JavaScript 代码混淆器

金琪琦

March 5, 2017

简介

JavaScript

- JavaScript 最初被设计用来作为简单的前端脚本
- 随着前端技术的不断发展，JavaScript 承担了越来越多的功能。它天生安全性不足的缺点得到了充分暴露
- JavaScript 程序解释执行，整个源代码都暴露在用户面前的特性，决定了代码混淆几乎是现阶段保护 JavaScript 代码唯一有效的手段。

代码混淆

代码混淆（Obfuscated code）是指将程序代码转换成一种功能上等价，但是难于阅读和理解的形式。Android 的 apk 就默认使用了代码混淆，使得反编译 APK 变得比较困难。

争议

争议

观点一

前端代码公开，没有秘密，本身代码就没有保护的意义。

争议

观点一

前端代码公开，没有秘密，本身代码就没有保护的意义。

观点二

正因为前端程序是以源码的形式展示在用户面前，才需要通过代码混淆的方式使得代码难以阅读从而增强前端代码的安全性，进一步保护自己的系统。

我的观点

- 前端代码天生的不安全性决定了，应该尽可能将重要的业务代码后移动。
- 但一方面，总可能会有一些需要在前端处理，又有一定的敏感性业务；
- 另一方面，前端的一些代码往往是攻击者猜测后端漏洞的入口。

因此对于一些重要的前端代码进行适当的混淆，能够增加攻击者破译的难度。保护前端代码的同时维护整个系统的安全。

现状

已有工具

- 开源工具：UglifyJS
- 商业软件：Jscrambler

现状

目前代码混淆在前端使用得并不多。

原因

这并不意味着前端代码不需要保护，或者对前端的代码混淆就没有意义。

而是因为前端的大多数代码并不涉及需要高安全的功能，代码混淆必然导致性能损失，对于轻量级的应用性能比安全更重要。

应用场景

例子

现在越来越多网站的验证码信息不再仅仅通过一张图片，而是从前端采集用户的操作信息返回给后台判断这一系列操作是否属于人类行为。面对这样一个前端代码，一旦知道了它的采集策略就很容易伪造信息。因此对这样重要的前端代码进行混淆是很必要的。

淘宝登录代码

淘宝登录界面通过 uab.js 程序来采集用户信息，而这个程序就用来加载一个经过混淆的 JavaScript 程序。

我的代码混淆器

JsPatronum

命名来自《哈利波特》防御魔法，守护神咒：Expecto Patronum 。

借鉴雅虎 JavaScript 代码压缩工具：YUI Compresser。

开发环境

- 语言：Java
- 平台：跨平台

混淆方法-代码压缩

代码压缩

高级语言中很多的元素都是给人看的，机器执行并不会用到它。例如：空白符号、注释、有意义的变量名等。因此，我们删除这些供人阅读的信息，一方面可以减少代码的体积增加传输效率，另一方面可以给用户阅读带来很大的困难，从而保护代码。

缺点

仅仅通过代码压缩，很容易被一些代码美化工具所还原。

作用

因此在一个完整的代码混淆器中，代码压缩通常会作为最后一步用来锦上添花。

混淆方法-正则替换

正则替换

代码混淆是一种功能等效源代码之间的转换，可以看作是字符串的转换，因此正则替换是一种处理字符串简单易行的方式。

缺点

然而正则替换只是纯粹的字符串转换，它不能了解程序的逻辑，所能做的工作有限，出错的概率也较高。

作用

通常使用正则替换来实现一些简单的代码混淆。

混淆方法-AST 替换

抽象语法树（AST）

在源代码的解释和编译过程中，语法分析器创建出抽象语法树，它是源代码的抽象语法结构的树状表现形式，树上的每个节点都表示源代码中的一种结构。一颗抽象语法树展示一个程序的完整语法结构，并不会包含真实语法中出现的每个细节，

抽象语法树替换

抽象语法树代表了一个程序的完整语法结构，那么我们可以通过对语法树的调整构造一个功能等效但难以阅读的混淆程序。

混淆步骤

1. 通过某个 JavaScript 引擎解析 javaScript 程序生成 AST；
2. 遍历语法树，并根据适当的混淆规则对语法树进行调整；
3. 通过 JS 引擎将调整后的语法树转换为 JS 源代码，这个代码就是混淆后的代码。

混淆策略

1. 变量名替换
   - 全局变量替换为 window 的属性调用
   - 属性调用替换为取元素操作 []
   - 局部变量名随机化
2. 常量混淆
   - 提取所有的字符串，通过字符数组打散
   - 常量编码转换
3. 控制流替换
   - 将普通的循环语句展开
   - 将顺序执行的代码放置在精心设计的循环之中

自我防御机制

通过以上的混淆策略，对一段 JavaScript 代码进行适当的混淆、压缩就能起到很好的防御作用。但是前端代码作为源码展示在用户面前，攻击者拥有足够的耐心，对代码进行深入的调试，仍然有可能理清代码的逻辑。

因此引入了 js 代码的自我防御机制

策略

1. 禁止代码格式化和变量重命名
2. 禁止代码调试
3. 域名绑定

未来

前端代码天然是容易被破译的，所以尽可能把敏感的代码移动到后端是必须的，代码混淆只是在特定情形下的一种折衷。

WebAssembly

谷歌、苹果、微软和 Mozilla 的工程师正合力创建 WebAssembly（又名 wasm），这是能够运用在未来浏览器中承诺可带来 20 倍更快性能的字节码（bytecode）。WebAssembly 项目创造全新的字节码（一种机器可读的指令集，能够更快为浏览器加载高级语言），让桌面和移动端浏览器相比较网页或者应用的整体源代码变得更加高效。

WebAssembly 能够使得在浏览器上运行的不再是程序的源代码，会大大提高程序被破译的难度。