int result = Math.floorDiv(n, d);
    }
}
```

```
public class CalculatorClient {                                              // Client
    public static void main(String[] args) {
        int numerator = 10;
        int denominator = 0;

        Calculator.doWork(numerator, denominator);
    }
}
```

Call Graph


doWork(int, int)
throw
IllegalArgumentException
calculate(int, int)
Math.floorDiv(n, d)

Analyzing Libraries



Class Hierarchy Analysis

3. Finding New Exceptions

For each library method transitively reachable (CHA) from the caller,
collect before & after sets of potentially thrown exceptions,

(i.e. `throw new XXXException/Error`)

and calculate the diff.

Methodology

1. Get libraries used by clients (old + new)
2. Find external library calls from client
3. Find library methods with new exceptions
4. **Filter new exceptions by taintedness**
5. Manually write test cases for flagged library methods

Can the client trigger the exceptions we have discovered?

“trigger” Image: WARHAMMER, Noun Project

Example of an Untriggerable Exception

```java
static CodedInputStream newInstance(
    final byte[] buf, final int off, final int len, final
       boolean bufferIsImmutable) {
  ArrayDecoder result = new ArrayDecoder(buf, off, len,
     bufferIsImmutable);
  try {
    result.pushLimit(len);
  } catch (InvalidProtocolBufferException ex) {
    // The only reason pushLimit() might throw an exception
    // here is if len is negative. Normally pushLimit()'s
    // parameter comes directly off the wire, so it's
    // important to catch exceptions in case of corrupt or
    // malicious data. However, in this case, we expect
    // that len is not a user-supplied value, so we can
    // assume that it being negative indicates a
    // programming error. Therefore, throwing an unchecked
    // exception is appropriate.
    throw new IllegalArgumentException(ex);
  }
  return result;
}
```

newInstance() from protobuf-java-4.30.1

4. Exception Filtering with Taint Analysis

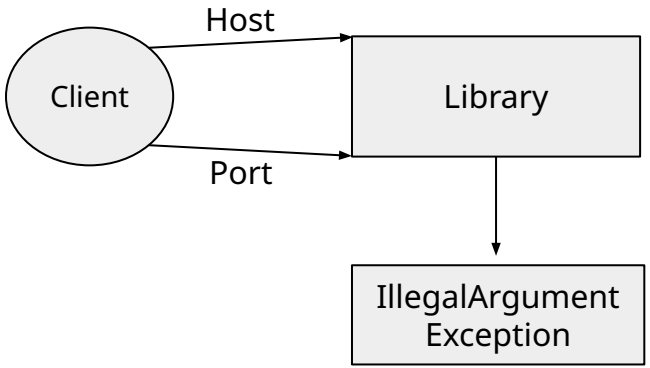
Use FlowDroid to do taint analysis:

- on methods potentially throwing unchecked exceptions
  (called transitively from client;
  identified using CHA callgraph in step 3).

4. Exception Filtering with Taint Analysis

Sources: All client-supplied values (library parameters)
(in earlier example, Host and Port)

Sinks: New exception constructors
(in example, IllegalArgumentException)



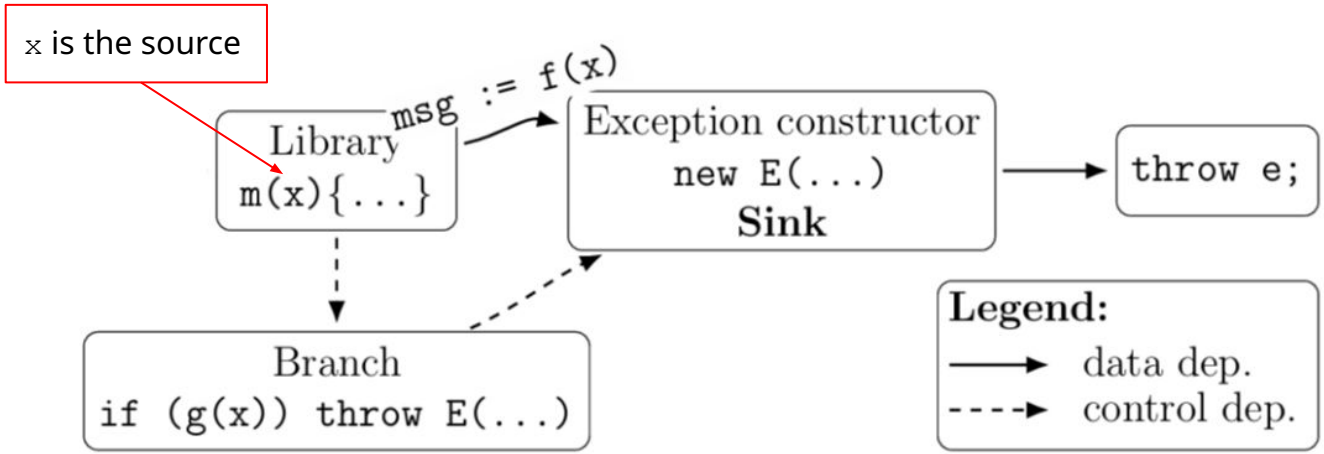
4. Exception Filtering with Taint Analysis

```
1  public class FlowDroidExampleCode {
2    public static int source() { return 1337; }
3
4    public void exampleTaintAnalysis() {
5      int temp = source(); int[] arr = new int[2];
6      arr[0] = temp; arr[1] = 19;
7      if (arr[0] == 1337) {
8        throw new RuntimeException("hello"); }
9      }
10 }
```

`throw new RuntimeException()` is tainted because client-controlled input (source()) affects the control flow that leads to the exception.

Taint analysis identifies explicit and implicit influences.

Exception Filtering with Taint Analysis



Taint analysis determines reachability using **explicit data dependences** and **implicit control-flow dependences** on the client-supplied value.

Methodology

1. Get libraries used by clients (old + new)
2. Find external library calls from client
3. Find library methods with new exceptions
4. Filter new exceptions by taintedness
5. **Manually write test cases for flagged library methods**

5. Verifying New Exceptions With Tests

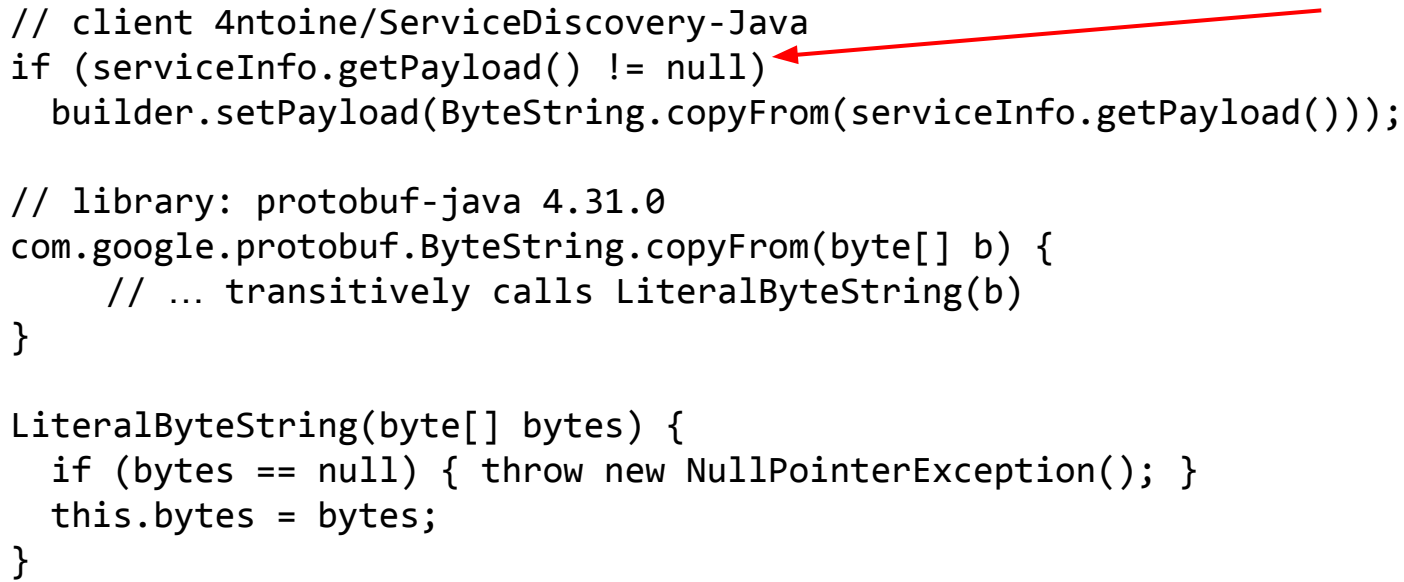
```
@Test
void testCreateAsyncClientThrowsExceptionForEmptyProxyHost() {
  HttpAsyncClient client = new HttpAsyncClient();

  IllegalArgumentException exception =
    assertThrows(IllegalArgumentException.class, () -> {
        client.createAsyncClient(true);
    });

    assertTrue(exception.getMessage()
    .contains("may not be empty"),
      "Expected exception due to empty hostname "+
      "after upgrading to httpcore-4.4.16");
}
```

A manually-written test case for HttpAsyncClientUtils (library httpcore).

5. Verifying New Exceptions With Tests

Sometimes we can't:

Client may explicitly check before calling the affected library method.

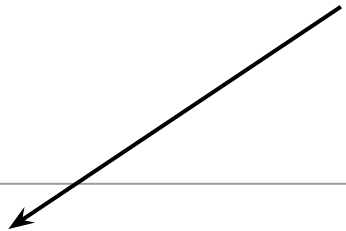


```
// client 4ntoine/ServiceDiscovery-Java
if (serviceInfo.getPayload() != null)
  builder.setPayload(ByteString.copyFrom(serviceInfo.getPayload()));

// library: protobuf-java 4.31.0
com.google.protobuf.ByteString.copyFrom(byte[] b) {
     // … transitively calls LiteralByteString(b)
}

LiteralByteString(byte[] bytes) {
  if (bytes == null) { throw new NullPointerException(); }
  this.bytes = bytes;
}
```

Some Additional Results

Exception Analysis Funnel

Client invocations of external methods	15,678
Invocations reaching exceptions passing taint analysis	1,708



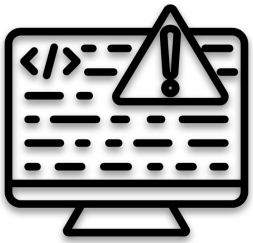
from 136/1011 clients

Top 15 client-library pairs

Client	Current Version	Latest Version	Number of Callsites	Reachable Callsites
codes.brewing.flinkexamples-1.0-SNAPSHOT	commons-logging-1.1.1	commons-logging-1.1.3	3479	436
api-2.0.2	gson-2.3	gson-2.13.1	453	209
cosyan-0.0.1-SNAPSHOT	json-20180130	json-20250517	365	112
TopicModelingTool	junit-4.11	junit-4.13.2	308	78
android-facebook-1.6	android-1.6_r2	android-4.1.1.4	154	77
indextank-engine-1.0.0	commons-cli-1.2	commons-cli-1.10.0	328	51
commons-pipeline-1.0-SNAPSHOT	commons-digester-1.7	commons-digester-2.1	76	48
codes.brewing.flinkexamples-1.0-SNAPSHOT	commons-codec-1.3	commons-codec-1.4	47	38
mrdpatterns-1.0-SNAPSHOT	hadoop-core-1.1.1	hadoop-core-1.2.1	466	36
rehttp	xembly-0.31.1	xembly-0.32.2	61	36
indextank-engine-1.0.0	log4j-1.2.16	log4j-1.2.17	33	33
Timeline-2.0.0	tablestore-4.11.2	tablestore-5.17.6	479	32
HospitalAction-1.0	poi-5.2.2	poi-5.4.1	42	28
MavenProject-0.0.1-SNAPSHOT	selenium-api-3.141.59	selenium-api-4.35.0	120	24
amazon-kinesis-aggregators-.9.2.9	commons-logging-1.1.1	commons-logging-1.3.5	105	23

Conclusions

Our Findings



Client applications invoke methods that contain newly added exceptions.

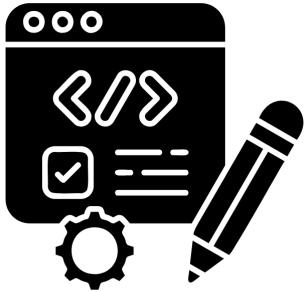
Test cases can be written to trigger these exceptions.

Most (89%) new unchecked exceptions occur in non-patch library upgrades

“change” Image: Sulistiana, Noun Project
“exception” Image: MUHAMMAT SUKIRMAN, Noun Project
“Test Case” Image: karyative, Noun Project

Our Findings

Client applications invoke methods that contain newly added exceptions.



Test cases can be written to trigger these exceptions.

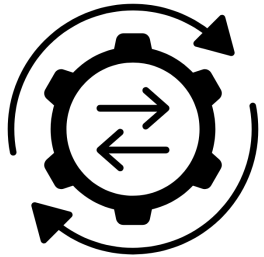
Most (89%) new unchecked exceptions occur in non-patch library upgrades

“change” Image: Sulistiana, Noun Project
“exception” Image: MUHAMMAT SUKIRMAN, Noun Project
“Test Case” Image: karyative, Noun Project

Our Findings

Client applications invoke methods that contain newly added exceptions.

Test cases can be written to trigger these exceptions.

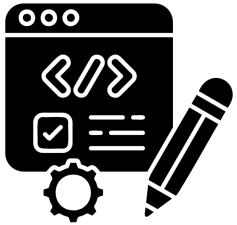


Most (89%) new unchecked exceptions occur in non-patch library upgrades

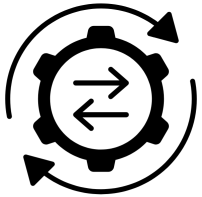
“change” Image: Sulistiana, Noun Project
“exception” Image: MUHAMMAT SUKIRMAN, Noun Project
“Test Case” Image: karyative, Noun Project

Our Findings

Client applications invoke methods that contain newly added exceptions.



Test cases can be written to trigger these exceptions.



Most (89%) new unchecked exceptions occur in non-patch library upgrades

Artifact & data:
https://doi.org/10.5281/zenodo.16788650

“change” Image: Sulistiana, Noun Project
“exception” Image: MUHAMMAT SUKIRMAN, Noun Project
“Test Case” Image: karyative, Noun Project

Syntactic Breaking Changes

```
public class Calculator {
  public int add(int a, int b)
  {
    return a + b;
  }
}
```

Before

```
public class Calculator {
  public double add(double a, double b)
  {
    return a + b;
  }
}
```

After

Changes to signatures of public methods

Syntactic Breaking Changes

```
public class StringUtils {
    public static String reverse(String input) {
        return new StringBuilder(input).reverse().toString();
    }
}
```



```
public class StringUtils {
    // Method reverse removed
}
```

Retractions of formerly-existing methods

Example for Syntactic Breaking Changes

```java
public class MyLibrary {
    public void greet() {
        System.out.println("Hello!");
    }
}
```

Library Code (old version)

```java
public class MyLibrary {
    public void greet(String name) {
        System.out.println("Hello!" + name + "!");
    }
}
```

Library Code (New version)

```java
public class ClientApp {
    public static void main(String[] args) {
        MyLibrary lib = new MyLibrary();
        lib.greet();
    }
}
```

Client Code

Detection of Syntactic Breaking Changes

Classes:

Status	Fully Qualified Name
MODIFIED	com.github.davidmoten.guavamini.Lists
NEW	com.github.davidmoten.guavamini.Maps
NEW	com.github.davidmoten.guavamini.Maps$Builder
NEW	com.github.davidmoten.guavamini.Maps$MapBuilder
MODIFIED (!)	com.github.davidmoten.guavamini.Preconditions
MODIFIED	com.github.davidmoten.guavamini.Sets

Tools like japicmp and Revapi can detect syntactic breaking changes