基础 3–扣代码补环境

Hook

目的

js hook 目的是找到函数入口以及一些参数变化，便于分析 js 逻辑，但是 hook 的能力远不及此。我们只借助其寻找函数入口。

底层原理

客户端拥有 js 的最高解释权，可以决定在任何时候注入 js 而服务器无法左右，只能通过检测和混淆手段令 hook 难度加大，但是却无法阻止。

定义

Hook 技术又叫做钩子函数，在系统没有调用该函数之前，钩子程序就先捕获该消息，钩子函数先得到控制权，这时钩子函数既可以加工处理（改变）该函数的执行行为，还可以强制结束消息的传递。简单来说，就是把系统的程序拉出来变成我们自己执行代码片段。

在 js 中，系统程序可以指浏览器 API，也可以指代码中实现的一些方法等

分类：手动 hook，自动 hook

步骤

1. 寻找 hook 点
2. 编写 hook 逻辑
3. Hook 公式

```JavaScript
函数Hook公式
old_func = func⍰func = function(argument){
```

```
  my task;
  return old_func .apply(argument)
}
func.prototype..... = .......
func ：要hook的函数
```

JavaScript

```
对象中的Hook公式
old_func = func⍁func = function(argument){
  my task;
  return old_func .apply(argument)
}
func.prototype..... = .......
func ：要hook的函数
```

弊端和缺陷

1. 函数 hook 一般情况下不会出现 hook 失败的情况，只有可能是 **proto** 模拟的不好导致被检测到了。
2. 属性 hook 当所有网站的逻辑都采用 Object.defineProperty 绑定后，属性 hook 就会失效。暂时没有发现好的解决方案。
3. 这里还有一种我没讲的 hook，叫局部 hook，原理是一样的，只不过需要在进入作用域的那一刻进行 hook。

Hook 插件：油猴脚本

油猴插件的文档攻略

官方文档 https://www.tampermonkey.net/documentation.php?ext=dhdg#api:GM_openInTab

方楠的一个入门版补充：入门篇 | 油猴开发指南 (scriptcat.org)

常用的参数

纯文本

```
@antifeature ：官方文档的意思：开发人员是否允许别人把脚本货币化。
@noframes：在主页上运行而不在iframes上运行。
@unwrap：卵用没有，官方文档的意思：在chrome上，不需要它，自动被忽略了（淦！）
@nocompat：一般默认，官方文档写了好长，大概的意思指支持标记浏览器。如：
@nocompat Chrome，这样就不能在火狐，浏览器运行它了。逆向中最重要的参数！！！！
@run-at： 指定油猴脚本在什么时候执行，默认是在所有js加载完成后执行。（那hook脚本还有
个屁用啊！）
参数：
@run-at document-start ：脚本尽快注入（相当于script断点之后，进入页面一瞬间注入）
@run-at document-body ：如果页面body元素存在，则注入
@run-at document-end ：脚本将在DOMContentLoaded事件发生注入。
@run-at document-idle：默认值。脚本将在DOMContentLoaded事件发生之后才注入。
@run-at context-menu：如果在浏览器上下文菜单中单击脚本（仅限于基于桌面Chrome的浏览器），则会注入脚本
```

范例：万能 hook-eval 函数

纯文本

```
// ==UserScript==
// @name         万能hook eval函数
// @namespace    http://tampermonkey.net/
// @version      0.1
// @description  eval-everything
// @author       An-lan
// @include      *
// @grant        none
// @run-at      document-start
// ==/UserScript==
(function() {
    alert('hook success');
    var eval_bk = eval;
    eval = function(val){
       console.log(val)
       return eval_bk(val);
    };
    eval.toString = function(){
         return "function eval() { [native code] }"
    };
    eval.length = 1;
})();
```

Hook 的一些补充

1. 局部函数的 hook
2. 全局变量的 hook
3. 一些常用的 hook 逻辑

```
纯文本

hook eval
hook Function
hook JSON.stringify
JSON.parse
hook cookie
hook window对象
```

验证码实现原理

session 与 cookie

cookie 是服务端识别客户的唯一标识的依据，客户在访问网站时候，服务端为了记住这个客户，会在服务端按照它的规则制作一个 cookie 数据，会将这个 cookie 数据保留在服务端一段时间，同时会给客户的一份它自己保留，这样就无需每次都要登录来认证自己了。
先来了解几个概念：

> 无状态的 HTTP 协议

协议是指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则，超文本传输协议(HTTP)是一种通信协议，它允许将超文本标记语言(HTML)文档从 Web 服务器传送到客户端的浏览器。HTTP 协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话。

> 会话（Session）跟踪：

会话，指用户登录网站后的一系列动作，比如浏览商品添加到购物车并购买。会话（Session）跟踪是 Web 程序中常用的技术，用来跟踪用户的整个会话。常用的会话跟踪技术是 Cookie 与 Session。Cookie 通过在客户端记录信息确定用户身份，Session 通过在服务器端记录信息确定

> 验证码的实现机理

从爬虫（验证码）角度来看，是以 set-cookie 建立了连接（以抓包情况演示验证码）（当然了，不一定是以 set-cookie 的形式，还有可能是以接口 API 形式返回，反正无论如何，必须让服务器给你标记一下）

js 逆向核心：抠代码 & 补环境

境界划分...

A. 缺啥补啥，稳扎稳打
B. 见文知义，化繁为简
C. 了然于胸，如履平地

抠代码优缺点

优点

1. 执行效率高
2. 并发能力强
3. 能感受到进度（抠出一行是一行）
4. 只要有耐心，笨办法也能成大事

缺点

1. 比较吃经验，只有勤学苦练才能更快
2. 若想精通则对 js 基础要求较高（尤其是对于浏览器 API 的掌握程度）
3. 网站即使微调，js 可能就会失效
4. 做不到完全还原浏览器的某些值，对风控响应不够及时。

补环境优缺点

优点

1. 复用性强、开发速度快
2. 有现成的库可供调用（如 jsdom）
3. 简单网站，相对于抠代码，对熟练度要求更低
4. 仅需对服务器的混淆代码做少量处理即可。

缺点

1. 占用资源大，计算速度可能慢
2. 若想高并发可能需要开发多种浏览器环境，增加时间成本
3. 对于极复杂的网站，可能更靠玄学。无法感受到进度的变化
4. 若想精通则对需对 node 指纹和浏览器 API 都足够精通

小结

1. 难的网站这两种方法都难。反而抠代码由于可以看到进度不容易轻易放弃。
2. 简单的网站补环境更爽，可能会秒破而且耗费不了多少资源。

浏览器指纹

定义

什么是浏览器指纹，为啥百度之前找不到?

因为现在的指纹都是安全厂商/逆向用户 意淫出来的名词，在之前我们都称之为 webAPI

浏览器 API 与之对应的是 nodeAPI，所以浏览器指纹这节课就很好讲了，直接两个网站怼脸上

```
纯文本

nodejsAPI http://nodejs.cn/api/path.html
webAPI  https://developer.mozilla.org/zh-CN/docs/Web/API
```

浏览器指纹 – web-api

全局相关： window，document
环境相关： navigator(包括经纬度在内都在这个接口里)，screen，history
请求相关：XMLHttpRequest fetch worker
dom 相关：canvas ，所有对 dom 节点操作，包括 jquery 等三方库以及自设导入接口
数据库相关： Storage IndexedDB cookie
其他：caches WebGL AudioContext WebRTC

浏览器指纹 – node-api

如果从安全开发者角度来看的话，找到 node 与 web 的异常地方，就可以实现逻辑检测。我也只是列举一点儿。

1. 全局变量 global
2. 2. 导包引擎 require 【这一条很危险】
3. 3. 可被重写的全局
4. 4. 绝大多数的 webAPI
5. 5. 全部的 dom 节点

调试技巧 – 插桩法

插桩法的使用需要技巧，流程如下

1. 中间人攻击
2. 找桩点，右键打桩
3. 看日志，分析正常执行逻辑
4. 抠出核心逻辑

某些极端情况下，插桩法可以实现环境自吐，也就是环境自吐法。实际上，某某宣传的环境自吐法，只是插桩的特殊情况罢了。对于 vmp 来讲，插桩法是一种行之有效的处理方案