文章编号：2095-2163(2020)02-0320-05 中图分类号：TP311.5 文献标志码：A

基于 Chrome 调试协议的 Chromium 浏览器 V8 引擎漏洞检测方法

秦梦远，傅忠传
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：为解决对 Chromium 浏览器中内嵌的 V8 引擎的漏洞检测困难问题，突破传统日志输出模式的种种不足，对 Chrome 调试协议进行研究，并基于 Chrome 调试协议安插自定义追踪 API，用于代替传统的日志输出，以解决日志输出方式存在的诸多局限性。在 Chromium 浏览器内安插特定的事件 API，添加 Hook 函数，做到了实时性与准确性，使输出信息的质量大幅提升。实验结果表明，使用基于 Chrome 调试协议的 V8 引擎漏洞检测方法，可有效辅助定位 V8 引擎中的漏洞位点，提高效率。

关键词：Chrome 调试协议；V8 引擎；漏洞检测

Chrome-debugging-protocol-based vulnerability detection method for V8 JavaScript engine embedded in Chromium Web browser

QIN Mengyuan, FU Zhongchuan
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】 In order to solve the problems of detecting vulnerabilities hidden inside V8 JavaScript engine, which is embedded in Chromium Web browser, and to overcome the shortage of traditional log method, the paper analyzes the Chrome debugging protocol, and proposes a new method based on the protocol. The paper modifies the implementation of Chrome debugging protocol inside target Chromium Web browser and uses them to overcome kinds of limitations existed in the traditional log method. The paper uses certain modified event APIs and inserts Hook functions to get information, therefore guarantees instantaneity and accuracy, and makes a great improvement in received information. The experiment result shows that the Chrome-debugging-protocol-based vulnerability detection method can effectively help locating vulnerability position inside V8 JavaScript engine, and improve the efficiency.

【Key words】 Chrome debugging protocol; V8 JavaScript engine; vulnerability detection

0 引 言

从上个世纪 90 年代至今，随着互联网技术的二次飞跃，商业机构参与建设互联网，网络逐渐成为人们日常生活中不可或缺的一部分。因此，大量以浏览器为入口进行穿透攻击，盗取用户个人信息，甚至破坏用户操作系统的恶意代码纷纷涌现。这种攻击方式不同于传统的端口攻击，由于攻击载荷是无害的 HTML 文本，需要通过浏览器的解析与渲染才能有效触发，本机防火墙无法对其进行有效防御。

Chrome 浏览器作为目前最流行的浏览器，其商业版本 Chrome 与对应的开源版本 Chromium，以及国内外各种基于 Chromium 浏览器内核开发的浏览器，占据了超过半数的浏览器市场份额。对于 Chromium 浏览器内核的漏洞，谷歌 Chrome 开发团队能够第一时间对其进行修复，并提供补丁，但对于那些基于 Chromium 浏览器内核的浏览器开发者而言，修复漏洞并非易事。由于对 Chromium 浏览器内核的核心功能缺乏足够透彻的了解，开发者往往不能自主定位漏洞产生的位点，更不必谈修复漏洞；而对 Chromium 浏览器内核的自主修改，往往又导致了无法及时移植最新版本的 Chromium 浏览器内核，从而使自行研发的浏览器暴露在漏洞威胁之中。

针对上述问题，本文首先对 Chromium 浏览器内核的工作原理进行了分析；对 Chrome 调试协议的原理和实现进行详细阐述；针对 Chrome 调试协议提出一种修改方案，使 Chromium 浏览器能够对外传输内部工作状态信息，并以此为基础构建漏洞扫描平台，在浏览器崩溃时回溯浏览器工作状态，定位漏洞产生位点，为修复漏洞提供有效建议。

1 Chromium 浏览器工作原理

1.1 渲染引擎结构与工作原理

Chromium 浏览器使用 Blink 引擎作为其渲染引擎。

Blink 引擎包含的主要模块，以及其在渲染网页

作者简介：秦梦远(1994-)，男，硕士研究生，主要研究方向：网络安全；傅忠传(1971-)，男，博士，副教授，硕士生导师，主要研究方向：网络安全。
通讯作者：傅忠传 Email：15124585561@163.com

收稿日期：2019-06-06

时的作用如图 1 所示。由图 1 可知，圆角矩形对应的结点表示一个 Blink 模块，矩形对应的结点表示原始文本，菱形对应的结点表示 Blink 内部数据结构。Blink 内，HTML 解析器负责对接收到的原始 HTML 文本进行解析，将原始 HTML 文本反序列化为内部 HTML DOM 结点数据结构，同时配合 DOM 树生成器，依据原始 HTML 文本的从属关系，连接这些 DOM 结点组成 DOM 树。CSS 解析器负责将原始 HTML 文本内包含的 CSS 样式表解析为 CSS 规则树。在 DOM 树生成完毕后，将 DOM 树与 CSS 规则树合并，使用渲染排版模块生成渲染树交给 Content 中的渲染模块实现网页的渲染。

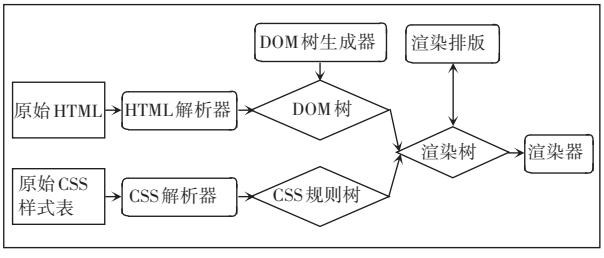

图 1　Blink 渲染引擎主要模块及工作流程

Fig. 1　Main modules of Blink render engine and the work flow

1.2　脚本引擎结构与工作原理

JavaScript 脚本语言因为其特有的面向对象特性[1]、事件驱动特性、安全性，得到广泛的应用。业内为统一 JavaScript 标准，提出了 ECMA－262[2]。Chromium 浏览器使用 V8 脚本引擎作为其 JavaScript 解析器。根据 W3C HTML 标准[3]，当网页渲染过程需要 JavaScript 脚本参与时（如处理<script>标签，或是处理 DOM 事件等），V8 脚本引擎才会启动。V8 引擎以其特有的即时（Just In Time，JIT）编译模式，从一众 JavaScript 引擎中脱颖而出，为 JavaScript 脚本语言带来了前所未有的性能提升。

V8 引擎本身分为以下几个部分：

（1）自行维护的内存池和配套的垃圾回收组件。内存池使用位索引（BitMap）来标记对象的生存状态，配合垃圾回收组件自动管理 JavaScript 对象的生命周期。

（2）虚拟机主体和内置执行引擎。虚拟机主体使用隔离机制（Isolate），不同的 Isolate 对象各自持有对应的内存池和全局对象（内置执行引擎），互相之间无法调用，由此构建了沙盒环境。

（3）JavaScript 编译器。包括抽象语法树生成器、字节码生成器和优化编译器。截至 2018 年 9 月，JavaScript 编译器采取编译到字节码的策略，初次执行时会使用字节码解释器运行代码，只有在同一段代码被执行了足够多次后，才会触发优化编译器，内联一部分函数调用并生成本地机器码。

（4）JavaScript 堆对象和配套方法。V8 为 C++ 的强类型与 JavaScript 的弱类型实现了一套堆上数据结构，用于接驳 JavaScript 数据存储。

V8 脚本引擎工作流程如下：

（1）初始化。对 V8 引擎的初始化几乎等同于对其内部对象 v8::internal::Isolate 的初始化。Isolate 对象初始化包括其内部的堆（v8::internal::Heap）的初始化，全局变量的初始化和上下文（Context）环境的初始化等。

（2）编译 JavaScript。首先需要构造 v8::String 对象。该对象可以使用 V8 引擎内部的字符串实现，也可自行实现其 v8::ExternalString 接口，接驳外部字符串实现。该对象包含的字符串为需要运行的 JavaScript 源代码。继而，调用 v8::Script::Compile 方法，将其转化为 V8 引擎的内部 Script 对象。

（3）运行。代码的运行过程会比编译简单些。V8 公共 API V8::Script::Run 会调用内部方法 v8::internal::Execution::Call 来完成对代码的动态调用。该方法内部会调用 Invoke 方法，提取出该段代码的入口函数和参数列表，以调用函数指针的方法唤醒该入口函数，并等待其返回值。V8 引擎在得到结果后，首先调用其成员方法 IsException 判断其是否为异常。如果为异常，则通知宿主 Isolate 对象异常详细信息，同时返回空结果；反之，则正常返回运行结果。

（4）垃圾回收。在 V8 引擎的内存占用达到一定值时，触发垃圾回收机制，V8 引擎会清理不再活跃的对象，以腾出内存空间。

2　Chromium 浏览器的安全设计

Chromium 浏览器内，V8 引擎为 Blink 引擎的子模块，需要接收来自 Blink 引擎的信息作为运行参数。这里将详细阐述传统的输出日志到文件的方法在该模式下的局限性。

2.1　V8 与 Blink 交互过程

在设计上，脚本引擎从属于渲染引擎，故 Blink 可直接调用 V8 引擎。Chromium 浏览器在设计之初就充分考虑了安全性，每个 Frame 标签（即<frame>、<iframe>，注意到主网页 Page 视为一个单独的 Frame）单独享有一个 V8 虚拟机，每个运行于特定 Frame 上的浏览器扩展（Extension）也单独享有一个 V8 虚拟机。一个标签页总计有 M 个 Frame 和 N 个扩展时（主体等同于一个扩展），其所包含的 Isolate

虚拟机共有 $M \times N$ 个。研究得到的 Blink 渲染引擎中页面、扩展和 V8 虚拟机的数量关系如图 2 所示。图 2 很好地解释了 Blink 内核的网页(Page)对象与 V8 虚拟机的对应关系[4]。其中的每一条线均代表了一个 Isolate 虚拟机。

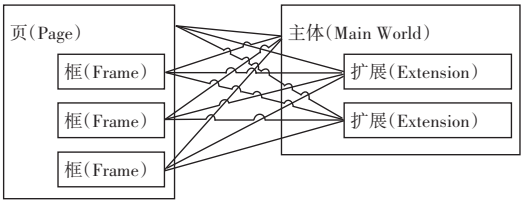

图 2 Blink 渲染引擎中页面、扩展和 V8 虚拟机的数量关系

Fig. 2 The relationship among pages, frames, extensions and v8 virtual machines

2.2 日志输出定位漏洞的局限性

由于 Chromium 浏览器执行严格的沙盒(Sandbox)策略,常规模式下,任何沙盒内的线程对本地文件目录、标准输入输出流的读写操作均被严格禁止,因此无论是使用日志手段输出到标准错误流,或是输出到本地文件,均不可行。

Chromium 浏览器提供内置启动参数--no-sandbox,该参数允许沙盒内进程突破沙盒限制,输出日志到文件或标准输出流/错误流。但 Chromium 浏览器出于性能与安全性的考虑,为多进程架构。图 2 中每个 Page 页面或是 Frame 页面,在操作系统内存充足的理想条件下,均为独立的进程,因此输出到标准错误流将存在争用,大幅影响性能,且存在来自 Chromium 其它模块日志输出的干扰;输出到同一个文件将存在读写锁,无法将日志写入同一个文件内。

Chromium 浏览器同时提供单进程运行参数--single-process,但此法会改变 Chromium 浏览器的运行行为,一些问题并不会在此模式下表现出来,一些功能将不可用,且工作线程仍然会从新进程中派生[5]。

由于 V8 引擎内部的复杂性,自行设计独立的日志系统往往容易产生程序异常行为,或是内部错误,因此,启用日志输出定位 Chromium 浏览器内部漏洞,往往事倍功半。

3 对 V8 引擎漏洞检测的方法

3.1 Chrome 调试协议简介

Chrome 调试协议(Chrome DevTools Protocol)是 Chrome 浏览器暴露给外部进行远程控制的通讯协议。Chrome 调试协议允许第三方工具测量、查看、调试和配置 Chromium、Chrome 以及其它基于 Blink 内核的浏览器。Chrome 的开发者工具就是利用了该协议来获取关键信息并呈现给开发者[6]。

Chrome 调试协议建立在 WebSocket 通讯协议之上,使用 JSON 格式文本传输内容。WebSocket 是一种基于 TCP 协议的,提供全双工通讯通道的计算机通讯协议。所有 Chrome 开发者工具(包括 Chrome 和 Chromium 浏览器自带的开发者工具 F12)均通过 WebSocket 协议远程连接到 Chromium 浏览器上,并获取相应的信息以供可视化呈现。

3.2 自定义检测指令

为了收集更多的有效信息,需要更改 Chromium 项目的源代码,增加额外的 API,以供自动化功能的实现。

考虑到代码整体功能的一致性,选择在调试协议的 Debugger 域添加自定义 API。

由于需要从 Chromium 浏览器端接收巨量数据,因此选择将新的 API 定义为事件。共计自定义 2 个 API 事件。对此可做阐释分述如下。

(1) beforeBytecodeEvaluated:该事件在每次 V8 内置的解释器解释一条字节码指令之前触发。

(2) AfterBytecodeEvaluated:该事件在每次 V8 内置的解释器解释一条字节码指令之后触发。

在自定义此两个 API 事件后,V8 虚拟机每次执行一条字节码,均会向漏洞扫描平台发送消息。当消息携带正在执行的字节码时,顺序接收的全部消息即可组成当前网页内 JavaScript 脚本的字节码执行流程。

与之相对应地,对字节码执行时的上下文环境获取,需要额外的代码支持。经过反复分析,选择添加钩子函数(Hook Functions)作为对前端 API 事件的支持函数。在钩子函数内,可以获取当前即将被执行的字节码、其对应的操作数、当前所在的上下文环境、所在的源代码和对应的字节码列表,以及当前字节码所对应的源代码偏移量信息。

根据 Chromium 项目源代码,JavaScript 协议定义于 js_protocol.json 中,其生成工具和配套模板定义于 inspector_protocol 目录中,生成的代码位于 gen/v8/src/inspector/protocol 目录中。修改协议的定义 JSON 文件,依照语法添加前端 API,即可获得对应的前端生成代码。在 Debugger 域添加的 API 最终会被作为纯虚方法生成到 gen/v8/inspector/protocol/Debugger.h 和同目录下的 Debugger.cpp 中。

在 v8-debugger-agent-impl.h 中定义触发方法 didBeforeBytecodeEvaluated,获取事件 API 所需的参

数，并主动调用 Debugger.cpp 生成代码中的事件 API。

V8 虚拟机执行 JavaScript 脚本，首先会将其编译成字节码，然后使用 Ignition 解释器进行逐字节码的运行。生成解释器的代码位于 interpreter-generator.cc 中。在需要监测的字节码解释器代码首部添加 Hook 函数，并向其传递字节码信息，处理传递来的字节码（bytecode）和对应的操作数（operands），并规范化存储。

在 debug.h 中定义 Debug::OnBeforeBytecodeEvaluated 方法，用于唤醒 beforeBytecodeEvaluated 事件。

Debug 类通过 DebugDelegate 类与 v8-debugger-agent-impl 交互，因此需要在 DebugDelegate 类内添加虚方法 HandleBeforeBytecodeEvaluated，并于 V8Debugger 类内将其具体实现。

修改后，完整的事件唤醒链的描述详见如下。

v8::internal::Hook_BytecodeTrace:

v8::internal::Debug::OnBeforeBytecodeEvaluated:

v8_inspector::V8Debugger::HandleBeforeBytecodeEvaluated:

v8_inspector::V8DebuggerAgentImpl::didBeforeBytecodeEvaluated:

v8_inspector::protocol::Debugger::Frontend::beforeBytecodeEvaluated:

v8_inspector::protocol::FrontendChannel::sendProtocolNotification:

4　实验结果与分析

为验证基于 Chrome 调试协议的 V8 引擎漏洞检测方法的有效性，选用测试平台为 64 位 Windows 平台，Chromium 版本 67，V8 引擎版本 6.8.255，对已知 V8 漏洞 CVE-2018-6149 进行检测。CVE-2018-6149 为 V8 引擎上的内存越界访问漏洞，黑客可通过精心布置的 HTML 文本触发该漏洞，导致 V8 引擎拒绝服务，或是执行黑客自定义的代码。CVE-2018-6149 的触发代码见如下。

```
var str2 = String.fromCharCode(0x2c);
var o2 = new Array(0x20000000);
String.prototype.split.call(o2,'');
```

测试平台上的 Chromium 浏览器，在访问测试网页后，于崩溃之前，传递回的最后一条自定义事件，其字节码为 Return，下一条即将执行的字节码为 Star r0。由于下一条字节码并未受到其执行后的事件消息，因此漏洞在该字节码处被触发，对 r0 寄存器的写操作发生了内存越界访问。

根据此处堆栈的上一条字节码 CallProperty2，寄存器 r4 的值为“split”，因而推断于该方法内部产生漏洞。阅读 V8 引擎源代码，split 方法定义于 builtins-string-gen.cc 中，为 StringPrototypeSplit，由 V8 团队设计的平台无关语言 CodeStubAssembler 书写[7]。在给定测试源代码的环境下，CVE-2018-6149 触发代码引发的执行分支的流程设计内容具体如下。

```
// If the separator string is empty then return the elements in the subject.
{
    Label next(this);
    GotoIfNot(SmiEqual(LoadStringLengthAsSmi(separator_string), smi_zero),
              &next);
    TNode<Smi> subject_length = LoadStringLengthAsSmi(subject_string);
    GotoIf(SmiEqual(subject_length, smi_zero), &return_empty_array);
    args.PopAndReturn(
        StringToArray(context, subject_string, subject_length, limit_number));
    BIND(&next);
}
```

该分支下，最终会执行到 AllocateFixedArray 方法处，该方法的部分生成代码信息详见如下。

```
Builtins_StringPrototypeSplit:
StringBuiltinsAssembler::StringToArray:
CodeStubAssembler::AllocateFixedArray:
{
…
TNode<IntPtrT> total_size =
  GetFixedArrayAllocationSize(capacity, kind, mode);
if (IsDoubleElementsKind(kind)) flags |= kDoubleAlignment;
// Allocate both array and elements object, and initialize the JSArray.
Node* array = Allocate(total_size, flags);
…
}
```

注意到 total_size 变量由 GetFixedArrayAllocationSize 方法进行计算,该方法会计算传入数组的长度信息,并返回其占用空间大小(以字节计)。此时,传入的数组为 o2,长度为 0x20000000,由于每个元素槽位均存储一个指针变量,在 64 位体系结构下其大小为 8 字节,因此对应总大小为:0x20000000 * 8 = 0x100000000。注意到 V8 的数组结构存在头信息,用于描述数组结构,该头信息占据 16 字节空间,因此最终 total_size 变量的值为 0x1000000010。

接下来进入 Allocate 方法,该方法会穿过许多调用堆栈,最终进入 AllocateRaw 方法内。AllocateRaw 方法的部分生成代码的表述形式如下。

```
Node * runtime_flags = SmiConstant(
    Smi::FromInt(AllocateDoubleAlignFlag::encode(needs_double_alignment) |
    AllocateTargetSpace::encode(AllocationSpace::LO_SPACE)));
Node * const runtime_result =
    CallRuntime(Runtime::kAllocateInTargetSpace, NoContextConstant(),
    SmiTag(size_in_bytes), runtime_flags);
```

注意到 SmiTag 方法,该方法将一个 64 位整数转化为 V8 内部的小整数(Small Integer,Smi)形式。对于小整数,V8 引擎使用 64 位有符号整数进行存储,仅使用其去除符号位的高 31 位存储数据,因此 SmiTag(val)等效于 val << 32。对于传入的变量 total_size,才会调用堆栈下对应 size_in_bytes 变量,其值为 0x100000010,超过 32 位无符号整数的表示极限,因此对其进行 SmiTag 操作,其实际表示的值将变成 0x10,远远小于期望值。

由于 Runtime_AllocateInTargetSpace 方法返回长度为 0x10 的数组,但程序并未发觉,仍尝试将 0x20000000 个数据依次写入该数组,于是写越界发生。

根据自定义 API 返回的结果,成功定位距漏洞位点最近的字节码,再以此为入口点,深入追踪 V8 引擎的源代码,最终成功定位漏洞位点,并发现漏洞产生原因。综上所述,基于 Chrome 调试协议的 V8 引擎漏洞检测方法,可以快速有效检出产生漏洞的代码区域,同时辅助精确定位漏洞代码,成功突破了传统日志输出的局限性,达到了实时性与有效性。

5 结束语

本文对使用 Chrome 调试协议对 V8 引擎进行漏洞检测的方法进行了介绍,研究了 Chromium 浏览器的工作原理与 Chrome 调试协议的实现细节。针对使用日志输出进行漏洞检测的局限性,提出使用 Chrome 调试协议进行漏洞检测的方法,并进行了验证。实验结果表明,该方法能够辅助定位 V8 引擎中的漏洞触发位点,并具有良好的操作性。

参考文献

[1] 张军林, 阳富民,胡贯容. JavaScript 语言解释器的设计与实现[J]. 计算机工程与应用, 2003(30):124.

[2] Ecma.Standard ECMA-262,2019 [EB/OL].[2019-06]. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[3] HTML Standard [EB/OL]. [2020-02-12].https://html.spec.whatwg.org/multipage/canvas.html.

[4] Haraken@. How Blink Works - Google Docs. [EB/OL].[2019-08-14]. https://docs.google.com/document/d/1aitSOucL0VHZa9Z2vbRJSyAIsAz24kX8LFByQ5xQnUg/edit?pli=1#.

[5] Debugging Chromium on Windows - The Chromium Projects [EB/OL]. [2019-05-01]. https://www.chromium.org/developers/how-tos/debugging-on-windows.

[6] Chrome DevTools Protocol Viewer [EB/OL].[2019-09-24]. https://chromedevtools.github.io/devtools-protocol/.

[7] CodeStubAssembler builtins. [EB/OL].[2019-05-01]. https://v8.dev/docs/csa-builtins.

(上接第 319 页)

"互联网+"模式来汇集多方的力量不断完善和弥补"医养文"养老模式在发展过程中的不足。营造一个良好的养老生态,促进中国养老服务事业的持续健康发展。

参考文献

[1] 李长远. "互联网+"在社区居家养老服务中应用的问题及对策[J]. 北京邮电大学学报(社会科学版), 2016,18(5):67.

[2] 本刊编辑部. 国务院印发关于积极推进"互联网+"行动的指导意见[J]. 中国能源,2015,37(9):1.

[3] 马龙飞, 乐嘉宜, 郁敏杰,等. 建立"互联网+医养结合"新模式的构想及探讨[J]. 上海医药, 2017,38(6):3.

[4] 张雷. 国务院关于加快发展养老服务业的若干意见[国发(2013) 35 号][J]. 标准生活,2015(3):44.

[5] 刘文俊, 孙晓伟, 张亮. 构建全民健康覆盖视角下"医养结合"养老服务模式的必要性[J]. 中国卫生经济, 2016,35(1):35.

[6] 徐志杰, 徐青松. "互联网+医疗"应用于"医养结合"的服务战略与保障路径—基于 SWOT-PEST 的模型分析[J]. 上海城市管理, 2016,25(5):28.

[7] 郭丽君, 鲍勇, 黄春玉,等. 中国养老服务医养结合模式的制度设计和政策建议[J]. 中国老年学杂志, 2018 ,38(11):2794.

[8] 徐广浩, 李传实, 崔瑞兰. 完善我国医养结合养老机构发展的思考[J]. 中国医药导报, 2018,15(18):150.

[9] 刘浩, 邹玲. 基于"互联网+"的智慧型医养新模式探讨[J]. 中国医院管理, 2018,38(5):56.