什么是 Hook?

Hook 中文译为钩子，Hook 实际上是 Windows 中提供的一种用以替换 DOS 下“中断”的系统机制，Hook 的概念在 Windows 桌面软件开发很常见，特别是各种事件触发的机制，在对特定的系统事件进行 Hook 后，一旦发生已 Hook 事件，对该事件进行 Hook 的程序就会收到系统的通知，这时程序就能在第一时间对该事件做出响应。在程序中将其理解为“劫持”可能会更好理解，我们可以通过 Hook 技术来劫持某个对象，把某个对象的程序拉出来替换成我们自己改写的代码片段，修改参数或替换返回值，从而控制它与其他对象的交互。

通俗来讲，Hook 其实就是拦路打劫，马邦德带着老婆，出了城，吃着火锅，还唱着歌，突然就被麻匪劫了，张麻子劫下县长马邦德的火车，摇身一变化身县长，带着手下赶赴鹅城上任。Hook 的过程，就是张麻子顶替马邦德的过程。

JS 逆向中的 Hook

在 JavaScript 逆向中，替换原函数的过程都可以被称为 Hook，以下先用一段简单的代码理解 Hook 的过程：

```
function a() {
  console.log("I'm a.");
}

a = function b() {
  console.log("I'm b.");
};

a()  // I'm b.
```

直接覆盖原函数是最简单的做法，以上代码将 a 函数进行了重写，再次调用 a 函数将会输出 `I'm b.`，如果还想执行原来 `a` 函数的内容，可以使用中间变量进行储存：

```
function a() {
  console.log("I'm a.");
}

var c = a;

a = function b() {
  console.log("I'm b.");
};

a()  // I'm b.
c()  // I'm a.
```

此时，调用 a 函数会输出 `I'm b.`，调用 c 函数会输出 `I'm a.`。

这种原函数直接覆盖的方法通常只用来进行临时调试，实用性不大，但是它能够帮助我们理解 Hook 的过程，在实际 JS 逆向过程中，我们会用到更加高级一点的方法，比如 `Object.defineProperty()`。

Object.defineProperty()

基本语法：`Object.defineProperty(obj, prop, descriptor)`，它的作用就是直接在一个对象上定义一个新属性，或者修改一个对象的现有属性，接收的三个参数含义如下：

`obj`：需要定义属性的当前对象；

`prop`：当前需要定义的属性名；

`descriptor`：属性描述符，可以取以下值：

通常情况下，对象的定义与赋值是这样的：

```
var people = {}  对象
people.name = "Bob"
people["age"] = "18"

console.log(people)
// { name: 'Bob', age: '18' }
```

使用 `Object.defineProperty()` 方法：

```
var people = {}

Object.defineProperty(people, 'name', {
   value: 'Bob',
   writable: true  // 是否可以被重写
})

console.log(people.name)  // 'Bob'

people.name = "Tom"
console.log(people.name)  // 'Tom'
```

在 Hook 中，使用最多的是存取描述符，即 get 和 set。

get：属性的 getter 函数，如果没有 getter，则为 undefined，当访问该属性时，会调用此函数，执行时不传入任何参数，但是会传入 this 对象

set：属性的 setter 函数，如果没有 setter，则为 undefined，当属性值被修改时，会调用此函数，该方法接受一个参数，也就是被赋予的新值，会传入赋值时的 this 对象

用一个例子来演示：

```
var people = {
  name: 'Bob',
};
var count = 18;

// 定义一个 age 获取值时返回定义好的变量 count
Object.defineProperty(people, 'age', {

  get: function () {
```

```
    console.log('获取值! ');
    return count;
  },
  set: function (val) {
    console.log('设置值! ');
    count = val + 1;
  },
});

console.log(people.age);
people.age = 20;
console.log(people.age);
```

输出：

```
获取值!
18
设置值!
获取值!
21
```

通过这样的方法，我们就可以在设置某个值的时候，添加一些代码，比如 `debugger;`，让其断下，然后利用调用栈进行调试，找到参数加密、或者参数生成的地方，需要注意的是，网站加载时首先要运行我们的 Hook 代码，再运行网站自己的代码，才能够成功断下，这个过程我们可以称之为 Hook 代码的注入，以下将介绍几种主流的注入方法。

Hook 注入的几种方法

Fiddler 插件注入

TamperMonkey 注入(推荐)

常用 Hook 代码

hook cookie

```
(function () {

  var cookieTemp = '';

  Object.defineProperty(document, 'cookie', {
    set: function (val) {
      if (val.indexOf('xxx') != -1) {
        debugger;
      }
      console.log('Hook捕获到cookie设置->', val);
      cookieTemp = val;
      return val;
    },
    get: function () {

      return cookieTemp;
```

```
    },
  });
})();
```

Hook Header

Header Hook 用于定位 Header 中关键参数生成位置，以下代码演示了当 Header 中包含`xxx`关键字时，则插入断点：

```
(function () {
    var org = window.XMLHttpRequest.prototype.setRequestHeader;
    window.XMLHttpRequest.prototype.setRequestHeader = function (key, value) {
        if (key == 'xxxx') {
            debugger;
        }
        return org.apply(this, arguments);
    };
})();
```

Hook URL

URL Hook 用于定位请求 URL 中关键参数生成位置，以下代码演示了当请求的 URL 里包含 `login` 关键字时

```
(function () {
    var open = window.XMLHttpRequest.prototype.open;
    window.XMLHttpRequest.prototype.open = function (method, url, async) {
        if (url.indexOf("login") != 1) {
            debugger;
        }
        return open.apply(this, arguments);
    };
})();
```

Hook JSON.stringify

`JSON.stringify()` 方法用于将 JavaScript 值转换为 JSON 字符串，在某些站点的加密过程中可能会遇到，以下代码演示了遇到 `JSON.stringify()` 时

```
(function() {
    var stringify = JSON.stringify;
    JSON.stringify = function(params) {
        console.log("Hook JSON.stringify —> ", params);
        debugger;
        return stringify(params);
    }
})();
```

Hook JSON.parse

`JSON.parse()` 方法用于将一个 JSON 字符串转换为对象，在某些站点的加密过程中可能会遇到，以下代码演示了遇到 `JSON.parse()` 时

```
(function() {
    var parse = JSON.parse;
    JSON.parse = function(params) {
        console.log("Hook JSON.parse —> ", params);
        debugger;
        return parse(params);
    }
})();
```

Hook eval

JavaScript `eval()` 函数的作用是计算 JavaScript 字符串，并把它作为脚本代码来执行。如果参数是一个表达式，`eval()` 函数将执行表达式。如果参数是 Javascript 语句，`eval()` 将执行 Javascript 语句，经常被用来动态执行 JS。以下代码执行后，之后所有的 `eval()` 操作都会在控制台打印输出将要执行的 JS 源码

```
(function() {
    // 保存原始方法
    window.__cr_eval = window.eval;
    // 重写 eval
    var myeval = function(src) {
        console.log(src);
        console.log("=============== eval end ===============");
        debugger;
        return window.__cr_eval(src);
    }
    // 屏蔽 JS 中对原生函数 native 属性的检测
    var _myeval = myeval.bind(null);
    _myeval.toString = window.__cr_eval.toString;
    Object.defineProperty(window, 'eval', {
        value: _myeval
    });
})();
```