1 EXECUTING CODE

Although npm places almost restrictions on what code developers can upload as a package, JavaScript programmers have settled on a number of best practices that we can use to find bugs.

- We ran our system on all 7514 packages.
- Of those packages, there were only 1823 where we could:
  - Download the code from source.
  - Install the dependencies.
  - Run all of the tests automatically.
- Of those, only 837 passed when we swapped out their entry point with one involving contracts.
- Of those, 274 failed when we ran their code using contracts.

2 CRITICAL ERRORS

2.1 Type Mismatch

These are the most clear kind of mismatch: The TypeScript expects one kind of primitive type, like `string`, but the JavaScript passes in something completely different. Examples of packages that fall into this category include:

- branca

2.2 Arity Mismatch

JavaScript ordinarily lets you pass as many arguments as you would like into a function:

```
const myAdd = (a, b) => a + b;
myAdd(1, 2, 3); // 3;
```

TypeScript wants to make sure each function takes the arguments it expects. However, sometimes, TypeScript developers miss all of the arguments that a JavaScript function can take. We've detected arity mismatch problems in the following packages:

3 UNDESIRED ERRORS

3.1 Message Mismatch

JavaScript's error handling system relies on strings: An error object in the language contains a "message" string that can be inspected at runtime. As such, JavaScript tests often work by creating functions that throw an exception and then checking that the message in the exception contains the error. An example of such a test might look like the following:

```
it("should throw an error if the value passed is not an array:", function() {
  (function() {
      unique("a", "b", "c");
  }).should.throw("array-unique expects an array.");
});
```

These tests present problems because our contract library throws errors related to blame that often do not match the string the test expects. As such, while these tests fail, they do not necessarily present a type error of the sort we desire—TypeScript is designed to guard against these cases.

Packages that fall into this category include:

- array-inital
- array-unique

3.2 Runtime Type Changes

JavaScript features a form of prototypal inheritance that lets developers modify the methods on classes at runtime. For example, one could write:

```
Array.prototype.push = function() { return "You tried to push an element onto an array!" };
[1, 2, 3].push(3)
```

Some packages provide convenience wrappers around this sort of prototype modification that expect different types than the ones in the ‘index.d.ts‘ file. Packages that fall into this category include:

- abbrev

3.3 Proxy Identity

Proxies in JavaScript sometimes don’t behave the same way as the identical object. That can cause tests to fail. Packages that fall into this category include:

- async-busboy
- basicauth-middleware