Proposal Update: Function Implementation Hiding

Stage 2 • Michael Ficarra • 76th Meeting of TC39, June 2020

Recap: Considered Use Cases

no information leaks

1. the function *does not* appear in stack traces
2. the function's source text (including parameter list) is not available through `F.p.toString`

no unintentional API

1. the function *does* appear in stack traces
2. the function has no file attribution or position info in stack traces
3. the function's source text (including name and parameter list) is not available through `F.p.toString`

appear as a built-in (for polyfilling)

1. the function *does* appear in stack traces
2. the function has no file attribution or position info in stack traces
3. the function is rendered as a `NativeFunction` by `F.p.toString`

call site insensitivity & refactoring helper functions

1. the function *does not* appear in stack traces

The Proposal, as Previously Presented

1. "sensitive", a directive for hiding implementation details and runtime behaviour:

   1. the function *does not* appear in stack traces

   2. N/A: the function has no file attribution or position info in stack traces
   3. the function is rendered as a `NativeFunction` by `F.p.toString`

2. "hide source", a second directive for hiding source text info:

   1. the function has no file attribution or position info in stack traces
   2. the function is rendered as a `NativeFunction` by `F.p.toString`

Relation to Considered Use Cases

no information leaks

1. the function *does not* appear in stack traces
2. the function's source text (including parameter list) is not available through `F.p.toString`

no unintentional API

1. the function *does* appear in stack traces
2. the function has no file attribution or position info in stack traces
3. the function's source text (including name and parameter list) is not available through `F.p.toString`

appear as a built-in (for polyfilling)

1. the function *does* appear in stack traces
2. the function has no file attribution or position info in stack traces
3. the function is rendered as a `NativeFunction` by `F.p.toString`

call site insensitivity & refactoring helper functions

1. the function *does not* appear in stack traces

Committee Feedback: Spec Text

```
19670      <emu-clause id="sec-function-definitions-static-semantics-presentinstacktraces">
19671        <h1>Static Semantics: PresentInStackTraces</h1>
19672        <emu-see-also-para op="PresentInStackTraces"></emu-see-also-para>
19673        <emu-grammar>FunctionBody : FunctionStatementList</emu-grammar>
19674        <emu-alg>
19675          1. If the Directive Prologue of |FunctionBody| contains a Sensitive Directive, return *false*.
19676          1. If |FunctionBody| occurs within a |ScriptBody|, |ModuleBody|, or |FunctionBody| that has a Directive Prologue th
19677          1. If the source text matched by |FunctionBody| is eval code resulting from a direct eval, then
19678            1. Return PresentInStackTraces of the |CallExpression| whose evaluation is the direct eval.
19679          1. Return *true*.
19680        </emu-alg>
19681      </emu-clause>
19682
19683      <emu-clause id="sec-function-definitions-static-semantics-hassourcetextavailable">
19684        <h1>Static Semantics: HasSourceTextAvailable</h1>
19685        <emu-see-also-para op="HasSourceTextAvailable"></emu-see-also-para>
19686        <emu-grammar>FunctionBody : FunctionStatementList</emu-grammar>
19687        <emu-alg>
19688          1. If the Directive Prologue of |FunctionBody| contains a Sensitive Directive or a Hidden Implementation Directive,
19689          1. If |FunctionBody| occurs within a |ScriptBody|, |ModuleBody|, or |FunctionBody| that has a Directive Prologue th
19690          1. If the source text matched by |FunctionBody| is eval code resulting from a direct eval, then
19691            1. Return HasSourceTextAvailable of the |CallExpression| whose evaluation is the direct eval.
19692          1. Return *true*.
19693        </emu-alg>
19694      </emu-clause>
```

no longer uses Execution Context

Committee Feedback: Separability

- Could we separate the proposal? *Yeah, I think so.*
- Proposal purpose:
  - support use cases impacted by abstraction-breaking runtime reflection APIs.
- Is there value in separating the proposal? *Yes, probably.*
  - "hide source" fairly uncontroversial at this point
  - use cases covered by "sensitive" might be too disparate
    - library & dev tooling use cases often would prefer to toggle it
    - security-related use cases require it to be perpetual
- We should separate it into
  - "function source hiding" proposal, with "hide source" advancing to stage 3
  - "stack trace censorship" proposal, remaining at stage 2
    - will explore separate solutions for use cases covered by "sensitive" today

Ecosystem Interactions: JS Self-Profiling API

§ 1. Introduction

This section is non-normative.

Complex web applications currently have limited visibility into where JS execution time is spent on clients. Without the ability to efficiently collect stack samples, applications are forced to instrument their code with profiling hooks that are imprecise and can significantly slow down execution. By providing an API to manipulate a sampling profiler, applications can gather rich execution data for aggregation and analysis with minimal overhead.

§ 1.1 Examples

This section is non-normative.

EXAMPLE 1

```
const profiler = await performance.profile({ sampleInterval: 10 });
for (let i = 0; i < 1000000; i++) {
     doWork();
}
const trace = await profiler.stop();
sendTrace(trace);
```

https://github.com/WICG/js-self-profiling/issues/23

Open Questions: Non-simple Parameter Lists

michaelficarra Member

We disallow the `"use strict"` directive in functions with non-simple parameter lists because it complicates the parsing of the parameter list. While the directives introduced by this proposal do not affect parsing, they do affect "earlier" code, e.g. functions defined in the parameter list. Is that a significant enough inconvenience for implementers that we want to disallow the commingling of these two features?

/cc **@bakkot** who raised this question

michaelficarra added the **question** label

ljharb (Jordan Harband) Member

It seems reasonable to apply the same restriction for the same reasons.

michaelficarra Member Author

@ljharb It's not quite the same reasons though. An implementation wouldn't have to re-parse the parameter list, they would just need to track the functions that they parsed in the parameter list until they've passed the directive prologue. This is a significantly smaller implementation burden.

https://github.com/tc39/proposal-function-implementation-hiding/issues/45