Analysis Report for: preflight.js

Overall Functionality

This C code implements a custom JavaScript module loader for the Node.js environment. It specifically targets `.jsc` files, which appear to be Brotli-compressed JavaScript code. The code decompresses the `.jsc` file, wraps it in a function that injects global variables (`module`, `exports`, `__dirname`, `require`) into the global scope, then executes the decompressed code within the current context. This process bypasses the Node.js's standard JavaScript module loading mechanism.

Function Summaries

* **`b(r, e, o, i)`:** This function copies enumerable properties from object `e` to object `r`, excluding property `o`. It uses `Object.getOwnPropertyNames`, `Object.defineProperty`, and `Object.getOwnPropertyDescriptor` for property manipulation. The `i` parameter seems to be an internal variable for handling enumerability. It returns the modified `r` object.

* **`s(r, e, o)`:** This function creates a new object (or uses an existing one if provided) and uses `b` to copy properties from `r` to it. It handles the case where `r` might be a ES module (`r.__esModule`). It returns an object with a `default` property containing `r`. This is a common pattern for creating ES module wrappers.

* **`c = s(require("module"))`, `a = s(require("v8"))`, `m = s(require("vm"))`:** These lines import and wrap the Node.js `module`, `v8`, and `vm` modules, respectively, using the `s` function.

* **`c.default._extensions[".jsc"] = (r, e) => { ... }`:** This is the core of the custom module loader. It registers a custom extension handler for `.jsc` files with Node.js. When Node.js encounters a `.jsc` file, this function is called.

* **`S(r)`:** This function parses the compressed JavaScript code. It extracts the length of the code from the first 4 bytes and creates a `vm.Script` object from this code, utilizing cached data for performance improvements. It returns a `vm.Script` object.

Control Flow

* **`c.default._extensions[".jsc"]`:** This function first reads the `.jsc` file synchronously using `fs.readFileSync` and decompresses it using `zlib.brotliDecompressSync`. It then sets the `__filename` global variable. The crucial part is the call to `r._compile(f, `${e}-wrapper`)(()=>S(i).runInThisContext())`. This compiles a wrapper function (defined in `f`) which injects globals, then runs the decompressed code using `vm.Script.runInThisContext()`. Error handling is minimal; a `cachedDataRejected` check exists but no other error checks are apparent.

* **`S(r)`:** This function reads the length of the decompressed code from the buffer, creates a `vm.Script` object and returns it. A check for `o.cachedDataRejected` handles potential errors during script creation.

Data Structures

The primary data structures are:

* **JavaScript Objects:** The code heavily relies on JavaScript objects, especially for property manipulation and module wrapping.
* **Buffers (Uint8Array):** The compressed JavaScript code is handled as a `Uint8Array` (or a similar buffer type).
* **vm.Script:** The decompressed JavaScript code is compiled into a `vm.Script` object for execution in the `vm` module.

Malware Family Suggestion

Given the code's functionality, it strongly resembles techniques used in obfuscation and code injection. The use of Brotli compression for obfuscation, the custom module loader, and the injection of global variables are all red flags. The bypassing of standard Node.js module loading mechanisms allows for potentially malicious code execution. There is no inherent self-replication, but the code enables loading and running arbitrary JavaScript code from `.jsc` files. Therefore, this code, if used maliciously, could be part of a **downloader** or a component within a larger **trojan** or **remote access trojan (RAT)**. It allows an attacker to deliver and execute arbitrary JavaScript code without being easily detected via standard antivirus techniques that check for known malicious signatures. The lack of error handling and security measures increases the risk that it could be easily exploited.