第 5 章　Chromium 软件指纹定制

Chromium 软件指纹定制涉及对浏览器软件指纹的获取、分析及修改。软件指纹包括但不限于 WebRTC、浏览器版本、操作系统、时区、语言设置及字体信息等。通过定制这些指纹，开发者可以增强用户隐私保护、进行安全测试或调试特定功能。本章将深入探讨 Chromium 软件指纹的获取方法及其定制技术，帮助开发者更好地理解软件相关指纹信息，以提升浏览器的安全性。

5.1　WebRTC 指纹

5.1.1　WebRTC 概述

WebRTC(Web Real-Time Communication)是一项支持网页浏览器进行实时音视频和数据传输的技术。它是由 W3C 和 IETF 组织开发的开放标准，旨在实现高质量的通信体验，且无须安装任何插件或专用软件。WebRTC 使开发者能够在网页和应用程序中实现点对点通信，适用于视频聊天、文件共享、在线会议等场景。

WebRTC 的功能主要通过以下几个核心组件实现。

(1) getUserMedia：用于获取用户的音视频设备，如摄像头和麦克风。

(2) RTCPeerConnection：用于在两个设备之间建立连接并传输音视频数据。

(3) RTCDataChannel：用于传输任意数据，如文件、文本消息等。

WebRTC 主要用于音视频数据传输，它会成为指纹的主要原因在于使用 WebRTC 可以直接获取本机的内网 IP 和外网 IP。当 WebRTC 在建立对等连接(peer-to-peer connection)时，需要找到通信的最佳路径，这涉及穿透 NAT(网络地址转换)和防火墙。为了解决这些问题，WebRTC 会尝试通过各种方法获取对等端的 IP 地址，包括内网 IP(局域网 IP)和外网 IP(公共 IP)。

WebRTC 在获取 IP 时，会使用交互式连接建立(Interactive Connectivity Establishment，ICE)协议来寻找和建立连接。ICE 协议会收集和交换多个“候选者”(candidate)，这些候选者包括不同的 IP 地址和端口，以找到最佳的连接路径。

这些候选者主要分为 3 种，分别是主机候选者(host candidate)设备的本地 IP 地址(内网 IP)；反射候选者(server reflexive candidate)通过 STUN(Session Traversal Utilities for NAT)服务器获取的外网 IP 地址；中继候选者(relay candidate)通过 TURN 服务器中继的 IP 地址。

(1) 主机候选者(host candidate)。

主机候选者是设备的本地 IP 地址(即内网 IP)。这是设备在本地网络中的唯一标识符。一般来说,获取的局域网 IP 的格式如 192.168.x.x 或 10.x.x.x 一样。主机候选者通常用于同一局域网内的直接连接。

(2) 反射候选者(server reflexive candidate)。

反射候选者通过 STUN 服务器获取设备外网 IP 地址。STUN 服务器帮助设备了解其在公共互联网上的 IP 地址和端口。通过反射候选者获取的是本地的外网 IP,如 203.0.113.x。

(3) 中继候选者(relay candidate)。

中继候选者的 IP 地址来自 TURN 服务器,这种服务器在双方无法直接通信时充当了中继节点。中继 IP 地址通常属于 TURN 服务器,而非用户的实际设备。中继传输确保了即使在严格的 NAT 或防火墙环境后,连接也能成功建立。

如果想通过 WebRTC 来获取本地 IP 地址,可以使用如下 JavaScript 代码:

```
async function getLocalIPs(callback) {
    const rtc = new RTCPeerConnection();
    rtc.createDataChannel("");
    //必须创建一个数据通道,否则 onicecandidate 不会被调用
    rtc.onicecandidate = (event) => {
        if (event.candidate) {
            console.log("ICE 候选:"+event.candidate.candidate);
            const ipRegex =
      /([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-fA-F0-9:]+(:[a-fA-F0-9:]+)+)/g;
            const ipMatches = event.candidate.candidate.match(ipRegex);
            if (ipMatches) {
                ipMatches.forEach(callback);
            }
        }
    };
    try {
        const offer = await rtc.createOffer();
        await rtc.setLocalDescription(offer);
    } catch (error) {
        console.error("Error creating offer:", error);
    }
}
//使用回调函数处理找到的 IP 地址
getLocalIPs((ip) => {
    console.log('找到的 IP 地址:', ip);
});
```

在浏览器的开发者工具中运行上面这段 JavaScript 代码之后,可以打印出本机的内网 IP,如下所示:

```
ICE 候选: candidate: 1970872015 1 udp 2113937151 198.18.0.1 62226 typ host
generation 0 ufrag YIU/ network-cost 999
找到的 IP 地址: 198.18.0.1
```

如果想要获取外网 IP,则需要借助 STUN 服务器来完成这个任务。STUN 服务器可以帮助设备了解其在公共互联网上的 IP 地址。以下是获取本机外网 IP 的代码,它使用免费的 STUN 服务器来获取外网 IP 地址:

```
async function getPublicIP(callback) {
    //使用一个免费的 STUN 服务器
    const rtc = new RTCPeerConnection({
        iceServers: [{ urls: "stun:stun.l.google.com:19302" }]
    });
    rtc.createDataChannel("");       //创建数据通道
    rtc.onicecandidate = (event) => {
        if (event.candidate) {
            console.log("ICE 候选者:", event.candidate.candidate);
            const ipRegex =
    /([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-fA-F0-9:]+(:[a-fA-F0-9:]+)+)/g;
            const ipMatches = event.candidate.candidate.match(ipRegex);
            if (ipMatches) {
                ipMatches.forEach(callback);
            }
        }
    };
    try {
        const offer = await rtc.createOffer();
        await rtc.setLocalDescription(offer);
    } catch (error) {
        console.error("Error creating offer:", error);
    }
}
//使用回调函数处理找到的 IP 地址
getPublicIP((ip) => {
    console.log('找到的外网 IP 地址:', ip);
});
```

这里使用了一个免费的 STUN 服务器(stun.l.google.com:19302),该服务器会帮助获取设备的公共 IP 地址。将其在浏览器的开发者工具中运行,可以得到以下结果:

```
ICE 候选者: candidate:1518449297 1 udp 1677729535 192.99.152.x 11174 typ srflx
raddr 198.18.0.1 rport 51784 generation 0 ufrag 4skb network-cost 999
找到的外网 IP 地址: 192.99.152.x
```

通过 WebRTC 的接口,网站可以直接获取到用户的内网 IP 地址和外网 IP 地址,接下来,只需要将其和用户的请求 IP 相比对,就可以判断 IP 地址是否是使用的代理 IP。因此,在进行 WebRTC 指纹信息的定制的时候,需要确保外网 IP 信息和代理 IP 一致。

5.1.2　WebRTC 内网 IP 定制

在 5.1.1 节通过 WebRTC 获取本机内网 IP 地址时，可以看出是每找到一个新的 ICE 候选者就从候选者字符串中提取 IP 地址，并将其传递给回调函数。

综上所述，要对 Chromium 源码中的 WebRTC 内网 IP 进行定制，需要找到候选者初始化的地方。先来确定一下 WebRTC 的 JavaScript 绑定函数的所在位置，WebRTC 作为一个数据传输通信的工具，应当属于独立的模块，peerconnection 是专门处理 WebRTC 功能的子模块。直接在 src\third_party\blink\renderer\modules\peerconnection 目录下找到 ICE 候选者相关的文件 rtc_ice_candidate.cc 进行定制修改即可。

该文件中关于候选者的部分源码如下：

```
String RTCIceCandidate::candidate() const {
  return platform_candidate_->Candidate();
}

String RTCIceCandidate::sdpMid() const {
  return platform_candidate_->SdpMid();
}

String RTCIceCandidate::address() const {
  return platform_candidate_->Address();
}

String RTCIceCandidate::protocol() const {
  return platform_candidate_->Protocol();
}

absl::optional<uint16_t> RTCIceCandidate::port() const {
  return platform_candidate_->Port();
}
```

回顾之前获取本机内网 IP 的 JavaScript 代码，它主要依赖以下函数：

```
rtc.onicecandidate = (event) => {
...
console.log("ICE 候选:"+event.candidate.candidate);
```

因此，较好的切入点是在 RTCIceCandidate::candidate 中对候选者的本地 IP 地址进行替换。定制代码先从传递的 JSON 对象中读取定制的本地 WebRTC IP 地址，然后检查候选者的类型是否为主机类型(host)，如果是，则使用正则表达式替换候选者字符串中的 IP 地址。具体代码如下：

```
//ruyi
std::string webrtc_private = *(json_reader->GetDict().FindString("webrtc_
private"));
```

```
if(platform_candidate_->Type()=="host"){
    String _candidate = platform_candidate_->Candidate();
    std::regex ip_regex("\\b(?:[0-9]{1,3}\\.){3}[0-9]{1,3}\\b");
    std::string dest = std::string(_candidate.Utf8().data());
    std::string replaced =regex_replace(dest, ip_regex,
        webrtc_private,std::regex_constants::format_first_only);
    return String(replaced);
        }
    }
//ruyi end
```

代码中的正则表达式\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b 匹配形如 xxx.xxx.xxx.xxx 的 IPv4 地址，在 WebRTC 获取本地 IP 的时候，候选者信息的格式具体如下：

```
a=candidate:1320071804 1 udp 2113937151 198.18.0.1 53544 typ host
  generation 0 network-cost 999
```

所以在判断候选者类型是 host 时，对其中的 IP 地址进行定制，即可修改本机的 WebRTC 内网 IP。

5.1.3 WebRTC 外网 IP 定制

定制修改外网 IP 的基本流程和修改内网 IP 的是一样的，只是在类型上有所区别。从 WebRTC 的基本概念中可以得知，候选者包括主机候选者、反射候选者和中继候选者。其中，涉及外网 IP 获取的是反射候选者，从而需要在获取内网 IP 的函数中进行类型判断。当类型为 srflx 时，要对候选者信息进行定制修改。

反射候选者的信息格式具体如下：

```
a=candidate:224160942 1 udp 1677729535 192.99.152.x 21108 typ srflx raddr 198.
18.0.1 rport 53544
```

可以将其中的 IP 地址全部更改为定制化的外网 IP。当候选者的类型为 srflx 时，代码会替换候选者字符串中的外网 IP 地址以保护用户隐私，实现代码如下：

```
if(platform_candidate_->Type()=="srflx"){
    String _candidate = platform_candidate_->Candidate();
    std::regex ip_regex("\\b(?:[0-9]{1,3}\\.){3}[0-9]{1,3}\\b");
    std::string dest = std::string(_candidate.Utf8().data());
    std::string replaced =regex_replace(dest, ip_regex,
          webrtc_public,std::regex_constants::format_first_only);
    std::regex ip_regex_after
        (R"(raddr (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) rport)");
    std::string replaced2 =regex_replace(replaced, ip_regex_after,
        webrtc_public,std::regex_constants::format_first_only);
    return String(replaced2);
}
```

正则表达式\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b 匹配形如 xxx.xxx.xxx.xxx 的 IPv4 地址。正则表达式 raddr (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) rport 匹配形如 raddr xxx.xxx.xxx.xxx rport 的字符串。通过两次正则表达式匹配，可以将其中存在的 IP 地址全部更改为定制的外网 IP 地址。

最后，本节选择的内网 IP 地址为 192.168.1.1，外网 IP 地址为 183.1.1.1。如图 5-1 所示，可以到在线指纹检测网站 browserleaks.com/webrtc 查看 WebRTC 指纹是否被修改成功。

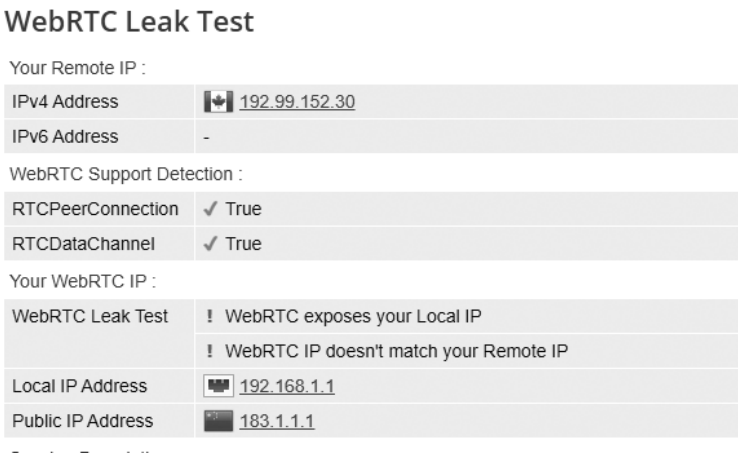

图 5-1　WebRTC 指纹

5.2　浏览器 navigator 指纹

5.2.1　navigator 指纹概述

在 JavaScript 中，navigator 是一个重要的对象，它提供了关于用户浏览器和操作系统的详细信息。通过 navigator，开发者可以检测浏览器的功能、状态和用户设置。下面详细介绍其用途。

navigator 是一个全局对象，可以通过 window.navigator 来访问。它包含多个属性和方法，用于获取有关浏览器和用户环境的信息，具体如下。

(1) navigator.productSub。

其用法如下：

```
console.log(navigator.productSub);
```

用途为返回一个表示浏览器子版本的字符串。在现代浏览器中，这个值通常是固定的。本机返回值为 20030107。这个属性很少在实际应用中使用，主要是为了保持与旧版本浏览器的兼容性。

(2) navigator.vendor。

其用法如下：

```
console.log(navigator.vendor);
```

用途为返回一个表示浏览器供应商名称的字符串。本机返回值为 Google Inc.。navigator.vendor 可以用来检测浏览器的供应商，可能会用在某些特定供应商相关的优化或调整中。

(3) navigator.vendorSub。

其用法如下：

```
console.log(navigator.vendorSub);
```

用途为返回一个表示浏览器供应商的子信息的字符串。在现代浏览器中，这个值通常是空字符串。本机返回值为“#”。这个属性在实际应用中很少使用，主要用于保持与旧版浏览器的兼容性。

(4) navigator.platform。

其用法如下：

```
console.log(navigator.platform);
```

用途为返回一个表示浏览器运行平台的字符串。本机返回值为 Win32。navigator.platform 可以用于检测用户的操作系统平台，从而提供相应的优化和调整。例如，针对不同操作系统提供不同的下载链接或提示信息。

(5) navigator.cookieEnabled。

其用法如下：

```
console.log(navigator.cookieEnabled);
```

用途为返回一个布尔值，指示浏览器是否启用了 Cookie。本机返回值是 true(启用)。它可以检测浏览器是否支持并启用了 Cookie，从而决定是否使用 Cookie 来存储用户会话信息或偏好设置。

(6) navigator.webdriver。

其用法如下：

```
console.log(navigator.webdriver);
```

用途为返回一个布尔值，指示浏览器是否由自动化工具(如 Selenium WebDriver)控制。

本机返回值为 false(未由自动化工具控制)。该属性可以用于检测网页是否在自动化测试环境中运行，这可以帮助开发者在自动化测试期间启用或禁用某些功能，或者用于反自动化检测来防止滥用。

(7) navigator.languages。

其用法如下：

```
console.log(navigator.languages);
```

用途为返回一个包含用户首选的语言的数组。本机返回值为[zh-CN,zh]。它可以用于根据用户首选的语言提供本地化内容和翻译服务。例如,根据首选语言设置网站的默认语言。

navigator 的这些属性和方法提供了丰富的信息,开发者可以利用这些信息来优化网页的表现和功能。下面是一个综合示例,展示如何使用这些属性和方法,代码如下:

```
console.log("Product Sub:", navigator.productSub);
console.log("Vendor:", navigator.vendor);
console.log("Vendor Sub:", navigator.vendorSub);
console.log("Platform:", navigator.platform);
console.log("Cookies Enabled:", navigator.cookieEnabled);
console.log("WebDriver Controlled:", navigator.webdriver);
console.log("Preferred Languages:", navigator.languages);
if (navigator.platform.startsWith("Win")) {
    console.log("提供 Windows 下载链接");
} else if (navigator.platform.startsWith("Mac")) {
    console.log("提供 macOS 下载链接");
} else if (navigator.platform.startsWith("Linux")) {
    console.log("提供 Linux 下载链接");
}
//示例应用:检查 Cookie 是否启用
if (navigator.cookieEnabled) {
    console.log("Cookie 已启用,可以存储用户会话信息");
} else {
    console.log("Cookie 未启用,请提示用户启用 Cookie 以获得最佳体验");
}
```

通过上述示例,开发者可以更好地理解如何利用 navigator 提供的各种信息来提升用户体验和应用的功能性。

5.2.2　navigator 指纹定制

在 Chromium 的代码库中,src\third_party\blink\renderer\core\frame 文件夹包含了与框架(frame)和窗口(window)相关的核心功能的实现。navigator 及其相关信息位于该文件夹下是因为这些信息与网页的框架和窗口环境密切相关。

navigator.cc 文件包含了上述接口,代码如下:

```
String Navigator::productSub() const {
    return "20030107";
}
String Navigator::vendor() const {
  return "Google Inc.";
```

```
}
String Navigator::vendorSub() const {
  return "";
}
String Navigator::platform() const {
  if (!DomWindow())
    return NavigatorBase::platform();
  const String& platform_override =
  DomWindow()->GetFrame()->GetSettings()->GetNavigatorPlatformOverride();
  return platform_override.empty() ? NavigatorBase::platform()
                                   : platform_override;
}
bool Navigator::cookieEnabled() const {
  if (!DomWindow())
    return false;
  Settings* settings = DomWindow()->GetFrame()->GetSettings();
  if (!settings || !settings->GetCookieEnabled())
    return false;
  return DomWindow()->document()->CookiesEnabled();
}
bool Navigator::webdriver() const {
  if (RuntimeEnabledFeatures::AutomationControlledEnabled())
    return true;
  bool automation_enabled = false;
  probe::ApplyAutomationOverride(GetExecutionContext(),
    automation_enabled);
  return automation_enabled;
}
String Navigator::GetAcceptLanguages() {
  if (!DomWindow())
    return DefaultLanguage();
  return DomWindow()
      ->GetFrame()
      ->GetPage()
      ->GetChromeClient()
      .AcceptLanguages();
}
```

由于其中的指纹信息修改的逻辑大多是一致的，因此可以封装一个公用的函数用于修改对应的指纹信息，代码如下：

```
String getfp_string(std::string name) {
    const base::CommandLine* ruyi_command_line = base::CommandLine::
      ForCurrentProcess();
    if (ruyi_command_line->HasSwitch(blink::switches::kRuyi)) {
```

```
            const std::string ruyi_fp = ruyi_command_line->
              GetSwitchValueASCII(blink::switches::kRuyi);
            absl::optional<base::Value> json_reader = base::JSONReader::
                 Read(ruyi_fp);
            std::string product_fp = * (json_reader->GetDict().
                FindString(name));
            return String(product_fp);
        }
        return "";
    }
```

该函数会在传递的 JSON 对象中查找需要的指纹参数，然后以字符串形式返回，如果没有找到，就返回空字符串。封装好之后，这个函数就可以被公用了。例如，如果要修改其中的 vendor 厂商信息，只需要传入 vendor 就可以得到返回值，之后再根据返回值进行判断，代码如下：

```
    //ruyi
    String result = getfp_string("vendor");
    if (result != "") {
        return result;
    }
    //ruyi 结束
```

navigator.cc 文件中的其余函数 webdriver、cookieEnabled 和 GetAcceptLanguages 等也可以按照该示例直接进行修改。

5.3　时区时间信息

5.3.1　时区时间信息概述

地球被划分为 24 个时区，每个时区大约对应 15 度的经度。每个时区通常以 UTC 为基准，通过增加或减少若干小时来表示本地时间。UTC(Universal Time Coordinated，协调世界时)是全球时间的标准。所有时区都以 UTC 为基准，通过加减偏移量来表示本地时间。例如，UTC+8 表示比 UTC 早 8 小时，UTC-5 表示比 UTC 晚 5 小时。此外，一些时区会根据夏令时(DST)进行调整，通常在夏季时将时钟拨快一小时。

以下是一些常见时区及其与 UTC 的对应关系。

(1) UTC-12:00：表示比 UTC 时间晚 12 小时。如 Baker Island。

(2) UTC-05:00：表示比 UTC 时间晚 5 小时。如美国东部时间(EST，标准时间)和美国中部时间(Central Standard Time，CST，夏令时)。

(3) UTC+00:00：表示与 UTC 时间相同。如格林尼治标准时间(GMT)和冰岛时间。

(4) UTC+08:00：表示比 UTC 时间早 8 小时。如中国标准时间(China Standard Time，CST)和新加坡时间。