9

JS-Reactor: 86% Size Reduction Progress

notes on JSR-Update, double encryption and the per-build shape changes

The demo input for JS-Reactor is `1.js` , 3,310 bytes of JavaScript. For a while the protected output for it was 2,490,159 bytes. Three kilobytes in, two and a half megabytes out. It ran fine, but a 2.5 MB script lags out most systems

Four fixes later the same input lands between 334 and 378 KB, median around 337 KB. The current artifact on disk is 367,095 bytes, about 86% off the old shipped build. Nothing was taken out of the protection. Every change is compression or output rewriting, the protections all still fire, and the full test chain re-ran clean after each one.

```
1.js  (input)              3,310 B
1.js  protected, before  2,490,159 B
1.js  protected, after     367,095 B   (band: 334 - 378 KB)
```

Same input, same protections, only the output got smaller.

The numbers

	before	after
protected output for 1.js (3,310 B)	`2,490,159 B`	`~334-378 KB`
reduction vs shipped artifact	`-`	`~86%`
String.fromCharCode calls in output	`13,676`	`19`
long float literals in output	`6,212 (~106 KB)`	`0`
encrypted payload surfaces (Uint16Array)	`733`	`733, intact`
size band, same input across rebuilds	`763 KB - 1.38 MB`	`334 - 378 KB`

Before column is the shipped build. After column is a band across rebuilds, not one measurement.

What was actually wrong

The pipeline ran string encryption after VM lowering. String encryption does what the name says, it walks the output and encrypts string literals per character, with position and key terms folded in. The VM also produces string literals, the hex pool payloads that hold the bytecode, which the VM decodes at runtime with its own keys.

Those payloads were already ciphertext. The string pass could not tell the difference, so it encrypted them again, per character, on top of the layer that was already there. One payload nibble ended up costing 27.7 bytes of source, in expressions shaped like this:

```
String.fromCharCode(enc^ka^kb^pos^t)
```

13,676 of these calls in one file. Around 2.16 MB, 87% of the output, spent re-encrypting data the VM decrypts itself anyway.

Second problem: size was random per build. The VM layout picks one of five pool, metadata and block-table shapes per build, that is part of the protection. Rebuild the same input and you get 937 KB, rebuild again and you get 1.38 MB. The observed band was 763 KB to 1.38 MB, and the shipped 2.49 MB build had drawn the worst shape of the five. Users had no way to know what they would get.

The full proof chain re-ran after the changes, workspace tests, both compatibility matrices, the release build through the real protected output path, and the protected demo under Node.

Live for all users

Both problems were boring once found. Encryption ran twice on the same data because nobody told the pass to stop, and output size was a side effect of a shape pick with no size budget. The fix list is short and none of it touches how hard the output is to reverse.

The smaller output is on the site now for all users, no new options, no config to change. Same input as before, 86% less file.