**Bytecode流处理相关论文解读**

**Author: MG193.7 程为民**

本档用于记录(不仅限于)V8引擎字节码处理相关论文的分析过程, 以及论文中方法论的落地实现. 分析论文的选取因素**取决于我对此论文使用技术以及方法论的感兴趣程度, 以及论文本身的市场价值和落地潜力**. 目前正在分析的论文:

1. Information Flow Control in WebKit's JavaScript Bytecode

<https://arxiv.org/pdf/1401.4339.pdf>

论文落地代码

<https://github.com/bichhawat/ifc4bc>

1. JTaint: Finding Privacy-Leakage in Chrome Extensions

<https://link.springer.com/chapter/10.1007/978-3-030-55304-3_29>

论文落地代码

<https://github.com/whucs303/JTaint>

**Information Flow Control in WebKit's JavaScript Bytecode**

Information-Flow-Control别名IFC. IFC作用于现代浏览器JS引擎的字节码层, 控制流分析相关, 可以分析显示以及隐式的分支. 根据其Introduction章节可知, 论文声称该工作可以检测恶意的外部库引用, 代码注入等一系列可以造成隐私泄露的安全问题.

**第三章. 技术背景**

论文在第三章简要阐述了信息流控制技术, 以及流的分类(显式流与隐含流), 相对于显式流, 对隐含流的追踪更加困难, 论文提出维护pc标签(pc与当前执行状态有关, 可理解为操作系统内的程序计数器)以追踪执行流, 当某个分支处理高机密变量H时, pc会被污染为高机密状态, 随后, pc的每次执行都会将其它分支以及访问到的变量污染为高机密状态, 以此追踪敏感操作的整个信息流走向, 并提取可能与高机密变量H有间接关联的隐含执行流. 这部分流跟踪思想贯穿整篇论文及其落地项目, 或许可以利用在后续的V8字节码调试器的开发中. 而关于机密状态, 这是属于信息流控制中的概念, 机密状态可以分为高和低, 低机密意味着可以信息可以被公开, 高机密则意味信息对外部不可访问(需要授权), 详细可查阅Wikipedia的[信息论条目](https://en.wikipedia.org/wiki/Information_flow_(information_theory)).

**第四章. 设计, 挑战, 见解与解决方案**

论文在第四章节首部提到, 其目的是通过小代价修改WebKit字节码compiler以实现预定工作, 具体实现在第六章内有详细说明, 该章节还提到论文的几乎所有工作都被限制在字节码解释器内. 该部分与我目前正在开发的V8字节码调试器来看, 有许多相似之处, 比如不得不妥协以小代价和侵入性来Hook解释器以及字节码跳板指令集等等, 所以目前来看, 我对论文如何实现该部分工作很感兴趣. 此处, 我提出一些关于该部分的待分析问题. // 20240422

* 论文为什么不得不修改编译器以及解释器, 通过对侵入性的妥协, 作者如何达到该工作的最终目的?
* 该部分的操作以及方法论是否能够为我所用, 如加速调试器的开发或者加速对某段程序的分析?

WebKit的字节码解释器是标准的堆栈机, 局部变量被存储在当前调用栈的寄存器内, 其中, 与JS功能相关的对象拥有特定的数据结构, 例如原型链, 变量环境, 作用域链, 函数对象等等, 论文在这一部分做的操作是为所有结构添加自定义的标签, 包括寄存器, 对象属性与作用域链指针等等. 该部分的目的似乎是方便隐式或显示的污点传播, 标签是一字大小(64bit), 每个比特均有其作用, 比如标识其所属域(比如google.com), 添加标签以位运算’或’进行.

论文提到在字节码层的工作会引出一系列与污点传播相关的有趣概念以及实现问题, 还有与威胁模型有关的有趣疑问, 所有的这些问题都会在论文的第四章说明. 其中的部分问题很通用, 且与JS本身无关, 比如论文结合动态分析与少量静态分析技术来处理非结构化的控制流与异常处理.

**处理非结构化控制流**

字节码层面的分支并非由if或while句柄控制的块结构化控制流, 相反, 由于字节码控制流的转移可以由单条字节码指令(例如op\_jfalse指令)产生, 故此, 论文定义除了函数调用操作, 字节码层面内的所有控制流都是非结构化的. 为了处理这种情况, 论文采用静态分析中的控制流图(CFG)技术. 并且利用到了图论中的即时后支配节点(IPD)概念, 论文的思想是在进入某个分支时将当前pc与其IPD推入栈中, 当执行到达IPD时再将pc弹出, 遇到eval等可能改变CFG的操作时, 分支对应的IPD也会随之变动, 论文声称该操作可以有效标记作用域的结束, 同时标志着安全上下文操作的结束. 但目前我仍不明白确定作用域或者安全上下文对于CFG分析有何作用.

根据GPT的回答, 非结构化的控制流会导致分支的不确定性, 使得确定作用域的工作更加复杂和难以进行. 确定作用域可以帮助识别某些局部变量的作用域, 以及其使用点, 从而提高程序分析的准确性, 确定作用域也可以优化控制流跟踪, 减少无效分支的分析, 简化异常处理, 减少分析的复杂性, 提高分析效率.

**处理异常与退出节点**

暂时不关心该处理 // 20240423

论文将WebKit的字节码分割为多种操作类型进行处理, 比如与原型链或对象相关的指令, 函数相关指令, 作用域相关指令, 异常相关指令等等, 这种分割操作值得学习, 在处理CFG时, 根据指令相关的操作类型可以做出很多灵活操作.

**论文落地代码IFC4BC**

根据论文内所提及的方法论以及理论上的代码实现可知, 作者对原生WebKit的编译器, 解释器以及其它模块进行了修改以达成论文预定的工作, 故其落地项目的代码结构本身与WebKit一致. 作者将该工作及其项目命名为”**IFC4BC**”, 为了定位IFC4BC对WebKit的修改以便后续分析, 需要对其所使用的WebKit分支以及Commit进行定位. 随后通过对比多个标志性文件, 可以确定该工作利用了safari-537.43-branch分支, 且可以定位到具体的Commit, 其Hash如下所示:

**309951b67906953f6a2d198e847f16e31df854b8**

分支对比使用了如下方案, 仅对比版本信息文件, xconfig文件, GNUmakefile列表等包含元数据的内容, 从而缩短对比时间进行高效查找. 最终利用WinMerge对比IFC4BC与此分支下的源码, 利用脚本消除部分Diff噪声后, 对比结果如下图所示, 明显可见工作修改过的模块. // 20240424



**第五章. 通用模型与IFC**

论文在第五章提到, 工作所使用的WebKit拥有147个字节码, 工作修改了其中69个字节码句柄, 剩下的字节码由于是”可序列化”或者是其它字节码的封装形式, 所以工作没有对其进行修改. 对于修改后的字节码, 论文则按照其操作类型进行分类. 论文还定义, WebKit中的概念”代码块”则是由函数内各代码指令所对应的字节码串组成, 当函数被创建或eval表达式被执行时, 解释器将生成并给出其对应的代码块, 随后, 工作将对代码块内的字节码串进行控制流分析, 并为此块构建CFG. 通常情况下, CFG是一个有向图, 且其节点就是块中的部分字节码, 其有向边即为可能的控制流, CFG将算出并记录图中每个节点的IPD. 而论文使用的IPD算法引用了数学家Lengauer与Tarjan在1979年给出的论文: ” A fast algorithm for finding dominators in a flowgraph”, 若CFG包含未捕获异常, 其出口节点(SEN)也会在分析时被给出.

CFG自然是一个不必再阐述的通用概念, 但我对于此处的疑问是, 论文的工作IFC4BC是如何抓取字节码串并给出CFG的? 由于前文的Diff与噪声消除工作已经完成, 故此, 可以直接从代码差异中找出此处论文工作的具体实现. 以下是分析日志: // 20240425

JavaScriptCore模块中的Interpreter#privateExecute函数会解释并执行当前调用帧所携带代码块内的所有字节码, 此时, 工作可以访问到帧内的代码块对象. 由于工作修改了CodeBlock.h, 并在其对象结构内部添加了StaticAnalyzer组件, 这使得工作可以灵活利用抓取到的代码块执行其它操作, 例如通过工作在组件内定义的StaticAnalyzer#genContextTable接口生成代码块的CFG.

// 20240425

WebKit在解释器内执行字节码时, 工作会干预解释过程, 并在特定的字节码句柄下操作工作自定义的栈结构pcstack, 同时为字节码正在操作的对象设定前文分析所提到的标签(可能用以进行污点传播), 或者从操作对象内取出标签并推入pcstack内, 不过目前暂不明确pcstack的具体作用.

// 20240426

// 当前优先级更高的任务应当是完成V8字节码调试器的开发, 先于开发来落地论文中的实现意义不大. 由于需求尚不明确, 而在论文中寻找需求的行为不符合实际, 因为需求只有在调试器开发完成后的实际应用中才能显现, 故此, 暂时停止对IFC4BC的分析 20240501